Implementacija proširene stvarnosti u mobilnim aplikacijama za Android

Alagić, Aldin

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:458120

Rights / Prava: Attribution-NoDerivs 3.0 Unported/Imenovanje-Bez prerada 3.0

Download date / Datum preuzimanja: **2024-03-28**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Aldin Alagić

IMPLEMENTACIJA PROŠIRENE STVARNOSTI U MOBILNIM APLIKACIJAMA ZA ANDROID

DIPLOMSKI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Aldin Alagić

Matični broj: 44924–16R

Studij: Informacijsko i programsko inženjerstvo

IMPLEMENTACIJA PROŠIRENE STVARNOSTI U MOBILNIM APLIKACIJAMA ZA ANDROID

DIPLOMSKI RAD

Mentor:

Izv. prof. dr. sc. Zlatko Stapić

Varaždin, rujan 2021.

Aldin Alagić

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema ovog rada je teorijski pregled koncepata prepoznavanja objekata i proširene stvarnosti te sistematizacija tehnologija za implementaciju istih u sklopu mobilnih aplikacija za Android platformu. Mogućnosti tehnologija za implementaciju prepoznavanja objekata i proširene stvarnosti su pobliže objašnjene putem programskih primjera koji su izrađeni korištenjem programskog jezika Kotlin i besplatnih biblioteka kao što su TensorFlow Lite i ARCore. Rad sadrži i praktični dio u sklopu kojeg je detaljno razrađen i prikazan cijeli proces razvoja Kotlin mobilne aplikacije koja koristi proširenu stvarnost. Prilikom izrade praktičnog dijela primijenjen je Android Jetpack, skup dobrih praksi dizajna i razvoja mobilnih aplikacija.

Ključne riječi: Prepoznavanje objekata; Proširena stvarnost; Kotlin; Android; Jetpack; TensorFlow; Aplikacija;

Sadržaj

1. Uvod 1
2. Metode i tehnike rada 2
3. Prepoznavanje objekata 3
3.1. Pristupi problemu 4
3.2. Povijesni razvoj 5
3.2.1. Prepoznavanje objekata temeljeno na strojnom učenju 6
3.2.2. Prepoznavanje objekata temeljeno na dubokom učenju 7
3.3. Područja primjene 10
3.4. Problemi i izazovi 12
4. Proširena stvarnost 13
4.1. AR uređaji 14
4.2. Tipologija 16
4.3. Povijesni razvoj 18
4.4. Područja primjene 21
4.4.1. Marketing i prodaja 21
4.4.2. Video igre 22
4.4.3. Medicina 23
4.5. Problemi i izazovi 24
5. Proširena stvarnost u Android uređajima 26
5.1. Kotlin 26
5.1.1. Programski primjeri 27
5.2. Android Jetpack 31
5.2.1. Arhitekturne komponente 32
5.2.2. Komponente korisničkog sučelja 33
5.2.3. Komponente ponašanja 34
5.2.4. Komponente osnove 34
5.3. Programske knjižnice 35

5.3.1. ARCore 35
5.3.2. CameraX 36
5.3.3. TensorFlow Lite 37
6. Implementacija proširene stvarnosti 38
6.1. Opis mobilne aplikacije 38
6.2. Dizajn rješenja 39
6.3. Razvoj mobilne aplikacije 42
7. Zaključak 56
Popis literature 57
Popis slika 63
Popis tablica 64
Popis programskih kodova 65
Prilozi 66

1. Uvod

Proširena stvarnost (engl. Augmented reality, AR) je u posljednjih desetak godina postala izrazito popularna. Naime, u relativno kratkom periodu čovjekov život se značajno promijenio, a kao jedan od najvećih uzroka toga zasigurno je sveprisutnost i dostupnost tehnologije. U takvim okolnostima je razumljivo da čovjek ima potrebu za pronalaskom što jednostavnijeg i pristupačnijeg načina interakcije s digitalnim svijetom današnjice. Upravo zbog toga koncept proširene stvarnost postaje istraživački veoma aktivno područje. Osjećaj proširene stvarnost se najčešće postiže ispisom informacija o sadržajima koji nas okružuju kao i stvaranjem novih virtualnih objekata, a u pojedinim slučajevima i cijelih virtualnih okolina koje su vidljive samo upotrebom određenih digitalnih uređaja. Pritom je bitno napomenuti da s ovakvim sadržajima nije moguće ostvariti fizičku interakciju.

Ideja proširene stvarnosti svoje mjesto je pronašla u brojnim područjima ljudskog djelovanja kao što su zabava, edukacija, medicina, marketing i mnogim drugim. Za temu ovog rada posebno je važna primjena proširene stvarnosti u svrhu otkrivanja i prepoznavanja objekata (engl. Object detection). Riječ je o složenom zadatku područja umjetne inteligencije zvanog računalni vid (engl. Computer vision) koji se bavi prepoznavanjem predmeta, životinja i ljudi. Danas, posebno je razvijena primjena prepoznavanja objekata u svrhu otkrivanja lica osoba, pješaka i okoline.

Kada se govori o načinima implementacije proširene stvarnosti, najznačajnije su mobilne aplikacije zbog svoje velike dostupnosti i jednostavnosti korištenja. Naime, većina osoba posjeduje pametne mobilne uređaje i bez njih ne može zamisliti svoju svakodnevnicu, a zahvaljujući brojnim novim tehnologijama savršeni su za implementaciju proširene stvarnosti. Jedna takva novija tehnologija je programski jezik Kotlin, koji bitno pojednostavljuje razvoj aplikacija za Android mobilne uređaje. Sve što je navedeno upućuje na izrazitu važnost i korisnost proširene stvarnosti, ali bitno je napomenuti da ova kao i svaka druga tehnologija ima i svoje nedostatke.

Cilj ovog rada je opisati teorijske postavke proširene stvarnosti i prepoznavanja objekata te potom napraviti sistematski pregled mogućnosti i tehnologija implementacije istih u mobilnim aplikacijama za Android platformu pisanih pomoću programskog jezika Kotlin. U praktičnom dijelu rada će biti detaljno prikazan proces razvoja i implementacije aplikacije koja koristi proširenu stvarnost.

2. Metode i tehnike rada

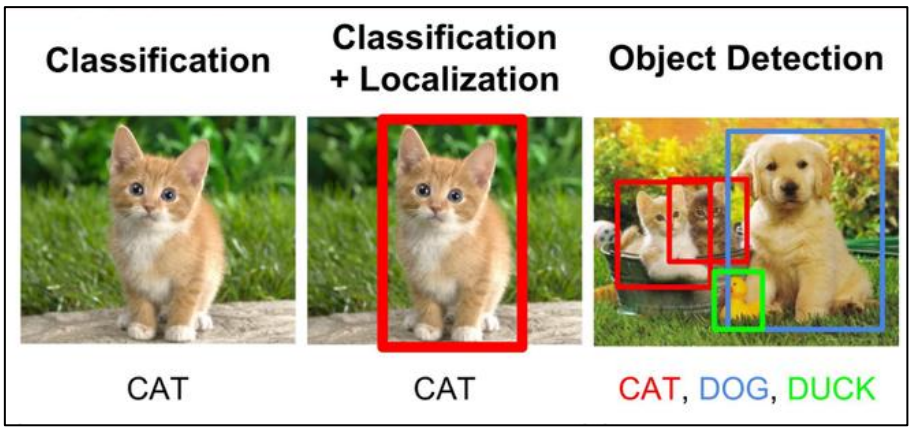
U prvom dijelu ovog rada napravljen je teorijski pregled prepoznavanja objekata i proširene stvarnosti te sistematizacija tehnologija implementacije istih u obliku mobilnih aplikacija za Android. Glavne metode istraživanja koje su korištene u ovom dijelu, kako bi se došlo do što bitnijih informacija te kako bi se objasnili spomenuti koncepti, su metode deskripcije i dedukcije, pri čemu su korišteni izvori preuzeti iz baza znanstvene literature kao što su Google Scholar, Hrčak, ResearchGate i dr. Prvi koncept koji je detaljno razrađen u ovom dijelu jeste prepoznavanje objekata. Ovdje se opisuje povijesni razvoj ove tehnologije, osnovne teorijske postavke te mogućnosti i načini implementacije. Potom slijedi detaljna razrada koncepta proširene stvarnosti. Ovaj dio započinje s opisom povijesnog razvoja tehnologije proširene stvarnosti i njenih osnovnih teorijskih postavki. Dodatno, napravljena je sistematizacija mogućnosti i tehnologija implementacije, s tim da je poseban naglasak stavljen na implementaciju proširene stvarnosti unutar mobilnih aplikacija korištenjem programskog jezika Kotlin te besplatnih knjižnica TensorFlow Lite, ML Kit i ARCore, što će biti potkrijepljeno programskim primjerima. Prilikom izrade ovog dijela korištena je metoda analize.

Nakon teorijskog dijela slijedi praktični dio rada unutar kojeg se postepeno opisuje razvoj i implementacija proširene stvarnost i detekcije objekata unutar Android mobilne aplikacije. Prilikom izrade mobilne aplikacije korišteno je razvojno okruženje Android Studio i programski jezik Kotlin. Dodatno, mobilna aplikacija je izrađena u skladu Android Jetpack, skupom dobrih praksi dizajna i razvoja mobilnih aplikacija.

3. Prepoznavanje objekata

Mnogi od nas ne shvaćaju u potpunosti složenost koja se krije iza našeg vida. Riječ je o jednom od pet osjeta koji nam omogućuje da prepoznajemo i otkrivamo svjetla, boje i oblike na različitim udaljenostima. Ljudsko oko je jedan od najsloženijih osjetilnih organa čovjeka. Ono ima mogućnost da percipira i obrađuje nekoliko desetina milijuna informacija o različitim oblicima i bojama u samo jednoj sekundi. Iz svega navedenog ne iznenađuje činjenica da to računalu predstavlja jedan od najsloženijih zadataka. Ovim problemom se bavi znanstvena i tehnološka disciplina koja se naziva računalni vid (engl. Computer vision). Konkretnije, riječ je o disciplini u kojoj se proučavaju i razvijaju sustavi koji se koriste za izvlačenje dodatnih informacija, odnosno informacija, koje računalu inače nisu dostupne, iz različitih vizualnih izvora kao što su slike i video zapisi [1]. Upravo je računalni vid ono što računalu daje sposobnost da razumije sadržaj koji se nalazi na slici ili videu.

Jedan od osnovnih zadataka računalnog vida jeste prepoznavanje i praćenje objekata. Prepoznavanje objekata je veoma složeno zbog velike raznolikosti objekata koji se nastoje percipirati. Prepoznavanje objekata je zapravo tehnika računalnog vida koja se bavi otkrivanjem objekata određene klase, kao što su ljudi, životinje, biljke, bolesti, vozila i sl. [2]. Osnovni cilj prepoznavanja objekata je razvoj modela i tehnika koje računalu pomažu da prepozna i klasificira objekte [3]. Bitno je napomenuti da se prepoznavanje objekata ne treba miješati s klasifikacijom objekata. Naime, klasifikacija objekata pruža informaciju koji je objekt prikazan na slici, dok prepoznavanje objekata pruža informaciju koji su objekti prikazani i gdje se oni točno nalaze, najčešće prikazivanjem okvira oko pojedinih objekata kao što je to prikazano na slici 1.



Slika 1. Razlika između klasifikacije i prepoznavanja objekata [5]

3.1. Pristupi problemu

Zbog velike kompleksnosti koja se krije iza preciznog prepoznavanja objekata očekivano je da će se vremenom pojaviti i više različitih pristupa ovom problemu. Trenutne vrste pristupa problemu prepoznavanju objekata su [2]:

- odozdo prema gore (engl. bottom-up approach),
- odozgo prema dolje (engl. top-down approach) i
- kombinacija ovih pristupa.

Pristup odozdo prema gore započinje s proučavanjem nižih odnosno jednostavnijih značajki slike, da bi se kasnije na osnovu rezultata iz ovog koraka uspostavljale hipoteze koje se onda dodatno proširuju prema pravilima konstrukcije i u konačnici procjenjuju korištenjem određenih funkcija [2]. Pravila konstrukcije te funkcije kojim se radi procjena ovise o tome koja je metoda iz pristupa odozdo prema gore odabrana. Problem ovog pristupa je u zahtjevnom pretraživanju i grupiranju značajki.

Pristupi odozgo prema dolje specifični su zbog faze učenja ili treniranja. Tijekom ove faze radi se na prepoznavanju značajki modela koje su posebne za određenu klasu objekata [2]. Ovaj pristup relativno brzo daje obećavajuće hipoteze, ali one se zato češće ispostave pogrešnima nego one proizvedene ostalim pristupima.

Pojedini aspekti prethodno navedenih pristupa često se kombiniraju pa većina autora ističe treću kategoriju pristupa, koja predstavlja kombinaciju pristupa odozdo prema gore i pristupa odozgo prema dolje. U ovoj skupini se kroz kombinaciju pozitivnih aspekata ostalih pristupa nastoje ispraviti njihovi pojedinačni nedostatci, kao što su problemi zahtjevnog pretraživanja i velik broj lažno pozitivnih hipoteza.

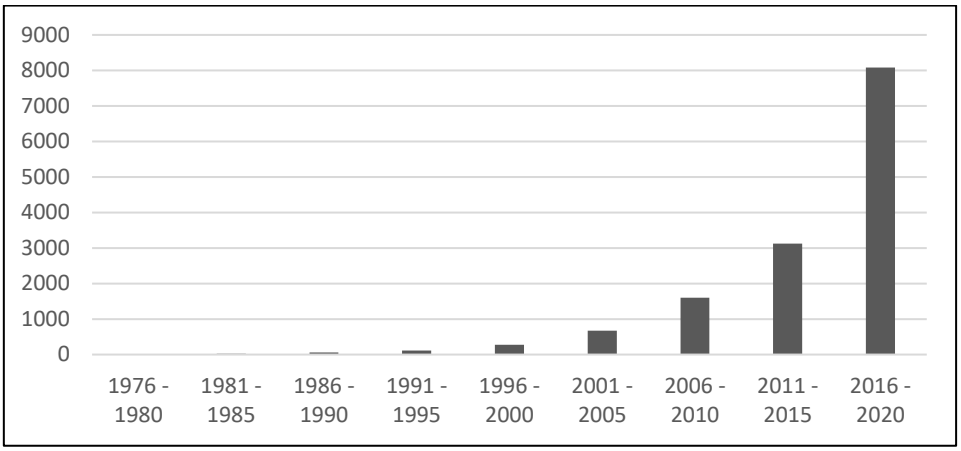
Primjer kombinacije pristupa odozdo prema gore i pristupa odozgo prema dolje predstavlja novi nadzirani detektor objekata Webly (engl. Novel webly supervised object detection, WebSOD), kojeg su 2020. godine predložili Wu Z. i sur. [3]. Riječ je o detektoru koji je koristio odozdo prema gore mehanizam utemeljen na bržem RCNN-u kako bi procijenio regije objekata na slikama koje bi pronalazio na internetu. Regije su procijenjene temeljem zajedničkih karakteristika postojećih i novih procijenjenih klasa objekata. Nad rezultatima ovog postupka se potom primjenjuje odozgo prema dolje pristup kako bi se izvršila klasifikacija procijenjenih regija. U konačnici se provodi dodatno optimiziranje kako bi se uklonile moguće greške u prepoznavanju objekata koje su nastale zbog nepravilnog preklapanja izvora s interneta i ciljanog izvora.

3.2. Povijesni razvoj

Prepoznavanja objekata je tek u zadnjih desetak godina steklo veliku popularnost u široj zajednici, ali sam koncept prepoznavanja objekata pojavljuje se mnogo prije toga. Prvi znanstveni radovi iz područja računalne vizije koji su bili vezani za prepoznavanje objekata pojavljuju se 70-ih godina prošlog stoljeća [6].

Kako bi se dodatno istaknuo nagli porast istraživačke aktivnost na temu prepoznavanja objekata, napravljen je pregled količine znanstvene literature na ovu temu po godinama putem internetskog pretraživača za znanstvenu literaturu, Google znalac (engl. Google Scholar). Pregled je napravljen korištenjem naprednog filtera za godine, opcije za pretragu naslova „allintitle“ i upotrebom ključne riječi „object detection“. Što se tiče ograničenja pregleda literature, bitno je napomenuti da su u obzir uzeti samo radovi koji su pisani na engleskom jeziku, sadrže pojam „object detection“ u naslovu i dostupni su na Google znalcu.

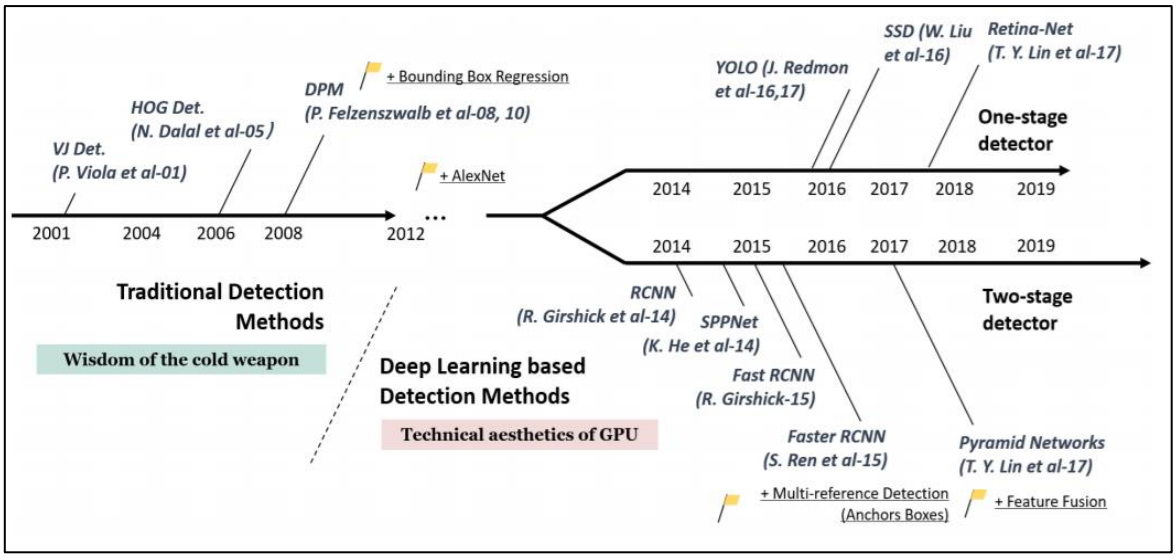
Rezultati pretraživanja pokazali su da se prva znanstvena literatura vezana za pojam prepoznavanja objekata počinje javljati još 1978. godine, kada su objavljena dva znanstvena rada. U narednih nekoliko godina broj znanstvenih radova se nastavio umjereno povećavati, da bi u zadnjoj deceniji započeo i eksponencijalno rasti (slika 2).



Slika 2. Porast broja znanstvenih radova o prepoznavanju objekata

Također, bitno je napomenuti da je samo u prvoj polovini 2021. godine objavljeno čak 1840 znanstvenih radova na temu prepoznavanja objekata što dodatno ukazuje na sve brži rast popularnosti i ove teme. Ovi rezultati upućuju na veliku važnost koju ova tema trenutno ima u znanstvenoj zajednici. Zbog naglog porasta važnosti drugih znanosti i tehnologija koje su srodne disciplini prepoznavanja objekata, kao što je to strojno učenje, može se očekivati da će se ovakav trend nastaviti i u budućnosti.

Povijesni razvoj discipline prepoznavanja objekata se može podijeliti na period tradicionalnog prepoznavanja objekata i period prepoznavanja objekata temeljenog na dubokom učenju (engl. Deep learning), s tim da većina znanstvene zajednice uzima 2014. godinu kao točku prijelaza između ova dva perioda [4]. Detaljan prikaz povijesnog razvoja discipline prepoznavanja objekata prikazan je na slici 3.



Slika 3. Vremenska crta razvoja prepoznavanja objekata [3]

3.2.1. Prepoznavanje objekata temeljeno na strojnom učenju

Period tradicionalnog prepoznavanja objekata se zasniva na primjeni strojnog učenja i inženjeringa značajki (engl. feature engineering) [7]. Strojno učenje je grana umjetne inteligencije koja sustavima pruža mogućnost učenja na temelju iskustva, što omogućuje rješavanje problema koji su prethodno bili nerješivi [8]. Primjer jednog takvog problema zasigurno je prepoznavanje objekata. Naime, bez tehnika strojnog učenja, kako bi se prepoznalo o kojem objektu se radi i gdje se on točno nalazi bilo bi potrebno razviti veoma složeno programsko rješenje. Ovakav program bi u isto vrijeme morao uzimati u obzir kut prikazivanja, svjetlost, različite pozadine i vrste oblika. S druge strane strojno učenje ovaj problem rješava učenjem značajki koje moraju postojati za pojedini objekt na temelju odgovarajućeg većeg skupa podataka.

Inženjering značajki je postupak upotrebe znanja iz određene domene kako bi se iz sirovih podataka prepoznale značajke i pretvorile u format koji se može upotrijebiti u strojnom učenju za rješavanje određenog problema [9]. Ovo je veoma bitan korak u strojnom učenju koji se često zanemaruje, iako dobro prepoznate značajke mogu mnogo pojednostaviti modeliranje

određenog problema. Osnovne metode koje se upotrebljavaju za prepoznavanje značajki objekata u ovom periodu su Houghova transformacija za prepoznavanje geometrijskih značajki, Harrisov detektor rubova za prepoznavanje značajki temeljem rubova objekta i invarijantnost razmjera i rotacije (engl. Scale-invariant Feature Transform, SIFT) koja ne ovisi o veličini i trenutnom položaju objekta [7].

Smatra se da je Viola-Jones detektor iz 2001. godine prvi značajniji napredak na području tradicionalnog prepoznavanja objekata [4] (slika 3). Ovo je prvi algoritam za prepoznavanje lica u stvarnom vremenu i bez ikakvih ograničenja. Sljedeći veliki napredak predstavlja histogram orijentiranih gradijenata (engl. Histogram of Oriented Gradients, HOG) detektor iz 2006. godine [4]. Ovaj detektor je imao mogućnost prepoznavanja objekata korištenjem računalne vizije i procesiranje slika. Posljednji veliki napredak iz ovog perioda prepoznavanja objekata se desio 2008. godine kada dolazi do pojave deformabilnog detektora na bazi dijelova (engl. Deformable Part-based Model, DPM). Ovo je prvi detektor s tehnikom regresijskog uokvirivanja (engl. bounding-box regression), odnosno načinom kako da se precizno lokaliziraju objekti [4].

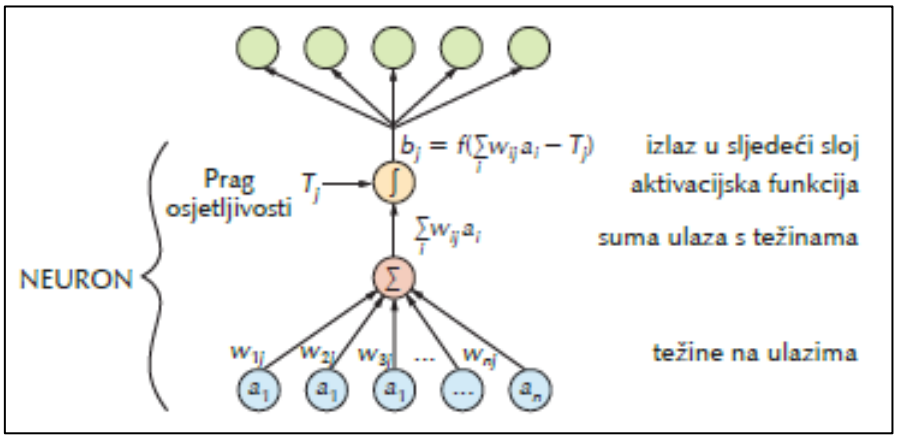
3.2.2. Prepoznavanje objekata temeljeno na dubokom učenju

Naglim porastom snage računala stvoreni su uvjeti za obradu velikih količina nestrukturiranih vrsta podataka kao što su govor, tekst te slikovni i video sadržaji koji su posebno bitni za prepoznavanje objekata. Za razliku od tradicionalnog perioda prepoznavanja objekata, metode dubokog učenja su omogućile prepoznavanje objekata bez prethodnog prepoznavanja i definiranja potrebnih karakteristika. „Duboko učenje predstavlja granu strojnog učenja koja se temelji na predstavljanju podataka složenim reprezentacijama na visokom stupnju apstrakcije do kojih se dolazi slijedom naučenih nelinearnih transformacija.“ [10]. Naziv duboko ili hijerarhijsko učenje potječe od načina na koji ono djeluje. Naime, u procesu razumijevanja složenog koncepta, duboko učenje nastoji ga prvo izgraditi iz jednostavnijih koncepata. Ovaj pristup može se zamisliti kao stablo iz teorije grafova koje ima veliki broj različitih razina, gdje čvorovi koji se nalaze na višim razinama predstavljaju složene koncepte koji su građeni iz većeg broja jednostavnijih koncepata koji se nalaze na nižim razinama. Primjer ovakve strukture bi bile rečenice, koje na nižoj razini imaju riječi, dok se na razini ispod riječi nalaze slogovi odnosno slova. Duboko učenje ima široku primjenu ne samo u prepoznavanju objekata, već i u brojnim drugim područjima umjetne inteligencije kao što su obrada prirodnog jezika, razumijevanje zvukova i govora. Duboko učenja temelji se na umjetnim neuronskim mrežama koje su dio strojnog učenja. Umjetne neuronske mreže načinom na koji su građene te načinom na koji djeluju nastoje na primitivan način imitirati

biološke neuronske mreže [10]. Naime, znanje u umjetnim neuronskim mrežama stječe se putem učenja, a način na koji se znanje prenosi i sprema kroz mrežu inspirirano je načinom na koji mozak djeluje. Svaka umjetna neuronska mreže sastoje se od sljedećih slojeva [11]:

- ulaznog sloja,
- jednog ili više skrivenih slojeva i
- izlaznog sloja.

Slojevi umjetne neuronske mreže sastoje se od čvorova odnosno umjetnih neurona. Kao i biološki neuron, umjetni neuron obrađuje signale odnosno podatke koristeći sinaptičke i somatske operacije. Ovaj jednostavni model neurona naziva se perceptron. Svaki pojedini čvor neuronske mreže ima određenu težinu i prag osjetljivosti [10]. Ulazni sloj umjetnih neurona prima vrijednosti ulaznih veličina odnosno podataka, a potom dolazi do izvršavanja sinaptičke funkcije pri čemu dolazi do množenja ulazne vrijednosti s težinskim koeficijentom [11]. Zbroj otežanih ulaza se uspoređuje s pragom osjetljivosti umjetnih neurona te u slučaju da je on veći od praga osjetljivosti, dolazi do generiranja izlazne vrijednosti za sljedeći sloj umjetnih neurona [11]. Na sličan način djeluju i neuroni u mozgu. Grafički prikaz umjetne neuronske mreže nalazi se na slici 4.



Slika 4. Pojednostavljeni prikaz umjetne neuronske mreže [11]

Kao što je već prethodno spomenuto, drugi način na koji umjetne neuronske mreže nastoje imitirati biološke neuronske mreže jeste stjecanje znanja putem učenja ili treniranja (engl. learning). Kada govorimo o učenju umjetnih neuronskih mreža govorimo zapravo o iterativnom postupku u kojem se iz iteracije u iteraciju radi na optimizaciji vrijednosti težinskih faktora umjetnih neurona na temelju nesukladnosti vrijednosti dobivene modelom umjetne neuronske mreže i stvarne vrijednosti određene promatrane veličine [11]. Postupak učenja umjetne neuronske mreže odvija se prema određenom pravilu učenja. Jedno od popularnijih pravila je pravilo širenja unatrag.

Struktura umjetne neuronske mreže je određena načinom na koji su čvorovi međusobno povezani. Danas, poznat je velik broj različitih vrsta umjetnih neuronskih mreža. Temeljem modela neurona od kojih se umjetna neuronska mreža sastoji te načinom na koji se signal širi kroz nju, razlikuju se sljedeće dvije vrste [11]:

- statičke unaprijedne (engl. feedforward) i
- dinamičke ili povratne (engl. feedback).

Od statičkih umjetnih neuronskih mreža najviše su u upotrebi višeslojne statičke mreže, dok su kod dinamičkih umjetnih neuronskih mreža najpopularnije višeslojne mreže sa zadrškom (engl. Time delay neural networks). Od posebne važnosti za prepoznavanje objekata su konvolucijske neuronske mreže (engl. convolutional neural networks). Riječ je o složenijoj vrsti višeslojne unaprijedne neuronske mreže koja se sastoji od istih slojeva kao i obična umjetna neuronska mreža, s tim da je razlika u posebnim konvolucijskim slojevima i slojevima sažimanja [12]. Konvolucijski sloj čine filteri koji imaju težine koje je potrebno savladati kako bi umjetna neuronska mreža proizvodila preciznije rezultate.

Pojavom dubokog učenja 2014. godine dolazi do prekretnice u razvoju discipline prepoznavanja objekata [4]. Potpomognuto snagom dubokog učenja, ova disciplina počinje da se razvija brže nego ikad prije. U ovom periodu razlikuje se dvofazno prepoznavanje objekata i jednofazno prepoznavanje objekata. Za razliku od jednofaznih modela koji u jednoj fazi obavljaju prepoznavanje objekata, dvofazni modeli slijede pristup po kojem se u prvoj fazi radi pronalazak svih mogućih slučajeva objekata, a potom u drugoj fazi dolazi do verifikacije u kojoj se potvrđuju prepoznati objekti.

Prvi dvofazni model prepoznavanja objekata bio je model prepoznavanja regija sa značajkama konvolucijskih neuronskih mreža (engl. Regions with Convolutional Neural Network features, RCNN). Ovaj model pojavio se 2014. godine, a predložio ga je autor R. Girshick [4]. U prvoj fazi ovog modela radi se na izvlačenju mogućih objekata iz određenog izvora. Na temelju ovih objekata uči se konvolucijska neuronska mreža kako bi se omogućilo izvlačenje karakterističnih značajki objekata. U drugoj fazi se koriste metoda potpornih vektora (engl. Support-vector machine) kako bi se prepoznali objekti u svakoj regiji, kao i njihove kategorije. Iako je RCNN detektor pružao daleko bolje rezultate u prepoznavanju objekata nego i jedan detektor do tad, on nije bio bez svojih nedostataka. Najveći problem bila je sporost prepoznavanja objekata zbog velikog broja redundantnih preračuna [4]. Kako bi se riješio ovaj problem, predložena je mreža udruženih prostornih piramida (engl. Spatial Pyramid Pooling Networks, SPPNet). Najveća prednost SPPNet-a je što omogućuje izračunavanje bitnih značajki iz slike samo jednom, dok se regionalne značajke ograničene duljine za potrebe

učenja modela izračunavaju odvojeno što smanjuje broj redundantnih računanja konvolucijskih značajki [4]. Ovaj pristup je bio brži, ali i dalje je imao nedostatke u vidu prepoznavanja objekata u više faza i optimiziranja samo povezanih slojeva neuronske mreže. Ovi problemi su riješeni kada je 2015. godine R. Girshick predložio model brzih regija sa značajkama konvolucijskih neuronskih mreža (engl. fast regions with convolutional neural network features, Fast RCNN), koji omogućuje da s istim postavkama mreže mogu istovremeno trenirati detektor i regresijsko uokvirivanje [4].

Prvi jednofazni detektor u periodu prepoznavanja objekata temeljem dubokog učenja bio je „gledate samo jednom“ detektor (engl. You Only Look Once, YOLO), čije i samo ime upućuje na činjenicu da je prethodni dvofazni pristup odbačen [13]. Za razliku od prethodnih dvofaznih detektora koji su prepoznavanje objekata obavljali analizirajući kroz njegove pojedine regije, YOLO je analizirao izvor kao cjelinu koristeći jednu konvolucijsku neuronsku mrežu. Ovaj model se pojavio 2015. godine, a predložen je od strane R. Josepha [13]. U narednih nekoliko godina R. Joseph je predložio više poboljšanih verzija YOLO detektora. Iako je YOLO imao znatno veću brzinu od prethodnih dvofaznih modela, njegova preciznost je bila manja, pogotovo kad je riječ o manjim objektima [4]. Ovaj nedostatak je nadoknađen kod jednostrukog kratkog detektora s višestrukim uokvirivanjem (engl. Single Short Multiplebox Detector, SSD), kojeg je 2015. godine predložio W. Liu [13]. Najveći doprinos ovog detektora su tehnike više referenci i više razlučivosti koje su znatno poboljšale preciznost prepoznavanja malih objekata od strane jednofaznih detektora. Naime, prethodni detektori su prilikom prepoznavanja objekata koristili samo gornje slojeve mreže, dok s druge strane SSD radi prepoznavanje objekata na više različitih slojeva mreže. Također, bitno je napomenuti da je riječ o prvom detektoru koji radi prepoznavanje objekata u stvarnom vremenu. Posljednji značajniji jednofazni detektor bio je RetinaNet. Ovaj detektor je imao još veću preciznost zahvaljujući posebnim funkcijama koje su olakšavale prepoznavanje objekata na vizualnim izvorima čije su pozadine zbog nedostatka kontrasta s bojom objekata posebno problematične [4]. RetinaNet model za prepoznavanje objekata predložen je 2017. godine od strane T. Y. Lin i njegovih suradnika [4].

3.3. Područja primjene

Iako disciplina prepoznavanja objekata postoji već duže od 50 godina, veću popularnost i primjenu počinje imati tek u posljednjoj deceniji zahvaljujući naglom razvoju kojeg je doživjela pojavom dubokog učenja 2014. godine. Danas, prepoznavanje objekata zbog brojnih svojih korisnosti ima široku upotrebu u velikom broju različitih područja, kao što su

prepoznavanje teksta, brojanje objekata, prepoznavanje lica osoba, autonomna vozila, prepoznavanje pješaka u saobraćaju, vizualne internetske tražilice, zabavne mobilne aplikacije, edukacija [4] [13].

Jedna od najbitnijih primjena discipline prepoznavanja objekata jeste prepoznavanje teksta. Ovo i nije iznenađujuće, s obzirom na količinu informacija koju ljudi pohranjuju u tekstualnom obliku. Naime, jedan od razloga iza toga je činjenica da tisućama godina čovjek nije imao drugog medija za pohranu informacija, već je bio primoran da zapisuje i čuva u knjigama sve što je želio sačuvati od zaborava. Također, čuvenje informacija u tekstualnom obliku je jedan od najjednostavnijih i najpraktičnijih medija u većini aspekata ljudskog života, kao što su zabava, edukacija, posao, saobraćaj i sl. Osnovni cilj prepoznavanja teksta jeste utvrđivanje postojanja teksta na određenoj slici te prepoznavanje njegovog sadržaja [4].

Primjena prepoznavanja objekata koja se često zanemaruje zbog lažne jednostavnosti koju odaje na prvi pogled jeste brojanje prepoznatih objekata. Primjeri ove primjene discipline prepoznavanja objekata je brojanje ljudi na nekom događaju, brojanje vozila koja su prošla određenim putem te brojanje životinja [13]. Neki od spomenutih primjera su do određene razine izvedivi od strane ljudi, ali postoje i primjeri u kojima to nije slučaj. Jedan takav primjer jeste brojanje mikroorganizama, za kojim postoji velika potreba u današnjem svijetu.

Posebno popularan primjer upotrebe metoda prepoznavanja objekata jeste prepoznavanje lica, koje usprkos tek nedavno stečenoj popularnosti predstavlja jedan od najstarijih primjera upotrebe računalne vizije i prepoznavanja objekata [13]. Popularni primjeri upotrebe mogućnosti prepoznavanja lica su: prepoznavanje ljudi na video snimkama, otključavanje i ostvarivanje pristupa digitalnim uređajima, prepoznavanje osmijeha prilikom fotografiranja te primjena digitalnih filtera i šminke na licu.

U današnjem vremenu sve više se spominje pojam autonomnih vozila, ponajviše zahvaljujući napredcima koje u tom području ostvaruju tvrtke kao što je Tesla. Upravo je razvoj tehnika prepoznavanja objekata omogućilo njihovu pojavu. Naime, kako bi se vozilo moglo samo navigirati ulicama grada, neophodno je da ima dobro razvijenu mogućnost prepoznavanja pješaka, ostalih vozila, saobraćajnih znakova te križanja i semafora.

Ovo su samo neke od brojnih mogućnosti i koristi koje nude metode prepoznavanja objekata, a daljnjim razvojem mogućnosti discipline prepoznavanja objekata te njoj bliskih područja kao što je duboko učenje zasigurno je da će njena primjena biti sve važnija i veća, zajedno s koristima koje ona nudi.

3.4. Problemi i izazovi

U prethodnom poglavlju navedeni su samo neki od brojnih primjera područja primjene discipline prepoznavanja objekata. Međutim, bitno je napomenuti da primjena metoda prepoznavanja objekata u svakom od navedenih područja ima i svoje nedostatke.

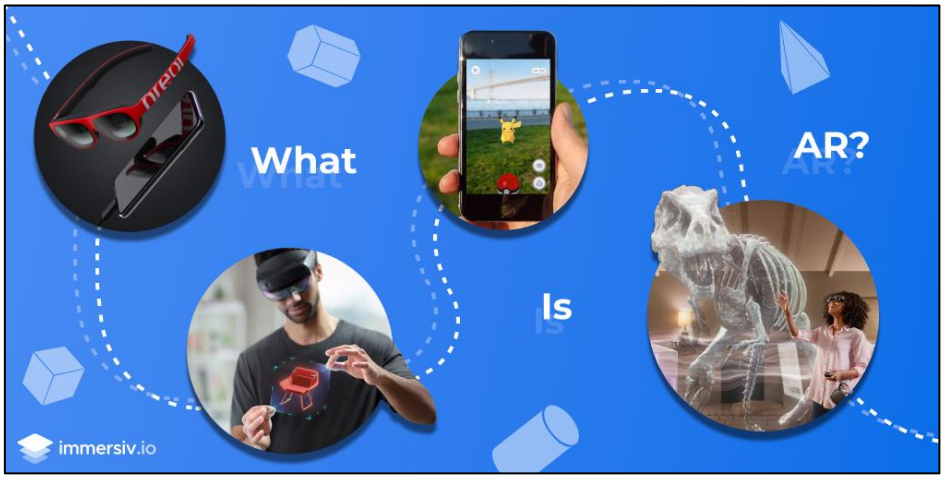
Problemi i izazovi primjene prepoznavanja objekata ovise o samoj domeni na koju se primjenjuje ova disciplina. Primjerice, područje prepoznavanja teksta suočava se s problemom postojanja velikog broja različitih jezika, simbola, fontova, boje. S druge strane izazovi prepoznavanja lica kriju se iza različitih boja kože, različitih ekspresija lica, prekrivenost dijela lica šminkom, kosom ili predmetima. Svoje probleme ima i područje autonomnih vozila. Naime prepoznavanje saobraćajnih znakova može biti osobito teško ili čak nemoguće u slučajevima kada je pojedini znak oštećen, prekriven ili uklonjen. Također, prepoznavanje saobraćajnih znakova i smjera kretanja puta može biti otežano u slučaju loših vremenskih prilika, kao što je velika količina padavina. Općenito se može reći da su problemi kojima se ova disciplina suočava najčešće vezani za varijabilnost broja objekata koje je potrebno prepoznati, varijabilnost veličine objekata te njihovu djelomičnu prekrivenost [13].

4. Proširena stvarnost

„Proširena stvarnost je poboljšana verzija fizičke odnosno prave stvarnosti čiji su objekti poboljšani računalno generiranim opažajnim informacijama i elementima kao što su to zvuk, video, grafika i haptika“ [14]. Jednostavnije rečeno riječ je o tehnologiji koja nam omogućuje da putem različitih digitalnih uređaja dobijemo *prošireni* pogled na svijet u svrhu poboljšavanja korisničkog iskustva i interakcije na jedan veoma jednostavan i zabavan način.

Pojmovi koji su veoma bliski proširenoj stvarnosti su virtualna stvarnosti (engl. virtual reality) i teleprisutnosti (engl. telepresence). Štoviše, zbog većeg broja sličnosti, proširena stvarnost i virtualna stvarnost često se greškom zamjenjuju i pogrešno upotrebljavaju. Razlika između ovih tehnologija je u tome što virtualna stvarnost korisnika u potpunosti uvodi i okružuje virtualnim odnosno računalno generiranim svijetom, dok proširena stvarnost nastoji samo *proširiti* stvarni svijet s generiranjem i prikazivanjem virtualnih objekata unutar njega [15]. Pojam teleprisutnosti označava osjećaj prisutnosti na nekoj udaljenoj lokaciji, odnosno lokaciji koja je udaljena od lokacije na kojoj se trenutno nalazimo [16]. Proširena stvarnost zapravo predstavlja jednu vrstu veze ili spoja virtualne stvarnosti i teleprisutnosti, koja uključuje kombinaciju stvarnog svijeta i virtualnog svijeta, interakciju s okruženjem u stvarnom vremenu te prikaz u 3D prostoru [17].

Velika popularnost koju proširena stvarnost trenutno ima te ubrzani razvoj tehnologije doveli su do pojave velikog broja različitih uređaja koji omogućavaju korištenje proširene stvarnosti u različite svrhe, kao što su zabava, edukacija pa čak i medicina (slika 5). Više o ovim uređajima te povijesnom razvoju i primjeni proširene stvarnost slijedi u narednim poglavljima rada.



Slika 5. Primjeri upotrebe proširene stvarnosti [18]

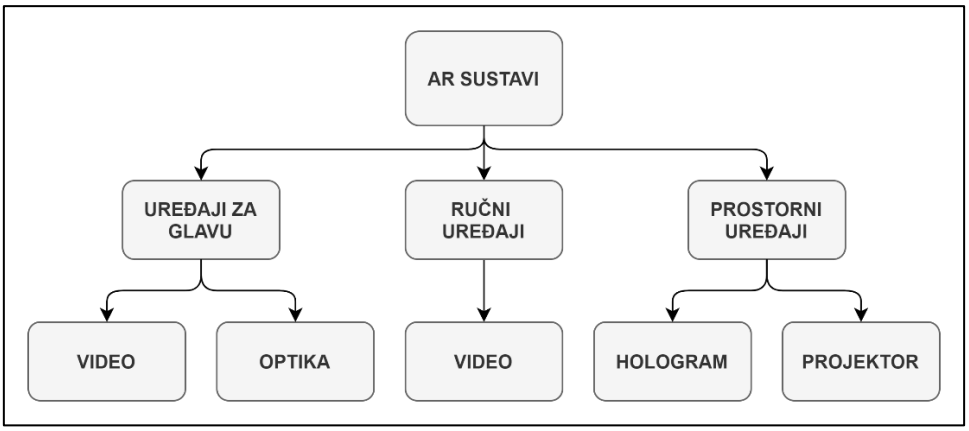
4.1. AR uređaji

Uslijed naglog razvoja kako programske podrške tako i sklopovlja, proširena stvarnosti postala je svima dostupna. Danas postoji velik broj uređaja različitih veličina i složenosti koji omogućuju implementaciju i prikazivanje proširene stvarnosti. Primjer takvih uređaja su pametni uređaji poput mobilnog telefona ili tableta kojeg danas posjeduje većina ljudi. Neovisno o kojem od tih uređaja je riječ, kako bi se omogućilo korištenje proširene stvarnosti, mora postojati sustav koji sadrži ili ima mogućnost da implementira sljedeće tri temeljne komponente [16]:

- generator scena,
- uređaj za prikazivanje i
- praćenje i osjet.

Generator scena su uređaji ili programi koji omogućuju prikaz računalno generiranih objekata u stvarnom svijetu [17]. Ovo je jedna od manje zahtjevnih komponenata AR sustava zbog činjenice da se u proširenoj stvarnosti većinom radi o prikazivanju manjeg broja jednostavnih objekata od kojih se ne zahtjeva da budu posebno kvalitetni i fotorealistični, kao što su tekstualni ispisi i različiti svakodnevni predmeti.

Uređaj za prikazivanje je još jedna komponenta AR sustava koja nije posebno zahtjevna [17]. Naime, u ovim sustavima se mogu koristiti i najjednostavniji jednobojni zasloni kako bi se prikazali računalno generirani objekti proširene stvarnosti. S druge strane, u sustavima za prikazivanje virtualne stvarnosti ova komponenta je mnogo složenija, jer je riječ o prikazivanju potpuno nove odnosno virtualne okoline. AR sustave na temelju uređaja za prikazivanje dijelimo na uređaje za glavu, ručne uređaje i prostorne uređaje [19]. Nadalje, ove kategorije imaju i svoje potkategorije kao što je to prikazano na slici 6.



Slika 6. Vrste prikaza proširene stvarnosti

Uređaji za glavu su različite vrste posebnih naočala koje se nose pomoću kacige ili remena za učvršćivanje. Ova vrsta uređaja se dalje dijeli na uređaje s prikazom putem videa i uređaje s prikazom putem optike, ovisno o tome je li riječ o prikazu proširene stvarnosti putem zaslona ili putem projiciranja svjetlosnih zraka [19]. U uređaje za glavu spadaju prozirni zasloni za glavu (engl. Head-mounted display, HMD), virtualni prikaz na mrežnici oka (engl. Virtual retinal display, VRD) te projektivni zasloni za glavu (engl. Head-mounted projective display, HMPD) [19]. Primjer ovih uređaja su Microsoft HoloLens, Google Glass i EyeTap.

Ručni uređaji zbog svoje niske cijene i velike dostupnosti predstavljaju najpopularniju vrstu prikaza proširene stvarnosti. Riječ je o uređajima koji se drže u ruci prilikom prikazivanja proširene stvarnosti. U ovu kategoriju spadaju ručni zasloni i ručni projektori. Primjeri ovih uređaja su mobilni telefon, iLamp i Pico projektor [19]. Implementacija proširene stvarnosti u mobilnim uređajima bit će detaljno razrađena u kasnijim poglavljima ovog rada.

Prostorni uređaji predstavljaju vrstu uređaja koji se više ne pomjeraju nakon inicijalnog pozicioniranja za prikazivanje proširene stvarnosti na određenoj površini [19]. Ova vrsta uređaja se dalje dijeli na uređaje s prikazom putem holograma i uređaje s prikazom putem projekcija [19]. Poznati primjer ovih uređaja su tzv. Head-up prikazi (engl. Head-up display, HUD). Oni se najčešće koriste za projiciranje korisnih informacija na različitim površinama, kao što je prikaz navigacijskih informacija na vjetrobranskom staklu vojnih aviona ili prikaz rezultata i preostalog vremena na parketu za vrijeme košarkaške utakmice. Problem kod ovakvog načina prikazivanja je što prikaz može biti iskrivljen i nejasan prilikom kretanja ili gledanja iz određenog kuta. Ovaj nedostatak riješen je pojavom 3D holografskih prikaza koji ne ovise o položaju gledatelja [19]. Primjer ovakvog uređaja je ARSyS TriCorder, koji se koristi u medicini od strane kirurga kako bi se olakšao operacijski zahvat [19].

Posljednja komponenta AR sustave je komponenta za praćenje i osjet. Ovo je daleko najsloženija komponenta iz razloga što stvarni i virtualni objekti moraju biti ispravno pozicionirani i poravnani u odnosu jedni na druge, inače se gubi osjećaj proširenog svijeta [17]. Ovdje je riječ o tzv. problemu registracije [16]. Brojna područja primjene proširene stvarnosti imaju striktne zahtjeve što se tiče preciznosti prikaza virtualnih objekata. Jedno takvo područje je medicina, gdje se proširena stvarnost ponekad koristi prilikom izvođenja operacija. Bitno je napomenuti da za razliku od prethodnih komponenti, ova komponenta je jednostavnija u sustavima virtualne stvarnosti. Glavni razlog iza toga je činjenica da u virtualnoj stvarnosti ne dolazi do spajanja stvarnog i virtualnog svijeta pa nema potrebe za međusobnim ujednačavanjem stvarnih i virtualnih objekata.

Pored ovih komponenti, sustavi moraju imati i odgovarajuće sklopovlje kako bi mogli procesirati i prikazivati proširenu stvarnost. Naime, generiranje scene koja će sadržavati kombinirani prikaz stvarnog okruženja i virtualnih objekata te njihovo generiranje i registriranje je veoma zahtjevno, stoga je neophodno da sustav ima dovoljno snažan procesor i grafičku karticu, kvalitetnu kameru i zaslon te određene senzore kao što su GPS i akcelerometar [20].

4.2. Tipologija

Ubrzanim razvojem proširene stvarnosti došlo je do pojave velikog broja različitih načina na koji ona može biti implementirana i korištena. Sukladno tome postoji i više načina klasifikacije proširene stvarnosti. Neki od popularnijih načina dijeljenja proširene stvarnosti su podjele koje se temelje na [21]:

- funkcionalnim karakteristikama,
- meti koja se nastoji *proširiti*,
- tehnikama proširene stvarnosti i
- odnosu s korisnikom.

Temeljem funkcionalnih karakteristika, vrste proširene stvarnosti moguće je svrstati u dvije osnovne kategorije, a to su izazvana proširena stvarnost (engl. Triggered augmentation) i proširena stvarnost temeljena na prikazu (engl. View-based augmentation), s tim da se ove kategorije dalje dijele na svoje potkategorije [22].

Izazvana proširena stvarnost temelji se na postojanju određenog okidača (engl. trigger) koji pokreće proces prikazivanja proširene stvarnosti [22]. Na temelju okidača razlikuju se četiri vrste izazvane proširene stvarnosti: proširena stvarnost temeljena na markerima (engl. Marker-based augmented reality), proširena stvarnost temeljena na lokaciji (engl. Location-based augmented reality), dinamička proširena stvarnost (engl. Dynamic augmentation) i kompleksna proširena stvarnost (engl. Complex augmentation) [22]. Okidač kod proširene stvarnosti temeljene na markerima može biti određena stvarna, fizička oznaka na papiru ili objektu. Prepoznavanjem unaprijed pripremljene oznake putem odgovarajućeg AR uređaja dolazi do pokretanja procesa generiranja scene proširene stvarnosti. Ova scena najčešće na neki način nadopunjuje prepoznati objekt. Proširena stvarnost temeljena na lokaciji koristi GPS koordinate AR uređaja kao okidač. Ovakva vrsta proširene stvarnost najčešće se koristi za davanje korisnih informacija o lokaciji na koju je osoba upravo stigla, kao što je festival, restoran, muzej i sl. Dinamička proširena stvarnost koristi promjene na određenom objektu kao okidač. Jedan popularan primjer upotrebe ove vrste izazvane proširene stvarnosti je dinamičko

isprobavanje kozmetike i odjeće [22]. Kompleksna proširena stvarnost predstavlja kombinaciju proširene stvarnosti temeljene na markerima i proširene stvarnosti temeljene na lokaciji. Ona omogućava povezivanje stvarnog fizičkog svijeta s dinamičkim informacijama koje najčešće dolaze s interneta. Primjer ove vrste izazvane proširene stvarnosti je kad se prilikom kretanja na zaslonu AR uređaja prikazuju informacije o okruženju kroz koje se prolazi.

Kod proširene stvarnosti temeljene na prikazu razlikujemo indirektna proširenja i neodređena digitalna proširenja [22]. Indirektna proširenja generiraju statični, izmijenjeni prikaz svijeta. Primjer ove vrste izazvane proširene stvarnosti je fotografiranje određenog objekta te pravljenje određenih *proširenja* u vidu promjene boje, veličine i oblika. Kod neodređene digitalne proširene stvarnosti ne postoji veza s stvarnim objektima koji se trenutno nalaze na zaslonu AR uređaja. Primjer su mobilne aplikacije koje generiraju različite scene proširene stvarnosti kako bi korisnik s njima bio u interakciji, a koje ni na koji način nisu povezane sa stvarnim objektima koji se nalaze u okruženju korisnika.

Veoma popularna i jednostavna podjela proširene stvarnosti je podjela na temelju same mete odnosno objekta koji se nastoji *proširiti*. Ovo je ujedno i prva opće prihvaćena podjela proširene stvarnosti [23]. Prema ovom pristupu proširena stvarnost se dijeli na proširenu stvarnost koja proširuje korisnika, proširenu stvarnost koja proširuje fizički objekt i proširena stvarnost koja proširuje okruženje koje se nalazi oko korisnika i objekta [23].

Proširena stvarnost koja proširuje korisnika predstavlja vrstu proširene stvarnosti koja je najčešće implementirana pomoću uređaja kojeg korisnik koristi kako bi dobio informacije o stvarnim objektima iz svog okruženja [23]. Ovdje je najčešće riječ o ručnom uređaju, primjerice mobilnom uređaju koji na svom zaslonu prikazuje informacije o stvarnim objektima. Još jedan primjer ručnog AR uređaja koji proširuje korisnika je Charade, pametne rukavice kojima osoba može putem gesti upravljati sadržajem prezentacije [24]. Postoje i uređaji koji se nose na glavi kao što su AR naočale koje projiciraju informacije o stvarnim objektima na zaslonu ili mrežnici korisnika. Primjeri ovakvih uređaja su svima poznati HoloLens i Google Glass.

Proširena stvarnost koja proširuje objekt je manje popularna vrsta proširene stvarnosti kod koje se stvarni fizički objekt proširuje ugradnjom uređaja na objektu ili unutar samog objekta [23]. Primjeri implementacije ove vrste proširene stvarnosti su PARCTab i Xerox EuroPARC uređaji, koji su bili izrazito popularni 90-ih godina [25] [26].

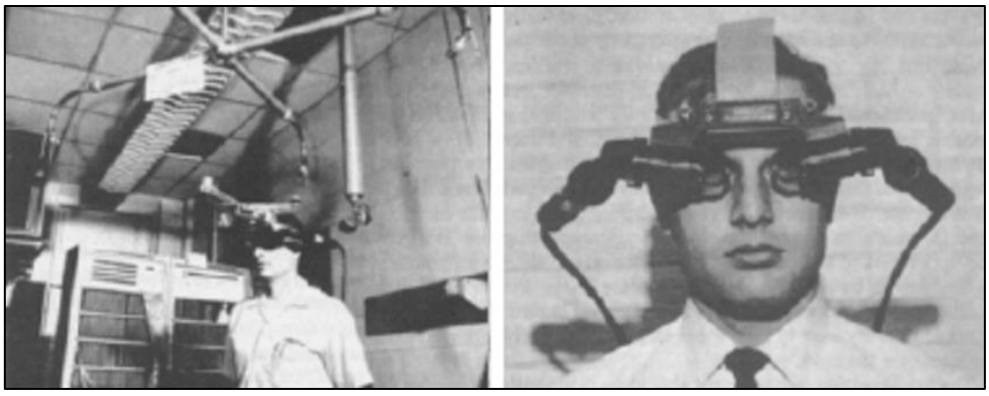
Proširena stvarnost koja proširuje okruženje predstavlja vrstu proširene stvarnosti kod koje ne postoji izravan utjecaj na korisnika AR uređaja i objekata u njegovoj neposrednoj blizini, već samo na njihovo okruženje [23]. Kod ove vrste proširene stvarnosti cilj je prikupljanje i pružanje informacija o okruženju korisnika te njihovo prikazivanje na objekte. Jedan od prvih

primjera implementacije ove vrste proširene stvarnosti je „Krueger's Video Place" iz 1975. godine, gdje se na zidu sobe prikazuje računalno generirani i kontrolirani lik koji se kreće i reagira na pokrete osobe koja se nalazi ispred zida [27]. Još jedan raniji primjer implementacije ove vrste proširene stvarnosti je „Put That There", gdje osoba prstom odabire virtualne objekte koji se prikazuju na nekoj površini i pokazuje gdje želi da se premjeste [28]. Moderniji primjeri predstavljaju brojne aplikacije koje na zaslonu mobilnog uređaja prikazuju izmijenjeno okruženje korisnika.

4.3. Povijesni razvoj

Smatra se da razvoj proširene stvarnosti započinje još 50-ih godina prošlog stoljeća s kinematografom M. Heiligom i njegovom vizijom o kinu budućnosti [29]. Naime, prema njegovoj viziji filmovi bi prilikom izvođenja u kinu utjecali na sva osjetila gledatelja. Heilig je svoju viziju o kinu budućnosti realizirao 1962. godine, izgradnjom uređaja kojeg je nazvao Sensorama [29]. Riječ je o mehaničkom uređaju koji je imao mogućnost puštanja odgovarajućih zvukova i mirisa, prikazivanja stereoskopskih 3D slika, pomicanje tijela gledatelja, a sve to kako bi se utjecalo na sva osjetila gledatelja te se dodatno dočarao film koji se prikazivao [29]. Primjer filma koji se prikazivao putem Sensorame je vožnja motociklom u New Yorku. Vožnja gradom je dočarana pomicanjem stolice u skladu s kretnjom motocikla, puhanjem zraka, puštanjem zvukova i mirisa New Yorka te prikazivanjem ulice kojom se motocikl kretao. Heilig na kraju nije uspio osigurati dovoljno financijskih sredstava kako bi nastavio raditi na daljnjem razvoju Sensorame. Ipak, pojava Sensorame je bio veoma važan trenutak u povijesti virtualne realnosti, ali i proširene stvarnosti jer se prvi put spominje mogućnost poboljšavanja različitih prizora iz stvarnog svijeta. Iako je ovaj trenutak bio početak razvoja proširene stvarnosti, do definiranja izraza „proširena stvarnost“ proći će još mnogo vremena.

Sljedeći veliki napredak u razvoju proširene stvarnosti bio je Damoklov mač (engl. The Sword of Damocles), kojeg je 1968. godine zajedno sa svojim studentima izumio profesor Ivan Sutherland [30]. Uređaj je nazvan po istoimenoj anegdoti koja aludira na stalno prisutnu opasnost kojom se suočavaju ljudi na moćnim pozicijama. Riječ je o prvom AR uređaju koji pomoću prozirnog zaslona prikazuje 3D grafiku u stvarnom svijetu [30]. Prilikom korištenja Damoklov mač bi bio pričvršćen mehaničkom rukom te pozicioniran na glavu korisnika zbog svoje težine te potrebe praćenja kretanje glave korisnika [31]. Damoklov mač pripada kategoriji prozirnih zaslona za glavu odnosno HMD uređajima za proširenu stvarnost (slika 7). Kao i Sensorama, ovaj uređaj je bio od izrazito velikog značaja za razvoj kako virtualne stvarnosti, tako i proširene stvarnosti.



Slika 7. Damoklov mač [30]

Nakon Damoklovog mača, 1975. godine se pojavljuje već spomenuti „Krueger's Video Place". Ovaj projekt je kombinirao sustav projekcije i nekoliko video kamera kako bi na zidu prikazivao računalno generirane likove te korisniku dao osjećaj da je riječ o interaktivnom okruženju [27]. Riječ je o interaktivnom okruženju, jer korisnik svojom kretnjom izaziva reakcije lika koji se prikazuje na zidu.

Konačno, 1990. godine dolazi do prve službene upotrebe izraza „proširena stvarnost“. Izraz su prvi put upotrijebili istraživači tvrtke Boeing, Tom Caudell i David Mizell, u svojem radu „Augmented reality: An application of heads-up display technology to manual manufacturing processes“ [32]. U ovom radu dolazi i do prvog uspoređivanja prednosti i nedostataka između proširene stvarnosti i virtualne stvarnosti. Ove godine dolazi i do pojave prvog pravog sustava proširene stvarnosti koji je nazvan „Virtual Fixtures“, a izumljen je od strane L. B. Rosenberg [31]. Ubrzo je objavljen i prvi značajniji rad na temu sustava proširene stvarnosti pod nazivom „Knowledge-Based Augmented Reality“ autora S. Feiner, B. MacIntyre i D. Seligmann [33]. U ovom radu, između ostalog, spominje se AR sustav nazvan KARMA, koji korisnicima putem proširene stvarnosti i prozirnog zaslona za glavu prikazuje kako da obavljaju osnovne operacije za održavanje pisača.

Iako je naziv „proširena stvarnost“ postao opće prihvaćen još od njegove pojave, tek 1997. godine dolazi do njegovog formalnog definiranja od strane R. Azume [17]. Naime, u svojem pregledu razvoja proširene stvarnosti, nazvanom „A survey of augmented reality", R. Azuma definira proširenu stvarnost kao kombinaciju stvarne i virtualne okoline koja je registrirana u 3D prostoru i interaktivna u stvarnom vremenu [17].

Do kraja 90-tih, proširena stvarnost je postala zasebno i to veoma popularno područje istraživanja [31]. Ovo potvrđuje i činjenica da je osnovano nekoliko znanstvenih konferencija na temu proširene stvarnosti. Primjer takvih konferencija su „International Workshop and Symposium on Augmented Reality“ i „International Symposium on Mixed Reality“ [31].

Također, osnovano je i nekoliko organizacija na istu temu, kao što je „International Symposium on Mixed and Augmented reality“ [31]. Ova organizacija je ubrzo postala jedan od najvećih simpozija za industriju i istraživanje koji omogućuje razmjenu problema i rješenja na području proširene stvarnosti. U ovom periodu došlo je i do velikog preokreta u dostupnosti tehnologije proširene stvarnosti. Naime, pojavljuju se različiti besplatni programski alati kao što je ARToolKit, koji omogućuju brži i jednostavniji razvoj aplikacija za proširenu stvarnost [31].

Sljedeći veliki događaj u povijesti proširene stvarnosti desio se 2000. godine, kada je B. Thomas razvio igru ARQuake [31]. Riječ je o prvoj AR igri za mobilne uređaje koja je namijenjena za igranje na otvorenom prostoru. Uređaj koji je pokretao igru bi najčešće bio smješten u torbi koju je korisnik nosio na leđima, a scene proširene stvarnosti prikazivane su putem prozirnog zaslona za glavu odnosno HMD-a. ARQuake temeljio se na GPS-u i naprednom sustavu praćenja koji je dozvoljavao do tad najviše dostupnih šest stupnjeva slobode [34]. Ovo je posebno važan događaj, jer u današnjem vremenu igre ove vrste predstavljaju jedan od najpopularnijih načina implementacije proširene stvarnosti. Primjeri trenutno popularnih igara ove vrste su „Pokemon GO“ i „Harry Potter: Wizards Unite“.

O brzom rastu važnosti proširene stvarnosti u ovom periodu svjedoči izvještaj „Horizon Report“ iz 2005. godine [35]. Prema ovom izvještaju govori se o stanju tehnologije proširene stvarnosti i predviđa se da će u sljedećih četiri do pet godina doći do još većih napredaka u tom području. Kao primjer navodi se sustav kamera koji ima sposobnost analiziranja fizičkog okruženja u stvarnom vremenu te izračun položaja objekata u takvom okruženju. Nakon svoje pojave, navedeni sustav postaje osnova za integraciju virtualnih objekata u većini AR sustava koji se intenzivno počinju pojavljivati u narednih nekoliko godina [31]. U ovom periodu dolazi do pojave veoma popularne i visoko nagrađivane AR igre zvane ARTennis. Ovo predstavlja prvu kolaborativnu igru za mobilne uređaje koja je u svojoj mehanici uključivala različite elemente proširene stvarnosti [34].

Nakon 2009. godine, područje proširene stvarnosti ulazi u jedan potpuno drugačiji period. U ovom periodu proširena stvarnost počinje da se komercijalizira i razvija brže nego ikad prije [34]. Razlog iza ovoga je veliki interes, očekivanja, ali i ulaganja svjetskih vodećih tvrtki u istraživanje i razvoj proširene stvarnosti. Ovdje najveći interes pokazuju Microsoft, Google i Facebook [34]. Sve ovo je rezultiralo pojavom AR uređaja Microsoft HoloLens i Google Glass (slika 8). HoloLens je složeni računalni AR uređaj za glavu s prozirnim zaslonom i većim brojem senzora koji omogućuju prikazivanje različitih informacija i virtualnih objekata u okruženju korisnika [34]. S druge strane, Google Glass je optički AR HMD uređaj kojim se upravlja putem dodira i glasovnih naredbi. Ovaj uređaj ima velik broj različitih funkcionalnosti kao što je prikazivanje informacija o okruženju korisnika, slikanje i snimanje [34].

Slika 8. Usporedba uređaja Google Glass i Microsoft HoloLens [35]

U ovom periodu dolazi do naglog razvoja sklopovlja i programske podrške za mobilne uređaje, zbog čega mobilni uređaji postaju savršena platforma za daljnji razvoj proširene stvarnosti. U ovom periodu dolazi do pojave trenda prema kojem većina implementacija proširene stvarnosti se zasniva na upotrebi SLAM (engl. Simultaneous localization and mapping) sustava. Riječ je o sustavu koji istovremeno nastoji izgraditi kartu nepoznatog okruženja i pratiti lokaciju agenata koji se kreću u njemu [34]. Ovaj trend je doveo do pojave većeg broja različitih varijacija ovog sustava, kao što su LSD-SLAM i DT-SLAM [34].

4.4. Područja primjene

Kako su istraživanja i razvoj proširene stvarnosti sve brže napredovali, povećavao se i broj područja koji bi mogao imati korist od primjene ove tehnologije. Trenutno, neka od područja u kojima proširena stvarnost ima veliku primjenu su marketing i prodaja, video igre, edukacija, vojska, medicina, automobilska industrija, turizam, arhitektura, arheologija, prevođenje [31]. U ovim područjima upotreba proširene stvarnosti donijela je višestruku korist u vidu pojeftinjenja usluga, poboljšanja kvalitete usluga, a u pojedinim slučajevima i stvaranje potpuno novih vrsta usluga.

4.4.1. Marketing i prodaja

Marketing i prodaja zasigurno predstavlja jedno od najpopularnijih područja primjene proširene stvarnosti. Prvi primjeri implementacije većinom se svode na reklame i kataloge za proizvode, koji se korištenjem markera mogu proširiti kako bi se dobili 3D prikazi proizvoda. Kako je tehnologija proširene stvarnosti napredovala, načini primjene proširene stvarnosti su postajali sve složeniji, ali i korisniji.

Proširena stvarnost je nedavno stekla široku primjenu u modnoj industriji. Primjer su JC Penney i Bloomingdale koji su u pojedinim svojim trgovinama ugradili posebne sobe za preslvačenje koje korisnicima omogućavaju da na sebi probavaju različite odjevne kombinacije bez potrebe za presvlačenjem [38]. Suzanne Harward, poznati australski brand za vjenčanice, svojim kupcima je omogućio pregledavanje vjenčanica na stvarnim modelima putem mobilne aplikacije [39]. Wanna Kicks svojim kupcima omogućuje da putem mobilne aplikacije probavaju i vide kako izgleda njihova obuća čak i u pokretu [38].

Proširena stvarnost se na sličan način počela primjenjivati i u kozmetičkoj industriji, gdje tvrtke poput L'Oreal, Sephora, Charlotte Tilbury i Rimmel koriste AR aplikacije kako bi svojim korisnicima omogućili da vide kako će određeni kozmetički proizvod izgledati na njihovom licu [40]. Proširena stvarnost se na sličan način koristi prilikom prikazivanja različitih filtera i efekata za lice na popularnim socijalnim mrežama kao što su Facebook, Instagram, Tiktok i Snapchat.

Primjena proširene stvarnosti je donijela niz pogodnosti kada je riječ o uređenju interijera. Tvrtke kao što su Ikea, Pottery Barn i Houzz svojim kupcima pružaju mogućnost da putem mobilne aplikacije pronađu proizvode koji će se najbolje uklopiti u njihov dom [41]. Primjer jedne takve aplikacije je Ikea Place, koja korisnicima omogućava da isprobaju više od 2.000 njihovih proizvoda, pritom uzimajući u obzir dimenzije i osvjetljenje prostora u kojem se proizvod isprobava. Na ovaj način korisnicima se omogućava vizualizacija i isprobavanje proizvoda prije kupovine, lakši odabir boje, veličine i budućeg položaja proizvoda te lakši pristup dodatnim informacijama vezanih za sam proizvod, kao što su opis, dimenzije i ocjene kupaca.

4.4.2. Video igre

Prethodno spomenuti ARQuake bio je tek skromna najava onog što će primjena proširene stvarnosti dovesti industriji video igara. Na samom početku primjena proširene stvarnosti u ovoj industriji svodila se na izradu eksperimentalnih AR uređaja koji zbog svoje složenosti i nepraktičnosti nisu bili namijenjeni za širu javnost. Međutim, razvoj pametnih uređaja je doveo do velikih promjena, od kojih je najznačajnija popularizacija video igara koje su namijenjene za mobilne telefone. U skladu s ovim trendom, dolazi do promjene načina na koji se proširena stvarnost primjenjuje u industriji video igara. Rezultat toga su brojne mobilne AR igre kao što su Ingress, Pokemon GO, Harry Potter: Wizards Unite, The Walking Dead: Our World i Jurassic World Alive [42].

Ingress je jedna od prvih popularnijih mobilnih AR igara, a razvijena je 2014. godine od strane tvrtke Niantic [42]. Cilj igre je pronaći i osvojiti portale koji su raspoređeni u stvarnom svijetu. Igra je 2018. godine dobila svoj nastavak koji je nosio naziv Ingress Prime.

Daleko najpopularnija mobilna, a ujedno i AR igra svih vremena je Pokemon GO, koju je razvio Niantic [42]. Riječ je o igri u kojoj igrač pokušava pronaći i uloviti pokemone koji se nalaze u stvarnom svijetu (slika 9). Kako bi pronašao pokemone, igrač koristi kameru svog mobilnog uređaja. Igra je vremenom proširena i nizom drugih mogućnosti, kao što je vođenje međusobnih bitki i razmjena pokemona. Iako je Pokemon GO izašao još 2018. godine, on je i danas veoma popularan.

Slika 9. Mobilna igra Pokemon GO [43]

Jedna od najnovijih mobilnih AR igara je Harry Potter: Wizards Unite [42]. Ova igra je po svojoj mehanici mnogo slična igri Pokemon GO, što i nije iznenađujuće s obzirom da je i ovu igru razvila tvrtka Niantic. Kao i u Pokemon GO, igrači istražuju stvarni svijet putem kamera svojih mobilnih uređaja, s tim da su pokemoni u ovom slučaju zamijenjeni s raznim magičnim predmetima i čudovištima.

4.4.3. Medicina

Primjena proširene stvarnosti u medicini je veoma istraživački aktivno područje. Iako brojni znanstveni radovi upućuju na moguće načine implementacije proširene stvarnosti u medicinske svrhe, za većinu njih tehnologija proširene stvarnosti nije još dovoljno razvijena.

Trenutno, primjena proširene stvarnost se u ovom području većinom svodi na pružanje dodatnih informacija prilikom izvođenja određenih operativnih postupaka [31][44]. Jedan primjer je vizualizacija vena na tijelu pacijenta kako bi se olakšao pronalazak mjesta za ubod. Proširena stvarnost se na sličan način koristi i prilikom operativnih postupaka, gdje se na tijelu pacijenta prikazuje njegova unutarnja anatomija što olakšava pronalazak odgovarajućeg

mjesta za pravljenje reza [31]. Nešto drugačiji primjeri primjene proširene stvarnosti u ovom području je upotreba uređaja Microsoft HoloLens prilikom obuke i pripreme za operativne zahtjeve, kao i olakšavanje navigacije slijepim osobama davanjem audio upozorenja ako se osoba nalazi preblizu zidu ili nekom objektu [45]. Bitno je napomenuti da se velik broj drugih ideja za implementaciju proširene stvarnosti još uvijek nalazi u fazi razvoja. Samo jedan takav primjer je mogućnost uvida u medicinsku povijest pacijenta putem njene vizualizacije nad tijelom pacijenta [31].

4.5. Problemi i izazovi

Prema svim navedenim mogućnostima i koristima koje nudi tehnologija proširene stvarnosti moguće je zaključiti da je ona u kratkom periodu zaista mnogo napredovala. Međutim, kao i svaka druga tehnologija i ona ima svoje probleme i izazove. Najznačajnije probleme predstavlja registracija, uklanjanje stvarnih objekata iz okruženja, socijalna prihvaćenost i potencijalna štetnost za oči u slučaju dužeg korištenja.

Problem registracije predstavlja jedno od najvećih ograničenja u daljnjem razvoju proširene stvarnosti [17]. Ovaj problem se temelji na zahtjevu da položaj virtualnih objekata mora biti u skladu sa stvarnim objektima i okruženjem kako bi se očuvao smisao scene proširene stvarnosti [17]. Važnost ispunjenja ovog zahtjeva ovisi o domeni primjene. Ako se za primjer uzme primjena proširene stvarnosti u video igrama kršenje ovog zahtjeva dovodi samo do smanjena realnosti prikazane scene. U slučaju da je riječ o medicini nepravilna registracija može imati katastrofalne posljedice kao što je pravljenje reza na pogrešnom mjestu ili postavljanje pogrešne dijagnoze. Problem registracije je mnogo značajniji u proširenoj stvarnosti, nego u virtualnoj stvarnosti u kojoj je cijela scena virtualna i ne postoji potreba za usklađivanjem virtualnog svijeta sa stvarnim svijetom [46]. S druge strane, u proširenoj stvarnosti ovaj problem je mnogo veći, jer ljudsko oko ima mogućnost zapažanja i najsitnijih detalja. Kako bi registracija scene proširene stvarnosti bila ispravna, potrebno je osigurati dovoljno visoku preciznost prikaza. Probleme u registraciji scene proširene stvarnosti moguće je svrstati u statičke i dinamičke [17]. Statički problemi predstavljaju vrstu problema registracije koji se dešavaju kada se korisnik i objekt kojeg on promatra nalaze u stanju mirovanja [17]. U ovu skupinu se ubrajaju problemi poput optičkog izobličenja, grešaka u sustavu praćenja objekata, mehanička odstupanja i pogrešne postavke prikaza scene proširene stvarnosti. Dinamički problemi su problemi registracije koji se javljaju u trenutku kada se korisnik ili objekt kojeg on promatra počne pomjerati [17]. Najčešći uzroci dinamičkih problema su različita kašnjenja u radu AR sustava. Kašnjenja se dešavaju u trenutku kada sustav praćenja izračuna

pozicije svih objekata i potom ih treba generirati i prikazati na zaslonu. Kašnjenja u prikazu objekata se dešavaju zbog činjenice da svaka komponenta sustava treba određeno vrijeme da obavi svoj zadatak [17]. Trenutno, dinamički problemi predstavljaju najveću i najznačajniju skupinu problema registracije. Neki od načina ublažavanja i uklanjanja dinamičkih problema registracije su smanjivanje kašnjenja sustava, usklađivanje vremena prikaza stvarnih objekata i računalno generiranih objekata te predviđanje budućih lokacija objekata.

Složen problem za tehnologiju proširene stvarnosti predstavlja uklanjanje stvarnih objekata iz scene proširene stvarnosti [37]. Kako bi se stvarni objekt uspješno uklonio iz prikaza potrebno je napraviti segmentaciju objekta. Problem koji je nedavno privukao veću pažnju je negativan utjecaj na vid korisnika u slučaju duže ili češće upotrebe AR uređaja [47]. Negativni utjecaj na vid korisnika se odražava kroz povećano naprezanje i umor očiju što dovodi do oštećivanja vida. Jedan od problema koji je prije predstavljao prepreku u razvoju proširene stvarnosti je problem prenosivosti i upotrebe AR uređaja na otvorenom okruženju [37]. Naime, prije pojave pametnih uređaja bilo je potrebno razviti posebne uređaje za prikaz proširene stvarnosti, a takvi uređaji su često bili složeni, veliki ili skupi. Primjer takvog uređaja je ARQuake uređaj, koji se nosio u torbi na leđima igrača. Neuspjeh AR uređaja Google Glass, pokazao je i problem socijalne prihvatljivosti proširene stvarnosti u slučaju kada je narušena privatnost ljudi. Naime, Google Glass je imao ugrađenu kameru koja se mogla koristiti za neprimjetno slikanje i snimanje osoba iz okruženja korisnika.

5. Proširena stvarnost u Android uređajima

Zbog naglog razvoja sklopovlja i programske podrške, mobilni uređaji postaju savršena platforma za daljnji razvoj proširene stvarnosti. Rad se upravo zbog ove činjenice fokusira na tehnologije implementacije proširene stvarnosti u pametnim mobilnim uređajima odnosno mobilnim aplikacijama za Android uređaje.

5.1. Kotlin

Implementacija proširene stvarnosti u mobilnim uređajima se svodi na razvoj složenih AR mobilnih aplikacija. Kako bi razvoj ovakvih mobilnih aplikacija bio moguć potreban je moćan, napredan i moderan programski jezik. Primjer jednog takvog programskog jezika je zasigurno Kotlin koji se prvi put javlja 2010. godine, a 2016. godine dobiva svoju prvu stabilnu produkcijsku inačicu [48]. „Kotlin je pragmatični, tipirani programski jezik namijenjen za Java virtualnu mašinu (engl. Java Virtual Machine, JVM) i Android platformu, koji kombinira objektno orijentirane i funkcionalne značajke programiranja i fokusira se na interoperabilnost, sigurnost, jasnoću i dobru podršku alata“ [48]. Kad se govori o interoperabilnosti misli se na mogućnost kombiniranja programskog koda napisanog u Kotlinu s Javom, jer se prilikom njegovog kompiliranja koda dobiva Java 6 *bytecode* [49]. Ovo je veoma bitno, jer prije pojave Kotlina, Java je bila daleko najpopularniji programski jezik za razvoj mobilnih aplikacija za Android platformu. Međutim, vremenom su pokazali se nedostatci Jave te je postalo očito da ona teško drži korak s modernijim programskim jezicima. Kao moderna alternativa se javlja programski jezik Kotlin, koji je vremenom postao veoma prihvaćen od strane Android zajednice, čak do te mjere da ga je Google 2019. godine proglasio preporučenim jezikom za razvoj Android mobilnih aplikacija [50]. Trenutno, većina dokumentacije za razvoj Android mobilnih aplikacija sadrži programske primjere pisane u Kotlinu, a pojedine popularne programske knjižnice za Android kao što je Jetpack Compose pisane su isključivo za ovaj programski jezik. Interoperabilnost s Javom je omogućila programerima Android mobilnih aplikacija da postepeno razvijaju i migriraju manje dijelove svojih aplikacija u Kotlin programski kod što je bitno lakši pristup od razvoja cijele aplikacije odjednom.

Iako je Kotlin prije svega programski jeziku za Android mobilne aplikacije, on se može koristiti i u brojne druge svrhe. Pomoću Kotlina moguće je razvijati web aplikacije, aplikacije namijenjene za izvođenje na poslužitelju, višeplatformne aplikacije pa čak i aplikacije koje primjenjuju strojno učenje [49].

Osim interoperabilnosti s Javom te podrškom za brojne različite platforme, Kotlin ima i niz drugih korisnih značajki i prednosti. Kao prvo Kotlin je zbog načina na koji je napravljen veoma jednostavan za učenje, posebno za Java programere [49]. Jedna od najvećih prednosti Kotlina je velika sažetost i čitljivost njegovog programskog koda [49]. Još jedna prednost Kotlina je sigurnost koda koja proizlazi iz njegove sažetosti, čitljivosti, ali i podrške za očuvanje *null* sigurnosti (engl. null safety). Ovo predstavlja osobinu Kotlina koja potiče programera da uvijek pazi na inicijalizaciju i moguća stanja varijabli putem posebnih operatora prilikom razvoja aplikacije kako ne bi došlo do greški i ispada u aplikaciji.

Pored svega navedenog, bitna strana Kotlina je njegova zrelost i velika podrška od strane tvrtke Google i Android zajednice [49]. Naime, Kotlin je projekt otvorenog koda koji se kontinuirano razvija još od 2011. godine. U tom periodu razvijen je velik broj programskih knjižnica, alata i okruženja namijenjenih za rad s Kotlinom [49]. Najznačajniji je Jetpack skup knjižnica i dobrih praksi za razvoj Android mobilnih aplikacija o kojem će se posebno govoriti u sljedećem poglavlju. Također, značajna je i KTX extensions knjižnica koja sadrži brojne značajke Kotlina kao što su korutine (engl. coroutines), proširujuće funkcije (engl. extension functions), lambda izraze, imenovane parametre. [49]. Zbog činjenice da je preko 60% od 1000 najpopularnijih aplikacija na Android Play trgovini razvijeno korištenjem Kotlina, može se zaključiti da će nastaviti trend rasta njegove popularnosti i podrške.

5.1.1. Programski primjeri

Kako bi se lakše prikazale prethodno navedene prednosti, koristit će se isječci programskih kodova koji objašnjavaju pojedine izmjene i novine koje donosi ovaj programski jezik. Kao uvod koristit će se jedan jednostavan primjer. Sve što je potrebno za izradu „Hello World“ aplikacije koja uključuje i rad s *null* operatorima je prikazano u programskom kodu 1.

```
fun main() {
    val message: String = getMessage("Aldin")
    println(message)
}

fun getMessage (name: String?): String {
    val text: String = name ?: "World")
    return "Hello $text"
}
```

Programski kod 1. Primjer jednostavne aplikacije napisane u Kotlinu

Iz navedenog primjera prvo je moguće uočiti način deklariranja funkcija u Kotlinu. Kao što je moguće vidjeti kod funkcije „getMessage“, funkcije se deklariraju korištenjem ključne riječi „fun“ te navođenjem identifikatora, parametara i povratnog tipa. U deklaraciji parametara

na prvom mjestu se nalazi identifikator varijable, a potom se navodi tip varijable koji je odvoje dvotočkom. Ovdje je moguće uočiti i upotreba null operatora „?“, koji označava da varijabla „name“ ne mora imati vrijednost odnosno da je *nullable*. Povratni tip funkcija navodi se odvojen dvotočkom nakon liste parametara. Varijable se navode u sličnom obliku kao i parametri, s tim da je razlika u navođenju da li je riječ o konstantnoj vrijednosti koja se označava sa „val“ ili promjenjivoj vrijednosti koja se označava sa „var“. Također, moguće je vidjeti još jedan *null safety* operator, a to je Elvis operator koji se označava sa „?:“. Ovaj operator se koristi kada se želi ponuditi alternativna vrijednost ako je izraz s lijeve strane *null.* U ovom slučaju ako varijabla „name“ nema vrijednost, koristit će se vrijednost „World“. U konačnici funkcija „getMessage“ će na osnovu prikazanog parametra korištenjem ključne riječi „return“ vratiti tekstualnu vrijednost „Hello Aldin“, koja će potom biti ispisana na zaslon korištenjem naredbe „println“. Ovaj primjer je dobar pokazatelj sažetosti i čitljivosti koda napisanog pomoću programskog jezika Kotlin.

Pored standardnog načina kontrole tijeka izvršavanja programskog koda korištenjem „if“ i „switch“ naredbi, Kotlin uvodi naredbu „when“ koja pruža veću sažetost i čitljivost koda prilikom definiranja mogućih uvjeta [49]. Upotreba „when“ naredbe prikazana je u programskom kodu 2.

```
fun main() {
    val result: Int = getResult(1, 2, '*')
    println(result)
}

fun getResult(a: Int, b: Int, operation: Char = '+'): Int {
    when (operation) {
        '+' -> return a + b
        '-' -> return a - b
        '*' -> return a * b
        else -> {
            println("Nepostojeca operacija")
            return 0
        }
}
```

Programski kod 2. Primjer korištenja Kotlin naredbe "when"

Kao što je moguće vidjeti, „when“ granjanje je sažetije i čitljivije od korištenja „switch“ naredbe ili većeg broja „if“ naredbi. Sintaksa naredbe se sastoji od navođenja varijable na osnovu čije vrijednosti se odlučuje o uvjetu koji će biti ispunjen. Pojedini uvjet se sastoji od navođenja vrijednosti varijable te bloka koda koji će biti izvršen u slučaju da je uvjet ispunjen, s tim da se pomoću ključne riječi „else“ navodi uvjet koji će biti ispunjen samo u slučaju da niti jedan drugi uvjet nije ispunjen. Ovaj primjer je moguće napisati u još sažetijem obliku ako se

uzme u obzir da u Kotlinu nije uvijek nužno navesti tipove varijabli ako se oni mogu predvidjeti na osnovu programskog koda. Također, jednostavnije funkcije koje vraćaju neku određenu vrijednost mogu biti napisane kao izraz koji vraća vrijednost. Ovo se temelji na činjenici da u Kotlinu skoro sve predstavlja izraz koji ima određenu vrijednost. Na osnovu navedenih pravila moguće je smanjiti veličinu koda navedenog primjera (programski kod 3).

```
fun main() {
    val result = getResult(1, 2, '*')
    println(result)
}

fun getResult(a: Int, b: Int, operation: Char = '+') = when (operation) {
    '+'  -> a + b
    '-'  -> a - b
    '*'  -> a * b
    else -> 0
}
```

Programski kod 3. Pojednostavljeni primjer korištenja Kotlin naredbe "when"

Kotlin osim standardnih „for“, „while“ i „do while“ petlji uvodi i „repeat“ petlju koja pomaže u čitljivosti i sažetosti koda [49]. Ova petlja pojednostavljuje izvršavanje programskog koda koji je potrebno ponoviti određeni broj puta. Upotreba Kotlin naredbe „repeat“ prikazana je u programskom kodu 4.

```
fun main() {
    printRectangle("*", 10)
}

fun printRectangle(symbol: String, length: Int){
    repeat (length) println(symbol.repeat(length))
}
```

Programski kod 4. Primjer korištenja Kotlin naredbe "repeat"

Programski kod koji se nalazi u bloku naredbe „repeat“ izvršiti će se 10 puta. Svaka njegova iteracija rezultirat će ispisom proslijeđenog simbola 10 puta korištenjem funkcije „repeat“. Ovo u konačnici dovodi do ispisa pravokutnika.

Mogućnost korištenja proširujućih funkcija predstavlja još jednu prednost Kotlina [49]. One omogućuju da se određena klasa proširi s dodatnom funkcijom koja obavlja neki proizvoljan zadatak vezan uz tu klasu (programski kod 5).

```
fun main() {
    val number: Int = 2
    val isEven: Boolean = number.isEven()
    println(isEven)
}
fun Int.isEven() = this % 2 == 0
```

Programski kod 5. Primjer korištenja proširujuće funkcije

U navedenom primjeru klasa cijelih brojeva „Int“ se proširuje s funkcijom „isEven“ koja vraća istinu ako je broj paran, a laž ako je broj neparan. Prilikom definiranja proširujuće funkcije bitno je da se ispred identifikatora, odvojeno točkom, navede klasa koja se proširuje.

Kotlin donosi više izmjena i novina kada je riječ o objektno orijentiranim konceptima kao što su sučelja, klase, objekti, enkapsulacija i nasljeđivanje. Najveća promjena je u načinu rada s konstruktorima klase, gdje Kotlin nudi mogućnost upotrebe primarnog i sekundarnog konstruktora te inicijalizacijskog bloka koji omogućava izvršavanje dodatnih radnji prilikom inicijalizacije objekta klase. Primjer korištenja ovih koncepata u Kotlinu prikazuje se u programskom kodu 6.

```
fun main() {
    val truck = Truck(2010, "MAN", 10)
    println(truck.weightLimit)
    truck.print()

    val car = Vehicle(2016, "AUDI")
    car.print()
}

interface Printer {
    fun print()
}

open class Vehicle(val year: Int, val brand: String) : Printer {
    init {
        println("Novo vozilo!")
    }

    open val maxWheels: Int = 4

    override fun print() {
        println("Godiste: $year | Brand: $brand")
    }
}

class Truck : Vehicle {
    val weightLimit: Int

    constructor(_year: Int, _brand: String, _weightLimit: Int)
    : super(_year, _brand) {
        weightLimit = _weightLimit
    }

    override val maxWheels: Int = 18

    override fun print() {
      println("Godiste: $year | Brand: $brand | Nosivost: $weightLimit")
    }
}
```

Programski kod 6. Primjena objektno orijentiranih koncepata u Kotlinu

Kao što je moguće vidjeti u primjeru se koristi sučelje „Printer“ koje se definira s ključnom riječi „interface“. Potom se definira klasa „Vehicle“ koja implementira sučelje „Printer“ što je definirano korištenjem dvotočke nakon liste parametara. Ova klasa u primarnom konstruktoru prima godište i brand vozila. U klasi se korištenjem ključne riječi „init“ definira inicijalizacijski blok koji ispisuje poruku prilikom inicijalizacije objekta klase „Vehicle“. Klasa korištenjem ključne riječi „override“ implementira funkciju „print“ sučelja „Printer“. Ova klasa je označena s ključnom riječi „open“ kako bi postala dostupna za nasljeđivanje. Ovu klasu nasljeđuje klasa „Truck“ istom sintaksom kao i kod implementacije sučelja. Ova klasa ima sekundarni konstruktor koji se definira ključnom riječi „constructor“. Bitno je napomenuti da sekundarni konstruktor mora delegirati izvršenje na drugi konstruktor klase ključnom riječi „this“ ili konstruktor roditeljske klase ključnom riječi „super“, što je i slučaj u ovom primjeru. Također, klasa ima svojstvo „weightLimit“ koje se inicijalizira u bloku sekundarnog konstruktora. Svojstva u Kotlinu imaju predefinirane „getter“ i „setter“ funkcije što bitno pojednostavljuje njihovo korištenje. U slučaju da je potrebno implementirati posebnu logiku, Kotlin pruža mogućnost nadjačavanja „getter“ i „setter“ funkcija. U preostalom dijelu klase „Truck“ nadjačava se vrijednost varijable „maxWheels“ i funkcije „print“.

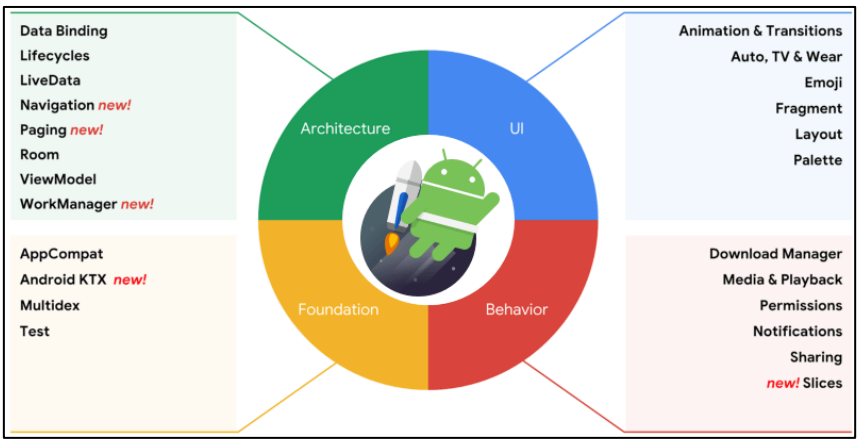
5.2. Android Jetpack

Riječ je o paketu programskih knjižnica (engl. libaries) koje pomažu programerima da slijede najbolje prakse, smanjuju veličinu i složenost koda te pišu kod koji dosljedno radi na različitim Android inačicama i uređajima [51]. Android Jetpack se prvi put pojavio 2018. godine, otkad počinje da se kontinuirano razvija zahvaljujući intenzivnom radu Android razvojnog tima [52]. Danas, Android Jetpack uključuje više od 90 programskih knjižnica i veoma je popularan u Android razvojnoj zajednici što potvrđuje podatak da čak 47% od 1000 najpopularnijih aplikacija na Android Play trgovini je razvijeno korištenjem njegovog skupa programskih knjižnica [52]. Bitno je napomenuti da u ovaj postotak nisu uključene aplikacije razvijene s poznatim standardnim knjižnicama Lifecycle i AppCompact čija upotreba prethodi pojavi samog Android Jetpacka.

Kako bi se Android programerima olakšalo učenje Android Jetpack programskih knjižnica, službena stranica za Android je pripremila svu potrebnu dokumentaciju, objašnjenja, „Codelabs“ tečajeve i programske primjere. Od primjera najznačajnija je „Sunflower“ aplikacija u kojoj se primjenjuje 16 Jetpack programskih knjižnica od kojih su najznačajnije AppCompact, AndroidKTX, Lifecycles, ViewModel, LiveData, Room, Navigation, WorkManager, Hilt, Fragment i Dana Binding [53].

Kao što je moguće vidjeti na slici 10, Android Jetpack programske knjižnice su podijeljene u četiri kategorije [51]:

- Arhitektura (engl. Architecture),
- Korisničko sučelje (engl. User interface),
- Osnova (engl. Foundation) i
- Ponašanje (engl. Behavior).

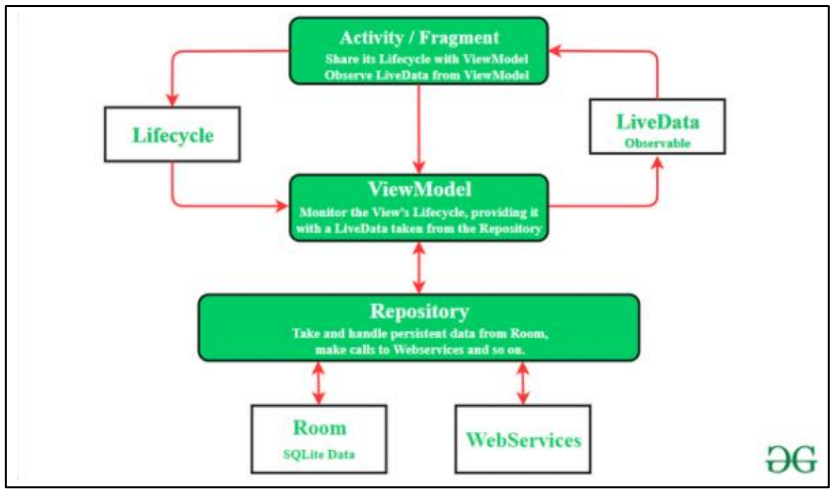


Slika 10. Android Jetpack programske knjižnice [54]

Kako bi se što jasnije prikazala suština i prednosti Android Jetpacka, u nastavku rada će biti razrađene najvažnije programske knjižnice iz prethodno navedenih kategorija.

5.2.1. Arhitekturne komponente

Najznačajnije programske knjižnice iz ove kategorije su Lifecycles, LiveData, ViewModel, Navigation, Room i WorkManager [51]. Lifecycles knjižnica sadrži klase i sučelja koja omogućuju Android programerima da razvijaju programske komponente koje su svjesne životnog ciklusa aplikacije te imaju mogućnost automatskog prilagođavanja njegovom trenutnom stanju [55]. LiveData i ViewModel knjižnice se zajedno koriste prilikom izgradnje model–view–viewmodel (MVVM) arhitekturnog uzorka dizajna (slika 11). Ovaj uzorak dizajna je preporučen za izgradnju arhitekture Android mobilnih aplikacija, jer omogućuje razdvajanje ovisnosti i razvoja korisničkog sučelja od razvoja programske logike aplikacije [55]. MVVM uzorak se sastoji od korisničkog sučelja kojeg čine aktivnosti i fragmenti te *viewmodela* koji prima podatke iz repozitorija te sadrži programsku logiku kojom se primljeni podaci obrađuju i pripremaju za prikaz u aktivnostima i fragmentima. Bitno je napomenuti da viewmodel mora uvijek biti u skladu s životnim ciklusom aktivnosti s kojom je povezan kako bi se očuvala ispravnost i perzistentnost korištenih podataka. Ovo se postiže pomoću programske knjižnice LiveData koja predstavlja klasu koja sadrži podatke čije stanje se može promatrati [55].



Slika 11. Dijagram MVVM arhitekturnog uzorka dizajna [55]

Bitna funkcionalnost velike većine mobilnih aplikacija je navigacija između različitih pogleda. Rad s navigacijom u Android mobilnim aplikacijama bitno je olakšan s programskom knjižnicom Navigation [55]. Pored jednostavnijih oblika navigacije, Navigation olakšava i implementaciju složenijih oblika kao što je *app bar* i *drawer* navigacija. Rad Android mobilnih aplikacija vezan za bazu podataka se još od samog početka oslanja na SQLite [55]. Međutim, vremenom postaju sve jasniji nedostatci ovakvog pristupa. Naime, najveći problemi u radu s SQLite bazom podataka je velika količina ponavljajućeg koda, nedostatak provjere ispravnosti SQL upita prilikom kompiliranja te nemogućnost pohrane *plain-old-Java* objekata. Svi ovi nedostatci su riješeni pojavom programske knjižnice Room [55]. Još jedna veoma korisna programska knjižnica iz ove kategorije je WorkManager koji omogućuje izvođenje pozadinskih zadataka u točno određeno vrijeme bez obzira na trenutno stanje aplikacije [55].

5.2.2. Komponente korisničkog sučelja

Programske knjižnice iz ove kategorije su Animation & Transition koji omogućava korištenje animacija i tranzicija, Auto koji omogućava *hands-free* upotrebu aplikacija, Emoji koji se koristi u radu s emotikonima, Pallete koji pruža podršku za rad s bojama te Fragment i Layout koji se koriste prilikom izgradnje korisničkog sučelja [56].

Jedna od najznačajnijih i najnovijih komponenti Android Jetpacka je Jetpack Compose koji zbog brojnih svojih funkcionalnosti prelazi okvire svih navedenih kategorija. Ipak, u ovom slučaju svrstan je u ovu kategoriju zbog činjenice da je njegova osnovna funkcionalnost najviše vezana za korisničko sučelje. Jetpack Compose tek nedavno dobio svoju inačicu 1.0 odnosno prvu produkcijsku inačicu. Jetpack Compose je novi okvir koji je baziran na programskom jeziku Kotlin, a namijenjen je za brži i lakši razvoj modernih grafičkih sučelja za Android mobilne aplikacije [57]. On omogućuje da se putem komponenti označenih s anotacijom „Composable“

jednostavno izgradi korisničko sučelje. Ove komponente potpuno zamjenjuju dizajn grafičkog sučelja korištenjem XML datoteka i potrebu za vezivanjem sučelja s određenim varijablama. Dizajn sučelja, kao i kontrola njegovog stanja putem različitih varijabli obavlja se na jednom mjestu, odnosno unutar same komponente što omogućava očuvanje principa jednog izvora istine (engl. Single source of truth principle) [57]. Velika prednost biblioteke je visoka interoperabilnost s postojećim bibliotekama koja omogućava postepeni prelazak postojećih aplikacija na Compose arhitekturu. Naime, Jetpack Compose podržava suradnju s XML pogledima (engl. view), MVVM arhitekturnim dizajnom te brojnim Jetpack bibliotekama kao što su Navigation, CameraX, Hilt, LiveData [57].

5.2.3. Komponente ponašanja

Najznačajnije programske knjižnice iz ove kategorije su DownloadManger, Media & Playback, Premissions, Notifications i Sharing [58]. DownloadManger je usluga Android sustava koja olakšava preuzimanje većih datoteka. Programske knjižnice Media & Playback olakšavaju integraciju multimedije u Android aplikacijama. Premissions predstavlja komponentu koja se odnosi na sustav prava za izvršavanje određenih akcija unutar aplikacije kao što su prava za pristup kameri, galeriji, kontaktima ili porukama mobilnog uređaja. Zbog privatnosti i sigurnosti podataka korisnika potrebno je obratiti posebnu pažnju na akcije koje aplikacija izvršava, prava koja su potrebna za njihovo izvršenje te na odgovarajući način obavijestiti korisnika i zatražiti njegovo dopuštenje. Notifications je komponenta koja uključuje prikazivanje obavijesti korisniku na različite načine. Sharing je komponenta koja se odnosi na funkcionalnost dijeljenja sadržaja na različite načine s ostalim korisnicima.

5.2.4. Komponente osnove

Programske knjižnice koje se nalaze u ovoj kategorije su AppCompact, Android KTX, Test i MultiDex [59]. Programska knjižnica AppCompact uključuje brojne komponente bez kojih je razvoj modernih Android mobilnih aplikacija nezamisliv. Primjer nekih od njih su sam AppCompact, CardView, RecyclerView, GridLayout, Preferences i Vector Drawable [59]. Android KTX je kolekcija Kotlin ekstenzija i alata koji omogućuju brži razvoj Android aplikacija. Test je komponenta koja se odnosi na područje testiranja Android aplikacije. MultiDex je možda i najznačajnija komponenta iz ove kategorije iz razloga što omogućuje zaobilaženje nedostataka .dex datoteka. Dex predstavlja format izvršne datoteka koja je namijenjena za izvršavanje na Dalvik Android virtualnoj mašini [59]. Bitan nedostatak Dex datoteka je što omogućavaju pohranu najviše 65.536 funkcija. Upravo zbog ovog nedostatka, MulitDex dijeli Dex datoteku na više datoteka kako bi omogućio pohranu većeg broja funkcija [59].

5.3. Programske knjižnice

Implementacija proširene stvarnosti i prepoznavanja objekata u mobilnim aplikacijama je veoma složen proces koji uključuje razvoj sustava koji omogućuje prepoznavanje i praćenje objekata te generiranje scene proširene stvarnosti. Razvoj ovakvog sustava zahtijeva visoku razinu znanja o konceptima programiranja, proširene stvarnosti i prepoznavanja objekata koju posjeduje mali broj programera mobilnih aplikacija. Kako bi se olakšao ovaj proces i izbjegao besmisao razvoja funkcionalnosti koje već postoje, moguće je koristiti neke od brojnih dostupnih i besplatnih programskih knjižnica. Kako je cilj rada implementacija proširene stvarnosti u svrhu prepoznavanja objekata, potrebno je uzeti u obzir programske knjižnice specijalizirane za implementaciju proširene stvarnosti te programske knjižnice specijalizirane za implementaciju tehnika strojnog učenja kako bi se ostvarila mogućnost prepoznavanja objekata. U nastavku rada bit će analizirani najpopularniji primjeri ovih programskih knjižnica kako bi se utvrdilo koje od njih najbolje odgovaraju cilju ovog rada.

5.3.1. ARCore

Programska knjižnica ARCore je besplatni SDK otvorenog koda za implementaciju proširene stvarnosti u mobilnim aplikacijama kojeg je 2018. godine razvio Google [60]. ARCore omogućava razvoj AR aplikacija za iOS i Android mobilne uređaje. Ova programska knjižnica rješava probleme praćenja lokalizacije korisnika, procjene svjetlosti okruženja te praćenje površina i objekata u okruženju [61]. Dva bitna problema koja ARCore ne rješava su prepoznavanje objekata te grafički prikaz rezultata prepoznavanja i lokalizacije objekata. Međutim, ARCore pruža mogućnost obrade i pohrane sadržaja kojeg hvata kamera mobilnog uređaja. Ovaj sadržaj se potom može koristiti u kombinaciji s programskim knjižnicama za strojno učenje kao što su ML Kit i Tensorflow kako bi se implementirala mogućnost prepoznavanja objekata [62]. Sadržaj kojeg hvata kamera moguće je pripremiti za obradu korištenjem funkcije „acquireCameraImage“ što je prikazano u programskom kodu 7.

```
fun Frame.tryAcquireCameraImage() =
    try {
        acquireCameraImage()
    } catch (e: NotYetAvailableException) {
        null
    } catch (e: RuntimeException) {
        handleAcquireCameraImageFailure(e)
    }

frame.tryAcquireCameraImage()?.use { image ->  //Obrada sadržaja }
```

Programski kod 7. Korištenje ARCore funkcija

U navedenom programskom isječku definirana je proširujuća funkcija „tryAcquireCameraImage“ klase „Frame“. Unutar funkcije poziva se funkcija „acquireCameraImage“ koja će vratiti sadržaj uhvaćen kamerom. Proširujuća funkcija sadrži i rukovanje s iznimkama u slučaju da dođe do greške u radu funkcije „acquireCameraImage“.

5.3.2. CameraX

Programska knjižnica CameraX bitno olakšava razvoj aplikacija koje na određeni način uključuju rad s kamerom mobilnog uređaja. Ona pruža konzistentan i pojednostavljen način rada s kamerom neovisno o kojoj vrsti Android uređaja se radi, s tim da je jedino ograničenje da uređaj mora imati 5.0 ili noviju inačicu Android sustava [63]. Dodatno, prednost CameraX programske knjižnice je u činjenice da omogućuje rad s kamerom koji je svjestan trenutnog životnog ciklusa aplikacije [63]. Sve ovo omogućuje da i manje iskusni Android programeri razvijaju funkcionalnosti koja koristi kameru uređaja i to u svega nekoliko linija programskog koda. Programski kod 8 pokazuje jedan primjer rada s CameraX knjižnicom.

```
fun bindPreview(
    lifecycleOwner: LifecycleOwner,
    previewView: PreviewView,
    cameraProvider: ProcessCameraProvider,
    analyzer: ImageAnalysis.Analyzer,
    executor: Executor
) {
    val preview = Preview.Builder().build().also {
        it.setSurfaceProvider(previewView.surfaceProvider)
    }

    val cameraSelector = CameraSelector.Builder()
        .requireLensFacing(CameraSelector.LENS_FACING_FRONT.build()

    cameraProvider.unbindAll()
    cameraProvider.bindToLifecycle(
        lifecycleOwner,
        cameraSelector,
        setupImageAnalysis(executor, analyzer),
        preview
    )
}
```

Programski kod 8. Primjer korištenja CameraX funkcija

U ovom primjeru se radi o pripremi za pokretanje i hvatanje sadržaja putem prednje kamere uređaja. Na samom početku se pomoću metode „setSurfaceProvider“ klase „Preview“ priprema zaslon aplikacije za prikaz sadržaja kojeg kamera hvata. Nakon toga se klasom „CameraSelector“ i odgovarajućim opcijama odabire zadnja kamera uređaja. Metodom „unbindAll“ gase sve trenutno otvorene kamere. Na kraju se metodom „bindToLifecycle“ vlastoručno napisana metoda „setupImageAnalysis“ zajedno s prethodno definiranim opcijama veže za životni ciklus aplikacije.

5.3.3. TensorFlow Lite

TensorFlow je besplatna programska knjižnica otvorenog koda koja predstavlja platformu za rad sa strojnim učenjem koju je razvio Google za unutarnje potrebe da bi ju tek kasnije učinio dostupnom za širu javnost [64]. Ova programska knjižnica uključuje poznate module TensorFlow Core i TensorFlow Lite [64]. TensorFlow Core daje najveću kontrolu nad pisanjem programskog koda za implementaciju tehnika strojnog učenja i on predstavlja osnovu za gradnju složenijih TensorFlow modula. TensorFlow Lite je okvir za duboko učenje koji se koristi za primjenu tehnika strojnog učenja u mobilnim i IoT uređajima [65]. U pozadini, TensorFlow koristi programski jezik Python kako bi obavio potrebne kalkulacije i obrade podataka te proizveo željene rezultate. Korištenje TensorFlow Lite okvira omogućava stvaranje visoke razina apstrakcije koja od programera krije sve detalje o radu i korištenju algoritama i modela strojnog učenja, kao što je implementacija i treniranje neuronske mreže [65]. U programskom kodu 9 je prikazano korištenje navedenog okvira.

```
val image = TensorImage.fromBitmap(bitmap)

val options = ObjectDetector.ObjectDetectorOptions.builder()
    .setMaxResults(1)
    .setScoreThreshold(0.7f)
    .build()

val detector = ObjectDetector.createFromFileAndOptions(
    this, // application context
    "ml_model.tflite",
    options
)

val results = detector.detect(image)
```

Programski kod 9. Primjer korištenja TensorFlow Lite funkcija

Kao što je moguće vidjeti u navedenom primjeru, Tensorflow Lite omogućava prepoznavanje objekata iz slike u nešto više od deset linija programskog koda. Prvi korak je priprema slike za daljnju obradu korištenjem funkcije „fromBitmap“ klase „TensorImage“. Nakon toga potrebno je pripremiti postavke i uvjete za prepoznavanje objekata, što je u navedenom primjeru učinjeno korištenjem klase „ObjectDetector“. Opcijom „setMaxResults“ je postavljeno da maksimalni broj rezultata bude jedan, dok je opcijom „setScoreThreshold“ minimalni prag prihvatljivosti za točnost rezultata postavljen na 70%. Sljedeći korak je priprema modela strojnog učenja za prepoznavanje objekata, što je učinjeno korištenjem funkcije „createFromFileAndOptions“ klase „ObjectDetector“. Kako bi ova funkcija pripremila model potrebno je proslijediti kontekst aplikacije, naziv datoteke modela strojnog učenja i prethodno definirane opcije. U konačnici, navedeni model, korištenjem funkcije „detect“, prepoznaje objekte iz prethodno pripremljene slike.

6. Implementacija proširene stvarnosti

U ovom dijelu rada se praktičnim primjerom nastoji postepeno prikazati cijeli proces implementacije proširene stvarnosti u svrhu prepoznavanja objekata u Android mobilnoj aplikaciji koja je razvijena pomoću programskog jezika Kotlin. Cilj je omogućiti dublje razumijevanje ovog koncepta prikazom praktične primjene proširene stvarnosti.

6.1. Opis mobilne aplikacije

Cilj praktičnog dijela rada je bio razviti mobilnu aplikaciju čija temeljna funkcionalnost na neki način implementira proširenu stvarnost. Jedan takav način primjene proširene stvarnosti je prepoznavanje objekata. Uzimajući to u obzir, u sklopu praktičnog dijela rada razvijena je aplikacija u kojoj korisnik u stvarnom svijetu nastoj pronaći popis objekata. Sukladno tome aplikacije je nazvana „ScavAR“. Pronalazak objekata svodi se na njihovo uočavanje putem kamere mobilnog uređaja. U trenutku kada je neki objekt uočen kamerom, algoritam prepoznavanja objekata određuje je li riječ o odgovarajućem objektu. U slučaju da jest, objekt je označen kao pronađen, dok u protivnom se potraga nastavlja. Korisnik tijekom potrage prolazi kroz različite razine, gdje se svaka razina razlikuje po vrsti i broju objekata te po težini njihovog pronalaska. Kada su svi objekti određene razine pronađeni, korisniku se otključava sljedeća, teža razina.

Slika 12. Logo aplikacije "ScavAR"

Osim navedenog, postoji i niz drugih zahtjeva koje „ScavAR“, kao jedna moderna i kvalitetna aplikacija, treba ispunjavati. Popis svih funkcionalnih i nefunkcionalnih zahtjeva se nalazi unutar tablice 2.

Tablica 1. Funkcionalni i nefunkcionalni zahtjevi aplikacije "ScavAR"

Funkcionalni zahtjevi	Nefunkcionalni zahtjevi
Kreiranje korisničkog računa	Podržavanje Android SDK-a 8.0 i iznad
Prijava korisničkim računom	Zastoji u radu aplikacije ispod 5 sekundi
Povrat lozinke korisničkog računa	Sigurnost i enkripcija korisničkih podataka
Prikaz objekata koje je potrebno pronaći	Primjena Kotlina i Android Jetpack najboljih principa i standarda razvoja mobilne aplikacije
Prikaz razina aplikacije	Skalabilnost i mogućnost proširenja aplikacije
Pronalaženje objekata korištenjem kamere	-
Pregled i uređivanje podataka korisnika	-

6.2. Dizajn rješenja

Kako bi prethodno navedeni zahtjevi bili ispunjeni, nije bilo dovoljno razviti samo mobilnu aplikaciju, već povezani sustav kojeg prvenstveno sačinjavaju mobilna aplikacija i poslužiteljska strana odnosno web servisi. Kako bi ovaj sustav bio kvalitetan i pouzdan, prilikom njegovog razvoja korištene su najmodernije tehnologije i alati. Ove tehnologije i alati su prikazani na slici 13.

Slika 13. Tehnologije i alati korištene u praktičnom dijelu rada

Prije samog početka razvoja mobilne aplikacije bilo je potrebno napraviti početni dizajn sučelja aplikacije, na osnovu čega bi planiranje i razvijanje aplikacije bilo jednostavnije i brže. Prilikom dizajniranja mobilne aplikacije korišten je programski alat Figma. Nakon ovog koraka započeo je razvoj Android mobilne aplikacije pri čemu je korišten Kotlin. Ovaj programski jezik je odabran zbog svojih brojnih prednosti i naprednih mogućnosti koje su već detaljno razrađene

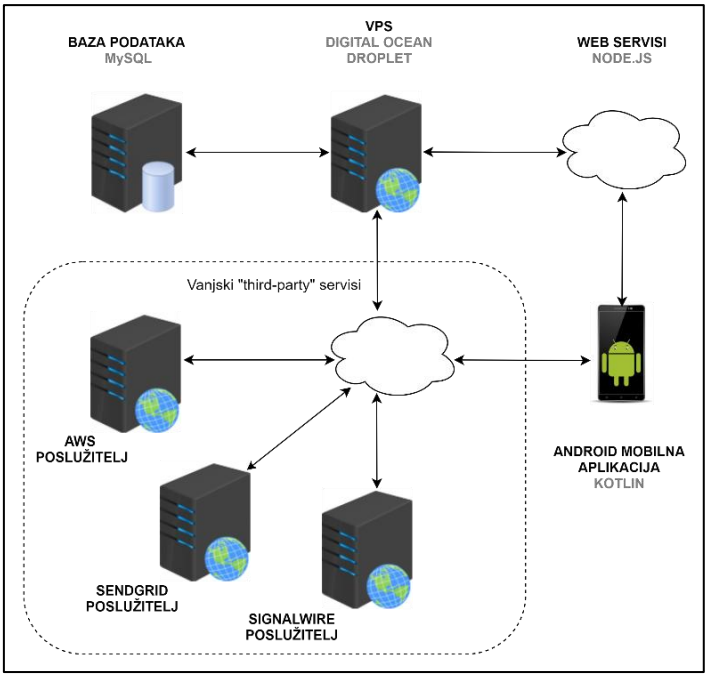
u ranijim poglavljima rada. Dodatno, u razvoju je korišten prethodno opisani Android Jetpack skup programskih knjižnica i dobrih praksi dizajna i razvoja mobilnih aplikacija. Što se tiče same implementacije proširene stvarnosti u svrhu prepoznavanja objekata, korištena je kombinacija programskih knjižnica CameraX i Tensorflow Lite. CameraX je olakšao inače složeni rad s kamerom Android mobilnog uređaja, kao što je obrada sadržaja kojeg kamera hvata. S druge strane, Tensorflow Lite je omogućio analiziranje i obradu navedenog sadržaja različitim tehnikama strojnog učenja u svrhu prepoznavanje objekata koji se u njemu nalaze. Prilikom cijelog razvoja „ScavAR“ mobilne aplikacije korišteno je razvojno okruženje Android Studio u svojoj Arctic Fox .

Kako bi se ispunili svi funkcionalni zahtjevi, pored mobilne aplikacije bilo je potrebno razviti i Web servise koji omogućavaju pohranu, čuvanje i dostavljanje podataka o korisnicima aplikacije „ScavAR“. Na taj način se omogućava registracija i prijava korisnika te dohvat podataka koji su potrebi za normalno izvođenje mobilne aplikacije. Poslužiteljska strana sustava razvijena je korištenjem JavaScript platforme Node.js koja nudi jednostavan, brz i siguran način razvijanja poslužiteljskih aplikacija visokih performansi. Node.js ima veliku podršku od strane programerske zajednice te u kombinaciji s aplikacijskim okvirom Express postaje još jednostavniji za korištenje. Kako bi se omogućila pohrana podataka u bazu, korišten je MySQL koji predstavlja jedan od najpopularnijih sustava za upravljanje relacijskim bazama podataka. Prednosti MySQL-a su sigurnost, velika podržanost od strane zajednice, portabilnost i otvorenost koda. Na poslužiteljskoj strani je korišten alat Docker koji značajno pojednostavljuje razvoj, implementaciju i pokretanje različitih vrsta aplikacija. Docker se zasniva na korištenju jedinki odnosno kontejnera koji se stvaraju na osnovu slike stanja (engl. image), koja u suštini predstavlja instalaciju određene inačice nekog programa. Docker kontejneri su međusobno izolirani te sadrže određeni program, programsku knjižnicu ili konfiguraciju koja se brzo i jednostavno instalira i koristi. Prilikom razvijanja sustava korišteni su Node.js i MySQL kontejneri. Bitno je još napomenuti da je prilikom testiranja i dokumentiranja Web servisa korišten Swagger. Riječ je o programskom alatu koji značajno olakšava pregled, održavanje i testiranje Web servisa putem jednostavnog i oku ugodnog korisničkog sučelja koje je javno dostupno putem IP adrese. Poslužiteljska strana sustava smještena je na udaljenom virtualnom privatnom poslužitelju (engl. Virtual Private Server, VPS) koji je postavljen korištenjem infrastrukture DigitalOceana koji trenutno slovi za jednog od najkvalitetnijih i najpopularnijih pružatelja usluga u oblaku. DigitalOcean omogućuje postavljanje i korištenje virtualnih privatnih poslužitelja putem svojih tzv. DigitalOcean Dropleta koji u suštini predstavljaju virtualne mašine (engl. Virtual Machine, VM) koje korisnik može koristiti u različite svrhe.

Uzimajući u obzir da sustav koristi slike za objekte koje korisnik treba pronaći, bilo je potrebno osigurati način njihove pohrane i dohvaćanja iz određenog online spremišta. Kako bi se ispunio taj zahtjev korišten je Amazon Simple Storage Service, poznatiji kao Amazon S3. Riječ o Amazon Web Service usluzi koja omogućuje korištenje sigurnog online spremišta za pohranu različitih vrsta objekata.

Sustav uključuje i korištenje elektroničke pošte i SMS poruka kako bi dostavio različite korisne informacije korisniku aplikacije, kao što je obavijest o registraciji korisnika i poveznici za aktiviranje računa ili o pokretanju postupka oporavka zaboravljene lozinke. Mogućnost korištenja elektroničke pošte u komunikaciji s korisnikom realizirana je putem SendGrid komunikacijske platforme, dok je komunikacija putem SMS poruka realizirana putem SignalWire komunikacijske platforme.

Arhitektura prethodno opisanog sustava prikazana je na slici 14. Kao što je to moguće vidjeti iz navedenog dijagrama, aplikacije koja je instalirana na Android mobilnom uređaju u svom radu koristi Web servise koji su zajedno s MySQL bazom podataka smješteni na VPS-u odnosno DigitalOcean *dropletu*. Poslužiteljska i mobilna aplikacija u svom radu koriste objekte iz AWS online spremišta kojem pristupaju putem S3 Web servisa. Također, sustav koristi SendGrid i SignalWire Web servise prilikom dostavljanja obavijesti korisniku aplikacije putem elektroničke pošte i SMS poruka.



Slika 14. Arhitektura praktičnog primjera implementacije proširene stvarnosti

Tijekom cijelog procesa razvoja opisanog sustava korišten je Git sustav verzioniranja programskog koda putem GitHub platforme za rad s Git repozitorijima. U ovom slučaju su korištena dva Git repozitorija, „scav-ar“ za mobilnu aplikaciju te „scav-ar-backend“ za poslužiteljsku aplikaciju. Također, putem GitHub platforme i njene funkcionalnosti GitHub Actions uspostavljena je kontinuirana integracija (engl. continuous integration) i kontinuirana isporuka (engl. continuous delivery) čime se automatizira postupak testiranja ispravnosti napravljenih promjena nad programskim kodom poslužiteljske aplikacije te postupak njegove integracije i konačne isporuke na određeni poslužitelj. Kako bi se pojednostavio rad s Git repozitorijima korišteno je Git grafičko sučelje SourceTree. Prilikom verzioniranja programskog koda korišten je GitHub Flow pristup prema kojem se svaka funkcionalnost aplikacije razvija na odvojenoj Git grani te po završetku razvoja spaja u glavnu granu za razvoj koja najčešće nosi naziv „develop“. Kada je aplikacija spremna za objavu, „develop“ grana se spaja u glavu granu koja se najčešće naziva „master“ ili „main“. Na ovaj način se olakšava suradnja, smanjuje broj mogućih problema i konflikata prilikom pravljenja izmjena nad programskim kodom te jasnije organizira razvoj aplikacije.

6.3. Razvoj mobilne aplikacije

Zbog same činjenice da je cilj ovog rada pokazati primjenu proširene stvarnosti u Android mobilnim aplikacijama, a ne njihov općeniti razvoj, u ovom dijelu rada fokus je stavljen na dio programskog koda koji je vezan za implementaciju proširene stvarnost, dok će ostatak biti tek površno obrađen.

Prije početka razvoja mobilne aplikacije „ScavAR“ bilo je potrebno preuzeti i instalirati razvojno okruženje Android Studio. U ovom slučaju je instalirana i korištena njegova inačica Arctic Fox 2020.3.1. Nakon pokretanja Android Studija za potrebe razvoja „ScavAR“ mobilne aplikacije je kreiran novi projekt s praznom aktivnošću te minimalnom podržanom inačicom Android sustava postavljenom na Android 8.0. Nakon kreiranja projekta bilo je potrebno napraviti početnu organizaciju njegove strukture koja je u skladu s Android Jetpack principima. Prvi paket koji je kreiran je „data" i on je namijenjen za grupiranje klasa i datoteka koje su vezane za rad s bazom podataka. Nakon njega je kreiran paket „di“ koji je kasnije korišten za smještanje klasa koje omogućuju korištenje injektiranja ovisnosti (engl. dependency injection). Sljedeći je paket „ui“ koji je namijenjen za komponente korisničkog sučelja. Za potrebe grupiranja pomoćnih klasa i datoteka kreiran je paket „utilities“. Nakon njega je kreiran paket „viewmodels“ u kojem će se kasnije nalaziti ViewModel klase. Na kraju je kreiran paket „workers“ za WorkManager klase. Izvan ovih paketa, smještena je glavna i jedina aktivnost

aplikacije. Aplikacija koristi jednu aktivnost jer se nastoji pratiti „Single Activity“ arhitekturni pristup razvoju aplikacije. Nakon pripreme strukture projekta, stvoreni su uvjeti za pisanje prvih linija programskog koda.

Razvoj mobilne aplikacije započeo je s pripremom aplikacije za korištenje injektiranja ovisnosti (engl. Dependency Injection). Ovaj pristup omogućuje da se u slučajevima kada pojedina klasa koristi funkcionalnost druge klase izbjegne stvaranje prevelike povezanosti različitih dijelova programskog koda čime se povećava njegova fleksibilnost, testabilnost i proširivost. U ovom slučaju je za implementaciju injektiranja ovisnosti korištena programska knjižnica Hilt koja je preporučena kada se radi s Android Jetpackom, jer podržava implementaciju većine njegovih koncepata. Prije nego što se može započeti s korištenjem mogućnostima koje nudi Hilt, potrebno je u glavnoj klasi aplikacije postaviti anotaciju „@HiltAndroidApp“, čime se označava da aplikacija koristi injektiranje ovisnosti putem Hilta. Na ovaj način se prilikom kompajliranja i pripreme aplikacije za pokretanje generiraju sve datoteke koje su potrebne za normalan rad programske knjižnice Hilt. Također, sve klase u kojima se koriste ovisnosti potrebno je označiti s anotacijom „@AndroidEntryPoint“. Hilt omogućava injektiranje ovisnosti putem vezivanja sučelja, konstruktora klase i svojstava klase. Koji od navedenih metoda injektiranja će se koristiti ovisi o više različitih uvjeta. U programskom kodu 10 se prikazuje postupak označavanja glavne klase aplikacije s anotacijom „@HiltAndroidApp“ te injektiranje ovisnosti „HiltWorkerFactory“ putem svojstva klase pri čemu je korištena anotacija „@Inject“.

```
@HiltAndroidApp
class ScavArApplication : Application(), Configuration.Provider {

    @Inject
    lateinit var workerFactory: HiltWorkerFactory

    override fun getWorkManagerConfiguration() =
                Configuration.Builder()
                            .setWorkerFactory(workerFactory)
                            .build()
}
```

Programski kod 10. Priprema „ScavAR“ aplikacije za korištenje programske knjižnice Hilt

Najčešće se koristi ubrizgavanje putem konstruktora sučelja, što je korišteno prilikom razvoja repozitorijskih klasa koje će naknadno biti pojašnjene. Kada je riječ o sučelju ili vanjskoj programskoj knjižnici, potrebno je definirati Hilt module koji će omogućiti injektiranje u navedenim slučajevima. Prilikom razvoja aplikacije definiran je Hilt modul „DatabaseModule“ koji omogućuje korištenje ovisnosti vezanih za rad s bazom podataka. Implementacija ovog modula se nalazi u programskom kodu 11.

```
@Module
@InstallIn(SingletonComponent::class)
object DatabaseModule {

    @Singleton
    @Provides
    fun provideAppDatabase(@ApplicationContext ctx: Context): AppDatabase {
        return AppDatabase.getInstance(ctx)
    }

    @Provides
    fun provideLevelDao(database: AppDatabase): LevelDao {
        return database.levelDao()
    }

    @Provides
    fun provideItemDao(database: AppDatabase): ItemDao {
        return database.itemDao()
    }

    @Provides
    fun provideLevelItemDao(database: AppDatabase): LevelItemDao {
        return database.levelItemDao()
    }

    @Provides
    fun provideRoleDao(database: AppDatabase): RoleDao {
        return database.roleDao()
    }

    @Provides
    fun provideUserDao(database: AppDatabase): UserDao {
        return database.userDao()
    }

    @Provides
    fun provideUserLevelDao(database: AppDatabase): UserLevelDao {
        return database.userLevelDao()
    }
}
```

Programski kod 11. Hilt modul „DatabaseModule"

Kao što je to moguće vidjeti iz programskog koda modula „DatabaseModule“, moduli se označavaju s anotacijom „@Module“ te anotacijom „@InstallIn“ kojom se označava gdje je u kontejneru svih ovisnosti potrebno smjestiti navedeni modul. Unutar modula se korištenjem anotacije „@Provides“ definiraju ovisnosti koje on pruža ostatku aplikacije. U slučaju da je riječ o ovisnosti koja mora biti samo jednom instancirana, kao što je „AppDatabase“ , tada se mora koristi anotacija „@Singleton“.

Kako bi se omogućio rad sa SQLite bazom podataka mobilne aplikacije korištena je prethodno opisana programska knjižnica Room. Tri osnovne komponente koje je potrebno implementirati kad se koristi Room su klasa za uspostavljanje veze s lokalnom bazom

podataka, entitetske klase koje su preslika tablica baze podataka te klase za pristup podacima (engl. Data access object, DAO) koje pružaju metode putem kojih se radi s podacima koji su pohranjeni u bazi podataka.

Prvi korak je bio rekreirati entitetske klase na osnovu prethodno prikazanog modela baze podataka. Na taj način su kreirane klase: „Role“, „User“, „CodeType“, „Code“, „Level“, „Item“, „UserLevel“, „LevelItem“ i „ActiveUserLevelItem“. Primjer implementacije entitetske klase prikazan je u programskom kodu 12.

```
@Entity(
    tableName = "item",
    foreignKeys = [
        androidx.room.ForeignKey(
            entity = Level::class,
            parentColumns = ["id"],
            childColumns = ["level_id"],
        ),
    ],
    indices = [Index("level_id")],
)
data class Item(
    @PrimaryKey
    @ColumnInfo(name = "id") val id: Int,
    @ColumnInfo(name = "level_id")  val levelId: Int,
    @ColumnInfo(name = "name")  val name: String,
    @ColumnInfo(name = "description")  val description: String,
    @ColumnInfo(name = "image")  val imageUrl: String,
    @ColumnInfo(name = "created_at")  val createdAt: Long,
) {
    override fun toString(): String {
        return name
    }
}
```

Programski kod 12. Entitetska klasa "Item"

Na samom početku prikazanog programskog koda se anotacijom „@Entity“ označuje da je riječ o Room entitetskoj klasi. Unutar ove anotacije se prosljeđuje naziv tablice koja se preslikava te vanjski ključevi i indeksi ako postoje. Prilikom definiranja vanjskog ključa potrebno je naznačiti entitetsku klasu te pripadajuće atribute ili stupce na osnovu kojih se uspostavlja veza. Kako je riječ o klasi koja ne sadrži vlastite metode možemo koristiti ključnu riječ „data“. Unutar konstruktora klase, korištenjem anotacije „@ColumnInfo“, označavaju se atributi odnosno svojstva entitetske klase. Na kraju se nadjačava metoda „toString“ tako da ispisuje naziv objekta. Nakon entitetskih klasa došao je red na DAO klase. Zbog toga je bilo potrebno isprogramirati klase: „RoleDao“, „UserDao“, „LevelDao“, „ItemDao“, „UserLevelDao“ te „LevelItemDao“. Kako bi se prikazala implementacija DAO klase za primjer je uzet programski kod klase „LevelItemDao“ koja ima najsloženiju implementaciju od svih implementiranih DAO klasa (programski kod 13).

```
@Dao
interface LevelItemDao
{

    @Query("SELECT * FROM level_item ORDER BY user_level_id")
    fun getLevelItems(): Flow<List<LevelItem>>

    @Query("SELECT * FROM level_item WHERE item_id = :itemId")
    fun getLevelItem(itemId: Int): Flow<LevelItem>

    @Query("SELECT item.id, item.name, item.description, item.image,
EXISTS(SELECT * FROM level_item WHERE item_id = item.id) as completed FROM
item JOIN user_level ON item.level_id = user_level.level_id WHERE
user_level.user_id = :userId AND user_level.completed=0 AND completed=0")
    fun getActiveLevelItems(userId: Int): Flow<List<ActiveLevelItem>>


    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertAll(items: List<LevelItem>)

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(item: LevelItem)

}
```

Programski kod 13. Dao klasa "LevelItemDao"

Kao što je to moguće vidjeti u prikazanom primjeru, DAO klase je prvo potrebno označiti kao sučelja s anotacijom „@Dao“. Unutar sučelja se anotacijom „@Query“ definiraju metode kojima se vrše SQL upiti koji su definirani unutar anotacije. Također, u slučaju unosa vrijednosti u bazu podataka moguće je koristiti anotaciju „@Insert“ kao što je to urađeno kod „insert“ i „insertAll“ metoda. Unutar „@Insert“ anotacije se navodi i način rješavanja konflikata u slučaju kad isti zapis već postoji u bazi podataka.

Kako je cilj da aplikacija bude razvijena prema Android Jetpack principima i MVVM arhitekturnom uzorku dizajna, bilo je potrebno razviti i repozitorijske klase putem koji će se pristupati i raditi s podacima. Na taj način su razvijene klase „UserRepository“, „LevelRepository“ i „ItemRepository“ (programski kod 14).

```
@Singleton
class ItemRepository @Inject constructor(private val itemDao: ItemDao,
private val levelItemDao: LevelItemDao) {

    fun getItems() = itemDao.getItems()
    fun getItem(id: Int) = itemDao.getItem(id)
    fun getLevelItems(id: Int) = levelItemDao.getLevelItems(id)
    fun getActiveLevelItems(id: Int)= levelItemDao.getActiveLevelItems(id)

    suspend fun markItemAsFound(itemId: Int, userLevelId: Int) {
      val item = LevelItem(itemId, userLevelId, system.currentTimeMillis())
        levelItemDao.insert(item)
    }
}
```

Programski kod 14. Repozitorijska klasa "ItemRepository"

Posljednji korak u implementaciji Room načina rada s lokalnom bazom podataka je programiranje klase baze podataka koja nosi naziv „AppDatabase“. U ovoj klasi potrebno je definirati sve entitetske klase i sve DAO klase koje se žele koristiti u radu s bazom podataka. Također, u ovom dijelu se definira metoda za dohvat uvijek iste instance ove klase, tako da u aplikaciji ne bi postajalo više različitih stanja baze podataka. Implementacija klase baze podataka prikazana je u programskom kodu 15.

```
@Database(
    entities = [
                Role::class,
                User::class,
                CodeType::class,
                Code::class,
                Level::class,
                Item::class,
                UserLevel::class,
                LevelItem::class
    ],
    version = 1, exportSchema = false
)
abstract class AppDatabase : RoomDatabase() {
    abstract fun levelDao(): LevelDao
    abstract fun itemDao(): ItemDao
    abstract fun userDao(): UserDao
    abstract fun roleDao(): RoleDao
    abstract fun userLevelDao(): UserLevelDao
    abstract fun levelItemDao(): LevelItemDao

    companion object {
        @Volatile private var instance: AppDatabase? = null
        fun getInstance(context: Context): AppDatabase {
            return instance ?: synchronized(this) {
                instance ?: buildDatabase(context).also { instance = it }
            }
        }
    ⋮
}
```

Programski kod 15. Klasa baze podataka "AppDatabase"

Kao što je moguće vidjeti u prikazanom programskom kodu, klasa baze podataka se označava s anotacijom „@Database“. Ovom anotacijom se definiraju entitetske klase, inačica baze podataka te isporuka sheme. Unutar same klase se definiraju DAO klase kao i metoda za dohvat instance ove klase.

Sukladno MVVM uzorku dizajna definirane su viewmodel klase koje će implementirati programsku logiku te putem repozitorijskih klasa rukovati s podacima. Ovaj način rada dodatno pomaže u izbjegavanju stvaranja prevelike povezanosti programskog koda. Prilikom razvoja „ScavAR“ aplikacije implementirani su „LevelViewModel“, „ItemViewModel“ i „ScanViewModel“. Implementacije viewmodel klase „ScanViewModel“ prikazuje se u programskom kodu 16.

```
@HiltViewModel
class ScanViewModel @Inject constructor(
    private val itemRepository: ItemRepository,
): ViewModel() {

    var currentScan: Scan? by mutableStateOf(null)

    fun getActiveLevelItems(userId: Int) {
        itemRepository.getActiveLevelItems(userId)
                      .asLiveData()
    }

    fun markItemAsFound(
       itemName: String, activeItems: List<ActiveLevelItem>,
       activeLevelId: Int
    ) {

        viewModelScope.launch {
            val item = activeItems.firstOrNull {
                 it.name == itemName.uppercase(Locale.getDefault())
            }

            if (item != null) {
                itemRepository.markItemAsFound(item.id, activeLevelId)
            }
        }

    }

}
```

Programski kod 16. Klasa „ScanViewModel“

Navedeni viewmodel je veoma bitan za rad aplikacije jer omogućava pohranu objekata koji su pronađeni. Prvo što je moguće primijetiti u navedenom programskom isječku je upotreba anotacije „@HiltViewModel“ koja omogućava injektiranje viewmodel klasa putem Hilta. Također, klasa koristi injektiranje putem konstruktora kako bi mogla pristupiti repozitorijskoj klasi „ItemRepository“. Klasa sadrži svojstvo „currentScan“ koje se koristi za pohranu sadržaja kojeg je kamera mobilnog uređaja uhvatila. Najbitniji dio klase je metoda „markItemAsFound“. Unutar ove metode se putem „viewModelScope“ pokreće korutina koja omogućuje izvršavanje određene programske logike bez blokiranja rada mobilne aplikacije. Ovakav način rada je bio neophodan za pozivanje metode s odgođenim izvršavanjem „markItemAsFound“. Metode koje imaju odgođeno izvršavanje u programskom jeziku Kotlin se označavaju korištenjem posebne ključne riječi „suspend“.

Sljedeća Android Jetpack knjižnica koja je bila korištena u razvoju mobilne aplikacije je WorkManager. Ova programska knjižnica je korištena kako bi se omogućilo neometano izvršavanje određenih pozadinskih zadataka aplikacije. Primjer jednog takvog zadatka je priprema početnog stanja baze podataka. Programski kod 17 sadrži dijelove klase „SeedDatabaseWorker“ koji obavljaju opisani zadatak.

```
@HiltWorker
class SeedDatabaseWorker @AssistedInject constructor (
    @Assisted context: Context,
    @Assisted workerParams: WorkerParameters
) : CoroutineWorker(context, workerParams) {

    override suspend fun doWork(): Result = withContext(Dispatchers.IO) {
        val levelsFilename = LEVELS_DATA_FILENAME

        try {
            val database = AppDatabase.getInstance(applicationContext)

            applicationContext.assets.open(levelsFilename).use {

              inputStream ->
              JsonReader(inputStream.reader()).use { jsonReader ->

                 val levelType = object : TypeToken<List<Level>>() {}.type
                 val levels: List<Level> = Gson()
                  .fromJson(jsonReader, levelType)
                  .levelDao().insertAll(levels)
               }
            }

            ⋮

            Result.success()
        } catch (ex: Exception) {
            Log.e(TAG, "Error seeding database", ex)
            Result.failure()
        }
    }
}
```

Programski kod 17. Worker klasa "SeedDatabaseWorker"

Navedeni programski isječak pokazuje još jedan način na koji Hilt podržava korištenje Android Jetpacka, a to je stvaranje uvjeta za injektiranje Worker klasa korištenjem anotacije „@HiltWorker“. Osim toga, u ovom slučaju je korišten i posebni način injektiranje konteksta aplikacije i parametara za rad Worker klase. Riječ je o asistiranom injektiranju koje omogućava korištenje klasa koje su unaprijed pripremljene za injektiranje. U klasi je nadjačana metoda „doWork“ unutar koje se dobavlja instanca baze podataka kako bi se u nju pohranila lista objekata klase „Level“. Ova lista objekata dobivena je čitanjem sadržaja JSON datoteke „levels“ korištenjem programske knjižnice „Gson“. Na isti način se učitava i pohranjuje u bazu podataka lista objekata klase „Item“. Nakon što su obavljeni svi zadaci opisani unutar Worker klase, uspješan ili neuspješan završetak njenog izvođenja mora se označiti korištenjem klase "Result“ i odgovarajućeg njenog objekta.

U razvoju korisničkog sučelja aplikacije, korištena je Android Jetpack knjižnica Compose. Upotreba ove programske knjižnice je bitno skratila veličinu programskog koda mobilne aplikacije te samim tim ubrzala i pojednostavila njen razvoj. Prvi korak u korištenju

knjižnice Jetpack Compose je bio razviti glavnu grafičku komponentu aplikacije koja je nazvana „ScavArApplication“. Implementacija ove komponente prikazuje se unutar programskog koda 18.

```
@Composable
fun ScavArApplication(finishActivity: () -> Unit) {
    ProvideWindowInsets {
        BlueTheme {
            val navController = rememberNavController()
            Scaffold(
                topBar = { TopBar() },
                backgroundColor = MaterialTheme.colors.background,
                bottomBar = { BottomBar(navController = navController) }
            ) { innerPaddingModifier ->
                NavGraph(
                    finishActivity = finishActivity,
                    navController = navController,
                    modifier = Modifier.padding(innerPaddingModifier)
                )
            }
        }
    }
}
```

Programski kod 18. Komponenta "ScavArApplication"

Kao što je to moguće vidjeti iz navedenog programskog koda, compose grafičke komponente označavaju se korištenjem anotacije „@Composable“. Unutar ove komponente se postavlja osnovna tema i struktura korisničkog sučelja aplikacije, kao što su gornja i donja traka te navigacijski graf kojim se definira način kretanja između različitih zaslona aplikacije. Sve ugrađene compose komponente sadrže i svojstvo „modifier“ koje omogućava pravljenje dodatnih izmjena u njihovom dizajnu. Nakon što je završen razvoj ove komponente, istu je potrebno pozvati u aktivnosti „MainActivity“ korištenjem metode „SetContent“.

Nakon ovog koraka, započet je razvoj ostalih grafičkih komponenti korisničkog sučelja aplikacije „ScavAR“. Najznačajnije razvijene grafičke komponenti su „Scan“, „ItemsScreen“, „Items“, „ItemsList“, „ItemCard“ i „CameraPreview“. Implementacija komponente „ItemsList“ prikazana je u programskom kodu 19.

```
@Composable
fun ItemList(items: List<ActiveLevelItem>) {
    LazyColumn (
        modifier = Modifier.fillMaxSize().padding(8.dp),
        verticalArrangement = Arrangement.spacedBy(8.dp),
    )  {
        items(items) { item ->
            ItemCard(item) }
    }
}
```

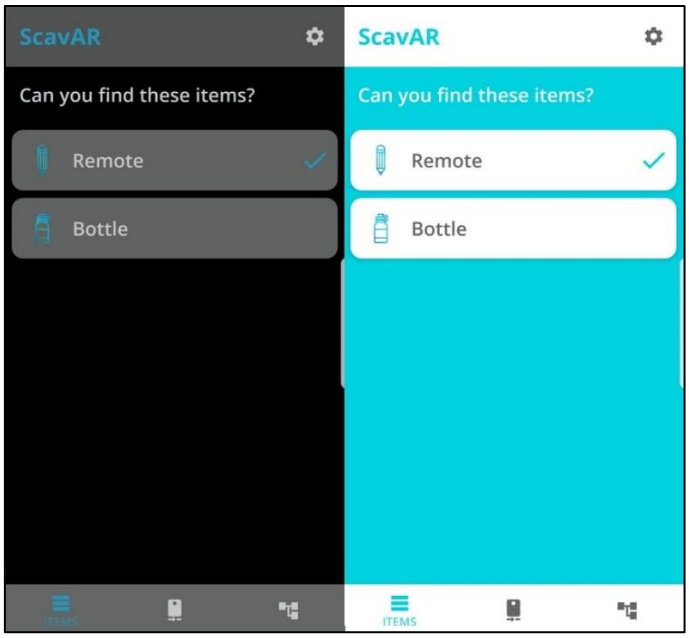
Programski kod 19. Komponenta „ItemList“

U navedenom programskom kodu „ItemList“ korištenjem ugrađene komponente „LazyColumn“ realizira RecyclerView. Riječ je o jedenoj od najčešće korištenih funkcionalnosti u Android mobilnim aplikacijama. Korištenje komponente „LazyColumn“ omogućava implementaciju ove funkcionalnosti u svega nekoliko linija koda umjesto nekoliko datoteka što je prethodno bio slučaj. Ova komponenta u sebi sadrži naredbu „items“ koja se koristi kako bi se generirala lista određenih komponenti. U ovom slučaju to je lista „ItemCard“ komponenti. Programski kod 20 prikazuje implementaciju ove komponente.

```
@Composable
fun ItemCard(item: ActiveLevelItem) {
    val image : String = S3_URL + item.imageUrl
    Card (
        modifier = Modifier.fillMaxWidth(),
        elevation = 4.dp,
        shape = RoundedCornerShape(12.dp),
    ) {
        Row {
            GlideImage(
                imageModel = image,
                contentScale = ContentScale.Crop,
                modifier = Modifier.size(64.dp).padding(8.dp)
            )
            Row(
                modifier = Modifier.height(64.dp).fillMaxWidth(),
                verticalAlignment = Alignment.CenterVertically,
                horizontalArrangement = Arrangement.SpaceBetween,
            ) {
                Text(
                    text = item.description,
                    style = MaterialTheme.typography.h6,
                    color = MaterialTheme.colors.secondary
                )
                if (item.completed) {
                    Icon(
                        imageVector = Icons.Default.Check,
                        contentDescription = "Checked",
                        tint = MaterialTheme.colors.primary,
                        modifier = Modifier.size(48.dp).padding(8.dp)
                    )
                }
            }
        }
    }
}
```

Programski kod 20. Komponenta „ItemCard“

U programskom kodu se primjenjuje veći broj ugrađenih komponenti. Prva koju je moguće uočiti je komponenta „Card“. Ova komponenta se koristi za prikaz kartice. Također, značajna je i komponenta „Row“, koja omogućuje horizontalno redanje komponenti. Jedna od najčešće korištenih komponenti je „Text“, koja se koristi za prikaz teksta unutar aplikacije. Na samom kraju se koristi komponenta za rad s ikonama. Sučelje koje grade „ItemList“ i „ItemCard“ komponente prikazano je na slici 15.



Slika 15. Prikaz glavnog zaslona aplikacije „ScavAR“ u noćnoj i dnevnoj temi

Za temu rada najbitniji dio korisničkog sučelja je zaslon koji prikazuje rad kamere prilikom prepoznavanja objekata. Ovo je implementirano s komponentom „CameraPreview“. Implementacija ove komponente prikazana je u programskom kodu 21.

```
@Composable
fun CameraPreview(analyzer: ImageAnalysis.Analyzer) {
    val lifecycleOwner = LocalLifecycleOwner.current
    val context = LocalContext.current

    val cameraProviderFuture = remember {
            ProcessCameraProvider.getInstance(context)
        }

    AndroidView(
        factory = { ctx ->
            val preview = PreviewView(ctx)
            val executor = ContextCompat.getMainExecutor(ctx)

             cameraProviderFuture.addListener({
                val cameraProvider = cameraProviderFuture.get()
                bindPreview(
                    lifecycleOwner,
                    preview,
                    cameraProvider,
                    analyzer,
                    executor
                )
            }, executor)
            preview
        },
        modifier = Modifier.fillMaxSize(),
    )
}
```

Programski kod 21. Komponenta „CameraPreview“

Na samom početku komponente „CameraPreview“ dohvaća se kontekst aplikacije i objekt „cameraProviderFuture“ koji se koristi prilikom rada s kamerom koji uzima u obzir životni ciklus aplikacije. Ova funkcionalnost je dio programske knjižnice CameraX, koja će u nastavku razvoja imati još važniju ulogu. Nakon toga se koristi „AndroidView“ komponenta kako bi se vratio Android pogled unutar kojeg se pomoću „PreviewView“ prikazuje rad kamere. U ovom dijelu se na objekt „cameraProviderFuture“ postavlja slušač događaja kojim se rad kamere povezuje s „analyzer“ detektorom sadržaja kojeg kamera hvata. Na samom kraju je korištenjem svojstva „modifier“ naznačeno da ovaj pogled zauzme punu širinu i visinu trenutno prikazanog zaslona aplikacije.

Nakon što je završen razvoj osnovnog dijela korisničkog sučelja aplikacije, započela je implementacija prepoznavanja objekata korištenjem proširene stvarnosti. U tu svrhu je prvo korištena programska knjižnica CameraX. Ona je omogućila jednostavniji rad s kamerom mobilnog uređaja. Programski kod 22 sadrži prikaz korištenja ove knjižnice kako bi se pripremio rad s kamerom u „ScavAR“.

```
private fun bindPreview(
    lifecycleOwner: LifecycleOwner,
    previewView: PreviewView,
    cameraProvider: ProcessCameraProvider,
    analyzer: ImageAnalysis.Analyzer,
    executor: Executor
) {

    val preview = Preview.Builder().build().also {
        it.setSurfaceProvider(previewView.surfaceProvider)
    }

    val cameraSelector = CameraSelector.Builder()
        .requireLensFacing(CameraSelector.LENS_FACING_BACK).build()

    cameraProvider.unbindAll()
    cameraProvider.bindToLifecycle(
        lifecycleOwner,
        cameraSelector,
        setupImageAnalysis(executor, analyzer),
        preview
    )
}

private fun setupImageAnalysis(executor: Executor, analyzer:
ImageAnalysis.Analyzer): ImageAnalysis {
    return ImageAnalysis.Builder()
        .setTargetResolution(Size(720, 1280))
        .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
        .build().apply {
            setAnalyzer(executor,analyzer)
        }
}
```

Programski kod 22. Implementacija rada s kamerom u „ScavAR“

U prikazanom programskom kodu priprema se pokretanje i hvatanje sadržaja korištenjem zadnje kamere mobilnog uređaja. Na početku se pomoću metode „setSurfaceProvider“ klase „Preview“ priprema zaslon aplikacije za prikaz sadržaja kojeg kamera hvata. U nastavku koda se klasom „CameraSelector“ i odgovarajućim opcijama odabire zadnja kamera uređaja. Metodom „unbindAll“ gase sve trenutno otvorene kamere kako bi se omogućilo pokretanje njene nove instance. Korištenjem „bindToLifecycle“ metoda „setupImageAnalysis“ zajedno s prethodno definiranim opcijama veže se za životni ciklus mobilne aplikacije. Unutar ove metode postavlja se način analiziranja i pohrane sadržaja uhvaćenog putem kamere uređaja.

Nakon što je kamera pripremljena za rad, sve što je preostalo je implementirati način prepoznavanja objekata iz sadržaja kojeg ona hvata. U realizaciji ove funkcionalnosti korištena je prethodno opisana programska knjižnica TensorFlow Lite. U programskom kodu 23 je prikazan način implementacije prepoznavanja objekata u aplikaciji „ScavAR“.

```
private fun runObjectDetection(context: Context, bitmap: Bitmap): Scan? {
    val image = TensorImage.fromBitmap(bitmap)
    val options = ObjectDetector.ObjectDetectorOptions.builder()
        .setMaxResults(5)
        .setScoreThreshold(0.4f)
        .build()
    val detector = ObjectDetector.createFromFileAndOptions(
        context,
        "model.tflite",
        options
    )

    val results = detector.detect(image)
    if (results.isNullOrEmpty()) return null
    val bestResult = results.maxByOrNull { it ->
        it.categories.first().score } ?: return null

    val category = bestResult.categories.first()
    val text = category.label
    val resultToDisplay = DetectionResult(bestResult.boundingBox, text)
    return Scan(frame = resultToDisplay.boundingBox, text = text)
}
```

Programski kod 23. Implementacija prepoznavanja objekata u aplikaciji „ScavAR“

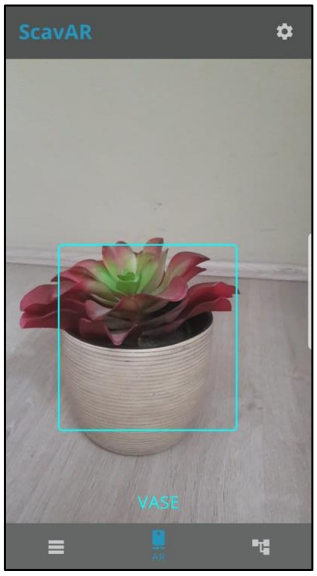
Na početku metode „runObjectDetection“ radi se pretvorba primljenog sadržaja u format koji je pogodan za rad programske knjižnice TensorFlow Lite. Pretvorba je napravljena korištenjem metode „fromBitmap“ klase „TensorImage“. Nakon toga su pripremljene postavke i uvjeti za prepoznavanje objekata korištenjem klase „ObjectDetector“. Opcijom „setMaxResults“ je postavljeno da maksimalni broj rezultata bude pet, dok je opcijom „setScoreThreshold“ minimalni prag prihvatljivosti za točnost rezultata postavljen na 40%. U nastavku metode priprema se model strojnog učenja za prepoznavanje objekata, što je učinjeno korištenjem metode „createFromFileAndOptions“. Kako bi ova funkcija pripremila

model potrebno je proslijediti kontekst aplikacije, naziv datoteke modela strojnog učenja i ostale prethodno definirane opcije. Na osnovu navedenog modela korištenjem funkcije „detect“, pokušavaju se prepoznati objekti iz prethodno pripremljene slike. U slučaju da su prepoznati određeni objekti, dohvaća se objekt koji s najvećom razinom točnosti. Na osnovu ovog objekta se priprema tekst i okvir objekta. Rezultati se na kraju prosljeđuju komponenti „Scan“ u kojoj se programskim kodom 24 prikazuje scena proširene stvarnosti.

```
drawRect(
    color = Color.Cyan,
    topLeft = offset,
    size = size,
    style = Stroke(6f, pathEffect = PathEffect.cornerPathEffect(15f))
)
Text(
     modifier = Modifier.fillMaxWidth().padding(bottom = 16.dp),
     text = scan.text.toUpperCase(Locale.getDefault()),
     textAlign = TextAlign.Center,
     fontSize = 20.sp,
     color = Color.Cyan
)
```

Programski kod 24. Prikazivanje scene proširene stvarnosti

Zaslon na kojem se izvršava prepoznavanje objekata i prikazivanje scene proširene stvarnosti je prikazan na slici 16. Kao što je moguće vidjeti na slici, lokacija prepoznatog objekta se prikazuje plavim okvirom. Ispod okvira se tekstualno prikazuje vrsta objekta.



Slika 16. Prikaz prepoznavanja objekata unutar "ScavAR"

7. Zaključak

Detaljna razrada tehnologije proširene stvarnosti pokazala je njen dugi put razvoja. Veoma skupo i nepraktično sklopovlje te složena programska podrška potrebna za realizaciju prvih primjera proširene stvarnosti, zbog naglog napretka tehnologije, je zamijenjena s pametnim mobilnim uređajima koje većina ljudi posjeduje te besplatnim AR mobilnim aplikacijama. Kao što je to prikazano na praktičnom primjeru, jednostavnije primjere AR mobilnih aplikacija je moguće relativno brzo razviti ako se koriste moderne tehnologije i alati. Iako je rad pokazao da proširena stvarnost već sad ima veoma veliku primjenu, smatra se da je ona još uvijek u svojim ranim fazama razvoja i da će daljnjim napretkom njene prave mogućnosti tek biti pokazane.

Popis literature

1. M. Hrga, "Računalni vid", Zbornik radova Veleučilišta u Šibeniku, sve. 1-2/2018, str. 207-216, 2018. [Na internetu]. Dostupno: Hrčak, http://hrcak.srce.hr/. [pristupano 02.08.2021].
2. L. Wang, J. Shi, G. Song, i I. Shen. "Object Detection Combining Recognition and Segmentation". Computer Vision – ACCV 2007, Computer Science, sve. 4843, 2017. [Na internetu]. Dostupno: Springer, https://link.springer.com/chapter/10.1007/978-3-540-76386-4_17. [pristupano 02.08.2021].
3. Z. Wu, W. Tao, G. Lin, i J. Cai, "Exploring Bottom-up and Top-down Cues with Attentive Learning for Webly Supervised Object Detection", ožu. 2020. [Na internetu]. Dostupno: https://arxiv.org/pdf/2003.09790.pdf. [pristupano 03.08.2021].
4. Z. Zou, Z. Shi, Y. Guo, i J. Ye, "Object Detection in 20 Years: A Survey", 16.05.2019. [Na internetu]. Dostupno: https://arxiv.org/pdf/1905.05055.pdf. [pristupano 03.08.2021].
5. Kaggle. "Object Recognition vs Object Detection vs Image Segmentation", 2020. [Na internetu]. Dostupno: https://www.kaggle.com/getting-started/169984. [pristupano 03.08.2021].
6. A. Andreopoulosa, i J. K. Tsotsosb, "50 Years of object recognition: Directions forward", Computer Vision and Image Understanding, sve. 117, izd. 8, str. 827-891, kol. 2013. [Na internetu]. Dostupno: ScienceDirect, https://www.sciencedirect.com/science/article/abs/pii/S107731421300091X#!. [pristupano 03.08.2021].
7. Z. Zou, "A Review of Object Detection Techniques", International Conference on Smart Grid and Electrical Automation (ICSGEA), kol. 2019. [Na internetu]. Dostupno: ResearchGate, https://www.researchgate.net/publication/337513199_A_Review_of_Object_Detection_Techniques. [pristupano 03.08.2021].
8. P. Črnčec i D. Andročec, "Primjena strojnog učenja na problemu kolorizacije crno-bijelih slika", Zbornik radova Veleučilišta u Šibeniku, sve.14, br. 3-4, str. 123-135, 2020. [Na internetu]. Dostupno: Hrčak, https://hrcak.srce.hr/248679. [pristupano 04.08.2021].
9. A. Zheng, i A. Casari, *Feature Engineering for Machine Learning: Principles and Techniques for Dana Scientists*, Sebastopol, CA, USA: O'Reilly Media, Inc., 2018.
10. N. Buduma, *Fundamentals of Deep Learning*, Sebastopol, CA, USA: O'Reilly Media, Inc., 2017, str. 7.
11. Ž. Ujević Andrijić i N. Bolf, "Osvježimo znanje: Umjetne neuronske mreže", Kemija u industriji, izd. 68, br. 5-6, str. 219-220, 2019. [Na internetu]. Dostupno: https://hrcak.srce.hr/220716. [pristupano 05.08.2021].

12. Y. LeCun, L. Bottou, Y. Bengio, i P. Haffner, "Gradient-based learning applied to document recognition", Proceedings of the IEEE, sve. 86, izd.11, str. 2278-2324, stu. 1998. [Na internetu]. Dostupno: IEEE Xplore, https://ieeexplore.ieee.org/document/726791. [pristupano 05.08.2021].
13. Anushka, C. Arya, A. Tripathi, P. Singh, M. Diwakar, K. Sharma, i H. Pandey, "Object Detection using Deep Learning: A Review", 2021. [Na internetu]. Dostupno: https://iopscience.iop.org/article/10.1088/1742-6596/1854/1/012012/pdf. [pristupano 05.08.2021].
14. P. Schueffel, "The Concise Fintech Compendium", 2017. [Na internetu]. Dostupno: ResearchGate, https://www.researchgate.net/publication/322819310_The_Concise_Fintech_Compendium. [pristupano 05.08.2021].
15. J. Carmigniani i B. Furht, "Augmented Reality: An Overview", srp. 2011. [Na internetu]. Dostupno: ResearchGate, https://www.researchgate.net/publication/227164365_Augmented_Reality_An_Overview. [pristupano 07.08.2021].
16. R. Silva, J. C. Oliveira i G. A. Giraldi, "Introduction to Augmented Reality", sij. 2003. [Na internetu]. Dostupno: ResearchGate, https://www.researchgate.net/publication/277287908_Introduction_to_augmented_reality. [pristupano 07.08.2021].
17. R. Azuma, "A survey of augmented reality", Teleoperators and Virtual Environments sve. 6, izd. 4, str. 355-385, kol. 1997. [Na internetu]. Dostupno: https://www.cs.unc.edu/~azuma/ARpresence.pdf. [pristupano 08.08.2021].
18. Immersiv.io, "What is Augmented Reality? How does it work? Let's see! ", srp. 2020. [Na internetu]. Dostupno: https://www.immersiv.io/blog/what-is-augmented-reality-definition. [pristupano 08.08.2021].
19. D. Krevelen i R. Poelman, "A Survey of Augmented Reality Technologies, Applications and Limitations", The International Journal of Virtual Reality. sve. 9, izd. 2, str. 1-20, lip. 2010. [Na internetu]. Dostupno: ResearchGate, https://www.researchgate.net/publication/279867852_A_Survey_of_Augmented_Reality_Technologies_Applications_and_Limitations. [pristupano 08.08.2021].
20. R. Metz, "Augmented Reality Is Finally Getting Real", MIT Technology Review, 2012. [Na internetu]. Dostupno: https://www.technologyreview.com/s/428654/augmented-realityis-finally-getting-real. [pristupano 11.08.2021].
21. J. Normand, M. Servieres i G. Moreaum, "A typology of augmented reality applications based on their tracking requirements", ožu. 2012. [Na internetu]. Dostupno: ResearchGate, https://www.researchgate.net/publication/245032078_A_new_typology_of_Augmented_Reality_applications. [pristupano 11.08.2021].

22. A. Edwards-Stewar, T. Hoyt i G. Reger, "Classifying different types of augmented reality technology", Annual Review of CyberTherapy and Telemedicine, sve. 14, str. 199-202, sij. 2016. [Na internetu]. Dostupno: ResearchGate, https://www.researchgate.net/publication/315701832_Classifying_different_types_of_augmented_reality_technology. [pristupano 12.08.2021].
23. W. Mackay, "Augmented Reality: Linking real and virtual worlds - A new paradigm for interacting with computers", srp. 2020. [Na internetu]. Dostupno: ResearchGate, https://www.researchgate.net/publication/2423811_Augmented_Reality_Linking_real_and_virtual_worlds_A_new_paradigm_for_interacting_with_computers. [pristupano 14.08.2021].
24. T. Baudel i M. Beaudouin-Lafon, "Charade: Remote control of objects using free-hand gestures", Communications of the ACM, sve. 36, izd. 7, str. 28-35, srp. 1993. [Na internetu]. Dostupno: ResearchGate, https://www.researchgate.net/publication/2378291_CHARADE_Remote_Control_of_Objects_using_FreeHand_Gestures. [pristupano 14.08.2021].
25. M. Weiser, "The Computer for the 21st Century", Scientific American, sve. 265, izd. 3, ruj. 1991. [Na internetu]. Dostupno: https://www.lri.fr/~mbl/Stanford/CS477/papers/Weiser-SciAm.pdf. [pristupano 14.08.2021].
26. W. Newman, M. Eldridge i M. Lamming, "PEPYS: Generating autobiographies by automatic tracking", Proceedings of the Second European Conference on Computer Supported Cooperative Work, sij. 1991. [Na internetu]. Dostupno: ResearchGate, https://www.researchgate.net/publication/220265912_PEPYS_Generating_Autobiographies_by_Automatic_Tracking. [pristupano 15.08.2021].
27. Krueger, M., *Artificial Reality,* New York City, NY, USA: Addison-Wesley, 1983.
28. R. Bolt, "Put-That-There: Voice and gesture at the graphics interface", Computer Graphics, ACM SIGGRAPH, sve. 14, br. 3, str. 262-270, 1980. [Na internetu]. Dostupno: ACM, https://dl.acm.org/doi/10.1145/965105.807503. [pristupano 15.08.2021].
29. M. L. Heilig, "EL Cine del Futuro: The Cinema of the Future", Presence: Teleoperators and Virtual Environments, sve. 1, izd. 3, str. 279–294, 1992. [Na internetu]. Dostupno: ACM, https://dl.acm.org/doi/10.5555/2870742.2870744. [pristupano 16.08.2021].
30. I. E. Sutherland, "A head-mounted three-dimensional display", Proc. Fall Joint Computer Conf., str. 757–764, pro. 1968. [Na internetu]. Dostupno: ACM, https://dl.acm.org/doi/10.1145/1476589.1476686. [pristupano 17.08.2021].
31. J. Carmigniani i B. Furht, "Augmented Reality: An Overview", str 4-6, srp. 2011. [Na internetu]. Dostupno: ResearchGate, https://www.researchgate.net/publication/227164365_Augmented_Reality_An_Overview. [pristupano 17.08.2021].

32. T. P. Caudell i D. W. Mizell, "Augmented reality: An application of heads-up display technology to manual manufacturing processes", Proc. Hawaii Int'l Conf. on Systems Sciences, velj. 1992. [Na internetu]. Dostupno: ResearchGate, https://www.researchgate.net/publication/3510119_Augmented_reality_An_application_of_heads-up_display_technology_to_manual_manufacturing_processes. [pristupano 18.08.2021].
33. S. Feiner, B. Macintyre i D. Seligmann, "Knowledge-Based Augmented Reality", Commun. ACM. sve. 36, str. 53-62., srp. 1993. [Na internetu]. Dostupno: ResearchGate, https://www.researchgate.net/publication/220420450_Knowledge-Based_Augmented_Reality. [pristupano 18.08.2021].
34. C. Arth, R. Grasset, L. Gruber, L. Tobias, A. Mulloni i D. Wagner, "The History of Mobile Augmented Reality", svi. 2015. [Na internetu]. Dostupno: ResearchGate, https://www.researchgate.net/publication/275974448_The_History_of_Mobile_Augmented_Reality. [pristupano 20.08.2021].
35. NMC: The New Media Consortium, "The Horizon Report", 2005. [Na internetu]. Dostupno: https://library.educause.edu/-/media/files/library/2005/1/csd3737-pdf.pdf. [pristupano 20.08.2021].
36. S. Narayanan, "Is Microsoft HoloLens a Tesla Roadster or a Google Glass? ", pro. 2016. [Na internetu]. Dostupno: https://1reddrop.com/2016/12/02/microsoft-hololens-tesla-roadster-google-glass. [pristupano 21.08.2021].
37. R. Van Krevelen, "Augmented Reality: Technologies, Applications, and Limitations", tra. 2007. [Na internetu]. Dostupno: ResearchGate, https://www.researchgate.net/publication/292150312_Augmented_Reality_Technologies_Applications_and_Limitations. [pristupano 20.08.2021].
38. S. G. Dacko, "Enabling smart retail settings via mobile augmented reality shopping apps", lis. 2016. [Na internetu]. Dostupno: ResearchGate, https://www.researchgate.net/publication/309217092_Enabling_smart_retail_settings_via_mobile_augmented_reality_shopping_apps. [pristupano 20.08.2021].
39. P. Bhageria, "Augmented Reality: The Latest Trend In The Fashion Industry", 2021. [Na internetu]. Dostupno: https://inhaabit.com/augmented-reality-the-latest-trend-in-the-fashion-industry. [pristupano 22.08.2021].
40. R. Arthur, "Augmented Reality Is Set To Transform Fashion And Retail", lis. 2017. [Na internetu]. Dostupno: https://www.forbes.com/sites/rachelarthur/2017/10/31/augmented-reality-is-set-to-transform-fashion-and-retail/#3ece57e33151. [pristupljeno: 22.08.2021].
41. M. C. White, "Before You Buy That Couch, an App Will Put It in Your Living Room", ruj. 2018. [Na internetu]. Dostupno: https://www.nytimes.com/2018/09/16/business/media/furniture-advertising-virtual-reality.html. [pristupano 22.08.2021].

42. L. Nguyen, "10 best augmented reality games and AR games for Android", lis. 2020. [Na internetu]. Dostupno: https://www.androidauthority.com/best-augmented-reality-games-ar-games-android-755298. [pristupano 22.08.2021].
43. Pokemon GO, "Catch Pokemon", 2021. [Na internetu]. Dostupno: https://pokemon golive.com. [pristupano 22.08.2021].
44. H. Ho-Gun, i H. Jaesung, "Augmented Reality in Medicine", Hanyang Medical Reviews sve. 36, izd. 4, br. 242, sij. 2016. [Na internetu]. Dostupno: ResearchGate, https://www.researchgate.net/publication/311467975_Augmented_Reality_in_Medicine. [pristupano 23.08.2021].
45. The Alliance of Advanced Biomedical Engineering, "Augmented Reality – Revolutionizing the Health Care", 2021. [Na internetu]. Dostupno: https://aabme.asme.org/posts/novel-augmented-reality-technology-to-revolutionize-the-health-care-industry. [pristupano 23.08.2021].
46. R. Azuma, "Tracking Requirements for Augmented Reality", Communications of the ACM, sve. 36, izd. 7, str. 50-51, srp. 1993. [Na internetu]. Dostupno: ACM, https://dl.acm.org/doi/10.1145/159544.159581. [pristupano 23.08.2021].
47. S. R. Ellis, F. Breant, B. Manges, R. Jacoby i B. D. Adelstein, "Factors influencing operator interaction with virtual objects viewed via headmounted seethrough displays: Viewing conditions and rendering latency", VRAIS'97: Proc. Virtual Reality Ann, Intl Symp, IEEE CS Press, str. 138–145, ožu. 1997. [Na internetu]. Dostupno: https://ieeexplore.ieee.org/document/583063. [pristupano 23.08.2021].
48. A. Berslav, "Kotlin 1.0 Released: Pragmatic Language for JVM and Android", sij. 2016. [Na internetu]. Dostupno: https://blog.jetbrains.com/kotlin/2016/02/kotlin-1-0-released-pragmatic-language-for-jvm-and-android. [pristupano 24.08.2021].
49. Kotlinlang, "Kotlin Language Documentation 1.5.21", velj. 2021. [Na internetu]. Dostupno: https://kotlinlang.org/docs/kotlin-reference.pdf. [pristupano 24.08.2021].
50. F. Lardinois, "Kotlin is now Google's preferred language for Android app development", svi. 2019. [Na internetu]. Dostupno: https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development. [pristupano 24.08.2021].
51. Android, "Android Jetpack", 2021. [Na internetu]. Dostupno: https://developer.android.com/jetpack. [pristupano 24.08.2021].
52. D. Wong, "11 Weeks of Android: Jetpack", srp. 2020. [Na internetu]. Dostupno: https://android-developers.googleblog.com/2020/07/11-weeks-of-android-jetpack.html. [pristupano 24.08.2021].
53. Android, "Sunflower", 2021. Github repozitorij. Dostupno: https://github.com/android /sunflower. [pristupano 25.08.2021].

54. Morris, "Android Jetpack", sij. 2019. [Na internetu]. Dostupno: https://androidwave.com/android-jetpack. [pristupano 25.08.2021].
55. GeeksForGeeks, "Jetpack Architecture Components in Android", kol. 2021. [Na internetu]. Dostupno: https://www.geeksforgeeks.org/jetpack-architecture-components-in-android. [pristupano 25.08.2021].
56. GeeksForGeeks, "UI Components of Android Jetpack", lip. 2021. [Na internetu]. Dostupno: https://www.geeksforgeeks.org/ui-components-of-android-jetpack. [pristupano 25.08.2021].
57. Android, "Build better apps faster with Jetpack Compose", 2021. [Na internetu]. Dostupno: https://developer.android.com/jetpack/compose. [pristupano 25.08.2021].
58. GeeksForGeeks, "Behaviour Components of Android Jetpack", velj. 2021. [Na internetu]. Dostupno: https://www.geeksforgeeks.org/behaviour-components-of-android-jetpack. [pristupano 25.08.2021].
59. GeeksForGeeks, "Foundation Components of Android Jetpack", velj. 2021. [Na internetu]. Dostupno: https://www.geeksforgeeks.org/foundation-components-of-android-jetpack. [pristupano 26.08.2021].
60. N. Paweł i W. Marek, "Capabilities of ARCore and ARKit Platforms for AR/VR Applications", sij. 2020. [Na internetu]. Dostupno: ResearchGate, https://www.researchgate.net/publication/333039077_Capabilities_of_ARCore_and_ARKit_Platforms_for_AR VR_Applications/link/5d24e096299bf1547ca75dfe/download. [pristupano 26.08.2021].
61. Google Developers, "ARCore", 2021. [Na internetu]. Dostupno: https://developers.google.com/ar/develop. [pristupljeno 27.08.2021].
62. Google Developers, "Use ARCore as input for Machine Learning models", 2021. [Na internetu]. Dostupno: https://developers.google.com/ar/develop/java/machine-learning. [pristupljeno 27.08.2021].
63. Android, "CameraX Overview", 2021. [Na internetu]. Dostupno: https://developer.android.com/training/camerax. [pristupljeno 28.08.2021].
64. TensorFlow, "TensorFlow Core", 2021. [Na internetu]. Dostupno: https://www.tensorflow.org/overview. [pristupljeno 29.08.2021].
65. TensorFlow, "For Mobile & IoT", 2021. [Na internetu]. Dostupno: https://www.tensorflow.org/lite. [pristupljeno 29.08.2021].
66. TensorFlow, "Object detection", 2021. [Na internetu]. Dostupno: https://www.tensorflow.org/lite/examples/object_detection/overview. [pristupljeno 29.08.2021].

Popis slika

Slika 1. Razlika između klasifikacije i prepoznavanja objekata [5] 3
Slika 2. Porast broja znanstvenih radova o prepoznavanju objekata 5
Slika 3. Vremenska crta razvoja prepoznavanja objekata [3] 6
Slika 4. Pojednostavljeni prikaz umjetne neuronske mreže [11] 8
Slika 5. Primjeri upotrebe proširene stvarnosti [18] 13
Slika 6. Vrste prikaza proširene stvarnosti 14
Slika 7. Damoklov mač [30] 19
Slika 8. Usporedba uređaja Google Glass i Microsoft HoloLens [35] 21
Slika 9. Mobilna igra Pokemon GO [43] 23
Slika 10. Android Jetpack programske knjižnice [54] 32
Slika 11. Dijagram MVVM arhitekturnog uzorka dizajna [55] 33
Slika 12. Logo aplikacije "ScavAR" 38
Slika 13. Tehnologije i alati korištene u praktičnom dijelu rada 39
Slika 14. Arhitektura praktičnog primjera implementacije proširene stvarnosti 41
Slika 15. Prikaz glavnog zaslona aplikacije u noćnoj i dnevnoj temi 52
Slika 16. Prikaz prepoznavanja objekata unutar "ScavAR" 55

Popis tablica

Tablica 1. Funkcionalni i nefunkcionalni zahtjevi aplikacije "ScavAR" 38

Popis programskih kodova

Programski kod 1. Primjer jednostavne aplikacije napisane u Kotlinu 27
Programski kod 2. Primjer korištenja Kotlin naredbe "when" 28
Programski kod 3. Pojednostavljeni primjer korištenja Kotlin naredbe "when" 29
Programski kod 4. Primjer korištenja Kotlin naredbe "repeat" 29
Programski kod 5. Primjer korištenja proširujuće funkcije 29
Programski kod 6. Primjena objektno orijentiranih koncepata u Kotlinu 30
Programski kod 7. Korištenje ARCore funkcija 35
Programski kod 8. Primjer korištenja CameraX funkcija 36
Programski kod 9. Primjer korištenja TensorFlow Lite funkcija 37
Programski kod 10. Priprema „ScavAR“ aplikacije za korištenje programske knjižnice Hilt 43
Programski kod 11. Hilt modul „DatabaseModule" 44
Programski kod 12. Entitetska klasa "Item" 45
Programski kod 13. Dao klasa "LevelItemDao" 46
Programski kod 14. Repozitorijska klasa "ItemRepository" 46
Programski kod 15. Klasa baze podataka "AppDatabase" 47
Programski kod 16. Klasa „ScanViewModel“ 48
Programski kod 17. Worker klasa "SeedDatabaseWorker" 49
Programski kod 18. Komponenta "ScavArApplication" 50
Programski kod 19. Komponenta „ItemList“ 50
Programski kod 20. Komponenta „ItemCard“ 51
Programski kod 21. Komponenta „CameraPreview“ 52
Programski kod 22. Implementacija rada s kamerom u „ScavAR“ 53
Programski kod 23. Implementacija prepoznavanja objekata u aplikaciji „ScavAR“ 54
Programski kod 24. Prikazivanje scene proširene stvarnosti 55

Prilozi

1. **Poveznica na repozitorij mobilne aplikacije:**

   https://github.com/aldin-alagic/scavenger-ar

2. **Poveznica na repozitorij poslužiteljske aplikacije:**

   https://github.com/aldin-alagic/scavenger-ar-backend

3. **Poveznica na Swagger dokumentaciju API servisa poslužiteljske aplikacije:**

   68.183.76.166/api-docs