Primjena proširene stvarnosti u navigaciji vozila

Pavleković, Daniel

Master's thesis / Diplomski rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:719257

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-11-30**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

PRIMJENA PROŠIRENE STVARNOSTI U NAVIGACIJI VOZILA

Diplomski rad

Daniel Pavleković

Osijek, 2018.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 04.09.2018.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Daniel Pavleković
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 857 R, 25.09.2017.
OIB studenta:	36951381191
Mentor:	Izv. prof. dr. sc. Josip Job
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Časlav Livada
Član Povjerenstva:	Dr. sc. Tomislav Galba
Naslov diplomskog rada:	Primjena proširene stvarnosti u navigaciji vozila
Znanstvena grana rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	Istražiti mogućnosti primjene proširene stvarnosti u navigaciji vozila korištenjem pametnog telefona. U praktičnom dijelu rada potrebno je izraditi aplikaciju koja će demonstrirati primjenjivost proširene stvarnosti na jednoj od mobilnih platformi.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	04.09.2018.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 18.09.2018.

Ime i prezime studenta:	Daniel Pavleković
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 857 R, 25.09.2017.
Ephorus podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom**: Primjena proširene stvarnosti u navigaciji vozila**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Josip Job

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD *1*

1.1. Zadatak diplomskog rada **1**

2. PROŠIRENA STVARNOST *2*

2.1. Proširena stvarnost na mobilnim uređajima **7**

2.2. Google Tango **8**

2.3. Google ARCore **9**

2.4. Apple ARKit **10**

2.4.1. Korištenje ARKit razvojnog okvira 11

2.4.2. ARKit virtualni svijet 12

3. IZVEDBA IOS APLIKACIJE *14*

3.1. Korištene tehnologije **14**

3.1.1. iOS operativni sustav 14

3.1.2. Swift 15

3.1.3. Cocoa (Touch) 16

3.1.4. Struktura iOS aplikacija 18

3.1.5. Alamofire 19

3.1.6. SwiftyJSON 19

3.1.7. ARCL 19

3.1.8. SWXMLHash 20

3.1.9. XCode 20

3.2. Pozicioniranje objekata pomoću ARKit razvojnog okvira **21**

3.2.1. Određivanje geografske pozicije uređaja 21

3.2.2. Određivanje 2D pozicije objekta u odnosu na uređaj 21

3.2.3. Određivanje visine objekta 24

3.2.4. Dodavanje objekta na prikaz proširene stvarnosti 26

3.2.5. Transformacija objekta 27

3.2.6. Rotacija objekta 28

3.3. Opis aplikacije **29**

3.3.1. Određivanje lokacije odredišta 34

3.3.2. Određivanje lokacije korisnika 39

3.3.3. Određivanje rute kretanja 39

3.3.4. Postavljanje prikaza proširene stvarnosti 43

3.3.5. Dodavanje prikaza pomoćne karte 45

3.3.6. Dodavanje posljednje upute 45

3.3.7. Dodavanje prve upute 46

3.3.8. Dodavanje cestovnih obilježja 48

3.3.9. Provjera brzine i promjena boje objekata 50

3.3.10. Provjera udaljenosti od idućeg skretanja i ažuriranje objekata 53

3.3.11. Provjera objekata rute 54

3.3.12. Provjera visine i vidljivosti objekata 54

3.3.13. Resetiranje i rotacija objekata 56

3.3.14. Korištenje aplikacije 57

4. ISPITIVANJE FUNKCIONALNOSTI APLIKACIJE *62*

4.1. Ispitivanje funkcionalnosti **62**

4.1.1. Brzina učitavanja objekata 62

4.1.2. Preciznost projekcije 62

4.2. Nedostatci sustava **65**

4.2.1. Nedostatci Apple Maps zemljopisnih karata 65

4.2.2. Nepreciznost lociranja uređaja 66

5. ZAKLJUČAK *67*
LITERATURA *68*
SAŽETAK *73*
ABSTRACT *74*
ŽIVOTOPIS *75*
PRILOG A. Dijagram toka aplikacije *76*

1. UVOD

Cilj ovog diplomskog rada je napraviti mobilnu aplikaciju koja će pomoću kamere i proširene stvarnosti pokazivati putanju kojom se korisnik treba kretati kako bi stigao do odabranog cilja. Motivacija za ovaj rad je nedostatak sličnih aplikacija na mobilnim platformama. U trenutku pisanja, u Apple trgovini aplikacija App Store, postoji samo jedna aplikacija koja nudi slično rješenje *ARCity - AR Navigation*, te je ograničena na 300 gradova i njihove znamenitosti. U trgovini Google Play također postoji jedna aplikacija naziva *AR GPS DRIVE/WALK NAVIGATION* koja nudi navigaciju uz pomoć proširene stvarnosti. Ideja jest da korisnik drži mobilni uređaj u automobilu u svom vidokrugu, te se ne mora oslanjati na zvučne upute, već može vizualno pratiti potreban smjer kretanja, a korištenjem kamere mobilnog uređaja zajedno sa proširenom stvarnosti ne smanjuje se vidljivost ceste.

1.1. Zadatak diplomskog rada

Istražiti mogućnosti primjene proširene stvarnosti u navigaciji vozila korištenjem pametnog telefona. U praktičnom dijelu rada potrebno je izraditi aplikaciju koja će demonstrirati primjenjivost proširene stvarnosti na jednoj od mobilnih platformi.

2. PROŠIRENA STVARNOST

Proširena stvarnost, u smislu računalne znanosti, je proces kombiniranja video materijala ili video ekrana i korisnih podataka koje generira računalo. [1] Ukratko, proširena stvarnost je kombiniranje imaginarnog i realnog, spajanje računalnog svijeta sa stvarnim svijetom. Proširena stvarnost omogućava prikaz objekata stvorenih pomoću računala u stvarnom svijetu.
Ideja o uređaju koji posjeduje mogućnost proširene stvarnosti se prvi put pojavljuje 1901. godine u kratkoj priči „*The Master Key: An Electrical Fairy Tale"* autora Lyman Frank Bauma gdje se opisuju naočale koje slovom opisuju karakter svakog čovjeka. [2]

Refleksni nišan

Godinu dana prije toga, 1900., Howard Grubb, irski optičar je stvorio refleksni nišan koji koristeći zrcala i leće grafički prikazuje osobi koja koristi oružje gdje će metak udariti. Refleksni nišani su ugrađivani 1930-ih u vojne avione. Prvo ih je implementirala francuska vojska, a zatim njemačka, britanska, američka i ruska vojska. [3]

HUD

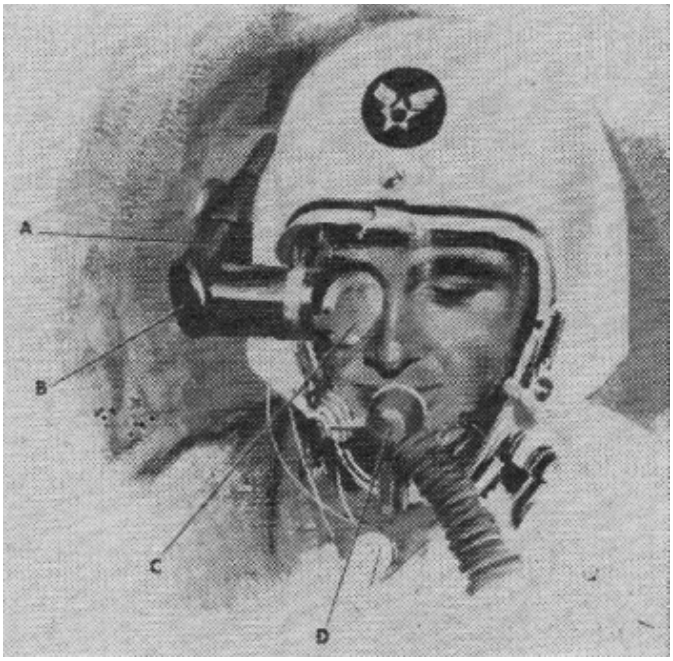
Refleksni nišan je preteča tzv. HUD-ova (eng. *head-up display*), odnosno uređaja koji pomoću zrcala reflektiraju sliku s ekrana na staklo koje se nalazi ispred korisnika. 1942. godine britansko ratno zrakoplovstvo je uspjelo spojiti refleksni nišan zrakoplova i katodnu cijev radara zrakoplova kako bi piloti lakše uočili metu bombardiranja, bez pomicanja glave prema dolje, odnosno prema instrumentima na ploči. [4] Prvi zrakoplov koji je imao ugrađeni HUD bio je Blackburn Buccaneer 1961. godine. Napravljen je od strane tvrtki Elliot Flight Automation(danas BAE Systems) i Cintel. Imao je sva obilježja modernih HUD-ova: optiku, katodnu cijev jake svjetlosti i programibilno generiranje valnih oblika, a prikazivao je nadmorsku visinu aviona i simbole koji su pomagali pilotu pri ciljanju. [5] Današnji HUD-ovi u zrakoplovima prikazuju minimalno zračnu brzinu, nadmorsku visinu, horizont, azimut, nagib aviona, te klizanje aviona. [6] 1980-ih godina HUD se počeo ugrađivati i u putničke avione, a 1988. je HUD prvi put ugrađen i u automobil (Oldsmobile Cutlass Supreme), te je prikazivao trenutnu brzinu i indikatore skretanja. [7] [8]

Sl. 2.1. Prikaz HUD-a u zrakoplovu [9]

Sl. 2.2. Prikaz HUD-a u automobilu (BMW E60) [10]

HMD

HMD (eng. *head-mounted display*) je uređaj, odnosno ekran koji se postavlja na kacigu u svrhu prikaza dodatnih informacija korisniku. [11] 1962. godine tvrtka Hughes Aircraft Company je predstavila je uređaj Electrocular. Uređaj se postavljao na kacigu, te se sastojao od male katodne cijevi i zrcala koje je odbijalo sliku na staklo koje se nalazilo ispred oka pilota. Razvijen je za potrebe astronauta. [12] Prva (vojna) upotreba HMD-a bila je 1970-ih godina kada je američka vojska koristila HMD u helikopteru AH-1G kako bi pilotu vizualizirala gdje je okrenut nišan mitraljeza helikoptera. Uz samu vizualizaciju, sustav je imao mogućnost lociranja pozicije glave pilota, pa je pilot mogao pomoću pokreta glave okretati mitraljez. Prvi HMD sustav koji je korišten u borbi je IHADSS sustav američke vojske korišten 1980-ih u Apache helikopteru. IHADSS je kombinirao prikaz termalne kamere i grafiku koja prikazuje informacije o poziciji helikoptera i meti koju pilot pogađa. HMD se koristi i u današnjim borbenim zrakoplovima i helikopterima. Prednost HMD sustava u odnosu na HUD jest činjenica da se pilot ne prilagođava ekranu, nego ekran pilotu, odnosno pilot ne mora gledati ispred sebe kako bi vidio informacije. Također, zbog mogućnosti lociranja pozicije glave pilota, HMD se dodatno može koristiti za upravljanje dijelovima zrakoplova (kao npr. mitraljezima). [13]

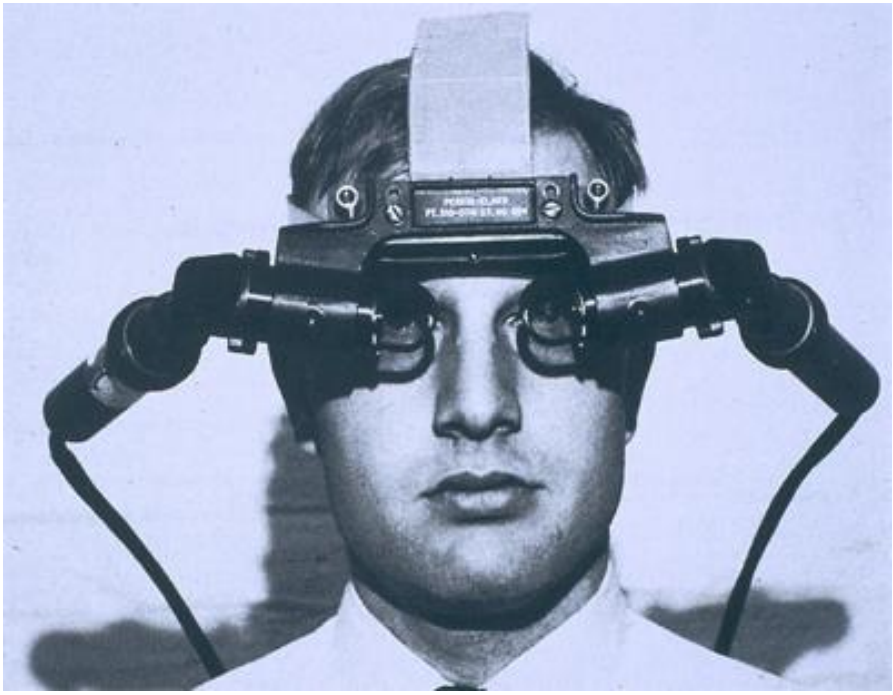


Sl. 2.3. Uređaj Electrocular [12]

Sl. 2.4. JHMCS HMD sustav [14]

Ivan Sutherland

Proširena stvarnost i virtualna stvarnost (u smislu 3D računalne grafike) prvi puta se pojavljuju 1965. godine kada Ivan Sutherland u svom eseju opisuje „ultimativni ekran“ koji će prikazivati sobu u kojoj računalo upravlja postojanjem materije. U takvoj bi sobi „stolica bila dovoljno dobra da se u nju može sjesti, lisice bi vezale ruke, a metak bi bio ubojit“. Tu se naravno opisuje virtualna stvarnost, međutim u tom istom eseju se spominje i proširena stvarnost kao mogućnost da korisnik pomoću takvih ekrana može krute objekte učiniti prozirnim, odnosno da može „vidjeti kroz materiju“. [15]



Sl. 2.5. Ivan Sutherland i njegov ultimativni ekran [16]

Ivan Sutherland je napravio prototip ultimativnog ekrana koji se može vidjeti na slici 2.5. Taj uređaj se sastojao od dva prozirna ekrana, od kojih se svaki nalazio na jednom oku. Informacije o pokretima glave se šalju računalu, koje na temelju programskog algoritma generira specifikaciju 3D linije u sobnim koordinatama. Zatim se ta informacija zajedno sa informacijom o perspektivi koja se koristi šalje množitelju matrice koji transformira podatke iz sobnih koordinata u očne koordinate, te ih šalje djelitelju odsječaka. Djelitelj odsječaka transformira podatke u 2D liniju koja odgovara koordinatama polja i šalje ih upravljačkom uređaju ekrana. Upravljački uređaj šalje signale katodnim cijevima koje stvaraju sliku na prozirnim ekranima. Nakon novog pomaka glave subjekta koji nosi uređaj na glavi postupak se ponavlja. [17]

Industrijska proširena stvarnost

1992. godine se prvi puta spominje pojam "proširena stvarnost" u radu T.P. Caudella i D.W. Mizella u tvrtki Boeing kada su osmislili način kako će olakšati rad svojim kolegama u tvrtki koji su postavljali kablove za ugradnju u zrakoplov Boeing 777. Redovna procedura sastojala se od provlačenja i vezanja žica oko klinova na pločama koje su dugačke od 6 do 9 metara(20 do 30 stopa), a nakon toga bi se ploče ugrađivale u avion. Problem se sastojao u tome da su radnici morali gledati konstantno u upute na papiru kako bi pravilno provukli žice. Caudell i Mizella su osmislili uređaj sličan naočalama, koji bi radnici nosili na glavi, te bi im uređaj preklopio sliku provučenih

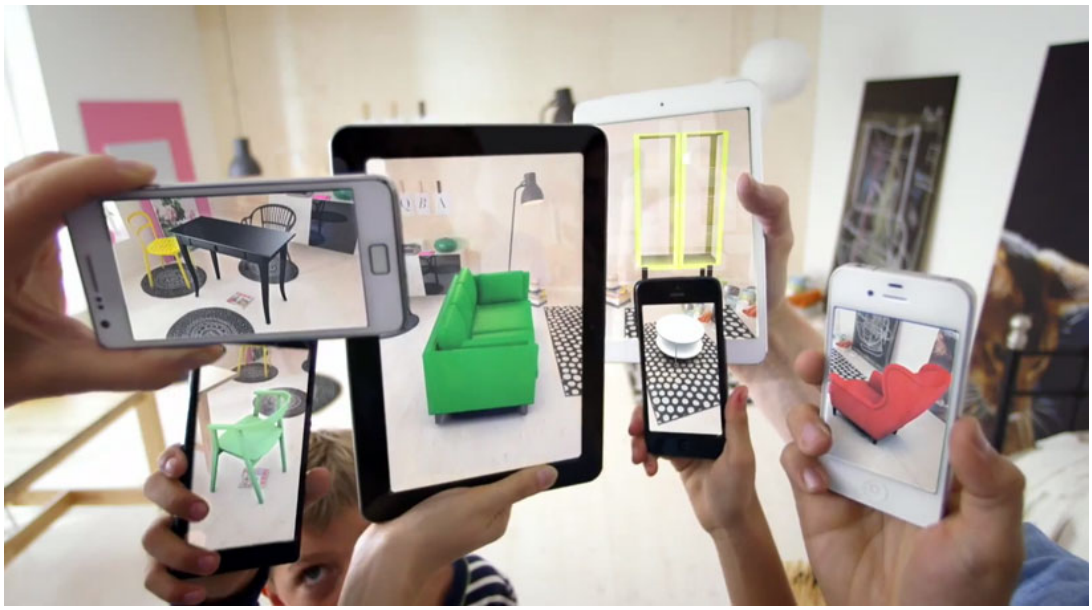
žica na praznu ploču. Tako su mogli raditi neometano. Međutim, sustav nije zaživio zbog lošeg odziva, koji je bio uzrokovan nedostatnom računalnom snagom. [18]

2.1. Proširena stvarnost na mobilnim uređajima

Proširena stvarnost se 2003. pojavljuje na dlanovniku (preteća pametnih telefona) u obliku Windows Mobile aplikacije za navođenje unutar zgrade. Aplikaciju su napravili Daniel Wagner i Dieter Schmalstieg koristeći Windows Mobile verziju ARToolKit biblioteke. ARToolKit biblioteka je biblioteka otvorenog koda koju su napravili Hirokazu Kato i Mark Billinghurst, a služi za praćenje poze ljudskog tijela. Prepoznaje šest stupnjeva slobode, te koristi crno-bijele pravokutne markere (tzv. *fiducial*) i pristup na temelju postojećih obrazaca za prepoznavanje objekata. [19]

Prva primjena proširene stvarnosti na mobilnom telefonu je bila 2008. kada je Daniel Wagner sa svojom skupinom napravio mobilnu aplikaciju koja je implementirala šest stupnjeva slobode za prepoznavanje objekata u prostoru. Aplikacija je uspjela postići brzinu od 20 Hz. [19]

Mobilne platforme koje danas podržavaju proširenu stvarnost, te imaju vlastitu okolinu za razvoj aplikacija koje koriste proširenu stvarnost su Google Android i Apple iOS. U trenutku pisanja(16.03.18.) postoje 23 mobilna uređaja koja podržavaju AR aplikacije, od toga su 13 Android mobilnih telefona i 10 Apple mobilnih telefona. AR platforma koju koristi Apple, odnosno iOS, jest ARKit, dok Google, odnosno Android, koristi ARCore.



Sl. 2.6. Prikaz proširene stvarnosti na Android i iOS uređajima [20]

2.2. Google Tango

Googleov projekt Tango je računalna platforma razvijena u tvrtki Advanced Technology and Projects, koja je dio Google grupacije. Tango je izdan 2014. godine kao platforma za razvoj aplikacija koje koriste proširenu stvarnost. Ima tri funkcionalnosti:

1. Praćenje pokreta - koristi akcelerometar, žiroskop i vizualna obilježja prostora (sa fotografija kamere mobilnog uređaja) kako bi odredio položaj i orijentaciju mobilnog uređaja u prostoru. [21]
2. Percepcija dubine - prepoznaje oblike objekata u prostoru koristeći Intelovu RealSense 3D kameru. RealSense 3D kamera se sastoji od 3 leće: obične 2D kamere, infracrvene kamere i infracrvenog laserskog projektora. [22] [21]
3. Učenje prostora - kombinirajući praćenje pokreta i percepciju dubine kreira se 3D model prostora i postavi se korisnika unutar tog modela. Model prostora se sprema u posebnu datoteku koja se zove *Area Description File* i može se koristiti u ostalim aplikacijama na mobilnom telefonu. [21]

Komercijalni mobilni uređaji koji su imali ugrađeno sklopovlje za korištenje Google Tango platforme su Lenovo Phab 2 Pro i Asus Zenfone AR.

U ožujku 2018. Google je ugasio Tango platformu zbog ARCore platforme.

2.3. Google ARCore

ARCore je SDK odnosno alat za razvoj softvera razvijen od strane tvrtke Google za stvaranje aplikacija koje koriste proširenu stvarnost. Izdan je prvog ožujka 2018. godine. Za razliku od Tango tehnologije koja je koristila poseban set senzora(Intel RealSense 3D), ARCore koristi 2D kameru kakvu koristi većina mobilnih telefona.

ARCore se sastoji od 3 tehnologije koje služe za izradu aplikacija proširene stvarnosti:

1. Praćenje pokreta - koristeći proces zvan simultana odometrija i mapiranje(engl. *concurrent odometry and mapping*) raspoznaje se pozicija korisnika u odnosu na okolinu. Promjena položaja se računa koristeći značajke(engl. feature point) sa slike koju generira kamera. Vizualna informacije se kombinira s podacima koje generiraju akcelerometar i žiroskop kako bi se odredila pozicija i orijentacija kamere u odnosu na okolinu.
2. Razumijevanje okoline - ARCore prepoznaje ravnine kao grupu značajki koje se nalaze na istoj vodoravnoj ili okomitoj površini. Također prepoznaje i rubove ravnina, te tako omogućava postavljanje virtualnih objekata na ravne površine.
3. Procjena svjetlosti - ARCore prepoznaje intenzitet svjetlosti okoline, te usklađuje boju i intenzitet svjetlosti virtualnih objekata kako bi povećao realizam.

Nakon što ARCore odredi okolinu i poziciju mobilnog uređaja u odnosu na okolinu, korisnik može praviti interakciju sa svijetom proširene stvarnosti tako da dodiruje ekran i dodaje virtualne objekte, briše virtualne objekte i pomiče ih.

ARCore sadrži i orijentirane točke(engl. *oriented points*) koje omogućuju postavljanje virtualnih objekata na površine koje nisu horizontalne.

Sidro(engl. *anchor*), u smislu ARCore alata, se odnosi na objekt koji se postavi na točno određeno mjesto u svijetu proširene stvarnosti. Prateći objekti(engl. *trackables*) su ravnine i točke u okolini. ARCore prati njihovu poziciju i prilagođava poziciju ostalih objekata(sidra) u slučaju da se prateći objekti pomaknu. Pojednostavljeno, sidra su virtualni objekti, dok su prateći objekti, objekti u stvarnom svijetu.

Funkcija proširenih fotografija(engl. Augmented Images) omogućava prepoznavanje određenih slika, te interakcija s njima u svijetu proširene stvarnosti. [23]

2.4. Apple ARKit

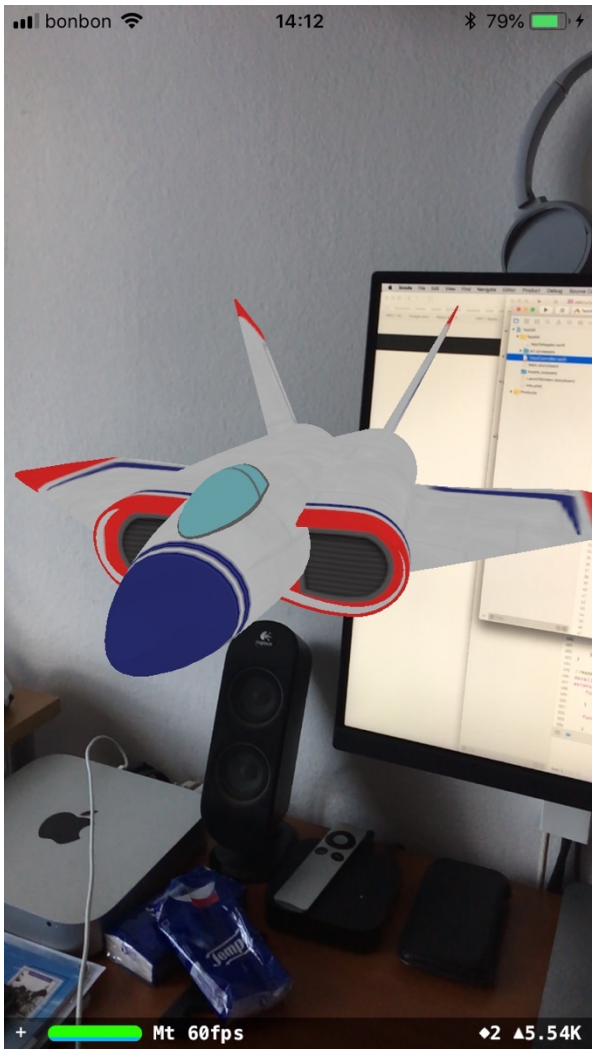
ARKit je paket za razvoj softvera (SDK) koji omogućava razvoj iOS aplikacija s podrškom za proširenu stvarnost. Izdan je u sklopu operativnog sustava iOS 11, 19. rujna 2017. godine. Kao i Googleov ARCore, ARKit koristi 2D kameru koja se nalazi u mobilnom telefonu u suradnji s ugrađenim senzorima(akcelerometar, žiroskop i barometar) kako bi omogućio prikaz proširene stvarnosti.

ARKit ima mogućnost praćenja okoline. Koristeći vizualno-inercijsku odometriju[1] ARKit stvara model koji opisuje poziciju i kretanje uređaja u stvarnom svijetu. ARKit također analizira i razumije sadržaj scene koju kamera vidi, pa koristeći metode testiranja pogotka(engl. *hit-testing*) mogu se naći točke u stvarnosti koje odgovaraju pozicijama u prikazu proširene stvarnosti. Također, ARKit može prepoznati ravnine u prostoru i vraćati informacije o njima, kao što su njihova pozicija i veličina. Koristeći prepoznavanje ravnina i testiranje pogodaka mogu se postavljati 2D ili 3D modeli u prikaz proširene stvarnosti, te vršiti interakcija s njima. [24]

Kao i ARCore, ARKit prepoznaje ambijentalno osvjetljenje, te prema njemu mijenja izgled objekata proširene stvarnosti kako bi scena proširene stvarnosti izgledala što realističnije.

Za one koji žele koristiti istu scenu proširene stvarnosti zajedno s drugim osobama, ARKit omogućava višekorisničko iskustvo. ARKit koristi *MultipeerSession* klasu i *MultipeerConnectivity* značajku kako bi omogućila višekorisničko iskustvo. Uređaj traži okolne uređaje, šalje zahtjev, drugi uređaj prima zahtjev, prihvaća i nakon toga oba korisnika dijele istu scenu proširene stvarnosti. Korisnici također mogu slati jedni drugima *ARWorldMap* objekt koji sadrži sve informacije o lokaciji uređaja u stvarnom prostoru. Nakon što korisnik primi *ARWorldMap* objekt, mora skenirati okolinu prije nego počne koristiti tuđi objekt. Nakon toga smješta objekte u scenu proširene stvarnosti tako da oba korisnika vide iste virtualne objekte na istom mjestu u prostoru. [25]

[1] Vizualno-inercijska odometrija je postupak kombinacije podataka senzora mobilnog uređaja koji detektiraju pokrete(akcelerometar, žiroskop i barometar), te informacija dobivenim računalnom analizom fotografija s kamere mobilnog uređaja. [24]



Sl. 2.7. Prikaz osnovne ARKit aplikacije

2.4.1. Korištenje ARKit razvojnog okvira

ARKit razvojni okvir je ugrađen u iOS operativni sustav, te se nalazi u sklopu XCode programskog paketa za razvoj iOS mobilnih aplikacija. Zbog toga, jedini korak koji je potrebno napraviti prije korištenja ARKit paketa jest postaviti ARKit SceneKit pogled ili ARKit SpriteKit pogled na određeni kontroler, te učitati ARKit biblioteku na početku svake datoteke u kojoj ga koristimo.
SceneKit pogled je dio SceneKit razvojnog okvira koji kombinira *engine* za renderiranje i aplikacijsko programsko sučelje(engl. *API*) za prikaz i manipulaciju 3D objektima. SpriteKit pogled je dio SpriteKit razvojnog okvira koji služi za prikaz i manipulaciju 2D objektima.
Kako bi se omogućilo praćenje uređaja u virtualnom svijetu potrebno je stvoriti objekt tipa *ARWorldTrackingConfiguration(),* te pri pokretanju sjednice proslijediti taj objekt objektu koji predstavlja pogled proširene stvarnosti. [26] [27]

```
@IBOutlet weak var ARView: ARSCNView!
let configuration = ARWorldTrackingConfiguration()
ARView.session.run(configuration)
```

Programski kod 2.1. Pokretanje ARKit pogleda

2.4.2. ARKit virtualni svijet

ARKit virtualni svijet ima svoj vlastiti koordinatni sustav. Ishodište koordinatnog sustava je na onom mjestu na kojem se nalazi korisnikov mobilni uređaj u trenutku kada se pokrene pogled proširene stvarnosti (ARKit pogled). Orijentacija koordinatnog sustava ovisi o postavkama koje se zadaju u varijabli *worldAlignment* objekta klase *ARConfiguration* [28]. Moguće vrijednosti varijable *worldAlignment* su:

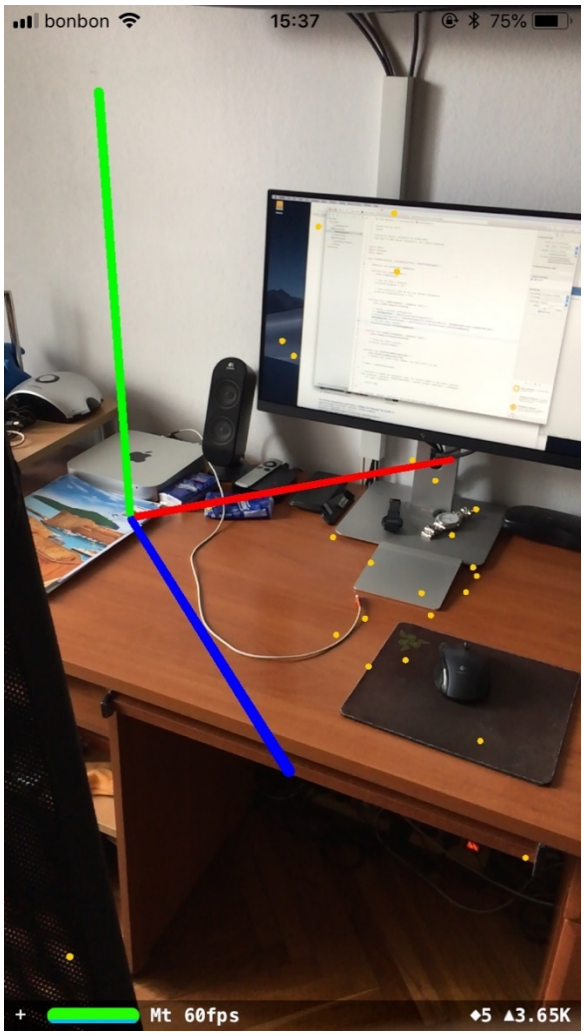
- *gravity* - y os koordinatnog sustava je paralelna sa silom gravitacije
- *gravityAndHeading* - y os koordinatnog sustava je paralelna sa silom gravitacije, a z os je orijentirana obrnuto od magnetnog sjevernog pola
- *camera* - usmjerenje koordinatnog sustava odgovara usmjerenju kamere

Objekti se u virtualni svijet dodaju u obliku čvorova. Čvorovi su strukturalni elementi SceneKit prikaza koji predstavlja poziciju i transformaciju u 3D koordinatnom sustavu na koju se može dodati geometrija, svjetlo, kamere, zvuk i animacija. Na svaki čvor se mogu dodavati podčvorovi, a korijenski čvor je koordinatni sustav cijelog prikaza. Transformacija koja se vrši nad čvorom utječe i na njegove podčvorove. Čvorovi imaju hijerarhiju u obliku stabla. Svaki čvor ima vrijednosti pozicije, rotacije i skaliranja. [29] Pozicija i skaliranje se određuje pomoću objekta klase *SCNVector3* koji predstavlja trodimenzionalni vektor. [30] Rotacija se određuje pomoću objekta klase *SCNVector4* koji predstavlja vektor koji sadrži normalizirane x, y i z komponente rotacijske osi, te kut rotacije. [31] Geometrija čvora može biti objekt tipa:

- *SCNPlane* - ravnina
- *SCNSphere* - sfera
- *SCNBox* - kvadar s mogućnosti zaobljenih rubova
- *SCNPyramid* - pravokutna četverostrana piramida
- *SCNTube* - valjak ili cijev

Na čvor je moguće dodati materijal, odnosno čvor može biti određene boje, imati određenu fotografiju ili videozapis. [32]

Osim čvorova mogu se koristiti i sidra(engl. *anchor*) koji predstavljaju poziciju i orijentaciju u stvarnom svijetu, koje se koriste za postavljanje objekata u prikaz proširene stvarnosti. Sidra mogu biti ravnina, objekt ili fotografija. Nad sidrima se ne može vršiti transformacija. [33] [34]



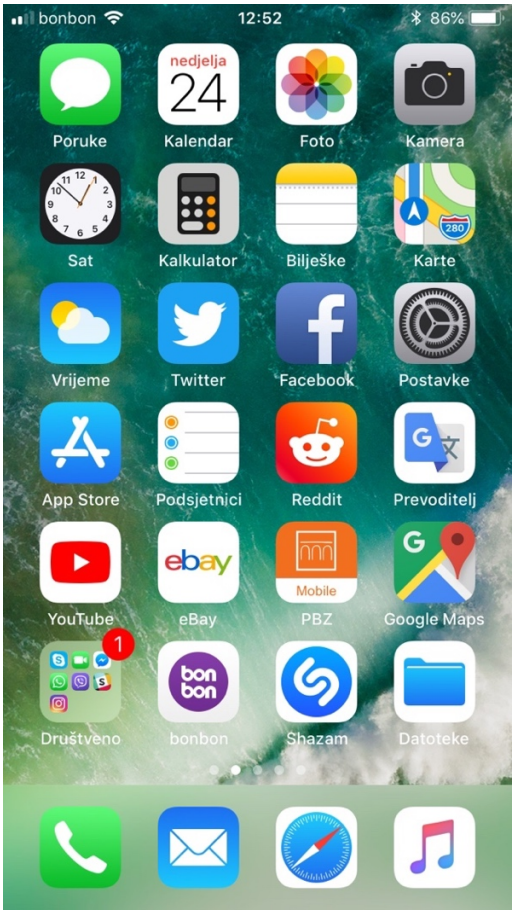
Sl. 2.8. Prikaz proširene stvarnosti sa prikazom ishodišta koordinatnog sustava i značajkama(žute točke) koje sustav prepoznaje sa fotografije

3. IZVEDBA IOS APLIKACIJE

3.1. Korištene tehnologije

3.1.1. iOS operativni sustav

iOS je operativni sustav razvijen od strane tvrtke Apple, prvenstveno za iPhone mobilne telefone. Prva verzija je izašla 2007. godine i pokretao ju je prvi model iPhone mobilnog telefona, iPhone 2G. Nedugo nakon toga, Apple je izdao još jedan uređaj koji je pokretao iOS sustav, a to je multimedijski uređaj iPod Touch. Naposljetku, 2010. godine izdan je i treći uređaj koji je pokretao iOS operativni sustav, a to je tablet uređaj iPad. iOS je Unixoidan operativni sustav, temeljen je na Darwin operativnom sustavu. Dijeli jezgru i osnovne komponente s macOS, odnosno Mac OS X operativnim sustavom. Svake godine dobije nove nadogradnje u obliku nove verzije(iOS 2, iOS 3, iOS 4…), a trenutna verzija je iOS 11.4. U trenutnoj verziji iOS operativnog sustava dodana je podrška za izradu nativnih iOS aplikacija koje koriste proširenu stvarnost, a koja se koristi u ovom radu.



Sl. 3.1. Izgled iOS operativnog sustava verzije 11.4.

3.1.2. Swift

Swift je programski jezik razvijen od strane tvrtke Apple Inc. Glavna odgovorna osoba za Swift programski jezik je Chris Lattner, koji je počeo raditi na njemu 2010. godine, a iduće godine su mu se pridružili i drugi programeri. Prva verzija je izdana u lipnju 2014. godine. Od tog trenutka Swift je, uz Objective-C, postao službeni programski jezik za razvoj iOS aplikacija. Prema tvrdnjama Chrisa Lattnera, pri kreiranju Swift programskog jezika preuzeo je ideje iz Objective-C, Rust, Haskell, Ruby, Python, C#, CLU i ostalih programskih jezika. U prosincu 2015. godine Swift je postao softver otvorenog koda, što znači da svaki programer može pridonijeti njegovom razvitku. [35]

Značajke Swift programskog jezika su:

- Programer ne mora eksplicitno odrediti tip varijable ili konstante već Swift implicitno zaključuje o kojem se tipu radi iz vrijednosti varijable ili konstante. Također, u Swiftu se varijable uvijek inicijaliziraju prije korištenja [36].
- Swift upravlja memorijom automatski pomoću tehnologije koja se zove ARC(engl. *Automatic Reference Counting*). ARC dodijeli komad memorije svaki puta kada se stvori novi objekt. U taj komad memorije se spremaju informacije o tipu klase objekta, kao i vrijednosti atributa objekta. Kada se objekt pridruži konstanti ili varijabli (ukoliko nije drugačije određeno) stvori se "jaka" referenca na objekt. Sve dok objekt ima barem jednu "jaku" referencu, ARC neće osloboditi njegovu memoriju. Ukoliko objekt nema niti jednu "jaku" referencu, ARC ga obriše iz memorije, odnosno oslobodi njegovu memoriju. Problem nastaje kada dva objekta imaju "jake" reference jedan na drugoga, a programer obriše(postavi na *nil*[2]) varijable ili konstante kojima su pridruženi. Tada oba objekta ostanu u memoriji iako im programer ne može pristupiti. Rješenje je postaviti "slabu" referencu na jedan od objekata pomoću ključne riječi "*weak"*. Tada ARC oslobodi memoriju objekta sa "slabom" referencom, te zbog toga drugi objekt nema više niti jednu "jaku" referencu na sebe, što znači da se oslobađa i njegova memorija [37].
- Swift provjerava indekse nizova kako ne bi došlo do greške pristupanja indeksu koji se ne nalazi u nizu [38].
- Svi cjelobrojni tipovi podataka(engl. *integer*) se provjeravaju za prekoračenje memorije(engl. *overflow*) [38].

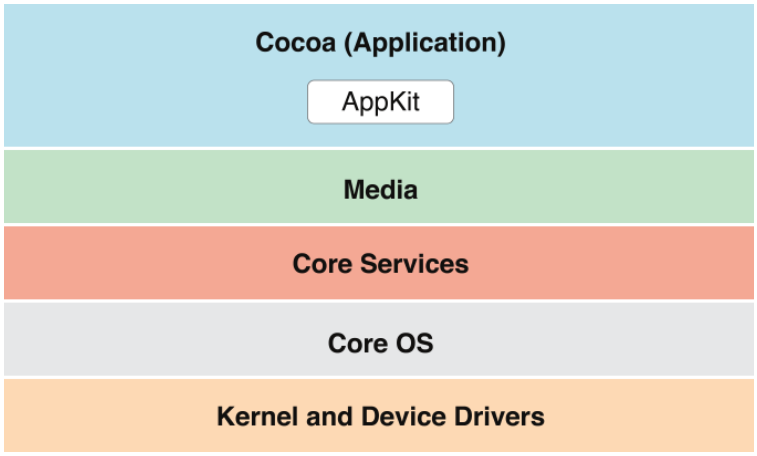
[2] "nil" je ključna riječ u Swift programskom jeziku za NULL vrijednost

- Koriste se opcionali(engl. *optional*) koji osiguravaju pravilno upravljanje s *nil* tipovima podataka. [38] Opcional je tip podatka koji može imati dva stanja [39]:
  1. Sadrži neku vrijednost
  2. Ne sadrži vrijednost. Njegova vrijednost je *nil*
- Sadrži upravljanje greškama koje omogućava kontrolirano upravljanje kvarovima unutar aplikacije. [38]

U trenutku pisanja, Swift je prema TIOBE indeksu 15. najčešće korišteni programski jezik [40].

3.1.3. Cocoa (Touch)

Cocoa je aplikacijski sloj macOS operativnog sustava zadužen za prikaz aplikacije i interakciju sa korisnikom. U njemu su implementirane mnoge značajke macOS operativnog sustava(centar za notifikacije, centar za igre, dijeljenje, Spotlight, Cocoa Auto Layout...), te sadrži mnoge razvojne okvire(engl. *framework*) kao što su Cocoa Umbrella, AppKit, Game Kit, Preference Panes, Screen Saver i Security Interface [41].



Sl. 3.2. Slojevi macOS operativnog sustava [41]

Cocoa Touch je posebna verzija Cocoa aplikacijskog sloja za iOS uređaje i sadrži sve potrebne razvojne okvire za razvoj iOS aplikacija. [42] Neki od razvojnih okvira se koriste i u ovom diplomskom radu, a to su: UIKit, Foundation, MapKit, Core Location, ARKit.

Foundation

Razvojni okvir Foundation sadrži osnovne funkcionalnosti za aplikacije koje koriste Cocoa aplikacijski sloj i ostale razvojne okvire koji se nalaze u Cocoa aplikacijskom sloju. Sadrži

osnovne tipove podataka, te klase i protokole za spremanje i postojanost podataka, procesiranje teksta, računske operacije s datumima i vremenom, sortiranje, filtriranje i mrežne funkcionalnosti. [43]

UIKit

UIKit je razvojni okvir potreban za razvoj iOS aplikacija, te sadrži osnovne komponente aplikacije. UIKit pruža okvir za izradu i prikaz korisničkog sučelja aplikacije, te infrastrukturu za upravljanje korisnikovim unosima. Također, osigurava glavnu nit koja služi za upravljanje međudjelovanjima korisnika, sustava i aplikacije. UIKit uz to osigurava i druge značajke kao što su podrška za animaciju, dokumente, crtanje, ispis, informacije o uređaju, upravljanje tekstom i ekranom, podrška za pretraživanje, pristupačnost, ekstenzije i upravljanje resursima. Slika 3.3. prikazuje strukturu UIKit aplikacije. Sažeto, UIKit se koristi za izradu i upravljanje svim komponentama koje služe za prikaz informacija korisniku i unos podataka od strane korisnika. [44]

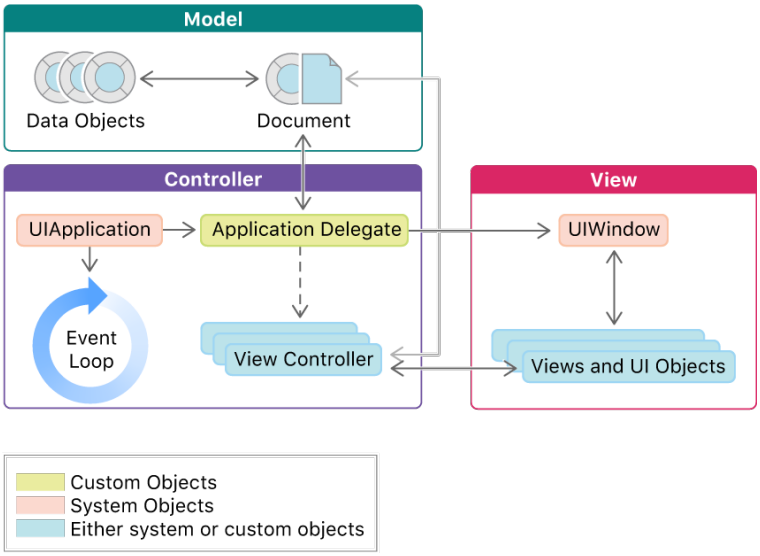
MapKit

MapKit razvojni okvir se koristi za ugradnju geografskih karata u prozore i prikaze. Sadrži mogućnost dodavanja bilješki i slojeva za prikaz interesnih točaka ili korisničkih odredišta. Putem MapKit razvojnog okvira moguće je dobiti različite upute za navigaciju ovisno o opcijama(vrsta prometala, cestarine, izbjegavanje autoceste...). Geografsku kartu i upute za navigaciju MapKit preuzima od Apple Maps usluge. [45]

Core Location

Core Location je razvojni okvir koji pruža informacije o lokaciji mobilnog uređaja. Može pružiti informacije o geografskoj lokaciji, nadmorskoj visini, orijentaciji i udaljenosti do najbližeg iBeacona. Koristi sve moguće sklopovlje ugrađeno u mobilni uređaj kako bi odredio lokaciju, pa tako prikuplja podatke pomoću: magnetometra, barometra, Wi-Fi, GPS i Bluetooth signala, te mobilnog signala. [46]

3.1.4. Struktura iOS aplikacija



Sl. 3.3. Struktura iOS aplikacije koja koristi UIKit [47]

UIKit aplikacije koriste MVC(engl. Model-View-Controller) obrazac. MVC obrazac označava da se aplikacija dijeli na model koji predstavlja logičku strukturu podataka, pogled koji predstavlja kolekciju klasa koje služe za prikaz sadržaja u aplikaciji i kontroler koji služi za komunikaciju između modela i pogleda.

UIApplication je centralna klasa za upravljanje i koordinaciju iOS aplikacije, te je uvijek singleton[3] klasa. Pruža API za upravljanje ponašanjem uređaja, usmjerava korisnikove unose do potrebnih objekata, održava listu otvorenih prozora, upozorava delegata na bitne događaje koji se događaju u izvršnom vremenu(engl. *runtime*), te upravlja međudjelovanjem aplikacija. [48]

Aplikacijski delegat (*UIApplicationDelegate*) je set metoda koje klasa UIApplication poziva kao odgovor na određene događaje u životu aplikacije. Programer može unutar metoda napisati kako će upravljati s određenim događajima. Metode se pokreću pri pokretanju aplikacije, u slučaju da aplikacija mijenja stanja(iz neaktivne u aktivnu, iz prvog plana u pozadinu i slično), te kao odgovor na notifikacije i posebne događaje(npr. upozorenje o nedostatku radne memorije). [49]

[3] Singleton je programski obrazac koji osigurava da postoji samo jedna instanca klase u programu. Kada se poziva singleton klasa, uvijek se vraća referenca na isti objekt. [72]

View Controller, odnosno klasa UIViewController je odgovorna za sve promjene sadržaja pogleda, odgovaranje na unose korisnika, promjenu veličine pogleda, upravljanje izgledom sučelja, te koordinaciju s drugim objektima unutar aplikacije. [50]
UIWindow je objekt koji je odgovoran za prikaz elemenata sučelja aplikacije. On je zapravo pozadina na koju programer stavlja elemente sučelja. Na UIWindow je moguće postaviti dodatne objekte tipa UIWindow, za prikaz dodatnih elemenata. [51]
Event Loop predstavlja glavni odsječak programskog koda koji se neprestano izvodi i odgovara na sve zahtjeve prema aplikaciji.
Model se sastoji od objekata koji sadrže podatke, te od objekta tipa UIDocument koji služi za skladištenje podataka u obliku podatkovnih struktura u datoteci koja se nalazi na disku. [47]

3.1.5. Alamofire

Alamofire je biblioteka napisana u Swift programskom jeziku, a koristi se za HTTP komunikaciju. Omogućava slanje HTTP zahtjeva, primanje odgovora poslužitelja i manipulaciju primljenim podacima. Moguće je također ulančavanje zahtjeva, enkodiranje podataka, preuzimanje podataka, postavljanje podataka na poslužitelj, validacija HTTP odgovora i slično. [52]

3.1.6. SwiftyJSON

SwiftyJSON je biblioteka koja služi za manipulaciju JSON datotekama u Swift programskom jeziku. Pomoću nje je moguće čitati i filtrirati podatke iz JSON datoteka. Inicijalizira se objekt klase JSON, kojemu se preda JSON datoteka. Nakon toga je moguće čitati podatke iz JSON datoteke i pretvarati ih u niz znakova ili u brojčane tipove podataka. [53]

3.1.7. ARCL

ARCL je biblioteka otvorenog tipa koja omogućava postavljanje objekata u ARKit svijet virtualne stvarnosti na temelju njegove geografske pozicije, odnosno geografske širine, dužine i nadmorske visine. Biblioteka sadrži klasu *SceneLocationView* koja nasljeđuje klasu Appleovu *ARSCNView* klasu i predstavlja prikaz proširene stvarnosti. Također sadrži klasu *LocationNode* koja nasljeđuje klasu *SCNNode*, te predstavlja objekt na određenoj geografskoj lokaciji i nadmorskoj visini. Naziv ARCL je skraćenica od ARKit Core Location, što znači da biblioteka koristi ta dva razvojna okvira

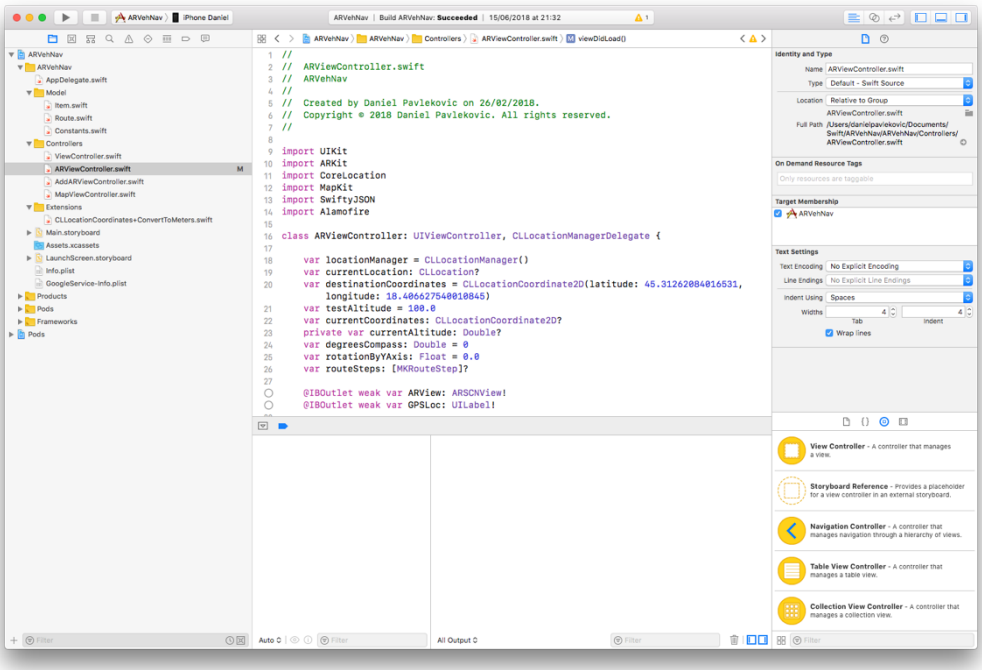
kako bi se postavio objekt u svijet proširene stvarnosti. Biblioteka kalibrira poziciju objekata. Na temelju pomaka korisnika u stvarnom svijetu (GPS) i pomaka u svijetu proširene stvarnosti određuje se korisnikova pozicija, a time i pozicija virtualnih objekata. Kut prema sjeveru se također kalibrira kako se korisnik kreće kroz virtualni svijet. [54]

3.1.8. SWXMLHash

SWXMLHash je biblioteka koja služi za parsiranje XML datoteka. Inicijalizira se objekt klase parse koji prima XML datoteku i kompletno ju parsira. Nakon toga moguće je pristupati određenim vrijednostima pomoću naziva korijenskih čvorova, pomoću indeksa čvora, pomoću atributa čvora i kombinacijama.

3.1.9. XCode

XCode je integrirano okružje za razvoj aplikacija(engl. *IDE*) koje se koristi za razvoj macOS, iOS, watchOS i tvOS aplikacija. Prva verzija XCode programa je izdana u jesen 2003. godine, bila je temeljena na programu Project Builder, ali sa drugačijim sučeljem, ZeroLinkom, Fix&Continue mogućnosti, distribuiranom podrškom za razvoj i Code Sense indeksiranjem. Prva verzija XCode programa koja je omogućavala razvoj iOS aplikacija je verzija 3.1. Aktualna verzija je verzija 9.4.1. XCode program je dostupan isključivo za macOS operativni sustav.



Sl. 3.4. Izgled sučelja XCode okružja za razvoj aplikacija

3.2. Pozicioniranje objekata pomoću ARKit razvojnog okvira

3.2.1. Određivanje geografske pozicije uređaja

U AR svijet potrebno je postaviti trodimenzionalne objekte koji će korisniku pomoći kretati se kroz stvarni svijet. AR svijet posjeduje vlastiti koordinatni sustav dok se za određivanje pozicije u stvarnom svijetu koriste zemljopisne koordinate i nadmorska visina. To znači da je potrebno prilikom postavljanja objekata u AR svijet napraviti poveznicu između ta sva svijeta. Način na koji je to napravljeno jest da se netom nakon pokretanja AR pogleda očitaju zemljopisne koordinate uređaja pomoću GPS uređaja koji se nalazi u mobilnom telefonu, te nadmorske visine pomoću barometra koji se također nalazi u sklopu mobilnog uređaja. Za dobivanje vrijednosti zemljopisnih koordinata i nadmorske visine se koristi biblioteka CoreLocation, te objekt klase *CLLocationManager*. Preciznost lociranja uređaja, odnosno vrijednost varijable *desiredAccuracy* objekta klase *CLLocationManager* je postavljena na *kCLLocationAccuracyBestForNavigation*, što označava najveću moguću preciznost. Pri odabiru ove postavke, uređaj koristi dodatne senzore kako bi što preciznije odredio lokaciju. [55] Službena dokumentacija ne navodi koji dodatni senzori se koriste, međutim na službenoj stranici tvrtke Apple, pod tehničkim specifikacijama uređaja iPhone 6s se navodi da se za lociranje koristi GPS, GLONASS, Galileo, QZSS, digitalni kompas, Wi-Fi, GSM odašiljači, te iBeacon mikrolokacijska usluga. [55]

3.2.2. Određivanje 2D pozicije objekta u odnosu na uređaj

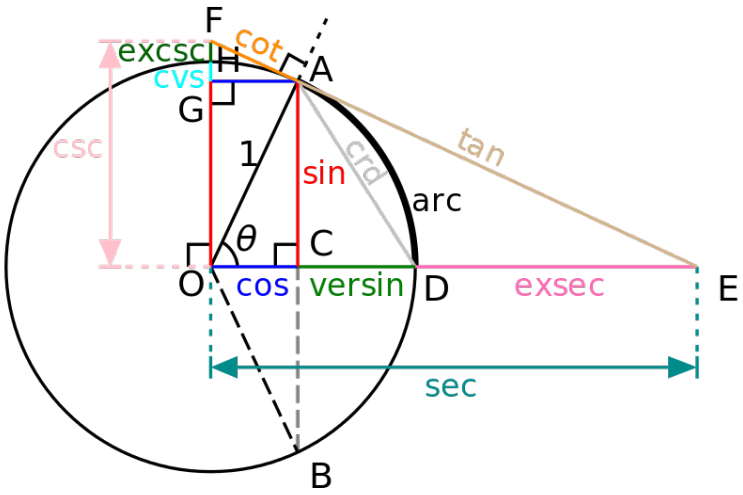
Koordinate objekta kojega je potrebno postaviti u AR svijet računaju se pomoću haversinus formule. Haversinus formula se koristi za računanje udaljenosti između dvije koordinate, odnosno između dvije točke na sferi. Zbog toga može doći do pogreške od 0.3%, jer Zemlja nije sfera, već elipsoid. [56] S obzirom da ovdje nije potrebna zračna, odnosno direktna udaljenost između dvije koordinate, već udaljenost po x osi i udaljenost po z osi, haversinus formula se upotrebljava dva puta. Ukoliko je početna točka X s pripadajućim koordinatama (x1,z1), a odredišna točka Y s pripadajućim koordinatama (x2,z2), prvo se traži udaljenost između točaka (x1,z1) i (x2,z1), što predstavlja udaljenost po osi x. Nakon toga se traži udaljenost između točaka (x1,z1) i (x1,z2), što predstavlja udaljenost po osi z. Početna točka X predstavlja zemljopisne koordinate koje se očitavaju pomoću objekta klase *CLLocationManager* u trenutku kada se otvori prikaz proširene

stvarnosti. Odredišna točka Y predstavlja zemljopisne koordinate objekta kojeg je potrebno prikazati u svijetu proširene stvarnosti.

S obzirom da svijet proširene stvarnosti ima ishodište u koordinatama (0,0), nakon što se izračunaju udaljenosti po x osi i z osi između točaka X i Y, te iste udaljenosti predstavljaju koordinate objekta kojega je potrebno postaviti u svijet proširene stvarnosti.

Haversinus formula

Haversinus formula se koristi za računanje udaljenosti između dvije točke na sfernoj površini. Haversinus označava pola inverznog sinusa (versinus). [57]



Sl. 3.5. Trigonometrijske funkcije [57]

$$haversin(\theta) = versin(\theta)/2 \tag{3-1}$$

gdje je θ kut u radijanima. Haversinus se također može izraziti kao:

$$haversin(\theta) = \sin^2(\theta/2) \tag{3-2}$$

Prije korištenja haversinus formule potrebno je izračunati razliku u zemljopisnoj širini, zemljopisnoj dužini i pretvoriti ih u radijane.

Udaljenost po geografskoj širini objekta i uređaja u radijanima:

$$\Delta\varphi = (\varphi_1 \cdot \pi)/180 - (\varphi_2 \cdot \pi)/180 \tag{3-3}$$

gdje je φ_1 zemljopisna širina objekta, a φ_2 zemljopisna širina uređaja.

Udaljenost po geografskoj dužini objekta i uređaja u radijanima:

$$\Delta\lambda = (\lambda_1 \cdot \pi)/180 - (\lambda_2 \cdot \pi)/180 \tag{3-4}$$

gdje je λ_1 zemljopisna dužina objekta, a λ_2 zemljopisna dužina uređaja.

Nakon toga se računa parametar 'a':

$$a = \sin^2(\Delta\varphi\,/2) + \cos(\varphi_1) \cdot \cos(\varphi_2) \cdot sin^2(\Delta\lambda/2) \tag{3-5}$$

gdje φ predstavlja zemljopisnu širinu izraženu u radijanima, a λ predstavlja zemljopisnu dužinu izraženu u radijanima. Nakon toga se računa parametar 'c'.

$$c = 2 \cdot \arctan\left(\sqrt{a}, \sqrt{1-a}\right) \tag{3-6}$$

gdje je 'a' prethodno izračunati parametar.

Nakon toga se može izračunati udaljenost pomoću jednadžbe:

$$d = R \cdot c \tag{3-5}$$

gdje je 'c' prethodno izračunati parametar, a R je polumjer sfere, odnosno polumjer Zemlje (R = 6378.137 km). [56]

Pomoću azimuta

Biblioteka ARCL za određivanje pozicije objekta u virtualnom svijetu koristi azimut. Prvo se odredi udaljenost objekta od trenutne lokacije korisnika po zemljopisnoj širini i dužini korištenjem ugrađene funkcije *distance(from:)* klase *CLLocation* koja vraća udaljenost u metrima. Nakon toga se udaljenosti po zemljopisnoj širini i dužini pretvore u vrijednosti zemljopisnih koordinata.

$$distRadLat = \frac{latM}{6360500} \tag{3-6}$$

$$distRadLong = \frac{longM}{5602900} \tag{3-7}$$

gdje 'distRadLat' predstavlja udaljenost po zemljopisnoj širini u radijanima, 'latM' je udaljenost zemljopisne širine u metrima, 'distRadLong' predstavlja udaljenost po zemljopisnoj dužini u radijanima, a 'longM' je udaljenost zemljopisne dužine u metrima. Zatim se zemljopisna širina i dužina objekta pretvaraju u radijane.

$$lat1 = lat \cdot \frac{\pi}{180} \tag{3-8}$$

$$lon1 = lon \cdot \frac{\pi}{180} \tag{3-9}$$

gdje je 'lat1' zemljopisna širina objekta u radijanima, 'lon1' zemljopisna dužina objekta u radijanima, 'lat' zemljopisna širina objekta u stupnjevima, a 'lon' zemljopisna dužina objekta u stupnjevima. Kada su te vrijednosti poznate, vrijeme je za određivanje udaljenosti pomoću azimuta.

$$lat2 = \text{asin}\left(\sin(lat1) \cdot \cos(distRadLat) + \cos(lat1) \cdot \sin(distRadLat) \cdot \cos(azimut)\right) \tag{3-10}$$

$$lon2 = lon1 + atan2(\sin(azimut) \cdot \sin(distRadLong) \cdot \cos(lat1), \cos(distRadLong) - \sin(lat1) \cdot \sin(lat2)) \quad (3\text{-}11)$$

gdje je 'lat2' nova vrijednost zemljopisne širine objekta, 'lon2' nova vrijednost zemljopisne dužine objekta, a 'azimut' je azimut koji je očitao mobilni uređaj. Na kraju se nove vrijednosti pretvore u stupnjeve.

$$latF = lat2 \cdot \frac{180}{\pi} \quad (3\text{-}12)$$

$$longF = lon2 \cdot \frac{180}{\pi} \quad (3\text{-}13)$$

gdje su 'latF' nova zemljopisna širina objekta u stupnjevima, a 'longF' nova zemljopisna dužina objekta u stupnjevima. [54]

3.2.3. Određivanje visine objekta

Uz x i z koordinate, postoji i y koordinata koja označava visinu na kojoj se objekt nalazi. Ta vrijednost se računa kao razlika između nadmorske visine na kojoj se nalazi mobilni uređaj u trenutku pokretanja pogleda proširene stvarnosti i nadmorske visine objekta, odnosno ceste na određenoj zemljopisnoj lokaciji. S obzirom da biblioteka MapKit ne pruža mogućnost dobivanja informacije o nadmorskoj visini na određenoj zemljopisnoj lokaciji, najbolji način za dobivanje nadmorske visine na određenoj zemljopisnoj lokaciji je korištenjem određenog internetskog servisa. U tom slučaju se koristi Open Elevation API tvrtke Mapquest. Šalje se GET zahtjev na zadanu html adresu, a server vraća odgovor u obliku JSON datoteke. Nadmorska visina se čita iz JSON datoteke. Očitava se trenutna nadmorska visina mobilnog uređaja pomoću biblioteke CoreLocation, te se y koordinata objekta određuje kao razlika između nadmorske visine uređaja i nadmorske visine očitane iz JSON datoteke. Primjer upita i odgovora sa Mapquest servera [58]:

```
GET
http://open.mapquestapi.com/elevation/v1/profile?key=KEY&shapeFormat=raw&latLngCollecti
on=39.74012,-104.9849,39.7995,-105.7237,39.6404,-106.3736
```

Programski kod 3.1. GET upit na Mapquest server

```
{
  "shapePoints":[
    39.74012,
    -104.9849,
    39.7995,
    -105.7237,
    39.6404,
    -106.3736
  ],
  "elevationProfile":[
    {
      "distance":0,
      "height":1616
    },
    {
      "distance":63.5583,
      "height":3910
    },
    {
      "distance":121.9561,
      "height":2501
    }
  ],
  "info":{
    "copyright":{
      "text":"© 2018 MapQuest, Inc.",
      "imageUrl":"http://api.mqcdn.com/res/mqlogo.gif",
      "imageAltText":"© 2018 MapQuest, Inc."
    },
    "statuscode":0,
    "messages":[

    ]
  }
}
```

Programski kod 3.2. Odgovor Mapquest servera u JSON obliku

3.2.4. Dodavanje objekta na prikaz proširene stvarnosti

ARSCNView

Za dodavanje objekta u ARSCNView prikaz proširene stvarnosti potrebno je stvoriti objekt tipa *SCNNode*, odrediti mu geometriju(kvadar, ravnina, sfera...), postaviti veličinu geometrijskog oblika i materijal geometrijskog oblika(boja, slika ili videozapis). Nakon toga je potrebno odrediti poziciju objekta pridruživanjem objekta tipa *SCNVector3* vrijednosti *position* objekta *SCNNode*. Objekt *SCNVector3* prima 3 parametra koji odgovaraju pozicijama na x, y i z osi. Nakon toga je potrebno objekt *SCNNode* proslijediti metodi *addChildNode* korijenskom ili postojećem čvoru prikaza proširene stvarnosti. [34]

```
let cubeNode = SCNNode(geometry: SCNBox(width: 0.1, height: 0.1, length: 0.1, chamferRadius: 0))
cubeNode.position = SCNVector3(0, 0, -0.2)
sceneView.scene.rootNode.addChildNode(cubeNode)
```

Programski kod 3.3. Primjer dodavanja čvora, odnosno objekta na prikaz proširene stvarnosti [34]

ARCL

Za dodavanje objekta u *SceneLocationView* prikaz proširene stvarnosti potrebno je stvoriti objekt tipa *LocationNode* koji prima jedan parametar, a to je objekt klase *CLLocation*. Objekt klase *CLLocation* sadrži zemljopisne koordinate i nadmorsku visinu. Objekt se postavlja pozivanjem metode *addLocationNodeWithConfirmedLocation* klase SceneLocationView koja kao parametar prima objekt klase *LocationNode.* Biblioteka u pozadini obavlja sve funkcije opisane u prethodnom potpoglavlju.

3.2.5. Transformacija objekta

Čvor se može transformirati pomoću varijable *transform* kojoj se pridružuje objekt klase *SCNMatrix4* koji predstavlja matricu veličine 4x4. Nove koordinate čvora se dobiju množenjem trenutnih koordinata čvora matricom transformacije kao što je prikazano na slici 3.6. [59] [60]

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} * \begin{bmatrix} m11 & m12 & m13 & m14 \\ m21 & m22 & m23 & m24 \\ m31 & m32 & m33 & m34 \\ m41 & m42 & m43 & m44 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$

koordinate matrica transformacije nove koordinate

Sl. 3.6. Dobivanje novih koordinata čvora [59]

Matrica transformacije se popunjava ovisno o tome što se želi napraviti sa čvorom. Način popunjavanja matrica je prikazan na slici 3.7.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Identitet

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ tx & ty & tz & 1 \end{bmatrix}$$

Translacija

$$\begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Skaliranje

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotacija oko osi x

$$\begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotacija oko osi y

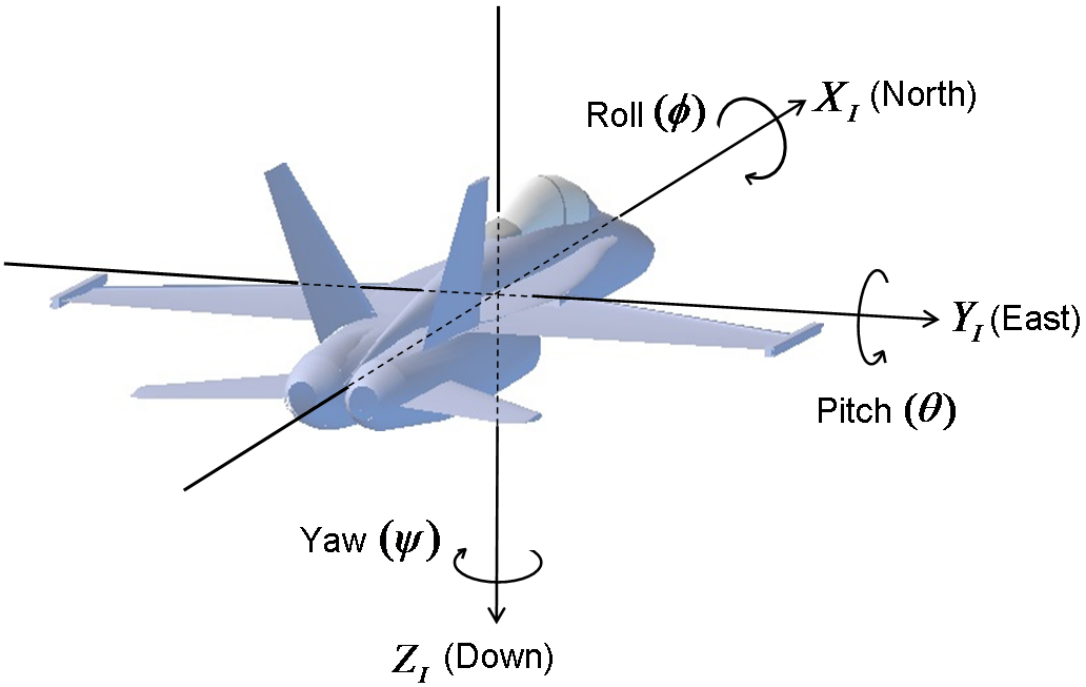
$$\begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotacija oko osi z

Sl. 3.7. Primjer popunjavanja matrice transformacije [59]

3.2.6. Rotacija objekta

Rotacija čvorova, uz korištenje transformacije, moguća je pomoću promjene varijabli *rotation* i *eulerAngles*. Rotacija pomoću varijable *rotation* je objašnjena u potpoglavlju 2.4.2. Varijabli *eulerAngles* se pridružuje objekt tipa *SCNVector3*, koji predstavlja trodimenzionalni vektor. Ukoliko se čvor želi rotirati po x osi, vektoru se mijenja prva veličina, po y osi se mijenja druga veličina, a po z osi se mijenja treća veličina. Također, vektoru je potrebno predati podatke u obliku radijana. Varijabla *eulerAngles* također sadrži varijable x, y i z, tako da nije potrebno pridruživati vektor, nego je moguće direktno njima mijenjati vrijednosti. [61]



Sl. 3.8. Primjer rotacije objekta po osima pomoću Eulerovih kutova [62]

3.3. Opis aplikacije

Aplikacija je napravljena za iOS operativni sustav. Sastoji se od 3 ekrana i pripadajućih kontrolera. Prvi ekran sadrži isključivo gumb za prijelaz na idući ekran, gdje korisnik odabire odredište. Na drugom ekranu se nalazi karta i tražilica. Korisnik može dugim pritiskom na kartu ili korištenjem tražilice odabrati odredište. Nakon odabira, lokacija odredišta se prikazuje na karti. Pritiskom na gumb "Prikaži" korisnik otvara treći ekran na kojemu se nalazi prikaz proširene stvarnosti. Ekran se sastoji od prikaza proširene stvarnosti, pomoćne karte, gumbova za rotaciju objekata, gumba za resetiranje prikaza i opisa iduće radnje. Korisniku se prikazuju 3D objekti na cesti, plave boje, koji ga usmjeravaju prema odredištu, a na svakom skretanju se prikazuje strelica koja usmjerava korisnika. Odredište je obilježeno crvenom strelicom koja je usmjerena u zemlju. Korisnik gumbovima može rotirati prikazane objekte, ukoliko nisu precizno postavljeni, a ukoliko ni tako ne može podesiti prikaz, pritiskom na gumb "Reset" može resetirati cijeli prikaz. To znači da aplikacija uklanja sve objekte, određuje novu rutu i postavlja nove objekte u prikaz. Na umanjenoj karti korisnik može vidjeti dio rute koju treba pratiti kako bi došao do odredišta. Također, na ekranu se korisniku ispisuje idući korak koji je potrebno napraviti.
Use-case dijagram aplikacije se nalazi na slici 3.9.

Aplikacija je napravljena pomoću MVC obrasca, pa uz kontrolere i prikaze, sadrži i modele. Sadrži 5 modela:

- ARViewControllerModel - služi kao pomoćni model ARViewControlleru, zbog smanjenja veličine samog kontrolera
- Route - Sadrži klasu RouteManager, strukturu Route i strukturu RouteStep. Klasa RouteManager sadrži sve potrebne atribute i metode za upravljanje objektima rute. Struktura Route sadrži niz objekata RouteStep koji sadrže koordinate i nadmorsku visinu pojedinog objekta koji predstavlja rutu kretanja.
- Speed - Sadrži klasu SpeedManager koja služi za dobavljanje maksimalne dozvoljene brzine za određene zemljopisne koordinate
- LocationManager - Klasa koja služi za određivanje trenutne pozicije korisnika, azimut, udaljenost između dvaju zemljopisnih koordinata, nadmorske visine za određene koordinate i udaljenost između korisnika i određenih koordinata
- Constants - Statička klasa koja sadrži URL API-ja za određivanje nadmorske visine na određenoj lokaciji

Klasni prikaz aplikacije se nalazi na slici 3.10.

Prije nego li se pokrene navigacija korisnika aplikacija mora napraviti sljedeće korake:

1. Odrediti lokaciju odredišta
2. Odrediti lokaciju korisnika
3. Odrediti rutu kretanja
4. Postaviti prikaz proširene stvarnosti
5. Dodati prikaz pomoćne karte
6. Dodati posljednju uputu
7. Dodati prvu uputu
8. Dodati cestovna obilježja

Nakon toga korisnik može početi voziti, odnosno započeti navigaciju. Tijekom korisnikove vožnje se odvijaju sljedeći koraci:

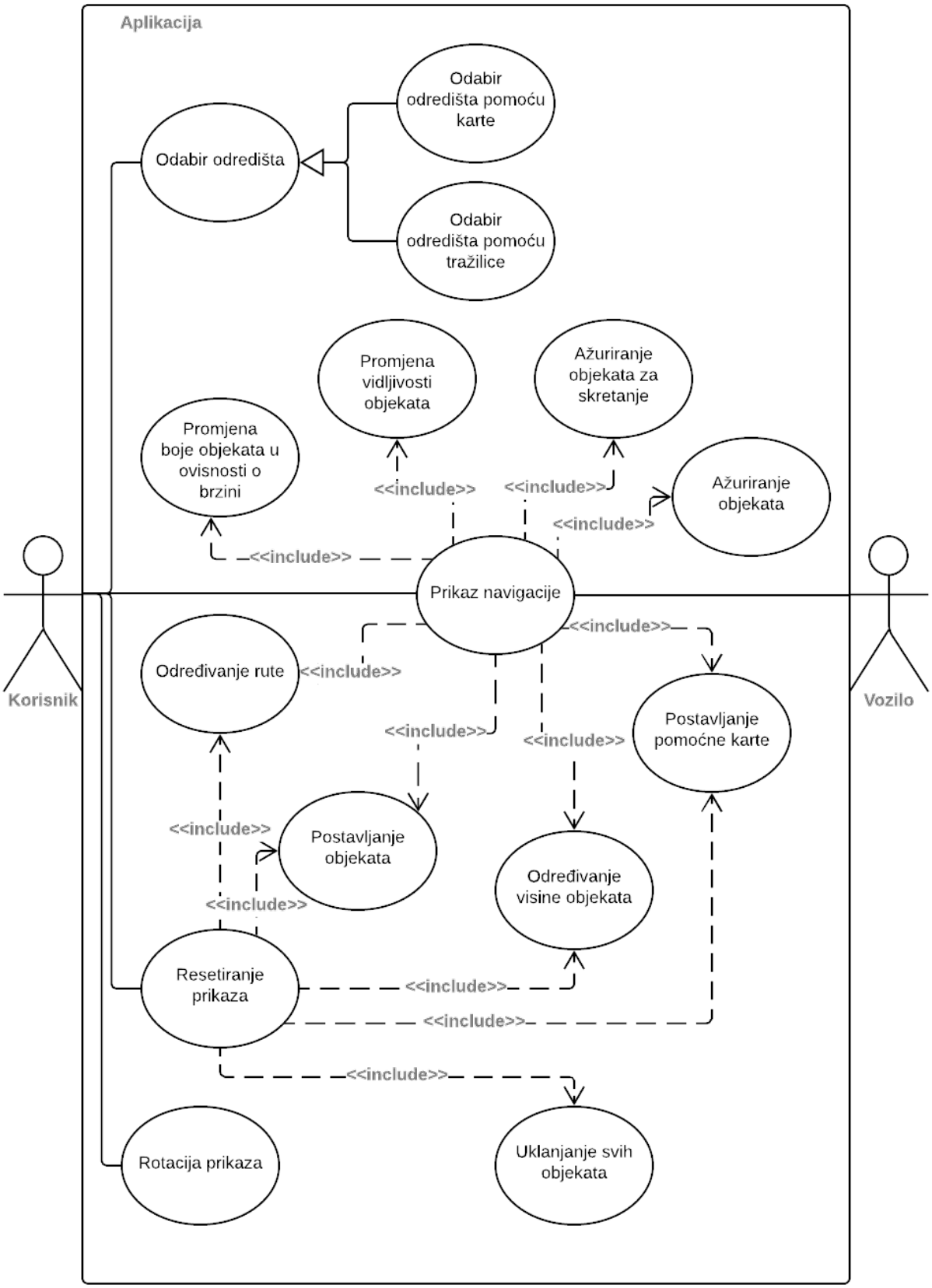
1. Provjera brzine i promjena boje objekata - Aplikacija provjerava trenutnu brzinu korisnika, maksimalnu dozvoljenu brzinu na toj lokaciji i ovisno o tome mijenja boju cestovnih objekata u plavu, narančastu ili crvenu boju.
2. Provjera udaljenosti od idućeg skretanja i ažuriranje objekata - Aplikacija provjerava je li udaljenost od idućeg skretanja manja od 20 metara. Ukoliko je, uklanja se trenutni objekt koji prikazuje skretanje i postavlja se idući. Također se mijenja opis idućeg koraka.
3. Provjera mogućih dodatnih objekata rute - Ukoliko ruta sadrži više od 200 cestovnih objekata, prikazuje se samo prvih 200. Aplikacija provjerava udaljenost od zadnjeg postavljenog objekta i ukoliko je manje od 25 metara, postavljaju se idućih 200 objekata.
4. Provjera visine i vidljivosti objekata - Ukoliko visina objekata nije zadana, aplikacija postavlja visinu objekata na visinu kamere. Ukoliko je visina objekata zadana, postavlja se visina objekata na visinu kamere samo za objekte koji su unutar 2 metra. Također, postavlja se vidljivost cestovnih objekata koji se nalaze unutar 50 metara od korisnika.

Dijagram toka aplikacije se nalazi kao Prilog A.

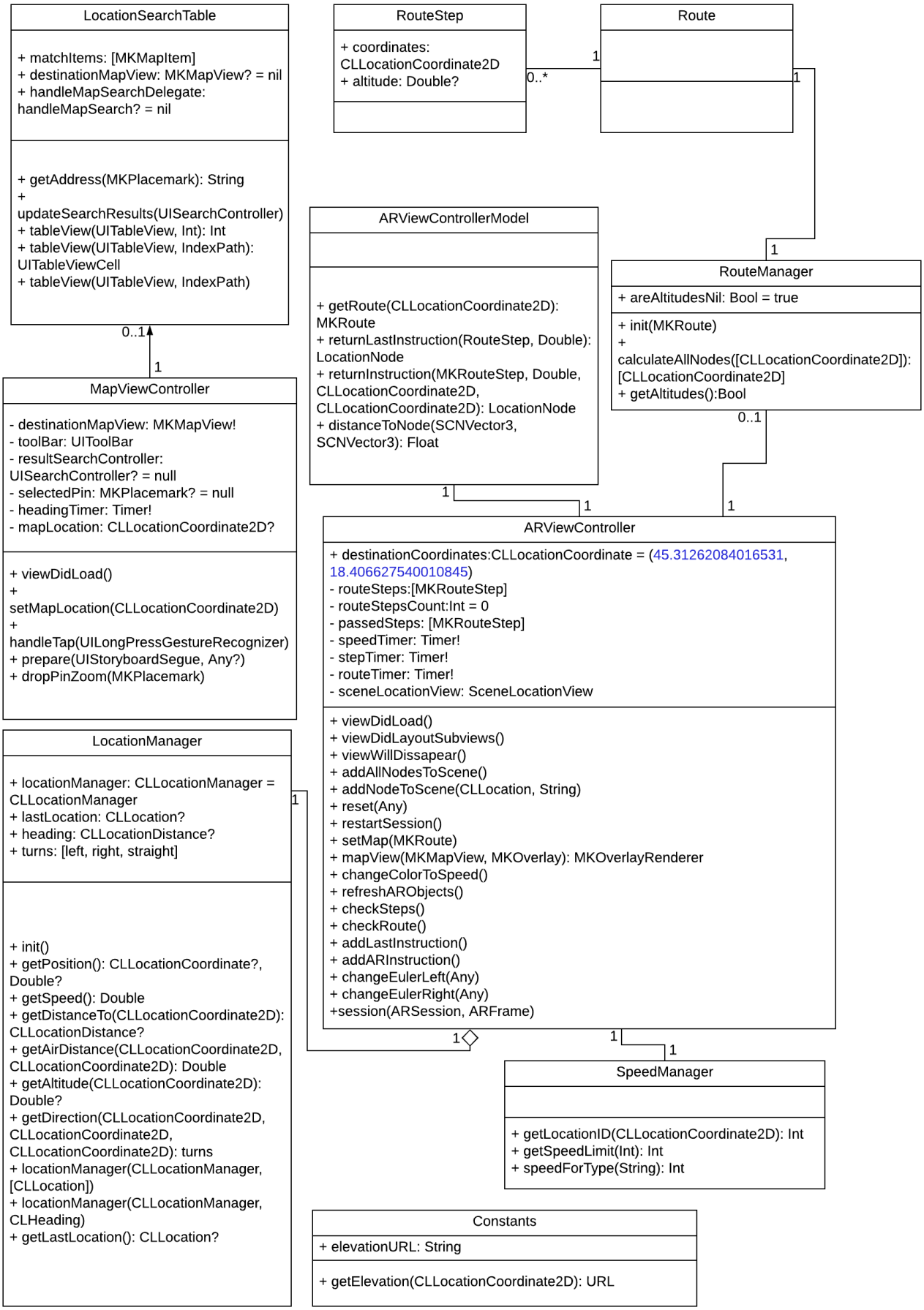
Prva tri koraka se ponavljaju u intervalima, a četvrti korak se odvija svakom promjenom prikaza proširene stvarnosti. Provjera brzine se odvija svake sekunde, a provjera udaljenosti od idućeg skretanja i provjera objekata rute se odvijaju svakih 500 milisekundi. Svi koraci su detaljnije objašnjeni dalje u tekstu.

Aplikacija je prevedena na hrvatski i engleski jezik.

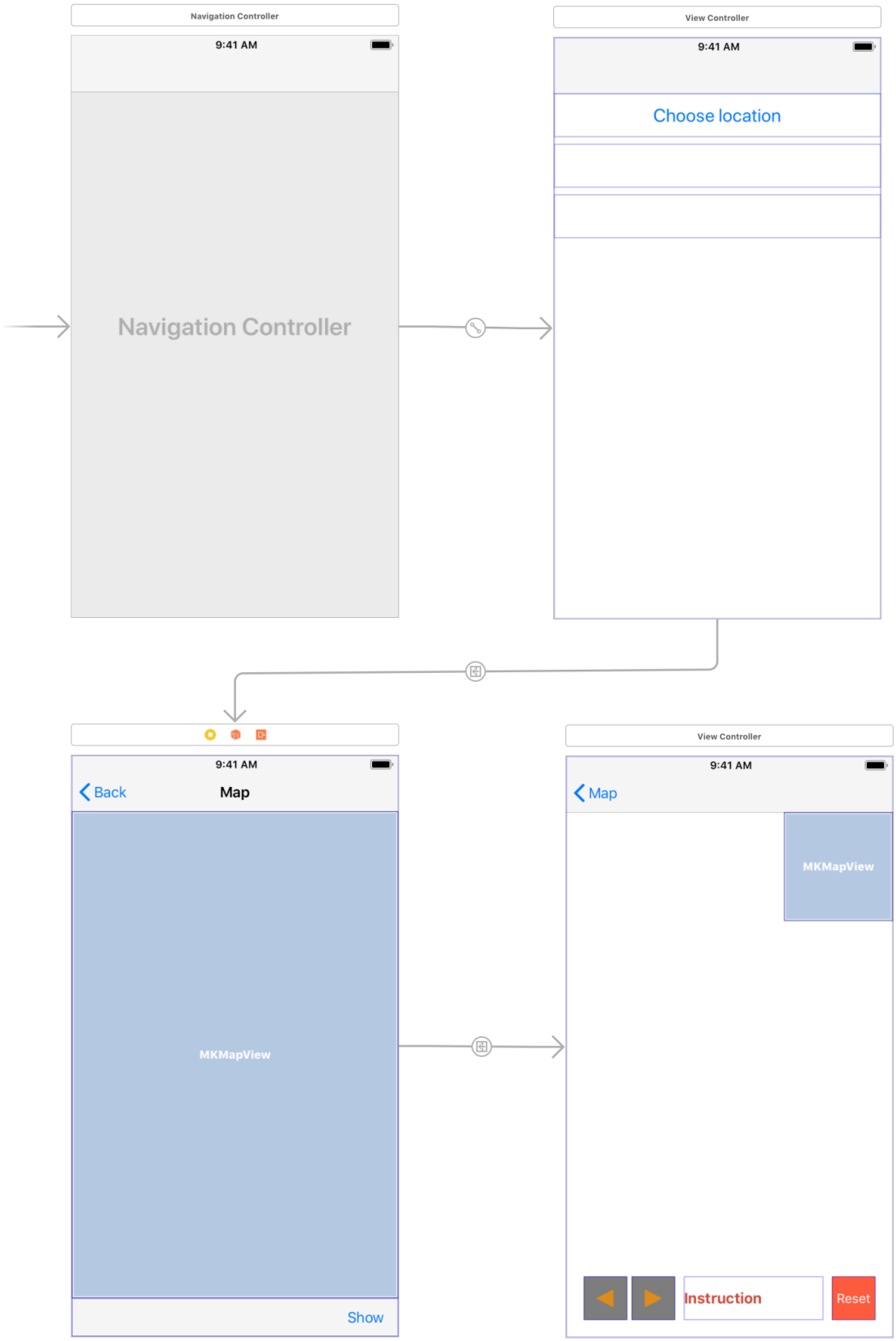
Izvorni kod aplikacije se nalazi na Github repozitoriju - ArVehNav [63].



Sl. 3.9. Use-Case dijagram aplikacije



Sl. 3.10. Klasni dijagram aplikacije



Sl. 3.11. Izgled aplikacije u XCode programskom alatu

3.3.1. Određivanje lokacije odredišta

Korisnik može odrediti lokaciju odredište na 2 načina:

- Odabirom lokacije na karti
- Korištenjem tražilice

Za određivanje zemljopisne lokacije odredišta putem karte zadužena je klasa MapViewController, za određivanje lokacije putem tražilice zadužene su klase MapViewController i LocationSearchTable, dok je za određivanje lokacije odredišta putem spremljene lokacije zadužena klasa ItemTableViewController.

Odabir lokacije na karti

Zemljopisna lokacija, odnosno zemljopisne koordinate na koje korisnik želi ići dobivaju se pomoću *MKMapView* prikaza zemljopisne karte i biblioteke MapKit. *MKMapView* i MapKit koriste Apple Maps za prikaz zemljopisne karte. Nakon što se aplikacija pokrene, korisnik pritisne gumb "Odaberi lokaciju", zatim se korisniku otvori novi prikaz koji sadrži kartu, tražilicu i gumb "Prikaži". Prilikom otvaranja prikaza, određuju se postavke karte(koliko je dugo potrebno pritisnuti kartu za određivanje lokacije, prikaz kompasa, skale i lokacije korisnika na karti), a na kartu se postavlja funkcija koja prepoznaje gesture, odnosno pritiske na kartu. Korisnik dugim pritiskom na kartu(0.5 sekundi) odabire zemljopisnu lokaciju na koju želi ići. Aplikacija pretvara lokaciju na karti u zemljopisnu lokaciju i sprema ju u varijablu *mapLocation*, te na kartu dodaje oznaku na mjesto odabrane lokacije. Nakon toga korisnik pritisne "Prikaži", te se otvara prikaz proširene stvarnosti. Aplikacija pomoću funkcije *prepareForSegue* prosljeđuje podatke o zemljopisnoj lokaciji između kontrolera koji sadrži *MKMapView* u kontroler koji sadrži *ARSCNView*.

```
override func viewDidLoad() {
        super.viewDidLoad()
        let gestureRecognizer = UILongPressGestureRecognizer(target: self, action:
#selector(MapViewController.handleTap(gestureRecognizer:)))
        gestureRecognizer.minimumPressDuration = 0.5
        gestureRecognizer.delaysTouchesBegan = true
        destinationMapView.delegate = self
        destinationMapView.showsScale = true
        destinationMapView.showsCompass = true
        destinationMapView.showsUserLocation = true
        gestureRecognizer.delegate = self as? UIGestureRecognizerDelegate
        destinationMapView.addGestureRecognizer(gestureRecognizer)
        ...
}
```

Programski kod 3.4. Određivanje postavki zemljopisne karte i dodavanje prepoznavanja gestura na zemljopisnu kartu

```
@objc func handleTap(gestureRecognizer: UILongPressGestureRecognizer) {
        if gestureRecognizer.state != UIGestureRecognizerState.began { return }
        let touchLocation = gestureRecognizer.location(in: destinationMapView)
        let locationCoordinate = destinationMapView.convert(touchLocation, toCoordinateFrom:
destinationMapView)
        mapLocation = locationCoordinate
        let mapAnnotations = destinationMapView.annotations
        destinationMapView.removeAnnotations(mapAnnotations)
        let pinPoint = MKPointAnnotation()
        pinPoint.coordinate = locationCoordinate
        destinationMapView.addAnnotation(pinPoint)
 }
```

Programski kod 3.5. Određivanje zemljopisne lokacije odredišta i postavljanje oznake na kartu [64]

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if segue.identifier == "ShowARViewSegue" {
            if let destinationVC = segue.destination as? ARViewController {
                if let coordinates = mapLocation {
                    destinationVC.destinationCoordinates = coordinates
                }          }}
```

Programski kod 3.6. Prosljeđivanje zemljopisnih koordinata na sljedeći pogled, odnosno pogled proširene stvarnosti

Odabir lokacije korištenjem tražilice

Tražilica, odnosno objekt klase UISearchBar je postavljena iznad karte. Tražilica također ima pripadajući kontroler tipa UISearchController koji upravlja rezultatima traženja. U ovoj aplikaciji to je LocationSearchTable kontroler koji sadrži UITableView, odnosno tablicu u kojoj prikazuje rezultate pretraživanja. Kada korisnik upiše određeni tekst u tražilicu pokreće se funkcija updateSearchResults koja stvara novi objekt klase MKLocalSearchRequest i prosljeđuje mu tekst koji je korisnik upisao zajedno sa regijom koja je prikazana na karti. Nakon toga se novostvoreni objekt prosljeđuje objektu klase MKLocalSearch koji šalje zahtjev na Appleov server i kao odgovor prima niz objekata tipa MKMapItem koji predstavljaju lokacije na karti. Ukoliko je zahtjev neuspješan, server vraća grešku koja se ispisuje u konzoli. Nakon toga se osvježava prikaz podataka u tablici pozivom metode *reloadData* klase *tableView*. Metoda *reloadData* poziva niz funkcija od kojih je prva *tableView(numberOfRowsInSection)* koja vraća broj ćelija u tablici, odnosno veličinu niza koji sadrži rezultate pretraživanja. Nakon toga se poziva funkcija tableView(cellForRowAt) koja prikazuje rezultate pretrage po ćelijama. U ćeliji se prikazuju naziv i adresa lokacije. Adresa se dobiva pomoću metode *getAddress* koja parsira rezultat pretraživanja i vraća podatak tipa String koji sadrži (ukoliko postoji) broj, ulicu, grad i administrativnu regiju lokacije. Ukoliko korisnik odabere neki od rezultata, poziva se funkcija *tableView(didSelectRowAt)* koja prosljeđuje zemljopisnu lokaciju, odnosno objekt tipa MKPlacemark funkciji *dropPinZoom* koja se nalazi u klasi MapViewController. Funkcija *dropPinZoom* postavlja oznaku na kartu, približi prikaz na postavljenu oznaku i postavi vrijednost varijable *mapLocation* pomoću funkcije *setMapLocation* na lokaciju odredišta. [65] Nakon toga se pomoću funkcije *prepareForSegue* (programski kod 3.6.) prosljeđuje podatak o lokaciji odredišta na sljedeći ekran (koji sadrži prikaz proširene stvarnosti).

```
override func viewDidLoad() {
        ...
        let locationSearchTable = storyboard!.instantiateViewController(withIdentifier:
"LocationSearchTable") as! LocationSearchTable
        resultSearchController = UISearchController(searchResultsController: locationSearchTable)
        resultSearchController?.searchResultsUpdater = locationSearchTable

        let searchBar = resultSearchController!.searchBar
        searchBar.sizeToFit()
        searchBar.placeholder = NSLocalizedString("searchPlaces", comment: "Search for places")
        navigationItem.titleView = resultSearchController?.searchBar

        resultSearchController?.hidesNavigationBarDuringPresentation = false
        resultSearchController?.dimsBackgroundDuringPresentation = true
        definesPresentationContext = true

        locationSearchTable.destinationMapView = destinationMapView
        locationSearchTable.handleMapSearchDelegate = self
    }
```

Programski kod 3.7. Postavljanje prikaza tražilice

```
class LocationSearchTable:UITableViewController{

    var matchItems:[MKMapItem] = []
    var destinationMapView: MKMapView? = nil
    var handleMapSearchDelegate:handleMapSearch? = nil

    func getAddress(item: MKPlacemark)->String{
        var address = ""
        if let number = item.subThoroughfare {
            address += number.description
        }
        if let itemAddr = item.thoroughfare{
            address += " \(itemAddr.description)"
        }
        if let city = item.locality{
            address += ", \(city.description)"
        }
        if let region = item.administrativeArea{
            address += ", \(region.description)"
        }
        return address
    }
}

extension LocationSearchTable:UISearchResultsUpdating{
    func updateSearchResults(for searchController: UISearchController) {
        guard let destinationMapView = destinationMapView,
```

```
            let searchBarText = searchController.searchBar.text else {
                return
            }
        let req = MKLocalSearchRequest()
        req.naturalLanguageQuery = searchBarText
        req.region = destinationMapView.region
        let search = MKLocalSearch(request: req)
        search.start { (data, err) in
            guard let data = data else{
                print(err?.localizedDescription)
                return
            }
            self.matchItems = data.mapItems
            self.tableView.reloadData()
        }   }}

extension LocationSearchTable{
    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) ->
Int {
        return matchItems.count
    }

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
UITableViewCell {
        guard let cell = tableView.dequeueReusableCell(withIdentifier: "cell") else{
            fatalError("Cell error")
        }
        let selected = matchItems[indexPath.row].placemark
        cell.textLabel?.text = selected.name
        cell.detailTextLabel?.text = getAddress(item: selected)
        return cell
    }

    override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        let selected = matchItems[indexPath.row].placemark
        handleMapSearchDelegate?.dropPinZoom(place: selected)
        dismiss(animated: true, completion: nil)
    }
}
```

Programski kod 3.8. Klasa koja upravlja rezultatima pretrage

```
extension MapViewController:handleMapSearch{
    func dropPinZoom(place: MKPlacemark) {
        selectedPin = place
        destinationMapView.removeAnnotations(destinationMapView.annotations)
        let selectedPlace = MKPointAnnotation()
        selectedPlace.coordinate = place.coordinate
        selectedPlace.title = place.name
        if let city = place.locality{
```

```
            selectedPlace.subtitle = city
        }
        setMapLocation(coord: place.coordinate)
        destinationMapView.addAnnotation(selectedPlace)
        let span = MKCoordinateSpanMake(0.05, 0.05)
        let region = MKCoordinateRegionMake(selectedPlace.coordinate, span)
        destinationMapView.setRegion(region, animated: true)
    }
}
```

Programski kod 3.9. Funkcija koja postavlja oznaku na kartu

3.3.2. Određivanje lokacije korisnika

Za određivanje lokacije korisnika koristi se klasa LocationManager koja sadrži objekt klase CLLocationManager. Klasa LocationManager je singleton klasa. Za određivanje pozicije korisnika koristi se metoda *getPosition* koja vraća korisnikove koordinate i nadmorsku visinu ukoliko ih je moguće očitati.

```
func getPosition() -> (Location: CLLocationCoordinate2D?, Altitude: Double?) {
        var curCoordinates: CLLocationCoordinate2D? = nil
        if let location = locationManager.location{
            curCoordinates = CLLocationCoordinate2D(latitude: location.coordinate.latitude,
longitude: location.coordinate.longitude)
        }
        let altitude = (locationManager.location?.altitude)
        return (curCoordinates, altitude)
    }
```

Programski kod 3.10. Funkcija koja vraća korisnikovu lokaciju i nadmorsku visinu

3.3.3. Određivanje rute kretanja

Za dobivanje rute koristi se funkcija *getRoute* vlastite klase ARViewControllerModel. Funkciji se predaju koordinate odredišta, a lokacija korisnika se određuje pomoću funkcije *getPosition*(programski kod 3.10.). Nakon toga se stvara novi objekt tipa MKDirectionsRequest kojemu se predaju početna lokacija, završna lokacija i vrsta prijevoznog sredstva, odnosno automobil. Nakon toga se pomoću objekta klase MKDirections šalje zahtjev za izračunavanje rute. Rezultat je niz mogućih ruta, te funkcija vraća prvu rutu iz niza.

```
func getRoute(destination destinationCoordinates: CLLocationCoordinate2D, onCompletion:
@escaping (MKRoute) -> Void) {
        let curCoord = LocationManager.shared.getPosition().Location
        guard let currentCoordinates = curCoord else{
            return
```

```
        }
        let startPlace = MKPlacemark(coordinate: currentCoordinates)
        let destPlace = MKPlacemark(coordinate: destinationCoordinates)
        let startItem = MKMapItem(placemark: startPlace)
        let destItem = MKMapItem(placemark: destPlace)
        let routeRequest = MKDirectionsRequest()
        routeRequest.source = startItem
        routeRequest.destination = destItem
        routeRequest.transportType = .automobile

        let directions = MKDirections(request: routeRequest)
        directions.calculate(completionHandler: { [weak self] response, error in
            guard let response = response else {
                if let error = error {
                    print(error.localizedDescription)
                }
                return
            }

            let route = response.routes[0]
            onCompletion(route)
        })
    }
```

Programski kod 3.11. Funkcija koja vraća rutu

Za upravljanje i spremanje rute koristi se klasa RouteManager koja sadrži objekt tipa Route. Objekt tipa Route sadrži niz objekata tipa RouteStep koji predstavljaju korake rute. Objekt tipa RouteStep sadrži zemljopisne koordinate i nadmorsku visinu. Klasa RouteManager u svom konstruktoru prima objekt klase MKRoute i poziva funkciju *calculateAllNodes* koja dodaje međukorake ukoliko je razlika između dva koraka veća od 10 metara. Koraci se dobivaju pomoću vrijednosti "polyline" objekta "route", te njegove vrijednosti "coordinates". Unutar funkcije *calculateAllNodes*, koraci se dodaju prema sljedećoj tablici:

Tab. 3.1. Dodavanje međukoraka

Udaljenost između koraka	Koraci dodani na svakih X metara
Od 20 do 100 metara	X = 20
Od 100 do 1000 metara	X = 50
Od 1000 do 5000 metara	X = 100
Preko 5000 metara	X = 200

Udaljenost između dva koraka se računa pomoću metode *getAirDistance* klase LocationManager. Metoda *getAirDistance* koristi haversinus formulu kako bi izračunala udaljenost između dva koraka.

```
init(forRoute route: MKRoute) {
        let steps = calculateAllNodes(steps: route.polyline.coordinates)
        for step in steps {
            self.route.steps.append(RouteStep(coordinates: step, altitude: nil))
        }
    }
```

Programski kod 3.12. Konstruktor klase RouteManager

```
public extension MKMultiPoint{
    var coordinates: [CLLocationCoordinate2D]{
        var rCoords = [CLLocationCoordinate2D](repeating: kCLLocationCoordinate2DInvalid,
count: pointCount)
        getCoordinates(&rCoords, range: NSMakeRange(0, pointCount))
        return rCoords
    }
}
```

Programski kod 3.13. Ekstenzija klase MKMultiPoint koja sadrži varijablu coordinates [66]

```
func calculateAllNodes(steps: [CLLocationCoordinate2D]) -> [CLLocationCoordinate2D] {
        var newSteps = [CLLocationCoordinate2D]()
        for (index, step) in steps.enumerated() {

            if index < steps.endIndex - 1 {
                let pointA = step
                let pointB = steps[index + 1]
                let diffLat = pointB.latitude - pointA.latitude
                let diffLong = pointB.longitude - pointA.longitude

                let distanceAirAB = LocationManager.shared.getAirDistance(currentLocation: pointA,
destinationLocation: pointB)
                var numPoints: Int = 0
                if (distanceAirAB > 20) && (distanceAirAB < 100) {
                    numPoints = Int(distanceAirAB / 20)
                } else if distanceAirAB > 100 && distanceAirAB < 1000 {
                    numPoints = Int(distanceAirAB / 50)
                } else if distanceAirAB > 1000 && distanceAirAB < 5000{
                    numPoints = Int(distanceAirAB / 100)
                } else if distanceAirAB > 5000{
                    numPoints = Int(distanceAirAB / 200)
                }

                let intervalLat = diffLat / (Double(numPoints) + 1)
                let intervalLong = diffLong / (Double(numPoints) + 1)

                if numPoints == 0 {
                    let coordinates = CLLocationCoordinate2D(latitude: pointA.latitude, longitude:
pointA.longitude)
                    newSteps.append(coordinates)
                } else {
                    for index in 1...numPoints{
```

```
                let coordinates = CLLocationCoordinate2D(latitude: pointA.latitude + intervalLat
* Double(index), longitude: pointA.longitude + intervalLong * Double(index))
                newSteps.append(coordinates)
            }
        }
      }
      newSteps.append(step)
    }
    return newSteps
  }
```

Programski kod 3.14. Funkcija za računanje međukoraka [67]

```
func getAirDistance(currentLocation: CLLocationCoordinate2D, destinationLocation:
CLLocationCoordinate2D) -> Double {
        let R = 6378.137
        let distanceLat = destinationLocation.latitude * Double.pi / 180 - currentLocation.latitude
* Double.pi / 180
        let distanceLong = destinationLocation.longitude * Double.pi / 180 -
currentLocation.longitude * Double.pi / 180
        let a = sin(distanceLat / 2) * sin(distanceLat / 2) + cos(destinationLocation.latitude *
Double.pi / 180) * cos(currentLocation.latitude * Double.pi / 180) * sin(distanceLong / 2) *
sin(distanceLong / 2)
        let c = 2 * atan2(sqrt(a), sqrt(1 - a))
        let d = R * c
        return d * 1000
  }
```

Programski kod 3.15. Funkcija za određivanje udaljenosti između dvije lokacije

Za određivanje nadmorske visine svih koraka koristi se metoda *getAltitudes* klase RouteManager. Metoda *getAltitudes* za svaki korak poziva metodu *getAltitude* klase LocationManager koja šalje zahtjev na Mapquest poslužitelj i pomoću klase Alamofire parsira nadmorsku visinu iz odgovora. Ukoliko server ne odgovori ili se ne uspije parsirati nadmorska visina iz odgovora, nadmorska visina se postavlja na vrijednost *nil* (odnosno *null*). Metoda *getAltitudes* također postavlja varijablu klase RouteManager "areAltitudesNil" na vrijednost *false* ukoliko barem jedan korak ima postavljenu visinu.

```
func getAltitudes(onCompletion: @escaping (Bool) -> Void) {
    let count = self.route.steps.count
    var finished = 0
    for (index,step) in self.route.steps.enumerated() {
      LocationManager.shared.getAltitude(destination: step.coordinates) { altitude in
        self.route.steps[index].altitude = altitude
        if altitude != nil{
          self?.areAltitudesNil = false
```

```
            }
            finished += 1
            if(finished == count){
                onCompletion(true)
            }           }       }   }
```

Programski kod 3.16. Funkcija za postavljanje nadmorskih visina za sve korake rute

```
func getAltitude(destination: CLLocationCoordinate2D, onCompletion: @escaping ((Double) ->
Void)) {
        var altitude: Double?
        Alamofire.request(Constants.getElevation(coordinates:    destination)).responseJSON    {
response in
            switch response.result {

            case .success(let data):
                let response = JSON(data)
                altitude = response["elevationProfile"][0]["height"].doubleValue
                onCompletion(altitude)

            case .failure(let error):
                print("Alamofire: \(error)")
                onCompletion(altitude)
            }           }   }
```

Programski kod 3.17. Funkcija za određivanje nadmorske visine određene lokacije

3.3.4. Postavljanje prikaza proširene stvarnosti

Za prikaz proširene stvarnosti koristi se klasa ARViewController. Na početku klase se deklariraju i inicijaliziraju varijable i konstante korištene u metodama klase. Nakon toga se nalazi poveznica na tekstualni objekt "GPSLoc" koji se prikazuje na ekranu, na gumb za resetiranje prikaza "resetBtn" i kartu koja prikazuje rutu kretanja "miniMap". U funkciji *viewDidLoad* se pokreće prikaz proširene stvarnosti metodom *run* klase SceneLocationView, te se prikaz postavlja unutar trenutnog prozora. Također se pokreće *timer* koji promatra brzinu korisnika i dozvoljenu brzinu, te ovisno o tim vrijednostima mijenja boju objekata proširene stvarnosti. Naposljetku se poziva funkcija *restartSession* koja je objašnjena dalje u tekstu (programski kod 3.39.).

Metoda *viewDidLayoutSubviews* postavlja prikaz proširene stvarnosti preko cijelog prozora.

Metoda *viewWillDisappear* se pokreće kada korisnik odluči izaći iz trenutnog prozora, te se zaustavljaju svi pokrenuti *timeri*, pauzira se prikaz proširene stvarnosti i uklanja iz prozora.

```
var destinationCoordinates = CLLocationCoordinate2D(latitude: 45.31262084016531, longitude:
18.406627540010845)
    private var routeSteps = [MKRouteStep]()
    private var routeStepsCount: Int = 0
    private var passedSteps = [MKRouteStep]()
    private var speedManager = SpeedManager()
    private var speedTimer: Timer!
    private var stepTimer: Timer!
    private var routeTimer: Timer!
    var itemAltitude: Double?
    private var sceneLocationView = SceneLocationView()
    private var routeManager: RouteManager?
    private let viewModel = ARViewControllerModel()
    private var startTime = DispatchTime.now()

    @IBOutlet weak var GPSLoc: UILabel!
    @IBOutlet weak var resetBtn: UIButton!
    @IBOutlet weak var miniMap: MKMapView!

    override func viewDidLoad() {
        super.viewDidLoad()
        miniMap.delegate = self
        sceneLocationView.run()
        view.addSubview(sceneLocationView)
        GPSLoc.text = NSLocalizedString("loading", comment: "Loading...")
        sceneLocationView.session.delegate = self
        speedTimer = Timer.scheduledTimer(timeInterval: 1, target: self, selector:
#selector(changeColorToSpeed), userInfo: nil, repeats: true)
        restartSession()
    }

    override func viewDidLayoutSubviews() {
        super.viewDidLayoutSubviews()
        sceneLocationView.frame = view.bounds
        view.sendSubview(toBack: sceneLocationView)
    }

    override func viewWillDisappear(_ animated: Bool) {
        speedTimer.invalidate()
        stepTimer.invalidate()
        routeTimer.invalidate()
        sceneLocationView.pause()
        sceneLocationView.removeFromSuperview()
    }
```

Programski kod 3.18. Postavljanje prikaza proširene stvarnosti

3.3.5. Dodavanje prikaza pomoćne karte

Na prikaz proširene stvarnosti se dodaje umanjena karta koja prikazuje rutu kojom se korisnik treba kretati. Funkcija *setMap* prima rutu kojom se korisnik treba kretati, dodaje oznaku na odredište, dodaje prikaz rute na kartu i postavlja praćenje korisnika na karti. Funkcija *mapView(rendererFor:)* postavlja izgled rute na karti na liniju plave boje i širine 4 točke.

```
func setMap(route: MKRoute){
      let destAnnotation = MKPointAnnotation()
      destAnnotation.coordinate = destinationCoordinates
      miniMap.showAnnotations([destAnnotation], animated: true)
      miniMap.add(route.polyline, level: MKOverlayLevel.aboveRoads)
      miniMap.setUserTrackingMode(.followWithHeading, animated: true)
   }

   func mapView(_ mapView: MKMapView, rendererFor overlay: MKOverlay) -> MKOverlayRenderer {
      let renderer = MKPolylineRenderer(overlay: overlay)
      renderer.strokeColor = UIColor.blue
      renderer.lineWidth = 4.0
      return renderer
   }
```

Programski kod 3.19. Postavljanje prikaza pomoćne karte [68]

3.3.6. Dodavanje posljednje upute

Posljednja uputa je 3D objekt koji pokazuje odredište na prikazu proširene stvarnosti. Posljednja uputa je prikazana kao crvena strelica usmjerena prema zemlji na mjestu odredišta. Postavlja se pomoću funkcije *addLastInstruction* klase ARViewController. Funkcija uzima zadnji korak rute, poziva funkciju *returnLastInstruction* klase ARViewControllerModel koja vraća objekt proširene stvarnosti, koji se zatim postavlja u prikaz proširene stvarnosti. Funkcija *returnLastInstruction* prema koordinatama i nadmorskoj visini vraća objekt LocationAnnotationNode koji predstavlja ravninu sa slikom crvene strelice okrenute prema zemlji i obilježjem SCNBillboardConstraint, što znači da se može vidjeti iz bilo koje pozicije korisnika u prikazu proširene stvarnosti. Također, objekt ima 20% prozirnosti.

```
func addLastInstruction() {
      if let step = routeManager?.route.steps.last {
         if let altitude = step.altitude {
            let node = viewModel.returnLastInstruction(step: step, altitude: altitude + 2)
            sceneLocationView.addLocationNodeWithConfirmedLocation(locationNode: node)
         } else if let altitude = LocationManager.shared.getPosition().Altitude {
            let node = viewModel.returnLastInstruction(step: step, altitude: altitude)
```

```
            sceneLocationView.addLocationNodeWithConfirmedLocation(locationNode: node)
        }       }   }
```

Programski kod 3.20. Funkcija za postavljanje posljednje upute

```
func returnLastInstruction(step: RouteStep, altitude: Double) -> LocationNode {
    let location = CLLocation(coordinate: step.coordinates, altitude: altitude + 6)
    let arrow = UIImage(named: "RedArrow")!
    let node = LocationAnnotationNode(location: location, image: arrow)
    node.tag = "lastNode"
    node.scaleRelativeToDistance = true
    node.opacity = 0.8
    return node
  }
```

Programski kod 3.21. Funkcija koja vraća objekt proširene stvarnosti

3.3.7. Dodavanje prve upute

Aplikacija dodaje uputu u obliku strelice (lijevo, desno ili uspravno) ukoliko korisnik treba skrenuti u drugu ulicu, izaći na određenom izlazu na kružnom toku i slično. Kako ne bi došlo do zabune, dodaje se samo prva iduća uputa. Uputa se dodaje pomoću funkcije *addARInstruction* klase ARViewController. Funkcija obriše sve instrukcije koje se već nalaze na prikazu proširene stvarnosti i nakon toga poziva funkciju *returnInstruction* klase ARViewControllerModel kako bi dobila sami objekt koji će staviti u prikaz proširene stvarnosti. Naposljetku postavlja objekt u prikaz proširene stvarnosti. Funkciji *returnInstruction* se prosljeđuje iduće skretanje, skretanje nakon toga, trenutna lokacija korisnika i nadmorska visina korisnika umanjena za jedan metar.

```
func addARInstruction() {
    for node in sceneLocationView.findNodes(tagged: "instructionNode") {
        sceneLocationView.removeLocationNode(locationNode: node)
    }
    let location = LocationManager.shared.getPosition()
    if let lastStepCoord = location.Location {
        if routeSteps.indices.contains(1) {
            let nextStepCoord = routeSteps[1].polyline.coordinate
            if let step = routeSteps.first {
                if let altitude = location.Altitude {
                    let node = viewModel.returnInstruction(step: step, altitude: altitude - 1, nextStep:
nextStepCoord, lastStep: lastStepCoord)
                    sceneLocationView.addLocationNodeWithConfirmedLocation(locationNode:
node)
                }           }       }   }   }
```

Programski kod 3.22. Funkcija koja dodaje sljedeću instrukciju

Funkcija *returnInstruction* vraća 3D objekt koji predstavlja okomitu ravninu sa slikom strelice, te na temelju vrijednosti koje vrati funkcija *getDirection* klase LocationManager odlučuje hoće li ravnina imati izgled strelice usmjerene desno, strelice usmjerene lijevo ili strelice usmjerene prema nebu. Objekt ima prozirnost 20%.

```
func returnInstruction(step: MKRouteStep, altitude: Double, nextStep: CLLocationCoordinate2D, lastStep: CLLocationCoordinate2D) -> LocationNode {
    let location = CLLocation(coordinate: step.polyline.coordinate, altitude: altitude + 4)
    let image: UIImage
    switch LocationManager.shared.getDirection(previous: lastStep, current: step.polyline.coordinate, next: nextStep) {
    case .right:
        image = UIImage(named: "RightArrow")!
    case .left:
        image = UIImage(named: "LeftArrow")!
    case .straight:
        image = UIImage(named: "StraightArrow")!
    }
    let node = LocationAnnotationNode(location: location, image: image)
    node.scaleRelativeToDistance = true
    node.tag = "instructionNode"
    node.opacity = 0.8
    return node
}
```

Programski kod 3.23. Funkcija koja vraća objekt proširene stvarnosti koji predstavlja instrukciju

Funkcija *getDirection* klase LocationManager na temelju trenutne, prethodne i iduće lokacije određuje da li se iduća lokacija nalazi lijevo, desno ili ravno od trenutne lokacije. Funkcija vraća jednu od mogućih vrijednosti tipa *turns*. [69]

```
enum turns{
    case left
    case right
    case straight
}

func getDirection(previous: CLLocationCoordinate2D, current: CLLocationCoordinate2D, next: CLLocationCoordinate2D) -> turns {
    let currentNew = CLLocationCoordinate2D(latitude: current.latitude - previous.latitude, longitude: current.longitude - previous.longitude)
    let nextNew = CLLocationCoordinate2D(latitude: next.latitude - previous.latitude, longitude: next.longitude - previous.longitude)
    if abs(currentNew.latitude*nextNew.longitude - currentNew.longitude*nextNew.latitude)<0.0000001{
```

```
            return .straight
        }
        else if currentNew.latitude * nextNew.longitude > currentNew.longitude * nextNew.latitude
{
            return .right
        } else {
            return .left
        }
    }
```

Programski kod 3.24. Funkcija koja vraća vrijednost lijevo, desno ili ravno

3.3.8. Dodavanje cestovnih obilježja

Poziva se funkcija *addAllNodesToScene* koja poziva metodu *getAltitudes* klase RouteManager. Kada metoda postavi nadmorsku visinu pozivaju se funkcije koje postavljaju objekte proširene stvarnosti.

```
func addAllNodesToScene() {
        if let routeMan = routeManager {
            routeMan.getAltitudes { [weak self] finished in
                if finished {
                    self?.addLastInstruction()
                    self?.routeStepsCount = routeMan.route.steps.count
                    self?.refreshARObjects()
                    self?.stepTimer = Timer.scheduledTimer(timeInterval: 0.5, target: self, selector:
#selector(self?.checkSteps), userInfo: nil, repeats: true)
                        if self?.routeTimer != nil {
                        self?.routeTimer.invalidate()
                    }
                    self?.routeTimer = Timer.scheduledTimer(timeInterval: 0.5, target: self, selector:
#selector(self?.checkRoute), userInfo: nil, repeats: true)
                }        }      }   }
```

Programski kod 3.25. Funkcija koja postavlja sve objekte u prikaz proširene stvarnosti

Funkcija *refreshARObjects* uklanja sve cestovne objekte postavljene u prikaz proširene stvarnosti. Ukoliko ruta ima manje od 200 koraka postavljaju se svi objekti koji predstavljaju korake u prikaz proširene stvarnosti. Ukoliko ima više od 200 koraka, postavlja se prvih 200 objekata. Pomoću *timera* "routeTimer" i funkcije *checkRoute* dodaju se ostali koraci.

```
func refreshARObjects() {
        if let routeManager = routeManager {
            if routeManager.route.steps.count > 200 {
                for object in sceneLocationView.findNodes(tagged: "roadMarker") {
```

```
            sceneLocationView.removeLocationNode(locationNode: object)
        }
        var i = 0
        for (index, step) in routeManager.route.steps.enumerated() {
            if i == 200 {
                print(routeManager.route.steps.count)
                return
            } else {
                var nodeLocation = CLLocation()
                if let altitude = step.altitude {
                    nodeLocation = CLLocation(coordinate: step.coordinates, altitude: altitude)
                } else if let currentAltitude = LocationManager.shared.getPosition().Altitude {
                    nodeLocation = CLLocation(coordinate: step.coordinates, altitude:
currentAltitude - 1)
                } else {
                    nodeLocation = CLLocation(coordinate: step.coordinates, altitude: 0)
                }
                addNodeToScene(destinationLoc: nodeLocation, tag: "roadMarker")
                routeManager.route.steps.remove(at: 0)
                i += 1
            }
        }
    } else {
        for step in routeManager.route.steps {
            var nodeLocation = CLLocation()
            if let altitude = step.altitude {
                nodeLocation = CLLocation(coordinate: step.coordinates, altitude: altitude)
            } else if let currentAltitude = LocationManager.shared.getPosition().Altitude {
                nodeLocation = CLLocation(coordinate: step.coordinates, altitude: currentAltitude
- 1)
            } else {
                nodeLocation = CLLocation(coordinate: step.coordinates, altitude: 0)
            }
            addNodeToScene(destinationLoc: nodeLocation, tag: "roadMarker")
            routeManager.route.steps.remove(at: 0)
    }       }    }   }
```

Programski kod 3.26. Funkcija za postavljanje objekata koji predstavljaju korake u prikaz proširene stvarnosti

Funkcija *addNodeToScene* prima objekt koji predstavlja lokaciju u stvarnom svijetu i na to mjesto dodaje objekt proširene stvarnosti. Funkcija postavlja objekt koji ima oblik cilindra, promjera 4 metra, visine 20 centimetara, plave boje i 20% prozirnosti.

```
func addNodeToScene(destinationLoc: CLLocation, tag: String) {
    let node = LocationNode(location: destinationLoc)
    node.tag = tag
    node.geometry = SCNCylinder(radius: 3, height: 0.2
        node.geometry?.firstMaterial?.diffuse.contents = UIColor.blue
```

```
        node.isHidden = true
        node.opacity = 0.8
    sceneLocationView.addLocationNodeWithConfirmedLocation(locationNode: node)
}
```

Programski kod 3.27. Funkcija za postavljanje pojedinog koraka u prikaz proširene stvarnosti

3.3.9. Provjera brzine i promjena boje objekata

Nakon što se otvori prikaz proširene stvarnosti postavlja se *timer* koji svake sekunde poziva funkciju *changeColorToSpeed* koja provjerava trenutnu brzinu kretanja korisnika i uspoređuje ju s dozvoljenom brzinom kretanja na toj lokaciji. Ukoliko je brzina kretanja manja od dozvoljene objekti proširene stvarnosti se postavljaju na plavu boju. Ukoliko se korisnik kreće brzinom između 100% i 110% dozvoljene brzine kretanja, boja objekata proširene stvarnosti se postavlja na narančastu, a ukoliko se korisnik kreće brzinom većom od 110% dozvoljene brzine boja objekata se postavlja na crvenu boju.

```
speedTimer = Timer.scheduledTimer(timeInterval: 1, target: self, selector: #selector(changeColorToSpeed), userInfo: nil, repeats: true)
```

Programski kod 3.28. Postavljanje timera za provjeru brzine

```
@objc func changeColorToSpeed() {
    if let location = LocationManager.shared.getLastLocation() {
        var currSpeed = location.speed
        let coordinates = location.coordinate
        print(currSpeed)
        if currSpeed >= 0 {
            currSpeed = currSpeed / 3.6
            speedManager.getLocationID(location: coordinates) {
                osm_id in
                self.speedManager.getSpeedLimit(osmID: osm_id) {
                    speed in
                    let nodes = self.sceneLocationView.findNodes(tagged: "roadMarker")
                    let maxSpeed = Double(speed) * 1.1
                    for node in nodes {
                        if node == nodes.last {
                            return
                        }
                        if (currSpeed > Double(speed)) && (currSpeed < maxSpeed) {
                            DispatchQueue.main.async {
                                node.geometry?.firstMaterial?.diffuse.contents = UIColor.orange
                            }
                        } else if currSpeed > Double(maxSpeed) {
                            DispatchQueue.main.async {
```

```
                    node.geometry?.firstMaterial?.diffuse.contents = UIColor.red
                }
            } else if currSpeed < Double(speed) {
                DispatchQueue.main.async {
                    node.geometry?.firstMaterial?.diffuse.contents = UIColor.blue
                }                }                }                }                }        }       }
}
```

Programski kod 3.29. Funkcija za promjenu boje objekata u odnosu na brzinu

Za određivanje dozvoljene brzine na korisnikovoj lokaciji koristi se klasa SpeedManager. Klasa SpeedManager se sastoji od 3 funkcije. Prva funkcija je *getLocationID* koja prima zemljopisne koordinate i pomoću klase Alamofire šalje zahtjev na OpenStreetMap poslužitelj koji vraća podatke o lokaciji. Očitava se OSM ID vrijednost lokacije.

```
func getLocationID(location: CLLocationCoordinate2D, onCompletion: @escaping (Int) -> Void)
   {

Alamofire.request("https://nominatim.openstreetmap.org/reverse?format=json&lat=\(location.lati
tude)&lon=\(location.longitude)&zoom=17&addressdetails=0").responseJSON
      { response in
         switch response.result {

         case .success(let data):
            let response = JSON(data)
            let osmID = response["osm_id"].intValue
            print("OSM_ID: \(osmID)")
            onCompletion(osmID)

         case .failure(let error):
            print(error)
            onCompletion(1)
         }      }   }
```

Programski kod 3.30. Funkcija koja vraća OSM ID za određenu lokaciju


```
{
 "place_id": "89735240",
 "licence": "Data © OpenStreetMap contributors, ODbL 1.0. https://osm.org/copyright",
 "osm_type": "way",
 "osm_id": "56261616",
 "lat": "45.3403396343524",
 "lon": "18.4209950004021",
 "display_name": "D7, Dračice, Kuševac, Osijek-Baranja County, 31400, Croatia",
 "boundingbox": [
  "45.3124837",
  "45.3437895",
```

```
    "18.3785974",
    "18.4211007"
  ]
}
```

Programski kod 3.31. Primjer odgovora poslužitelja

Druga funkcija klase SpeedManager je *getSpeedLimit* koja na temelju OSM ID vrijednosti šalje zahtjev na OpenStreetMap poslužitelj, te poslužitelj vraća informacije o lokaciji, među kojima su informacije o vrsti ceste i moguća informacija o dozvoljenoj brzini. Ukoliko postoji informacija o dozvoljenoj brzini, funkcija parsira i vraća dozvoljenu brzinu. Ukoliko ne postoji informacija o dozvoljenoj brzini funkcija parsira vrijednost o vrsti ceste te poziva funkciju *speedForType* koja na temelju vrste ceste vraća vrijednost o dozvoljenoj brzini. Ukoliko se ne može parsirati niti vrijednost o vrsti ceste, funkcija vraća dozvoljenu brzinu 50 km/h. [70]

```
func getSpeedLimit(osmID: Int, onCompletion: @escaping (Int) -> Void)
  {
    Alamofire.request("https://www.openstreetmap.org/api/0.6/way/\(osmID)").responseData
    { [weak self] response in
      if let data = response.data
      {
        let xml = SWXMLHash.parse(data)
        if let speed = try? xml["osm"]["way"]["tag"].withAttribute("k",
"maxspeed").element?.attribute(by: "v")?.text
        {
          print(speed!)
          onCompletion(Int(speed!)!)
        }
        else if let roadType = try? xml["osm"]["way"]["tag"].withAttribute("k",
"highway").element?.attribute(by: "v")?.text
        {
          print(roadType)
          let speed = self?.speedForType(type: roadType!)
          print(speed!)
          onCompletion(speed!)
        }
        else
        {
          print("default speed")
          onCompletion(50)
        }      }    }  }

  func speedForType(type: String) -> Int
  {
    switch type {
    case("highway"): // autocesta
      return 130
```

```
        case("trunk"): // brza cesta
            return 110
        case("primary"): // glavna cesta
            return 90
        case("pedestrian"): // pješačka zona
            return 0
        case("living_street"): // zona smirenog prometa
            return 10
        default:
            return 50
        }
    }
```

Programski kod 3.32. Funkcije za određivanje dozvoljene brzine kretanja

```
<osm version="0.6" generator="CGImap 0.6.1 (23300 thorn-03.openstreetmap.org)" copyright="OpenStreetMap and contributors"attribution="http://www.openstreetmap.org/copyright" license="http://opendatacommons.org/licenses/odbl/1-0/">
<way id="56261616" visible="true" version="10" changeset="42189857" timestamp="2016-09-15T21:59:31Z" user="ediyes" uid="1240849">
<nd ref="296138712"/>
...
<nd ref="163002676"/>
<tag k="highway" v="primary"/>
<tag k="lanes" v="2"/>
<tag k="maxspeed" v="90"/>
<tag k="ref" v="D7"/>
<tag k="source" v="survey"/>
<tag k="surface" v="asphalt"/>
</way>
</osm>
```

Programski kod 3.33. Primjer odgovora poslužitelja

3.3.10. Provjera udaljenosti od idućeg skretanja i ažuriranje objekata

Nakon što se postave svi objekti u prikaz proširene stvarnosti, postavlja se *timer* koji poziva funkciju *checkSteps* svakih 500 milisekundi (programski kod 3.25.).

Funkcija *checkSteps* provjerava udaljenost korisnika od idućeg skretanja. Ukoliko se korisnik nalazi unutar 20 metara od idućeg skretanja, uklanja se trenutna uputa za skretanje i postavlja se iduća uputa za skretanje (3D objekt strelice). Također, ispisuje se iduća uputa na ekran.

```
@objc func checkSteps() {
    DispatchQueue.global(qos: .background).async {
        if self.routeSteps.count > 0 {
            self.addARInstruction()
```

```
            if let distance = LocationManager.shared.getDistanceTo(locationCoords:
(self.routeSteps.first?.polyline.coordinate)!) {
                if distance < 20 {
                    self.passedSteps.append(self.routeSteps.first!)
                    self.routeSteps.remove(at: 0)
                }           }
            DispatchQueue.main.async {
                self.GPSLoc.text = self.routeSteps.first?.instructions
            }       }   }   }
```

Programski kod 3.34. Funkcija za provjeru udaljenosti korisnika od idućeg koraka

3.3.11. Provjera objekata rute

Ukoliko ruta ima više od 200 koraka, zbog performansi aplikacije postavlja se samo prvih 200 koraka u prikaz proširene stvarnosti, nakon toga se svakih 500 milisekundi poziva funkcija *checkRoute* koja provjerava udaljenost između korisnika i prvog idućeg koraka koji nije prikazan. Ukoliko je udaljenost manja od 25 metara, postavljaju se idućih 200 (ili manje, ukoliko nema 200 koraka) u prikaz proširene stvarnosti pomoću funkcije *refreshARObjects* (programski kod 3.26.).

```
@objc func checkRoute() {
    if (routeManager?.route.steps.count)! > 0 {
        if let distance = LocationManager.shared.getDistanceTo(locationCoords:
(self.routeManager?.route.steps.first?.coordinates)!) {
            if distance < 25 {
                print(distance)
                refreshARObjects()
            }       }   }   }
```

Programski kod 3.35. Funkcija za provjeru objekata rute

3.3.12. Provjera visine i vidljivosti objekata

Funkcija *session(didUpdate frame)* se poziva svaki puta kada se promjeni kadar u prikazu proširene stvarnosti. Provjerava jesu li postavljene visine koraka rute, te ukoliko nisu postavlja sve objekte rute u prikazu proširene stvarnosti na visinu jedan metar ispod pogleda kamere. Ukoliko jesu, postavlja visinu jedan metar ispod pogleda kamere samo onim objektima koji se nalaze unutar 2 metra visine od kamere. Također provjerava udaljenost objekata od korisnika, te ukoliko je objekt udaljen manje od 50 metara od korisnika postavlja objekt vidljivim, inače je objekt korisniku nevidljiv, odnosno sakriven. Udaljenost između korisnika i objekta se provjerava pomoću funkcije *distanceToNode* klase ARViewControllerModel. Funkciji se predaje trenutna pozicija kamere i pozicija objekta unutar prikaza proširene stvarnosti.

```
func session(_ session: ARSession, didUpdate frame: ARFrame) {
    if let camPosition = session.currentFrame?.camera.transform.columns.3 {
        if let altitudesAreNil = routeManager?.areAltitudesNil {
            if altitudesAreNil {
                for node in sceneLocationView.findNodes(tagged: "roadMarker") {
                    let position = SCNVector3(camPosition.x, camPosition.y, camPosition.z)
                    let distance = viewModel.distanceToNode(from: position, to: node.presentation.worldPosition)
                    if distance < 50 {
                        node.isHidden = false
                    } else {
                        node.isHidden = true
                    }
                    node.position.y = camPosition.y - 1
                }
                for node in sceneLocationView.findNodes(tagged: "instructionNode") {
                    node.position.y = camPosition.y + 1
                }
                for node in sceneLocationView.findNodes(tagged: "lastNode") {
                    node.position.y = camPosition.y + 2
                }
            } else {
                for node in sceneLocationView.findNodes(tagged: "roadMarker") {
                    if abs(node.position.y - camPosition.y) < 2 {
                        node.position.y = camPosition.y - 1
                    }
                    let position = SCNVector3(camPosition.x, camPosition.y, camPosition.z)
                    let distance = viewModel.distanceToNode(from: position, to: node.presentation.worldPosition)
                    if distance < 50 {
                        node.isHidden = false
                    } else {
                        node.isHidden = true
                    }
                }
            }
        }
    }
```

Programski kod 3.36. Funkcija koja postavlja visinu objekata i vidljivost objekata

```
func distanceToNode(from node1: SCNVector3, to node2: SCNVector3)->Float{
    let distance = SCNVector3(node1.x - node2.x, node1.y - node2.y, node1.z - node2.z)
    return sqrtf(distance.x * distance.x + distance.y * distance.y + distance.z * distance.z)
}
```

Programski kod 3.37. Fukcija koja vraća udaljenost između dvije vektorski određene pozicije

3.3.13. Resetiranje i rotacija objekata

Nakon što su postavljeni objekti u prikaz proširene stvarnosti, korisnik može resetirati prikaz, te rotirati objekte u prikazu proširene stvarnosti. Korištenjem gumbova sa strelicama lijevo i desno objekti se rotiraju za 1 stupanj u pozitivnom, odnosno negativnom smjeru. Rotacija se postiže pozivanjem funkcija *moveSceneHeadingAntiClockwise* za rotaciju u pozitivnom smjeru, odnosno funkcije *moveSceneHeadingClockwise* za rotaciju u negativnom smjeru. Te dvije funkcije mijenjaju Eulerov kut y osi ishodišta koordinatnog sustava proširene stvarnosti.

```
@IBAction func changeEulerLeft(_ sender: Any) {
    sceneLocationView.moveSceneHeadingAntiClockwise() }

  @IBAction func changeEulerRight(_ sender: Any) {
    sceneLocationView.moveSceneHeadingClockwise() }
```

Programski kod 3.38. Funkcije za rotaciju objekata proširene stvarnosti

Korisnik pritiskom na gumb "Reset" poziva funkciju *restartSession* koja uklanja sve objekte proširene stvarnosti, određuje novu rutu, postavlja prikaz pomoćne karte i postavlja objekte proširene stvarnosti. Također, funkcija zaustavlja *timere* "stepTimer" i "routeTimer". Funkcija *restartSession* se poziva i nakon učitavanja prikaza proširene stvarnosti.

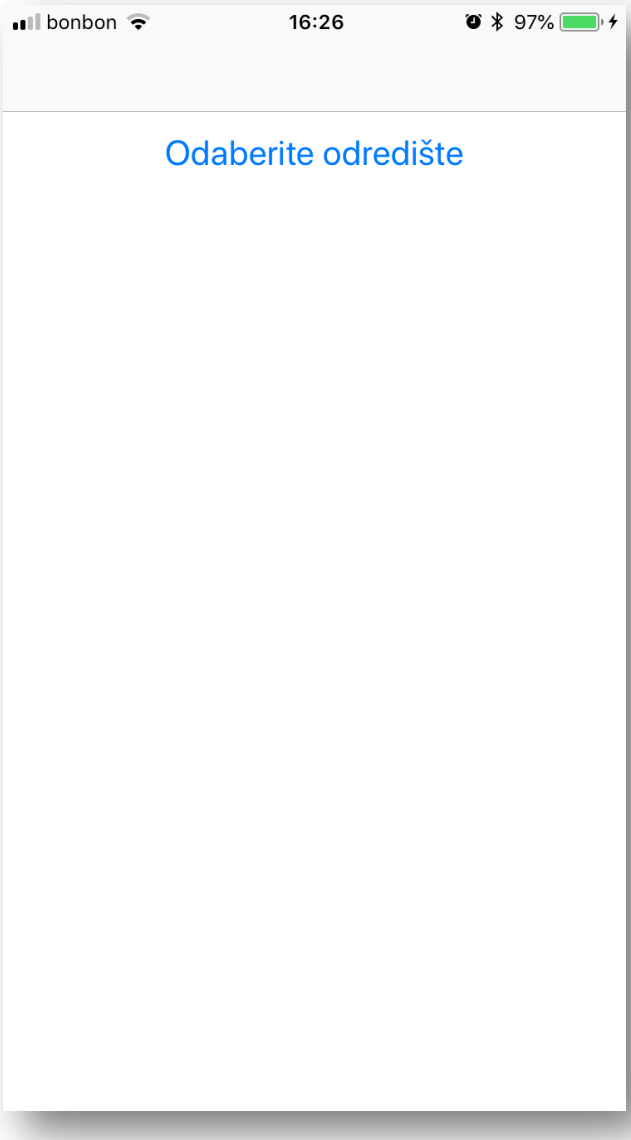
```
func restartSession() {
    startTime = DispatchTime.now()
    resetBtn.isEnabled = false
    GPSLoc.text = NSLocalizedString("loading", comment: "Loading...")
    if stepTimer != nil {
        stepTimer.invalidate()
    }
    if routeTimer != nil {
        routeTimer.invalidate()
    }
    sceneLocationView.session.pause()
    for node in sceneLocationView.findNodes(tagged: "roadMarker") {
        sceneLocationView.removeLocationNode(locationNode: node)
    }
    for node in sceneLocationView.findNodes(tagged: "lastNode") {
        sceneLocationView.removeLocationNode(locationNode: node)
    }
    sceneLocationView.run()
    viewModel.getRoute(destination: destinationCoordinates) { [weak self] route in
        self?.routeSteps = route.steps
        self?.routeManager = RouteManager(forRoute: route)
        self?.addARInstruction()
        self?.setMap(route: route)
        self?.addAllNodesToScene()
```

```
    }
}
```

Programski kod 3.39. Funkcija za resetiranje prikaza

3.3.14. Korištenje aplikacije

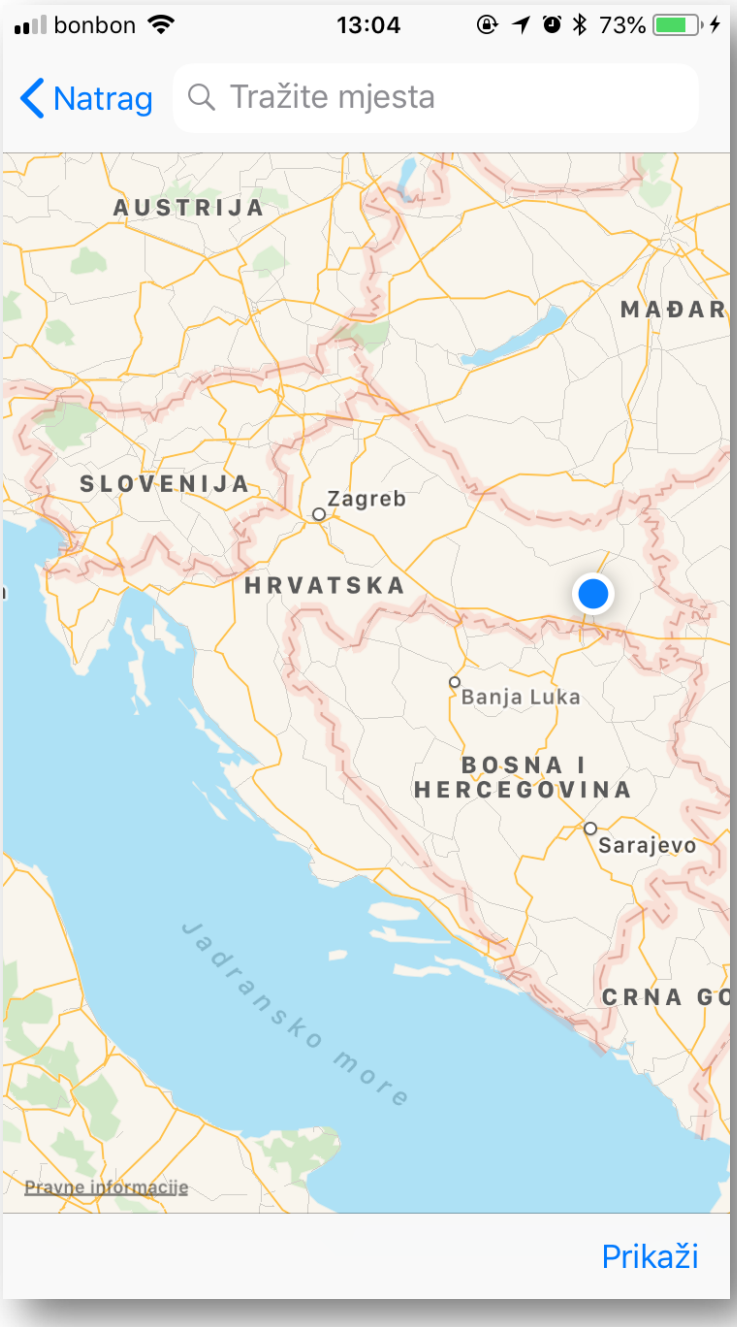
Korisnik nakon otvaranja aplikacije odabire gumb "Odaberi lokaciju"



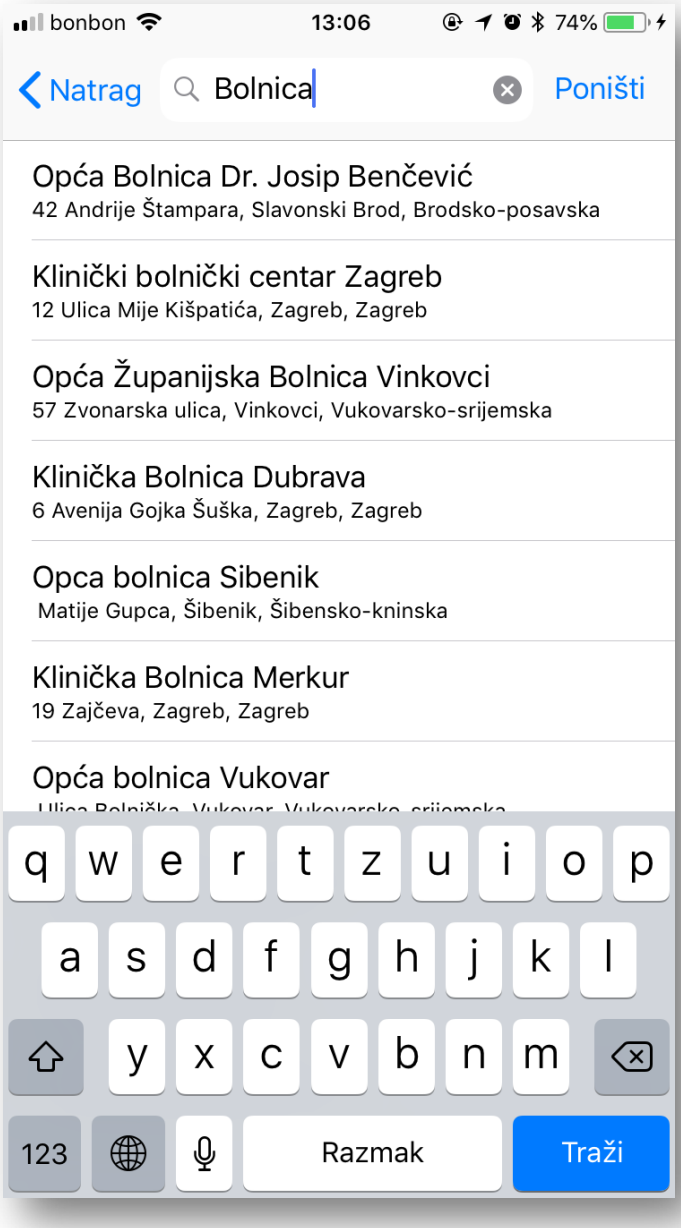
Sl. 3.12. Početni ekran

Nakon toga se otvara prikaz koji sadrži kartu i tražilicu. Korisnik može odabrati lokaciju dugim pritiskom(0.5 sekundi) na kartu ili korištenjem tražilice. Ukoliko pritisne na tražilicu može upisati naziv ili adresu lokacije. Kako korisnik upisuje tekst, tako mu se prikazuju rezultati pretrage.

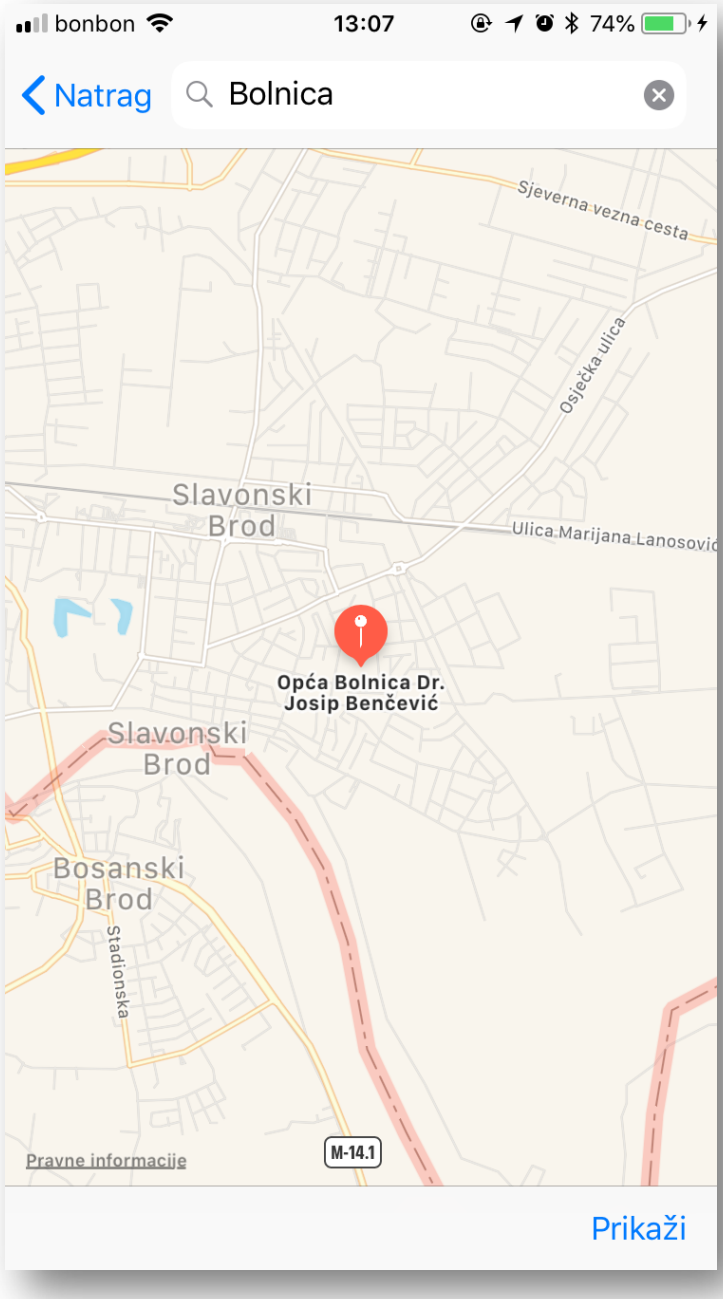
Korisnik odabirom jednog od rezultata odabire lokaciju odredišta koje mu se prikazuje na karti. Pritiskom na gumb "Prikaži" otvara se prikaz proširene stvarnosti.



Sl. 3.13. Prikaz karte i korisnikove lokacije



Sl. 3.14. Prikaz tražilice



Sl. 3.15. Prikaz odabranog odredišta na karti

Nakon što je odabrana odredišna lokacija, korisnik pritiskom na gumb "Prikaži" otvara prikaz proširene stvarnosti. Prikaz proširene stvarnosti sadrži objekte koji korisnika vode do odredišta, opis idućeg koraka, gumbove za rotaciju virtualnih objekata i gumb za resetiranje virtualnih objekata.

Sl. 3.16. Prikaz proširene stvarnosti

Sl. 3.17. Prikaz posljednje upute u obliku strelice

4. ISPITIVANJE FUNKCIONALNOSTI APLIKACIJE

4.1. Ispitivanje funkcionalnosti

Testiranje je provedeno na uređaju iPhone 6s.

4.1.1. Brzina učitavanja objekata

Tab. 4.1. Rezultati testiranja učitavanja lokacija i opterećenja sustava

Br.	Naziv odredišta	Broj koraka	Vrijeme učitavanja	Opterećenje procesora	Zauzeće memorije	Udaljenost rute
1.	Gimnazija Đakovo	49	3,73758775 s	43%	128,4 MB	478 m
2.	Dom Zdravlja Đakovo	186	8,37387388 s	53%	130,9 MB	1495 m
3.	Satnica Đakovačka	198	8,94486346 s	51%	137,2 MB	5600 m
4.	Gorjani	321	13,6518472 s	50%	183 MB	10 000 m
5.	FERIT Trpimirova	1588	61,130227125 s	60%	160,7 MB	51 000 m

Iz tablice 4.1. se može uočiti kako s porastom broja koraka linearno se povećava vrijeme učitavanja. Opterećenje procesora nema linearni porast s porastom broja koraka. Zauzeće memorije ima porast s brojem koraka, ali iz 4. i 5. primjera se uočava da ovisi i o drugim faktorima.

4.1.2. Preciznost projekcije

Kompas koji se nalazi unutar uređaja ne očitava vrijednost azimuta sa 100% sigurnosti. Zbog toga uređaj očita krivi kut otklona prema sjeveru, a posljedica toga je nepravilna usmjerenost koordinatnog sustava (z os treba biti okrenuta prema jugu). Posljedica nepravilno usmjerenog koordinatnog sustava je krivo računanje pozicije objekata proširene stvarnosti, te činjenica da objekti ne prate pravilno putanju navigacije.

Klasa CLLocationManager pri očitavanju vrijednosti azimuta daje vrijednost o maksimalnoj mogućoj devijaciji između očitanog azimuta i stvarnog azimuta. Pri testiranju, nakon svake očitane vrijednosti azimuta ispisana je vrijednost maksimalne moguće devijacije vrijednosti azimuta. Testiranje je provedeno tako da je izmjereno vrijeme koliko je aplikacija radila, maksimalna devijacija očitanja u tom trenutku i stvarno odstupanje objekata (koliko su objekti pomaknuti u odnosu na stvarni položaj ceste).

Tab. 4.2. Testiranje odstupanja vrijednosti kompasa

Pokušaj broj	Proteklo vrijeme (u sekundama)	Maksimalna moguća devijacija (u stupnjevima)	Stvarno odstupanje (u stupnjevima)
1	37	15	10
2	28	15	10
3	15	15	3
4	21	15	13
5	49	15	2

Sl. 4.1. Prikaz odstupanja položaja objekata od stvarnosti

Kako bi se smanjila vidljivost odstupanja objekata u odnosu na stvarne položaje, jer može zbuniti osobu koja upravlja vozilom, smanjena je vidljivost objekata tako da se prikazuju samo cestovni objekti unutar 50 metara (slika 4.3.).

Maksimalna devijacija koja se javlja prilikom pokretanja aplikacije je 35 stupnjeva, međutim kao što je vidljivo iz tablice 4.2. maksimalna devijacija se s vremenom smanji na 15 stupnjeva.

Sl. 4.2. Pogreška u očitanju vrijednosti azimuta s kompasa

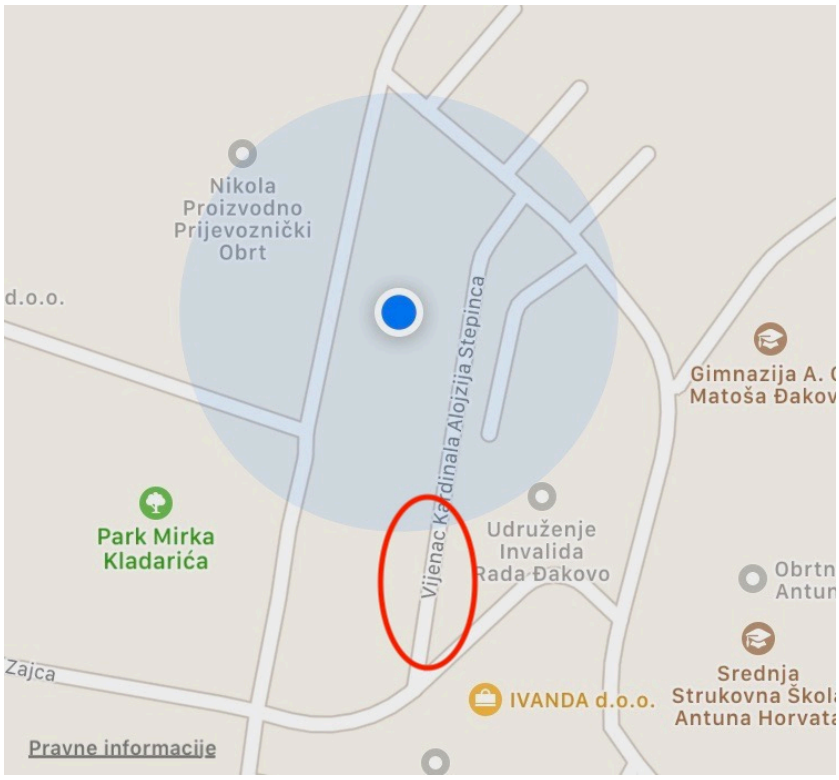
Sl. 4.3. Novi prikaz gdje se prikazuju samo cestovni objekti unutar 50 metara

4.2. Nedostatci sustava

4.2.1. Nedostatci Apple Maps zemljopisnih karata

Nedostatak koji se javlja su pogreške koje se nalaze u Apple Maps kartama u obliku nepostojećih ulica. Zbog toga aplikacija pokušava usmjeriti korisnika kroz ulice koje ne postoje ili kroz ulice koje su slijepe(dok je na kartama pokazano da nisu slijepe).

Moguće rješenje je korištenje usluge Google Maps zemljopisnih karti koje su ispravne.



Slika 4.3. Dio ceste označen crvenom elipsom ne postoji

4.2.2. Nepreciznost lociranja uređaja

Točnost lociranja korisnika na otvorenom mjestu je nekoliko metara(oko 4 metra). Iako je odabrana opcija za najtočnije moguće lociranje uređaja, ukoliko se korisnik nalazi unutar zgrade ili kuće točnost lociranja može biti do 150 metara. Zbog toga se može dogoditi da prilikom pokretanja aplikacije, ukoliko postoji druga cesta u blizini ceste na kojoj se korisnik nalazi, korisnik dobije prikazanu rutu na cesti kojom se ne kreće. [54] Biblioteka ARCL djelomično rješava problem nepreciznosti lociranja na način da uspoređuje pomake korisnika u proširenoj stvarnosti i očitanja lokacije uređaja, te pomoću toga kompenzira nepreciznost samog uređaja. [54]

5. ZAKLJUČAK

U ovom diplomskom radu napravljena je mobilna aplikacija koja korisniku omogućava navigaciju pomoću proširene stvarnosti. Korisnik ima mogućnost odabira lokacije na koju želi ići, a nakon toga mu se prikazuje pogled proširene stvarnosti. Aplikacija je napravljena za iOS operativni sustav koristeći Swift programski jezik i ARKit razvojni okvir koji omogućuje stvaranje aplikacija koje koriste proširenu stvarnost. Aplikacija pomoću Apple Maps usluge pronalazi najkraću rutu, te od Mapquest poslužitelja dobiva informacije o visini objekata. Nakon toga, koristeći biblioteku ARCL postavlja objekte u virtualni svijet, odnosno prikaz proširene stvarnosti. Tijekom rada aplikacije se ažurira prikaz proširene stvarnosti promjenom boje objekata, promjenom pozicije objekata, te uklanjanjem i dodavanjem pojedinih objekata kada je potrebno. Ciljevi iz uvodnog poglavlja su postignuti, premda postoji prostor za poboljšanje preciznosti prikaza rute u prikazu proširene stvarnosti, ponajviše na području određivanja točne vrijednosti azimuta. Aplikacija se može proširiti dodatnim mogućnostima kao što su autentikacija korisnika i omogućavanje korisniku spremanje vlastitih lokacija, kao i učitavanje rute do spremljenih lokacija.

LITERATURA

[1] W. L. Hosch, »Augmented reality,« [Mrežno]. Dostupno: https://www.britannica.com/technology/augmented-reality. [Pokušaj pristupa 11 lipanj 2018].

[2] D. Pescovitz, »"The Master Key": L. Frank Baum envisions augmented reality glasses in 1901,« Mote & Beam, 10 Listopad 2012. [Mrežno]. Dostupno: https://web.archive.org/web/20130522153011/thttp://moteandbeam.net/the-master-key-l-frank-baum-envisions-ar-glasses-in-1901. [Pokušaj pristupa 11 Lipanj 2018].

[3] Liberator Crew, »Gun Sights,« [Mrežno]. Dostupno: http://liberatorcrew.com/15_Gunnery/08_sights.htm. [Pokušaj pristupa 01 Rujan 2018].

[4] I. White, The History of Air Intercept Radar and the British Nightfighter: 1935-1959, Casemate Publishers, 2007.

[5] BAE Systems, »The evolution of the Head-Up Display,« BAE Systems, [Mrežno]. Dostupno: https://www.baesystems.com/en/feature/our-innovations-hud#. [Pokušaj pristupa 01 Rujan 2018].

[6] Federal Government of the United States, *14th Code Of Federal Regulations, Part 91,* Office Of Federal Regulations, 2018.

[7] Airline Ratings, »What is a HUD and how does it work?,« AirlineRatings, 07 Veljača 2017. [Mrežno]. Dostupno: https://www.airlineratings.com/did-you-know/what-is-a-hud-and-how-does-it-work/. [Pokušaj pristupa 01 Rujan 2018].

[8] C. Neiger, »How Head-up Displays Work,« HowStuffWorks, [Mrežno]. Dostupno: https://auto.howstuffworks.com/car-driving-safety/safety-regulatory-devices/head-up-display2.htm. [Pokušaj pristupa 01 Rujan 2018].

[9] Joschiki, »File:AIII in 1000ft.JPG,« Wikimedia Project, 27 Ožujak 2007. [Mrežno]. Dostupno: https://commons.wikimedia.org/wiki/File:AIII_in_1000ft.JPG. [Pokušaj pristupa 01 Rujan 2018].

[10] Affemitwaffe, »File:E60hud.JPG,« Wikimedia Project, 24 Veljača 2008. [Mrežno]. Dostupno: https://commons.wikimedia.org/wiki/File:E60hud.JPG. [Pokušaj pristupa 01 Rujan 2018].

[11] Technopedia, »Head-Mounted Display (HMD),« Technopedia Inc., [Mrežno]. Dostupno: https://www.techopedia.com/definition/2342/head-mounted-display-hmd. [Pokušaj pristupa 01 Rujan 2018].

[12] RF Cafe, »RF Cafe,« 28 Travanj 2014. [Mrežno]. Dostupno: http://www.rfcafe.com/references/popular-electronics/third-eye-space-explorers-july-1962-popular-electronics.htm. [Pokušaj pristupa 01 Rujan 2018].

[13] M. M. Bayer, C. E. Rash i J. H. Brindle, Helmet-Mounted Displays: Sensation, Perception and Cognition Issues, Fort Rucker, Alabama: U.S. Army Aeromedical Research Laboratory, 2009.

[14] Spectron Engineering Inc, »Joint Helmet Mounted Cueing System (JHMCS),« Spectron Engineering Inc, [Mrežno]. Dostupno: http://www.spectronengineering.com/03_Solutions%20for%20Specific%20Aircraft%20Displays%20(HMD).htm. [Pokušaj pristupa 01 Rujan 2018].

[15] T. Hollerer i D. Schmalstieg, »Introduction to Augmented Reality,« Pearson Education, Informit, 10 Lipanj 2016. [Mrežno]. Dostupno:

http://www.informit.com/articles/article.aspx?p=2516729&seqNum=2. [Pokušaj pristupa 11 Lipanj 2018].

[16] D. Pape, »Homework 2,« Dave Pape, [Mrežno]. Dostupno: http://resumbrae.com/ub/dms423_f08/06/. [Pokušaj pristupa 24 lipanj 2018].

[17] I. E. Sutherland, »A head-mounted three dimensional display,« The University Of Utah, Salt Lake City, Utah, 1968.

[18] R. Metz, »Augmented Reality Gets to Work,« MIT Technology Review, 24 Veljača 2014. [Mrežno]. Dostupno: https://www.technologyreview.com/s/524626/augmented-reality-gets-to-work/. [Pokušaj pristupa 11 Lipanj 2018].

[19] L. G. R. G. T. L. M. D. S. D. W. Clemens Arth, »The History of Mobile Augmented Reality,« Inst. for Computer Graphics and VisionGraz University of Technology, Austria, Graz, 2015.

[20] OyundariZorigtbaatar, »File:Augmented-reality.jpg,« 20 ožujak 2016. [Mrežno]. Dostupno: https://commons.wikimedia.org/wiki/File:Augmented-reality.jpg. [Pokušaj pristupa 24 lipanj 2018].

[21] R. Amadeo, »Google Tango review: Promising Google tech debuts on crappy Lenovo hardware,« Conde Nast Inc., 26 Prosinac 2016. [Mrežno]. Dostupno: https://arstechnica.com/gadgets/2016/12/google-tango-review-promising-google-tech-debuts-on-crappy-lenovo-hardware/. [Pokušaj pristupa 12 Lipanj 2018].

[22] A. Piltch, »Intel RealSense 3D: What It Is and What You Do With It,« Purch Group, Inc, 14 Siječanj 2015. [Mrežno]. Dostupno: https://www.tomsguide.com/us/intel-realsense-guide,news-20286.html. [Pokušaj pristupa 12 Lipanj 2018].

[23] Google Inc, »Fundamental Concepts,« Google Inc, 8 Svibanj 2018. [Mrežno]. Dostupno: https://developers.google.com/ar/discover/concepts. [Pokušaj pristupa 12 Lipanj 2018].

[24] Apple Inc., »Understanding World Tracking in ARKit,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/arkit/understanding_world_tracking_in_arkit. [Pokušaj pristupa 19 lipanj 2018].

[25] Apple Inc., »Creating a Multiuser AR Experience,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/arkit/creating_a_multiuser_ar_experience. [Pokušaj pristupa 23 lipanj 2018].

[26] Apple Inc., »SceneKit,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/scenekit. [Pokušaj pristupa 25 lipanj 2018].

[27] Apple Inc., »SpriteKit,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/spritekit. [Pokušaj pristupa 25 lipanj 2018].

[28] Apple Inc., »Apple Developer,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/arkit/arconfiguration.worldalignment. [Pokušaj pristupa 28 05 2018].

[29] Apple Inc., »SCNNode,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/scenekit/scnnode. [Pokušaj pristupa 25 lipanj 2018].

[30] Apple Inc., »position,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/scenekit/scnnode/1408026-position. [Pokušaj pristupa 25 lipanj 2018].

[31] Apple Inc., »SCNVector4,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/scenekit/scnvector4. [Pokušaj pristupa 25 lipanj 2018].

[32] M. Azam, »ARKit - Geometry, materials, nodes gestures Oh my!,« A Medium Corporation, 7 listopad 2017. [Mrežno]. Dostupno: https://medium.com/@azamsharp/arkit-geometry-materials-nodes-gestures-oh-my-c4bc04e8aadf. [Pokušaj pristupa 25 lipanj 2018].

[33] Apple Inc., »ARAnchor,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/arkit/aranchor. [Pokušaj pristupa 25 lipanj 2018].

[34] Apple Inc., »Providing 3D Virtual Content with SceneKit,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/arkit/arscnview/providing_3d_virtual_content_with_scenekit. [Pokušaj pristupa 25 lipanj 2018].

[35] C. Lattner, »Chris Lattner's Homepage,« [Mrežno]. Dostupno: http://nondot.org/sabre/. [Pokušaj pristupa 20 lipanj 2018].

[36] Apple Inc., »About Swift,« Apple Inc., [Mrežno]. Dostupno: https://swift.org/about/. [Pokušaj pristupa 20 lipanj 2018].

[37] Apple Inc., »Automatic Reference Counting,« Apple Inc., [Mrežno]. Dostupno: https://docs.swift.org/swift-book/LanguageGuide/AutomaticReferenceCounting.html. [Pokušaj pristupa 20 lipanj 2018].

[38] Apple Inc., »About Swift,« Apple Inc., [Mrežno]. Dostupno: https://docs.swift.org/swift-book/. [Pokušaj pristupa 20 lipanj 2018].

[39] Apple Inc., »The Basics,« Apple Inc., [Mrežno]. Dostupno: https://docs.swift.org/swift-book/LanguageGuide/TheBasics.html#//apple_ref/doc/uid/TP40014097-CH5-ID330. [Pokušaj pristupa 20 lipanj 2018].

[40] TIOBE Software BV, »TIOBE Index for June 2018,« TIOBE Software BV, [Mrežno]. Dostupno: https://www.tiobe.com/tiobe-index/. [Pokušaj pristupa 20 lipanj 2018].

[41] Apple Inc., »Cocoa Application Layer,« Apple Inc., 16 rujan 2015. [Mrežno]. Dostupno: https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/OSX_Technology_Overview/CocoaApplicationLayer/CocoaApplicationLayer.html. [Pokušaj pristupa 20 lipanj 2018].

[42] Apple Inc., »Cocoa (Touch),« Apple Inc., 06 travanj 2018. [Mrežno]. Dostupno: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/Cocoa.html#//apple_ref/doc/uid/TP40008195-CH9-SW1. [Pokušaj pristupa 20 lipanj 2018].

[43] Apple Inc., »Foundation,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/foundation?changes=_5#overview. [Pokušaj pristupa 20 lipanj 2018].

[44] Apple Inc., »UIKit,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/UIKit. [Pokušaj pristupa 20 lipanj 2018].

[45] Apple Inc., »MapKit,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/mapkit. [Pokušaj pristupa 20 lipanj 2018].

[46] Apple Inc., »Core Location,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/corelocation. [Pokušaj pristupa 20 lipanj 2018].

[47] Apple Inc., »About App Development with UIKit,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/uikit/about_app_development_with_uikit. [Pokušaj pristupa 23 lipanj 2018].

[48] Apple Inc., »UIApplication,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/uikit/uiapplication. [Pokušaj pristupa 20 lipanj 2018].

[49] Apple Inc., »UIApplicationDelegate,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/uikit/uiapplicationdelegate?hl=et. [Pokušaj pristupa 23 lipanj 2018].

[50] Apple Inc., »UIViewController,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/uikit/uiviewcontroller. [Pokušaj pristupa 23 lipanj 2018].

[51] Apple Inc., »UIWindow,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/uikit/uiwindow?changes=_2. [Pokušaj pristupa 23 lipanj 2018].

[52] Alamofire, »Alamofire,« Github, [Mrežno]. Dostupno: https://github.com/Alamofire/Alamofire. [Pokušaj pristupa 25 lipanj 2018].

[53] SwiftyJSON, »SwiftyJSON,« Github, [Mrežno]. Dostupno: https://github.com/SwiftyJSON/SwiftyJSON. [Pokušaj pristupa 25 lipanj 2018].

[54] A. Hart, »ARKit-CoreLocation,« Github, 31 svibanj 2017. [Mrežno]. Dostupno: https://github.com/ProjectDent/ARKit-CoreLocation. [Pokušaj pristupa 25 lipanj 2018].

[55] Apple Inc., »Apple Developer,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/corelocation/kcllocationaccuracybestfornavigation. [Pokušaj pristupa 01 06 2018].

[56] C. Veness, »Movable Type Scripts,« Movable Type Ltd, [Mrežno]. Dostupno: https://www.movable-type.co.uk/scripts/latlong.html. [Pokušaj pristupa 05 06 2018].

[57] spk578, »Distance on a sphere: The Haversine Formula,« 5 listopad 2017. [Mrežno]. Dostupno: https://community.esri.com/groups/coordinate-reference-systems/blog/2017/10/05/haversine-formula. [Pokušaj pristupa 24 lipanj 2018].

[58] Mapquest, »Open Elevation API,« Mapquest, [Mrežno]. Dostupno: https://developer.mapquest.com/documentation/open/elevation-api/elevation-profile/get/. [Pokušaj pristupa 24 lipanj 2018].

[59] Apple Inc., »Core Animation Basics,« Apple Inc., 09 ožujak 2015. [Mrežno]. Dostupno: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/CoreAnimation_guide/CoreAnimationBasics/CoreAnimationBasics.html#//apple_ref/doc/uid/TP40004514-CH2-SW18. [Pokušaj pristupa 25 lipanj 2018].

[60] Apple Inc., »transform,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/scenekit/scnnode/1407964-transform. [Pokušaj pristupa 25 lipanj 2018].

[61] Apple Inc., »eulerAngles,« Apple Inc., [Mrežno]. Dostupno: https://developer.apple.com/documentation/scenekit/scnnode/1407980-eulerangles. [Pokušaj pristupa 25 lipanj 2018].

[62] CHRobotics LLC., »Understanding Euler Angles,« CHRobotics LLC., [Mrežno]. Dostupno: http://www.chrobotics.com/library/understanding-euler-angles. [Pokušaj pristupa 25 lipanj 2018].

[63] D. Pavleković, »ArVehNav,« GitHub, Inc., 02 Rujan 2018. [Mrežno]. Dostupno: https://github.com/dpavlek/ARVehNav.git. [Pokušaj pristupa 03 Rujan 2018].

[64] D. Pavleković, »AddEventViewController, Tifosi,« Github, 23 kolovoz 2017. [Mrežno]. Dostupno:

https://github.com/dpavlek/Tifosi/blob/master/Tifosi/AddEventViewController.swift. [Pokušaj pristupa 25 lipanj 2018].

[65] R. Chen, »iOS Tutorial: How to search for location and display results using Apple's MapKit,« ThornTechnologies, 12 Siječanj 2016. [Mrežno]. Dostupno: https://www.thorntech.com/2016/01/how-to-search-for-location-using-apples-mapkit/. [Pokušaj pristupa 24 Kolovoz 2017].

[66] S. Mishali, »MKPolyline+Ext.swift,« Github, 08. Srpanj 2018.. [Mrežno]. Dostupno: https://gist.github.com/freak4pc/98c813d8adb8feb8aee3a11d2da1373f. [Pokušaj pristupa 28. Kolovoz 2018.].

[67] Max, »How to calculate the points between two given points and given distance?,« Stack Exchange Inc, 21 Siječanj 2014. [Mrežno]. Dostupno: https://stackoverflow.com/questions/21249739/how-to-calculate-the-points-between-two-given-points-and-given-distance. [Pokušaj pristupa 28 Kolovoz 2018].

[68] A. Knopper, »Draw Route with MapKit Tutorial,« Squarespace, 29 Veljača 2016. [Mrežno]. Dostupno: https://www.ioscreator.com/tutorials/draw-route-mapkit-tutorial. [Pokušaj pristupa 01 Rujan 2018].

[69] Cedric, »Angle between vectors, positive or negative,« gamedev.net, 23 veljača 2003. [Mrežno]. Dostupno: https://www.gamedev.net/forums/topic/141307-angle-between-vectors-positive-or-negative/. [Pokušaj pristupa 24 kolovoz 2018].

[70] A. Gudibande, »DriveSafe,« GitHub, 4 kolovoz 2016. [Mrežno]. Dostupno: https://github.com/sfhacks/DriveSafe/blob/master/Drive/SpeedLimitFinder.swift. [Pokušaj pristupa 27 kolovoz 2018].

[71] V. Muravev, »SceneKit : how to get distance between two SCNNode ? (ObjC and Swift),« Stack Exchange Inc., 07 Rujan 2017. [Mrežno]. Dostupno: https://stackoverflow.com/questions/46081137/scenekit-how-to-get-distance-between-two-scnnode-objc-and-swift. [Pokušaj pristupa 01 Rujan 2018].

[72] Apple Inc., »Singleton,« Apple Inc., 06 travanj 2018. [Mrežno]. Dostupno: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/Singleton.html#//apple_ref/doc/uid/TP40008195-CH49. [Pokušaj pristupa 20 lipanj 2018].

[73] GeekyAnts, »Introduction to Firebase,« Medium Company, 28 prosinac 2017. [Mrežno]. Dostupno: https://hackernoon.com/introduction-to-firebase-218a23186cd7. [Pokušaj pristupa 25 lipanj 2018].

[74] D. Pavleković, »LoginViewModel, LoginRegister,« Github, 29 kolovoz 2017. [Mrežno]. Dostupno: https://github.com/dpavlek/LoginRegisterSwift/blob/master/LoginRegister/LoginViewModel.swift. [Pokušaj pristupa 25 lipanj 2018].

SAŽETAK

Naslov: PRIMJENA PROŠIRENE STVARNOSTI U NAVIGACIJI VOZILA

U ovom diplomskom radu napravljena je iOS aplikacija koja omogućuje korisniku navigaciju vozila uz pomoć proširene stvarnosti. Aplikacija prikazuje rutu kojom se korisnik treba kretati kako bi došao do svog odredišta pomoću 3D objekata u prikazu proširene stvarnosti. Korisnik odabere odredište na karti, te nakon toga otvori prikaz proširene stvarnosti gdje može vidjeti 3D reprezentaciju njegove rute. Korisnik ima opciju ručnog podešavanja usmjerenja rute (lijevo ili desno) pomoću gumbova. U prvom poglavlju dan je opis zadatka, što je potrebno i koja je motivacija za izradu ovog rada. U drugom poglavlju je opisana povijest proširene stvarnosti i razvojni okviri za Android i iOS mobilne uređaje. U trećem poglavlju su opisane tehnologije koje se koriste u ovom radu, a najvažnije su iOS operativni sustav, Swift programski jezik, UIKit, MapKit i Core Location razvojni okviri, biblioteka ARCL, te pomoćne biblioteke. Zatim je opisan način postavljanja objekata u prikaz proširene stvarnosti pomoću ARKit razvojnog okvira. Nakon toga je opisana izvedba prethodno spomenute iOS aplikacije. U četvrtom poglavlju je opisano testiranje mogućnosti aplikacije i nedostatci. Naposljetku, u zaključku je ukratko opisana napravljena aplikacija, te mogućnosti poboljšanja.

Ključne riječi: iOS, navigacija, proširena stvarnost, ARKit, Swift

ABSTRACT

Title: VEHICLE NAVIGATION USING AUGMENTED REALITY

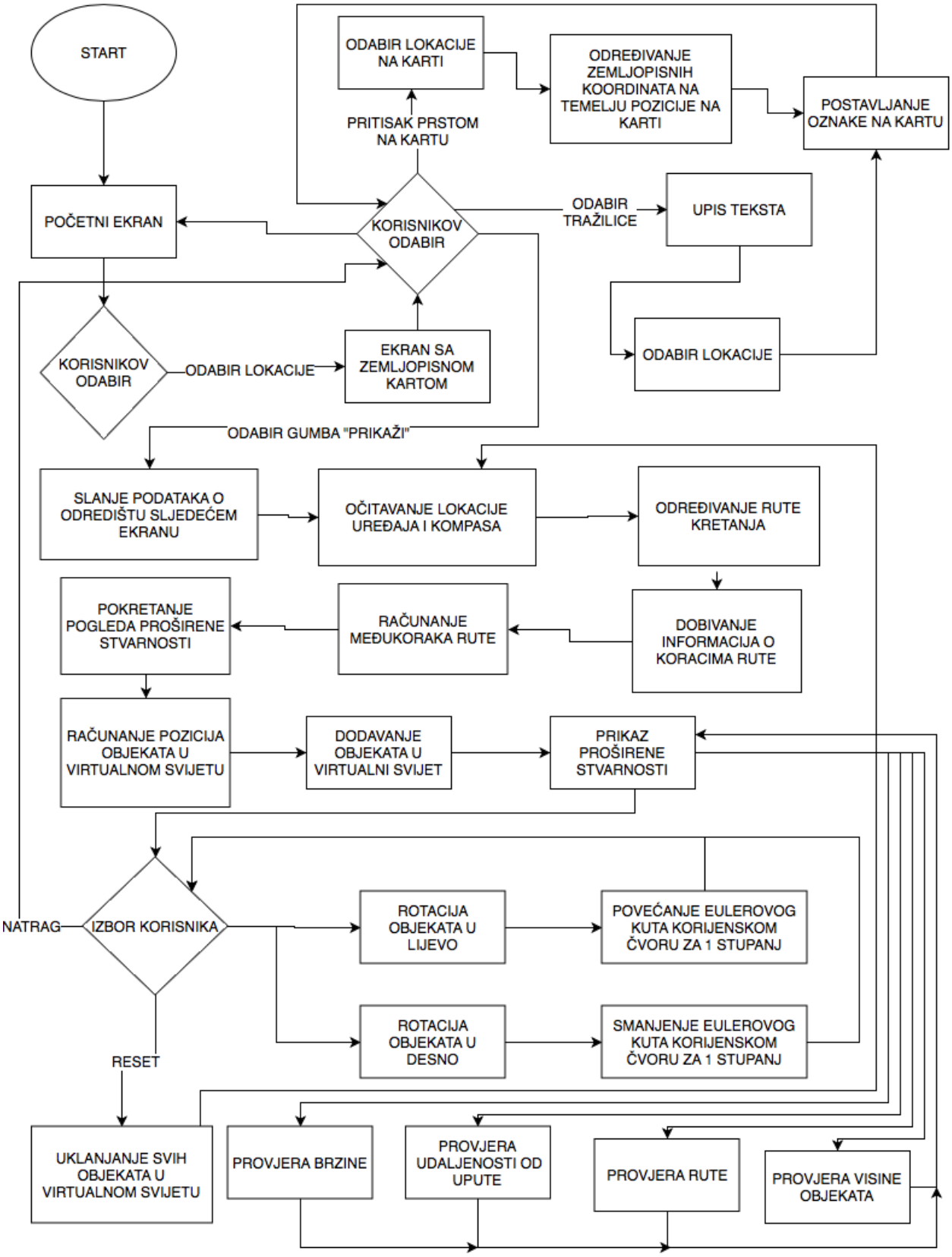
The main goal of this paper was making an iOS application which enables the user to navigate the vehicle with the help of augmented reality. Application shows the route which user needs to navigate in the form of 3D objects which are placed in the augmented reality scene. User chooses the destination on the map and then opens the augmented reality view which shows the representation of his route using 3D objects. User has the option of rotating the placed objects (left or right) using the on-screen buttons as well as resetting the view. First chapter gives the description of the goals, what is necessary to do and what is the motivation for this paper. Second chapter contains the history of augmented reality as well as Android and iOS native frameworks for creating augmented reality applications. Third chapter contains the description of technologies used in the making of this paper. Most important are iOS operating system, UIKit, MapKit and CoreLocation frameworks, ARCL library, as well as the additional libraries. Further, it describes the placing and transforming the 3D objects in augmented reality scene. It also contains the description of the workings of the aforementioned iOS application. Fourth chapter describes testing and limitations of the application. At last, the conclusion gives the description of the application as well as possible improvements.

Keywords: iOS, navigation, augmented reality, ARKit, Swift

ŽIVOTOPIS

Daniel Pavleković rođen je u Đakovu, 31. kolovoza 1994. godine. Pohađao je Osnovnu školu Vladimira Nazora u Đakovu. Već u trećem razredu dolazi u doticaj s programiranjem u obliku programskog jezika LOGO. Redovno sudjeluje u natjecanjima iz informatike u osnovnoj i srednjoj školi. Nakon završetka osnovne škole upisuje Gimnaziju Antuna Gustava Matoša u Đakovu, prirodoslovno-matematički smjer. U srednjoj školi se upoznaje sa C programskim jezikom. Nakon srednje škole upisuje preddiplomski studij elektrotehnike na Elektrotehničkom fakultet Osijek, ali nakon prve godine se prebacuje na preddiplomski smjer računarstva na istom fakultetu. 2016. godine diplomira na istom fakultetu (koji je sada promijenio naziv u Fakultet Elektrotehnike, Računarstva i Informacijskih Tehnologija) i stječe titulu prvostupnika inženjera računarstva. Iste godine upisuje diplomski studij računarstva na Fakultetu Elektrotehnike, Računarstva i Informacijskih Tehnologija u Osijeku. U srpnju i kolovozu 2017. godine odrađuje stručnu praksu u tvrtki Cobe u Osijeku, gdje se upoznaje s programskim jezikom Swift i razvojem aplikacija za iOS operativni sustav.

PRILOG A. Dijagram toka aplikacije



Sl. PA.1. Dijagram toka aplikacije