

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 533

**Android aplikacija za planiranje misije
ronioca**

Zagreb, lipanj 2013

Sadržaj

Uvod.....	1
Funkcionalni zahtjevi	2
Slanje i primanje poruka.....	2
Interesne točke i generacija puta za pretraživanje	3
Primanje podataka sa senzora.....	4
Korištene tehnologije.....	6
Android operacijski sustav.....	7
Razvoj za Android platformu	9
Komponente aplikacije.....	9
Aktivnosti	10
Prihvatanje emisije	11
Bluetooth tehnologija.....	12
Implementacija	12
Komunikacija i spajanje.....	13
Google Maps	14
Google Maps Android API v2	14
Programsko ostvarenje	16
Razvojna okolina.....	16
Razvojni uređaj	17
Korisničko sučelje	18
Glavna aktivnost	19
Mapa.....	20
Komunikacija	22
Izranjanje	24
AndroidManifest.xml.....	25
Komunikacija.....	27
BluetoothChatService.java.....	27
ConnectThread	29
ConnectedThread	31
DeviceListActivity	33
Parse	34
Predefinirane poruke.....	36
Fotografija.....	38
Mapa	38

Dodavanje i brisanje oznaka.....	38
Generacija puta za pretraživanje	39
Testiranje i pokretanje aplikacije	42
Zaključak	45
Sažetak	46
Abstract	47
Literatura	48

Tablica slika

Slika 1 Format poruka	3
Slika 2 Primjer rada algoritma za generaciju staze za pretraživanje	4
Slika 3 Android operacijski sustav	8
Slika 4 Životni ciklus aktivnosti	11
Slika 5 Logo Bluetooth specifikacije	12
Slika 6 Google maps iz preglednika	14
Slika 7 Ikona Diver Assistant aplikacije	16
Slika 8 Docklight.....	16
Slika 9 Android Developer Tols razvojno okruženje	17
Slika 10 Napredne postavke za razvoj	18
Slika 11 Hijerarhija pogleda.....	19
Slika 13 Glavna aktivnost	19
Slika 12 NonSwipeableViewPager	20
Slika 14 Kartica sa mapom.....	21
Slika 15 Fragment mapa	21
Slika 16 Kartica za komunikaciju	22
Slika 17 Spajanje sa uređajima	23
Slika 18 Fragment komunikacija.....	23
Slika 19 Izranjanje	24
Slika 20 Isječak koda korisničkog sučelja fragmenta za izranjanje.....	25
Slika 21 Potrebna dopuštenja.....	26
Slika 22 Postavke aplikacije	27
Slika 23 Connect metoda	28
Slika 24 Connected metoda	28
Slika 25 Stop metoda	29
Slika 26 Konstruktor dretve za uspostavljanje konekcije	29
Slika 27 DeviceType razred.....	30
Slika 28 Run metoda razreda ConnectThread	31
Slika 29 Cancel metoda razreda ConnectThread.....	31
Slika 30 ConnectedThread	32
Slika 31 run metoda	32
Slika 32 write metoda	33
Slika 33 BroadcastReceiver	33
Slika 34 Odabir uređaja.....	34
Slika 35 Parse konstruktor	34
Slika 36 Run metoda razreda Parse.....	35
Slika 37 new_pos metoda Parse razreda	35
Slika 38 DefaultMessage razred.....	36
Slika 39 DefaultMessageList	37
Slika 40 Slanje predefinirane poruke	37
Slika 41 takePictureHandler	38
Slika 42 Dodavanje oznaka na mapu	38
Slika 43 Brisanje oznake	39

Slika 44 drawPath metoda	40
Slika 45 Generirani poligon	41
Slika 46 clearAll metoda	41
Slika 47 Ikona aplikacije Diver Assistant	42
Slika 48 Kartica za komunikaciju	42
Slika 49 Kartica za planiranje i pregled misije	43
Slika 50 Poslane i primljene poruke u alatu Docklight	44
Slika 51 Poslane i primljene poruke u kartici za komunikaciju	44

Uvod

Ronioci djeluju u okruženju gdje su izloženi opasnostima vezanim uz nedostatak navigacijskih sposobnosti. Korištenje naprednih tehnologija poput podvodnih tableta može olakšati ove aktivnosti i učiniti ih sigurnijima. U sklopu diplomskog rada potrebno je implementirati tri funkcionalnosti Android aplikacije: 1) prethodno razvijen algoritam za planiranje područja pretraživanja korištenjem Google Maps-a implementirati kao Android aplikaciju i prilagoditi ga za ronioce; 2) imajući na umu da će tablet komunicirati s više vanjskih jedinica putem Bluetootha, potrebno je istražiti mogućnosti spajanja i čitanja podataka s više Bluetooth uređaja istovremeno; 3) razviti sučelje s predefiniranim skupom tekstualnih poruka koje ronilac može odabrati i poslati Bluetooth vezom na vanjski uređaj.

Prilikom implementacije potrebnih značajki potrebno je uzeti u obzir korisnički doživljaj, odnosno okolnosti u kojima će ronilac koristiti tablet računalo. Sam tablet nalaziti će se u oklopljenom kućištu i koristiti će se uz pomoć indukcijske olovke. Također treba glavne elemente sučelja učiniti lako dostupnima i vidljivima. Potrebno je omogućiti spajanje više senzora preko Bluetooth protokola i omogućiti oporavak podataka od pogreške zbog same podvodne komunikacije, sa posebnim naglaskom kod prijenosa poruka visoke važnosti.

Funkcionalni zahtjevi

Za ostvarenje zadane aplikacije potrebno je ostvariti nekoliko korisničkih slučajeva odnosno zahtjeva:

- Slanje i primanje poruka
- Dodavanje interesnih točaka i generacija puta za pretraživanje
- Primanje podataka sa senzora

Svaki korisnički slučaj potrebno je prilagoditi i uvjetima u kojima će aplikacija biti korištena. Svaki element korisničkog sučelja mora biti odgovarajuće veličine i lako dostupan. Osim toga tok korisničkih slučajeva mora imati malo interakcije sa samim korisnikom, kako bi brže obavljao radnje i treba omogućiti automatsko prepoznavanje situacije kada je ronilac u nevolji i slati poruke upozorenja. Kako se sva komunikacija odvija ispod vode, potrebno je i to uzeti u obzir prilikom odabira tehnologije razmjena informacija te i njenoj implementaciji. Kod razmjene poruka potrebno je definirati i sam protokol odnosno format same poruke pri čemu treba voditi računa o prethodno navedenim uvjetima.

Slanje i primanje poruka

Osim tekstualnih poruka, protokol treba prilagoditi i za ostale tipova poruka koji su interesantne korisniku. Kako bi se korisniku omogućilo da sa što manje interakcije komunicira s površinom, protokol mora sadržavati predefinirane poruke kao što su: Da, Ne, SOS i slično, kako bi jednim klikom mogao poslati takvu poruku. Kako je riječ o podvodnom pretraživanju potrebno je ostvariti i razmjenu fotografija sa površinom te interesnih točaka, odnosno poligona koji se treba pretraživati. Format poruke treba podržati sljedeće tipove:

- Ručno unesena tekstualna poruka
- Predefinirana tekstualna poruka
- Fotografija
- KML format
- Pozicija i dubina

Kako je sama komunikacija ograničena, potrebno je tako definirati protokol (Slika 1) da se šalje što manja količina poruka, tako da se u jednu poruku može grupirati više informacija. Komunikacija ispod vode je dosta ograničena i treba uzeti u obzir

0	position init
1	position
2	position+msg
3	position+img
4	position+default
5	position+msg+def
6	position+img+def
7	position+kml
8	position + CHK
9	position+msg + CHK
10	position+msg + CHK
11	position+default + CHK
12	position+msg+def + CHK
13	position+img+def + CHK

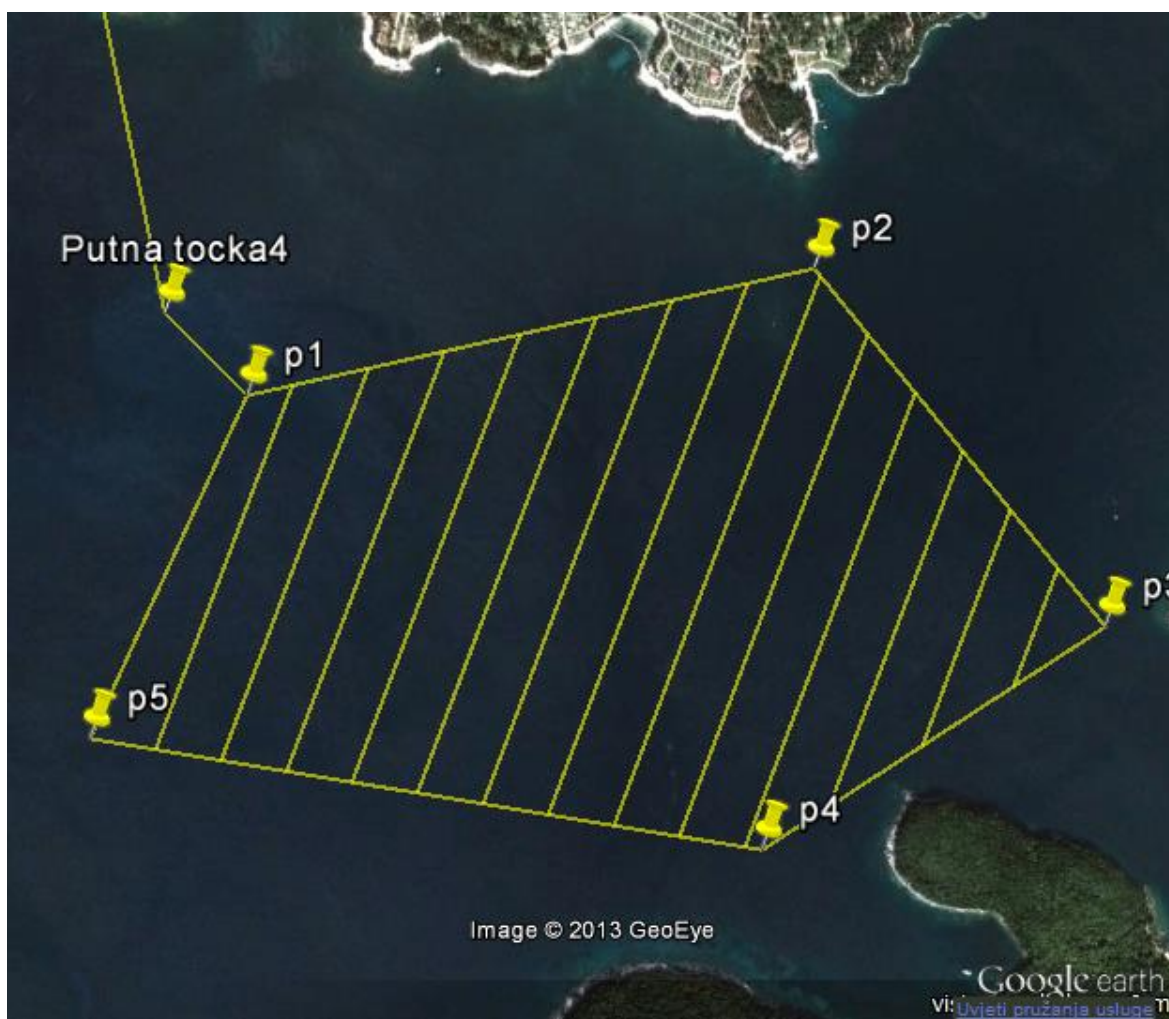
Slika 1 Format poruka

moгуćnost gubitka poruka, tako da je za bitnije poruke potrebno ostvariti i algoritam za provjeru uspješnosti komunikacije te ponovno slanje poruka koje nisu primljene. Također treba i ograničiti samu duljinu poruke kako bi vjerojatnost pogreške prilikom razmjene bila što manja. Kod fotografija nije potreban prijenos fotografije visoke kvalitete, fotografija se prilikom slanja smanjuje i lokalno na uređaj se sprema originalna veličina za naknadno pregledavanje.

Interesne točke i generacija puta za pretraživanje

Kako je jedna od zadaća aplikacije planiranje misije [7], potrebno je integrirati geografsku kartu koja će sadržavati interesne točke i poligone za pretraživanje zadanog područja [6]. Korisnik aplikacije mora imati mogućnosti dodavanja i brisanja točaka, informaciju o svojoj trenutnoj lokaciji i mogućnost generacije staze

za pretraživanje na zadanom poligonu. Generirani poligon je prikazan na Slici 2.



Slika 2 Primjer rada algoritma za generaciju staze za pretraživanje

Unos i brisanje točaka mora biti jednostavno i precizno, također sama interakcija sa kartom treba biti logična i treba koristiti principe interakcije koji su otprije poznati korisniku. Primanje lokacije i poligona treba biti transparentno za korisnika, i staza za pretraživanje treba se generirati na samom uređaju kako bi se smanjio broj potrebnih točaka odnosno podataka koji se šalju na uređaj. Sam algoritam je prethodno razvijen.

Primanje podataka sa senzora

Kako bi ronilac imao informacije o svojoj okolini, potrebne su mu informacije sa senzora. Kako je bežična komunikacija ispod vode ograničena, potrebno je prilagoditi protokol za razmjenu informacija sa senzora. Sama tehnologija za bežičnu komunikaciju može diktirati i ograničiti broj senzora s kojih je moguće primiti informacije. Također podaci moraju biti prikazani korisniku u jasnom

formatu. Osnovni senzori koji moraju biti dostupni za rad aplikacije su dubinomjer i GPS uređaj. Dubinomjer se koristi kod plana izranjanja, a GPS za prikaz trenutnu lokacije ronioca.

Korištene tehnologije

Za ispunjavanje svih zahtjeva potrebno je odabrati tehnologiju u kojoj je moguće sve to izvesti. Zahtjevi koji su uzimani u obzir prilikom odabira platforme su otvorenost, skalabilnost i dobra podrška za razvoj te podrška za sve funkcionalne zahtjeve koje su navedeni. Kao platforma za razvoj odabran je Android operacijski sustav za mobilne uređaje. Nakon odabranog operacijskog sustava potrebno je odabrati i ostale tehnologije, za prijenos i prikaz podataka. Kod bežičnog prijenosa podataka izbor pada na Bluetooth. Za prikaz točaka i pozicije korisnika korišteno je rješenje Google Maps zbog dobre podrške na samom Android operacijskom sustavu.

Android operacijski sustav

Android [1] je prvi otvoreni operacijski sustav za mobilne uređaje pokrenut od strane Google Inc. i vođen od strane Open Handset Alliance grupe. To je grupa koja danas broji preko 80 tehnoloških kompanija među kojima se nalaze T-Mobile, HTC, Intel, Motorola, Qualcomm i drugi. Njihov je cilj ubrzati inovacije na području mobilnih operacijskih sustava, a samim time ponuditi krajnjim kupcima bogatije, jeftinije i bolje iskustvo korištenja. Svojstvo Androida jest da je modularan i veoma prilagodljiv pa postoje slučajevi njegovog prenošenja na razne uređaje poput čitača elektronskih knjiga, mobilne telefone, prijenosnike i multimedijske svirače.

Android Inc. su osnovali Andy Rubin, Rich Miner, Nick Sears i Chris White u listopadu 2003. godine kako bi razvijali programe za pametne mobilne uređaje koji bi uzimali u obzir korisničke postavke te njegovu lokaciju. Nakon dvije godine gotovo tajnog rada Google je odlučio kupiti Android te tada počinju spekulacije o ulasku Googlea na tržište pametnih telefona. Osnivači i ključni programeri, osnaženi Googleovim programerima, na tržište donose mobilnu platformu temeljenu na linuxovoj jezgri koja bi trebala biti potpuno prilagodljiva zahtjevima korisnika.

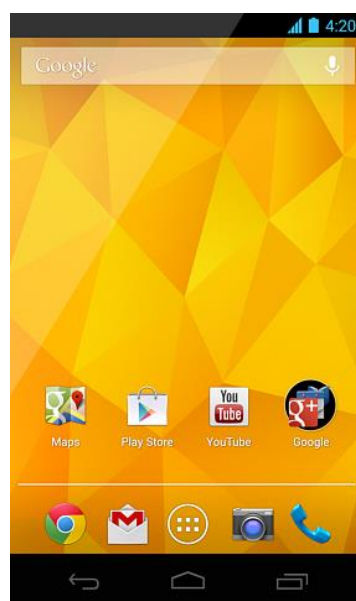
U studenome 2007. godine osnovana je Open Handset Alliance (OHA) s ciljem stvaranja javnog standarda za mobilne uređaje. Glavni inicijator i ovoga puta bio je Google koji je okupio 34 tvrtke iz različitih domena mobilne industrije poput proizvođača mobilnih telefona, programera aplikacija, mobilnih operatera i sličnih.

Istoga dana, 5. studenog 2007. godine, OHA otkriva mobilnu platformu otvorenog koda baziranu na Linux jezgri – Android. Ovo je prvo javno predstavljanje Androida kao operativnog sustava, a prvi komercijalni uređaj u koji je bio ugrađen Android OS bio je T-Mobile G1, tajvanskog proizvođača pametnih telefona HTC (poznat i pod nazivom HTC Dream). Osnivanjem ovog saveza, Android je bačen u utrku sa ostalim mobilnim platformama na tržištu: iOS (Apple), Windows Phone (Microsoft), Symbian (Nokia, Sony Ericsson), Palm (HP), Bada (Samsung).

Od samih početaka Android je zamišljen kao projekt otvorenog koda te je od 21. listopada 2008. godine dostupan cjeloviti kôd pod Apache licencom. S druge strane, proizvođačima uređaja nije dozvoljeno korištenje Android zaštićenog imena ukoliko Google ne certificira uređaj kao kompatibilan. Također, uređaji

trebaju zadovoljavati određene kriterije kako bi dobili pristup aplikacijama zatvorenog koda poput Android Market aplikacije. Ona služi za pretraživanje i instalaciju aplikacija koje su na tržištu, a ne dolaze ugrađene u sam uređaj. Jedina iznimka od politike otvorenog koda jesu verzije 3.0, 3.1 i 3.2 kodnog imena Honeycomb kako ne bi došlo do instaliranja istih na mobilni uređaj. Razlog je taj da je ta verzija namijenjena uporabi na tablet računalima. Najnovija stabilna verzija Androida je 4.2.2 kodnog imena Jelly Bean od 11.02.2013.

Android platforma je prilagođena za uporabu na uređajima sa većim zaslonima poput pametnih telefona koji koriste 2D ili 3D grafičku knjižnicu. Za pohranu podataka koristi se SQLite relacijski sustav za upravljanje bazom. Karakteristike ovog softvera jesu njegova knjižnica koja u svega 275 kB implementira većinu SQL standarda. U odnosu na druge sustave za upravljanje bazama, SQLite nije zasebni proces već je sastavni dio aplikacije koja pristupa bazi podataka. Za prijenos poruka koriste se SMS i MMS servisi sa podržanim prikazom razgovora. Android je do verzije 2.3 podržavao 26 različitih jezika, a izlaskom Gingerbreada ta brojka se udvostručuje i danas je podržano čak 57 različitih jezika. Uređaj sa instaliranim Android Froyo sustavom kao i oni noviji imaju mogućnost korištenja uređaja kao žičane (putem USB kabela) ili bežične pristupne točke za pristupanje internetu. Prije verzije 2.2 ta mogućnost je postojala samo kroz korištenje neslužbenih aplikacija ili ukoliko je proizvođač uređaja omogućio tu funkcionalnost. Na slici 3 je prikazan izgled Android operacijskog sustava.



Slika 3 Android operacijski sustav

Iako je većina aplikacija pisana u Java programskom jeziku, na Android uređajima ne postoji Java Virtual Machine pa tako nije moguće izvršavati Java byte kod. Za pokretanje Java aplikacija, Android koristi Dalvik virtualni stroj.

Razvoj za Android platformu

Android aplikacije napisane su programskom jeziku Java. Alat Android SDK prevodi kod zajedno sa ostalim podacima i resursima u Android paket. Android paket je arhiva koji ima ekstenziju .apk. Sav kod koji se nalazi apk arhivi smatra se kao jedna aplikacija i to je datoteka koje uređaji pokretani Android operacijskim sustavom koriste za instalaciju aplikacije.

Jednom kada je instalirana na uređaj, svaka aplikacija radi u zasebnom zaštićenom okruženju:

- Android operacijski sustav je višekorisnički Linux sustav u kojem je svaka aplikacija zaseban korisnik
- Po pretpostavljenim postavkama sustav dodjeljuje svakoj aplikaciji jedinstveni korisnički identifikator. Sustav postavlja dopuštenja za sve datoteke u aplikaciji kako bi im mogla samo određena aplikacija odnosno korisnik pristupiti
- Svaki proces izvodi se u zasebnom virtualnom stroju kako bi kod aplikacije radio u izolaciji od ostalih aplikacija
- Svaka aplikacija izvodi se u zasebnom procesu. Android pokreće proces kada bilo koja komponenta aplikacije zahtjeva izvršavanje. Proces se gasi kada više ne postoji potreba ili ukoliko je potrebno oslobađanje memorije za druge aplikacije

Komponente aplikacije

Komponente aplikacije [4] su osnovni dijelovi Android aplikacije. Svaka komponenta je druga točka kroz koju sustav može ući u aplikaciju. Nisu sve komponente ulazne točke za korisnike. Postoje četiri tipa aplikacijskih komponenti. Svaka ima svoju namjenu i svoj životni ciklus.

- Aktivnosti (*Activities*)
- Servisi (*Services*)
- Dostavljač sadržaja (*Content Providers*)

- Prihvaćanje emisije (*Broadcast Receivers*)

Za izradu aplikacije korištene su aktivnosti i prihvaćanje emisije.

Aktivnosti

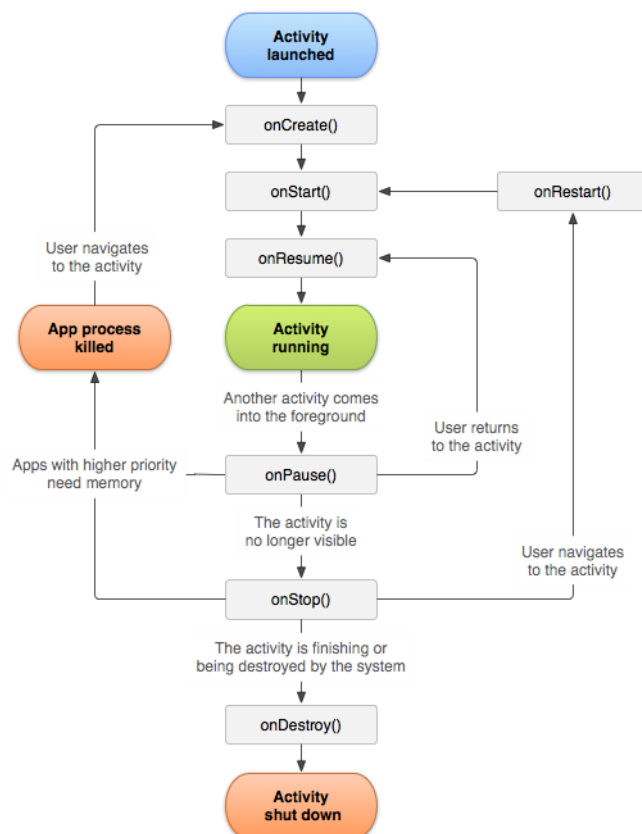
Aktivnost je aplikacijska komponenta koja sadrži ekran sa kojim korisnik ima interakciju kako bi postigao određeni cilj kao što je obavljanje poziva, fotografiranje, slanje elektroničke pošte, pregled mape i slično. Svaka aktivnost nalazi se u prozoru u kojem se iscrtava korisničko sučelje. Sam prozor obično popunjava cijeli ekran iako može biti manji.

Aplikacija se obično sastoji od više aktivnosti koje su međusobno povezane. Obično postoji aktivnost koja se specificira kao glavna. Ona se pokreće prilikom prvog pokretanja aplikacije. Nakon toga svaka aktivnost može pokretati druge aktivnosti, prilikom čega se prethodna aktivnost zaustavlja, ali se njeno stanje sačuva.

Kada se aktivnost zaustavlja zbog druge aktivnosti, ona je obavještena da je stanje promijenjeno kroz povratnu metodu životnog ciklusa aktivnosti. Postoji nekoliko metoda koje aktivnost može izvoditi zbog promijene stanja:

- Stvaranje
- Zaustavljanje
- Nastavak rada
- Uništavanje

Svaka povratna metoda nudi mogućnost izvršavanje određenog posla koji je potreban za tu povratnu metodu. Cijeli životni ciklus aktivnosti prikazan je na slici 4.



Slika 4 Životni ciklus aktivnosti

Prihvatanje emisije

Aplikacijska komponenta za prihvatanje emisije odgovara na objave na razini sustava. Osim sustava i sama aplikacija može započeti emisiju, u konkretnom slučaju emisija se radi prilikom završetka otkrivanja novih uređaja prilikom zahtjeva za spajanjem na drugi Bluetooth uređaj. Ta aplikacijska komponenta nema korisničko sučelje, ona može stvoriti obavijest u traci za obavijesti kako bi obavijestio korisnika. Najčešće se ne koristi za obavljanje puno posla, u većini slučajeva služi za pokretanje određenih servisa kako bi se izvršio neki posao.

Bluetooth tehnologija

Bluetooth [2] (Slika 5) je standard za bežični prijenos podataka na kratke udaljenosti (korištenjem kratko-valnih radio transmisija na pojasu 2400-2480 MHz) sa prijenosnih i fiksnih uređaja i za kreiranje osobnih mreža (*Personal Area Network*) sa velikom razinom sigurnosti. Prijenos se može odvijati između dva ili više uređaja.



Slika 5 Logo Bluetooth specifikacije

Standard je razvijen od strane telekom operatera Ericsson 1994. godine, početna ideja je bila razviti bežičnu alternativu za RS-232 protokol koji koristi podatkovne kabele. Moguće je i spajanje nekoliko uređaja i rješava problem sinkronizacije.

Bluetooth vodi organizacija *Bluetooth Special Interest Group*, koja ima više od 18000 članova kompanija u području telekomunikacija, računarstva, mreža i potrošačke elektronike. Bluetooth je standardiziran kao IEEE 802.15.1, ali standard se više ne održava. Grupa SIG vodi razvoj specifikacije, upravlja sa kvalifikacijskim programom i štiti patent. Kako bi se uređaj vodio kao Bluetooth uređaj, on mora podržavati standarde definirane od strane grupe SIG. Mreža patenata je također potrebna za implementaciju tehnologije i licencira se samo kvalificiranim uređajima.

Implementacija

Bluetooth radi na rasponu od 2400 do 2483.5 MHz (uključujući sigurnosni raspon). Bluetooth koristi radio tehnologiju koja se naziva *frequency-hopping spread spectrum*. Podaci koji se prenose rasporede se u pakete i svaki paket se prenosi u jednom od 79 postojećih Bluetooth kanala. Svaki kanal ima pojasnu širinu od 1 MHz. Prvi kanal je na frekvenciji od 2402 MHz, a zadnji na 2480 MHz sa razmakom od 1 MHz. Obično radi na 1600 skokova u sekundi sa uključenom adaptivnim frekvencijskim-skokom (*Adaptive Frequency-Hopping*).

Bluetooth je paketno orijentirani protokol sa *master-slave* strukturom. Jedan *master* može komunicirati sa najviše 7 *slave* uređaja u *piconet* mreži. Svi uređaji

dijele otkucaj *master* uređaja. Razmjena paketa bazira se na osnovnom otkucaju kojeg postavlja *master* koji radi na intervalima od 312.5 μ s. Dva otkucaja čine isječak od 625 μ s, a dva isječka čine par isječaka od 1250 μ s. U jednostavnom slučaju koji koristi pakete od jednog isječka, *master* prenosi u parnim isječcima, a prima u neparnim isječcima. Paketi mogu biti veličine 1, 3 ili 5 isječaka, ali u svim slučajevima *master* počinje razmjenu u parnom isječku, a *slave* počinje razmjenu u neparnom isječku.

Bluetooth omogućava siguran način za spajanje i razmjenu informacija između uređaja kao što su faksevi, mobilni telefoni, telefoni, prijenosna računala, osobna računala, GPS prijamnika. Početni dizajn je bio za tehnologiju sa niskom širinom pojasa.

Komunikacija i spajanje

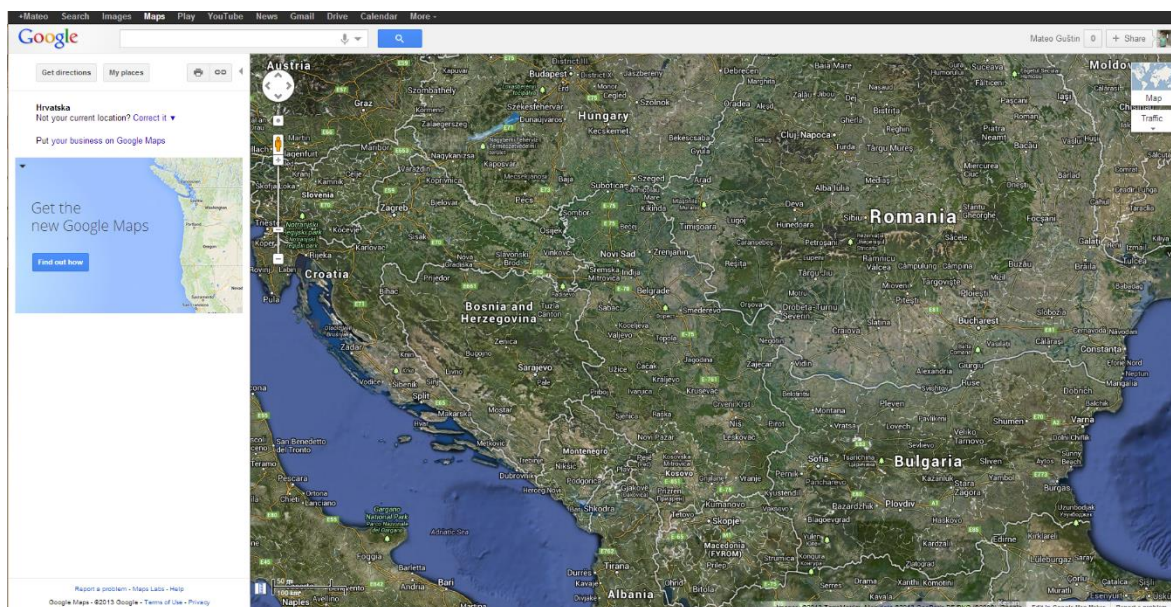
Master Bluetooth uređaj može komunicirati sa maksimalno sedam uređaja u *piconet* mreži (ad-hoc računalna mreža), iako na svim stvarnim uređajima nije moguće postići toliki broj uređaja. Uređaji mogu i razmjenjivati uloge po dogovoru, odnosno *slave* uređaj može postati *master* uređaj (koristi se kod uređaja za prijenos zvuka).

Bluetooth Core specifikacija omogućava i spajanje 2 ili više *piconet* mreže koje formiraju *scatternet* u kojoj određeni uređaji simultano imaju ulogu *master* uređaja u jednoj *piconet* mreži, a *slave* ulogu u drugoj *piconet* mreži.

Google Maps

Google Maps je usluga besplatnih digitalnih mrežnih karata, koje čine osnovu mnogih servisa i usluga, od pregledavanja satelitskih snimaka, planiranje putovanja, lociranje traženih mjesta, itd.

Dopušta jednostavnu implementaciju na različite web stranice, kombiniranje sa drugim aplikacijama, razvoj dodatka i prilagođavanje specifičnim potrebama. Zasnovana na istoj tehnologiji postoji i kao zasebna aplikacija namijenjena instaliranju i korištenju na pojedinim osobnim računalima sa vezom na internet, Google Earth, virtualni globus. Aplikacija je dostupna i iz web preglednika (Slika 6).



Slika 6 Google maps iz preglednika

Google Maps Android API v2

Sa Google Maps Android API, moguće je dodavati mape bazirane na podacima iz Google Maps usluge u Android aplikaciju. API automatski odrađuje pristup Google Maps poslužiteljima, prijenos podataka, prikaz mape i odgovore na korisničke geste na mapi. Također moguće je koristiti API za dodavanje markera, poligona i dodatnih slojeva običnoj mapi, te promjenu pogleda na mapu u određenom području. Ti objekti nude dodatne informacije za lokacije na mapi i nude interakciju korisnika sa mapom. API nudi dodavanje sljedećih grafičkih elemenata na mapu:

- Ikone postavljene na određenu poziciju na mapi (*Markers*)
- Setove linijskih segmenata (*Polylines*)

- Zatvorene segmente (*Polygons*)
- Slike u digitalnom formatu postavljene na određenu poziciju na mapi (*Ground Overlays*)
- Setovi slika prikazani na osnovnoj mapi (*Tile Overlays*)

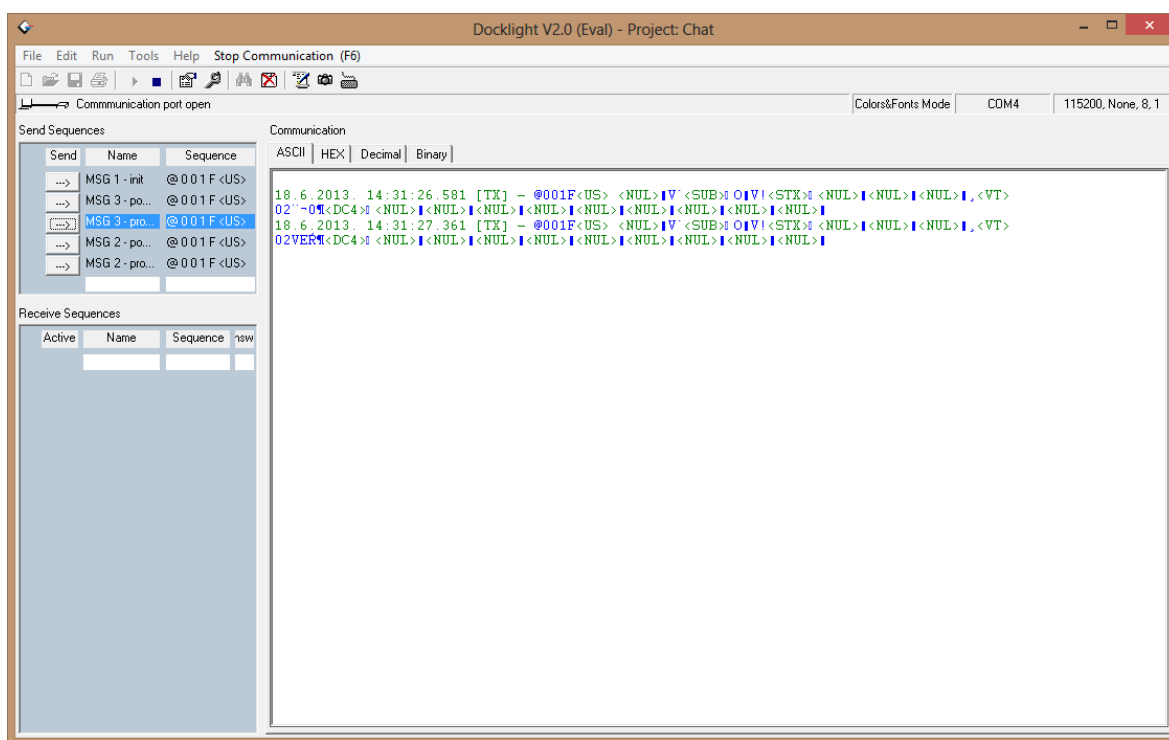
Programsko ostvarenje

Kako bi se zadovoljili svi funkcionalni zahtjevi, aplikacija je ostvarena na operacijskom sustavu za mobilne uređaje Android. Također napravljene su preinake na samom uređaju. Minimalna verzija Apija koja je korištena je 16, a ciljana 17 što je ujedno i najnovija verzija koja nudi najnovije mogućnosti za razvoj Android aplikacija. Aplikacija je nazvana Diver Assistant. Logo aplikacije je prikazan na slici 7.



Slika 7 Ikona Diver Assistant aplikacije

Za testiranje komunikacije korišten je alat Docklight [8] (Slika 8) koji omogućuje slanje i primanje podataka preko različitih komunikacijskih protokola.

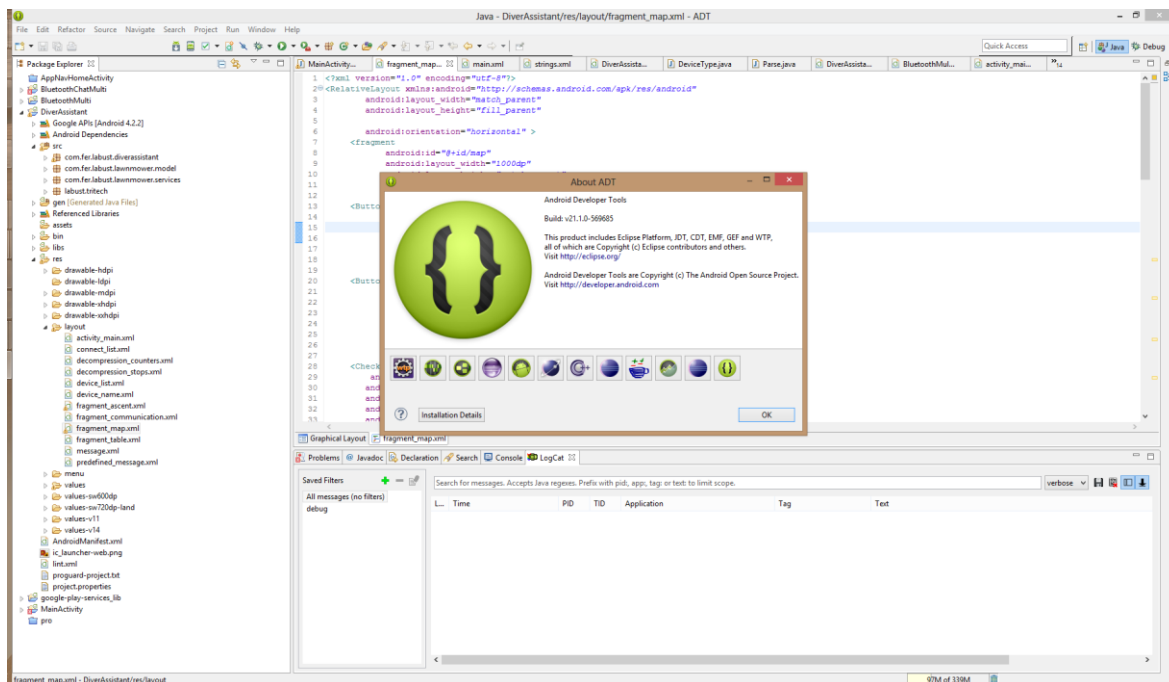


Slika 8 Docklight

Razvojna okolina

Kao razvojna okolina korišten je program Eclipse sa dodatkom Android Developer Tools (Slika 9). ADT je Java IDE sa naprednim mogućnostima koje pomažu u

izgradnji, testiranju, otklanjanju grešaka i stvaranje Android aplikacija. Njegova je prednost što je otvorenog koda i pokreće se na većini operacijskih sustava.



Slika 9 Android Developer Tools razvojno okruženje

On nudi i emulator za pokretanje Android aplikacija, koji pomaže u testiranju ukoliko se ne koristi stvarni Android uređaj.

Za preuzimanje potrebnih paketa korišten je Android SDK Manager, koji automatski preuzima potrebne pakete i dodaje ih u projekt, te brine se o nadogradnjama.

Za verzioniranje i kolaboraciju korišten je sustav Git koji nudi distribuiranu kontrolu verzije i upravljanje sa izvornim kodom. Alat koji je korišten je EGit, odnosno dodatak za Eclipse.

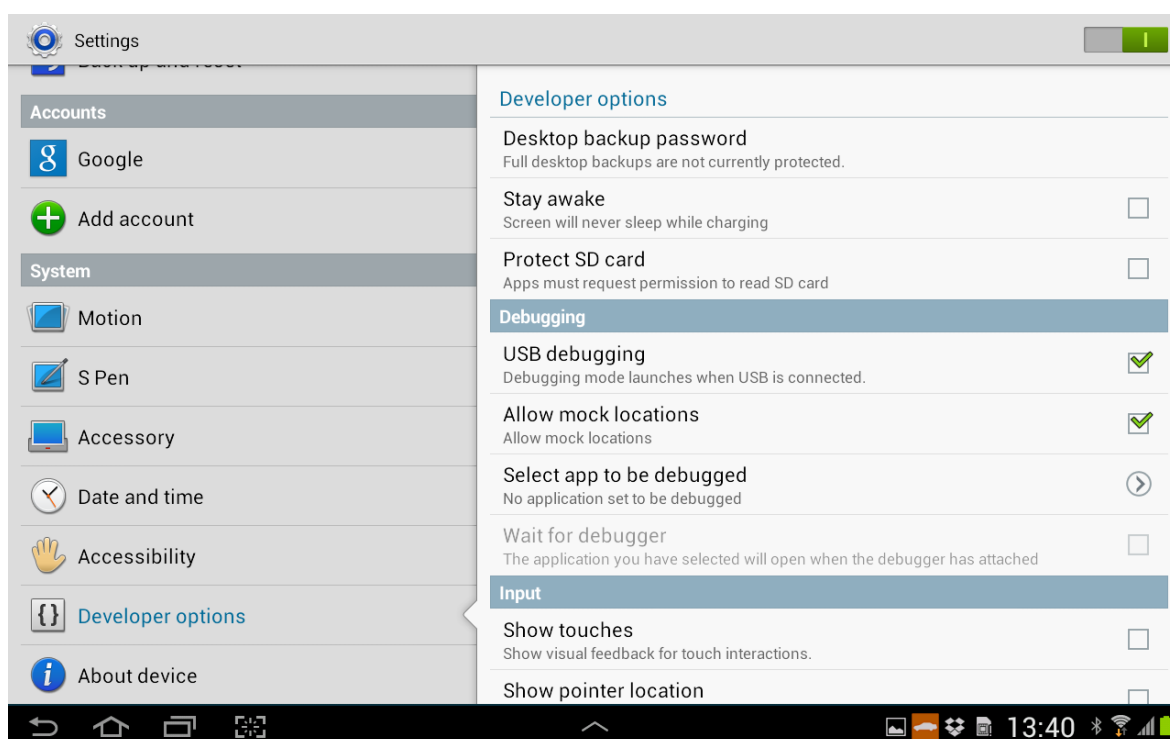
Razvojni uređaj

Za izradu aplikacije korišten je stvarni uređaj, zbog jednostavnijeg pokretanja i otklanjanja grešaka, te naprednim postavkama. Uređaj koji je korišten je Samsung Galaxy Note 10.1 zbog njegove prikladne veličine i specifikacija. Neke od bitnih specifikacija:

- Dimenzije: 262 x 180 x 8,9 mm
- Težina: 600g

- Veličina ekrana: 10,1 inch
- Procesor: Quad-core 1.4 GHz Cortex-A9
- Radna memorija: 2 GB
- Operacijski sustav: Android O, v4.1.2 (Jelly Bean)
- Senzori: Akcelerometar, žiroskop i kompas
- S Pen induktivna olovka za interakciju sa sučeljem
- Bluetooth povezivost

Sam uređaj nudi i napredne postavke za otklanjanje grešaka i testiranje koje je potrebno uključiti u postavkama (Slika 10).



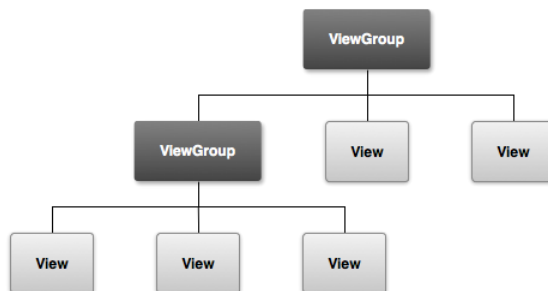
Slika 10 Napredne postavke za razvoj

Kako tablet nije vodootporan potrebno je koristiti vanjsko kućište. Vanjsko kućište napravljeno je od kaljenog stakla kako ne bi propuštalo vodu i izdržalo pritisak vode na velikim dubinama. Kako bi interakcija sa uređajem i dalje bila moguća modificirana je induktivna olovka za unos, odnosno pojačana je kako bi signal prolazio kroz 10 mm debelo staklo.

Korisničko sučelje

Svi elementi korisničkog sučelja [5] u Android aplikaciji grade se korištenjem View i ViewGroup objekata. View je objekt koji iscrtava element na ekranu s kojim

korisnik ima interakciju, ViewGroup je objekt koji sadrži druge objekte tipa View i ViewGroup (Slika 11).



Slika 11 Hijerarhija pogleda

Kod izrade korisničkog sučelja korišteni su objekti ViewPager i Fragment. ViewPager [3] je objekt koji upravlja rasporedom elemenata, slično kao i ViewGroup i služi za listanje stranicama sadržaja. ViewGroup sadrži fragmente u kojima se nalaze kartice, odnosno svaki fragment je zasebna kartica. Fragmenti predstavljaju ponašanje djela korisničkog sučelja u aktivnosti. Moguće je staviti više fragmenata u jednu aktivnost. Fragment ima kao i aktivnost svoj životni ciklus, dobiva unose korisnika i mogu se dodavati i brisati dok se aktivnost izvodi.

Korisničko sučelje sastoji se od tri dijela:

- Mapa
- Komunikacija
- Izračun za izranjanje

Samo sučelje raspoređeno je u tri dijela korištenjem kartica, odnosno svaki dio se nalazi u zasebnoj kartici. Važnije informacije i elementi kao što su trenutna dubina i slanje SOS poruke nalaze se na samom vrhu kako bi bili uvijek dostupni.

Glavna aktivnost

Glavnu aktivnost predstavlja ViewPager (Slika 13) koji drži ostale elemente, odnosno NonSwipeableViewPager.

```
1 <com.fer.labust.diverassistant.NonSwipeableViewPager xmlns:android="http://schemas.android.com/apk/res/android"
2   xmlns:tools="http://schemas.android.com/tools"
3   android:id="@+id/pager"
4   android:layout_width="match_parent"
5   android:layout_height="match_parent"
6   tools:context=".MainActivity" />
```

Slika 12 Glavna aktivnost

ViewPager po zadanom ponašanju omogućuje listanje karticama, kako je to problematično zbog prikaza karte dodan je razred NonSwipeableViewPager.

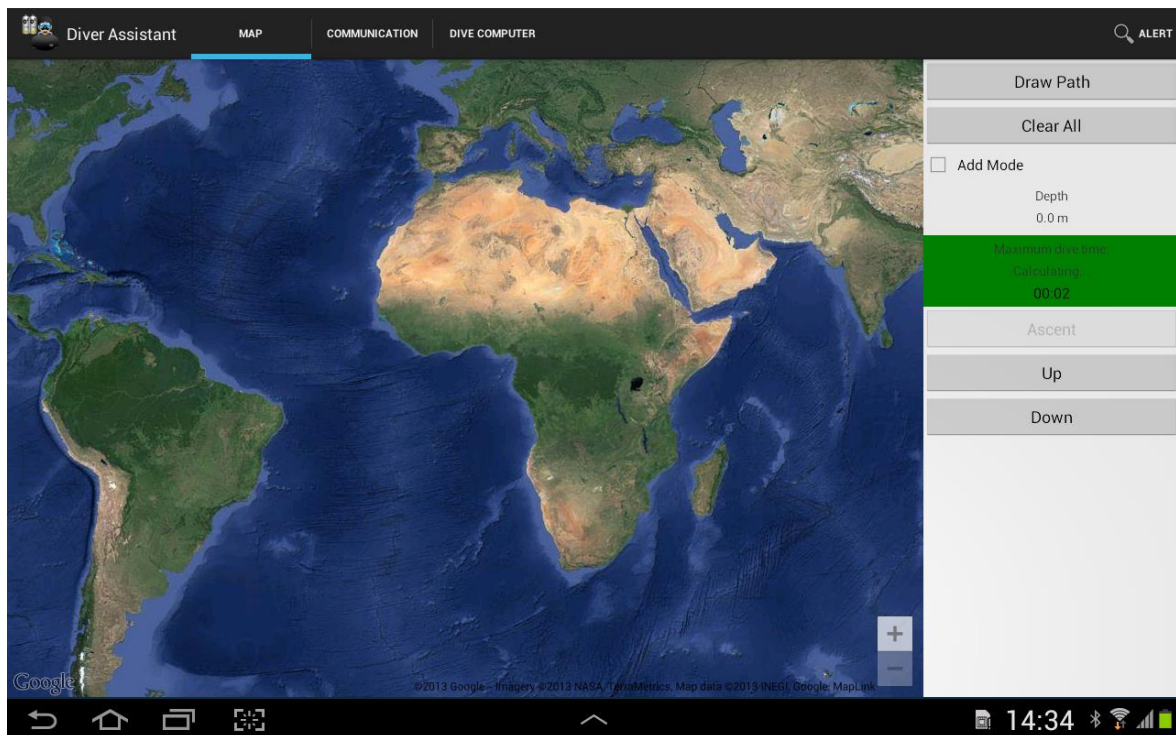
```
1 package com.fer.labust.diverassistant;
2
3+ import android.content.Context;
4
5
6
7
8 public class NonSwipeableViewPager extends ViewPager {
9
10-     public NonSwipeableViewPager(Context context) {
11         super(context);
12     }
13
14-     public NonSwipeableViewPager(Context context, AttributeSet attrs) {
15         super(context, attrs);
16     }
17
18-     @Override
19     public boolean onInterceptTouchEvent(MotionEvent arg0) {
20         // Never allow swiping to switch between pages
21         return false;
22     }
23
24 }
```

Slika 13 NonSwipeableViewPager

NonSwipeableViewPager (Slika 12) nasljeđuje ViewPager i nadjačava metodu onInterceptTouchEvent koja zabranjuje listanje kartica.

Mapa

Na prvom mjestu nalazi se zemljopisna karta na kojoj se nalazi trenutna pozicija ronioca, staza za pretraživanje i akcija za započinjanje izranjanja. Izgled kartice sa mapom je prikazan na slici 14.



Slika 14 Kartica sa mapom

Točke se dodaju uključivanjem načina za dodavanjem tako da se uključi kućica „Add Mode“. Nakon toga točke se dodaju klikom na mapu. Točke se brišu odabirom željenog markera, nakon čega se pojavljuje dijalog sa potvrdom brisanja. Odabirom opcije „Draw Path“ iscrta se staza za pretraživanje. Svi elementi sa ekrana se brišu pokretanjem akcije „Clear All“.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="fill_parent"
5      android:orientation="horizontal" >
6      <fragment
7          android:id="@+id/map"
8          android:layout_width="1000dp"
9          android:layout_height="match_parent"
10         android:name="com.google.android.gms.maps.SupportMapFragment" />
11
12     <Button
13         android:id="@+id/button_draw_path"
14         android:layout_width="wrap_content"
15         android:layout_height="wrap_content"
16         android:text="@string/draw_path"
17         android:layout_toRightOf="@+id/map"
18         android:layout_alignParentRight="true"/>

```

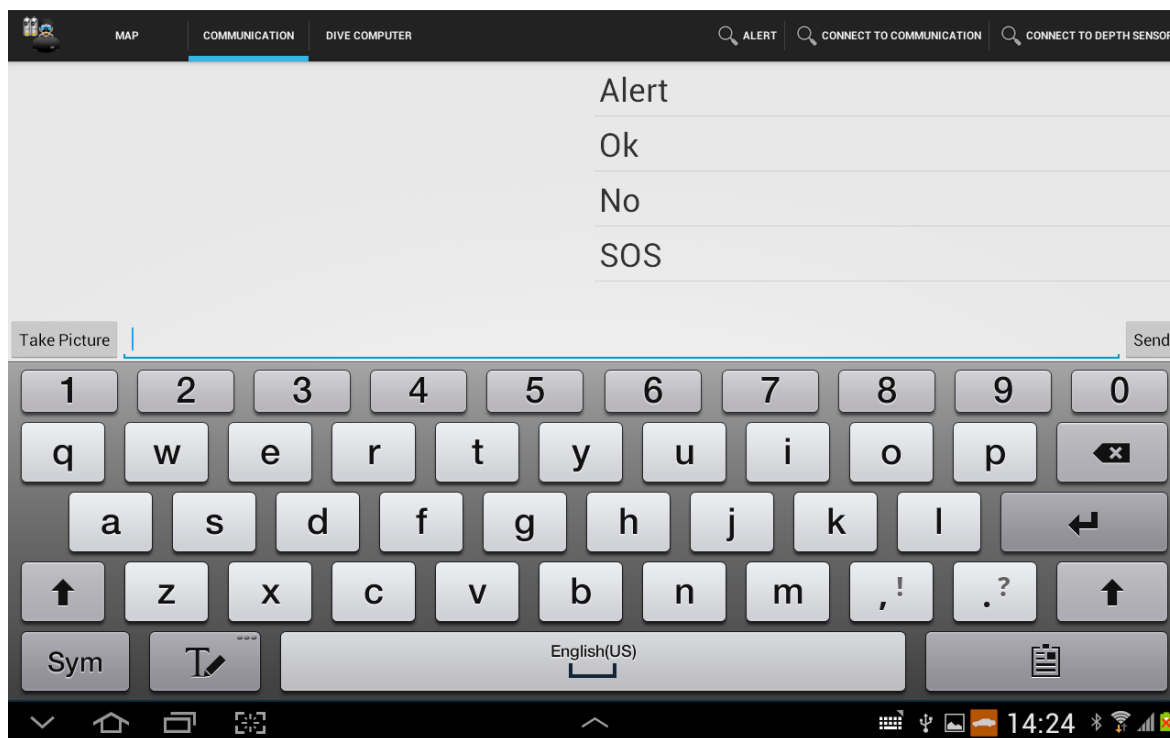
Slika 15 Fragment mapa

Za izradu kartice sa mapom korišten je element RelativeLayout. Element RelativeLayout sadrži raspored tako da se elementi prikazuju u relativnom odnosu.

Odnosno koriste se atributi lijevo od, desno od, iznad prilikom slaganja rasporeda. Na slici 15 je prikazan dio fragmenta.

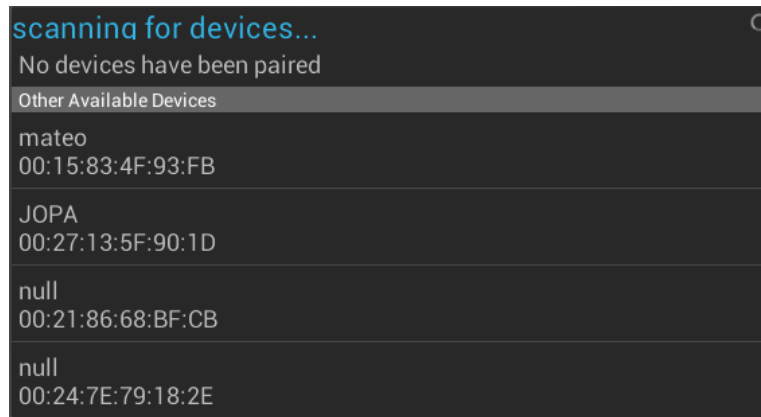
Komunikacija

Kartica za komunikaciju (Slika 16) sadrži listu predefiniranih poruka koje se šalju čim se odaberu, okvir za unos teksta te gumbovi za okidanje slike i slanje napisane poruke. Također prilikom prelaska na karticu za komunikaciju pojavljuju se na samom vrhu aplikacije opcije za spajanje na senzore i komunikaciju.



Slika 16 Kartica za komunikaciju

Snimljena slika sprema se lokalno u punoj rezoluciji. Odabirom opcije za spajanje pojavljuje se dijalog sa listom uparenih Bluetooth uređaja i moguće je skenirati nove dostupne uređaje (Slika 17).



Slika 17 Spajanje sa uređajima

Za stvaranje rasporeda sučelja (Slika 18) koristi se element `LinearLayout`. `LinearLayout` grupira sve unutarnje elemente u jednom smjeru, vertikalno ili horizontalno. Smjer se specificira sa atributom `android:orientation`. Svi unutarnji elementi su poslagani jedan iza drugog, tako da vertikalna lista ima samo jedan element po redu, a horizontalna će biti visoka jedan red.

```

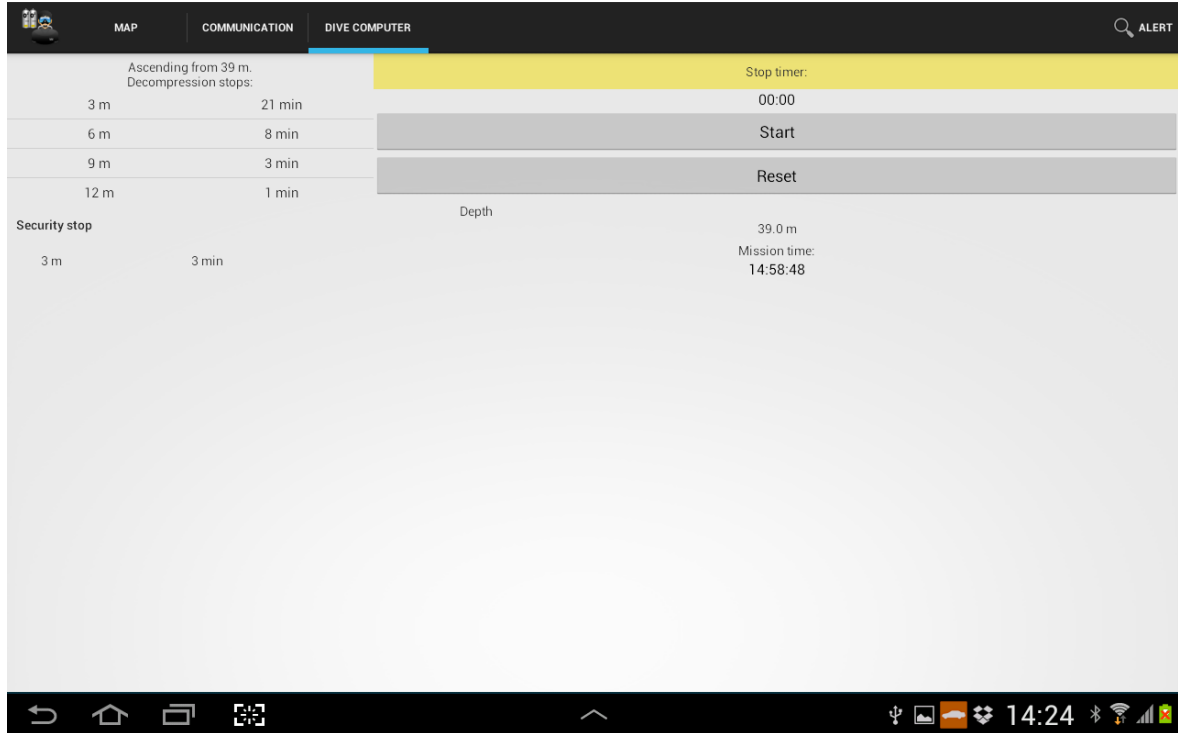
1 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2   xmlns:tools="http://schemas.android.com/tools"
3   android:layout_width="match_parent"
4   android:layout_height="match_parent"
5   android:orientation="vertical"
6   tools:context=".BluetoothChat" >
7   <LinearLayout
8     android:layout_width="match_parent"
9     android:layout_height="fill_parent"
10    android:layout_weight="1"
11    android:orientation="horizontal" >
12    <ListView
13      android:layout_width="fill_parent"
14      android:layout_weight="1"
15      android:id="@+id/in"
16      android:layout_height="match_parent"
17      android:stackFromBottom="true"
18      android:transcriptMode="alwaysScroll" />
19    <ListView
20      android:layout_width="fill_parent"
21      android:layout_weight="1"
22      android:id="@+id/predefined_messages"
23      android:layout_height="match_parent"
24      android:transcriptMode="alwaysScroll" />
25
26 </LinearLayout>

```

Slika 18 Fragment komunikacija

Izranjanje

Kartica „Dive Computer“ (Slika 19) sadrži sve potrebne informacije proračunate ovisno o parametrima dostupnima sa senzora roniocu za sigurno izranjanje na površinu.



Slika 19 Izranjanje

Na slici 20 prikazan je isječak koda korisničkog sučelja kartice za izranjanje. Za dizajn sučelja korišten je element `TableLayout`. `TableLayout` formira sučelje kao tablicu i svaki element predstavlja jedan redak u tablici odnosno `Table Row`

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     android:orientation="horizontal" >
6
7     <TableLayout
8         android:layout_width="400dp"
9         android:layout_height="match_parent" >
10
11         <TableRow
12             android:id="@+id/tableRow1"
13             android:layout_width="wrap_content"
14             android:layout_height="wrap_content" >
15
16             <TextView
17                 android:id="@+id/depthLevel"
18                 android:layout_width="400dp"
19                 android:layout_height="wrap_content"
20                 android:ems="14"
21                 android:gravity="center"
22                 android:paddingTop="5dp"
23                 android:text="a" />
24         </TableRow>
25
26         <TableRow
27             android:id="@+id/tableRow2"
28             android:layout_width="400dp"
29             android:layout_height="wrap_content" >

```

Slika 20 Isječak koda korisničkog sučelja fragmenta za izranjanje

AndroidManifest.xml

Svaka aplikacija mora imati AndroidManifest.xml datoteku u svojem korijenskom direktoriju. Ta datoteka predstavlja osnovne informacije o aplikaciji, i samom Android sustavu, odnosno informacije koje sustav mora imati prije nego što može izvršavati kod. Neke od informacija koje se nalaze u manifestu:

- Naziv aplikacije
- Popis aplikacijskih komponenti (aktivnosti, servisi...)
- Određuje koji procesi će izvoditi aplikacijske komponente
- Dopuštenja koja mora imati aplikacija kako bi pristupala zaštićenim dijelovima APIja
- Minimalna verzija Android APIja koju zahtjeva aplikacija
- Knjižnice koje su potrebne za rad aplikacije

Kako bi aplikacija mogla dobiti pristup određenim sistemskim dijelovima potrebno je zatražiti dopuštenje od korisnika za pristup. Dopuštenja se navode u AndroidManifest.xml datoteci (Slika 21).

```
<uses-sdk
    android:minSdkVersion="16"
    android:targetSdkVersion="17" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<permission
    android:name="com.example.mapdemo.permission.MAPS_RECEIVE"
    android:protectionLevel="signature"/>
<uses-permission android:name="com.example.mapdemo.permission.MAPS_RECEIVE" />
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WRITE_SETTINGS"/>
<uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
<uses-feature
    android:glEsVersion="0x00020000"
    android:required="true"/>
<uses-feature android:name="android.hardware.camera" />
<uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION"/>
```

Slika 21 Potrebna dopuštenja

Potrebna dopuštenja su:

- Bluetooth
- Mape
- Internet
- Vanjsko spremište podataka
- Upis postavki
- Različita dopuštenja za dohvaćanje geografske pozicije


```

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name_label"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.fer.labust.diverassistant.MainActivity"
        android:label="@string/app_name"
        android:screenOrientation="landscape">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".DeviceListActivity"
        android:label="@string/select_device"
        android:theme="@android:style/Theme.Holo.Dialog"
        android:configChanges="orientation"
        android:screenOrientation="landscape"/>
    <meta-data
        android:name="com.google.android.maps.v2.API_KEY"
        android:value="AIzaSyDJU2Tb48Ubr876SlHxul42ZBgnkwdL7_w"/>
</application>

```

Slika 22 Postavke aplikacije

Na slici 22 su prikazane postavke aplikacije iz datoteke AndroidManifest.xml. Aplikacija se sastoji od dvije aktivnosti: Glavna aktivnost i aktivnost sa listom uređaja. Glavna aktivnost izvodi cijelu aplikaciju i ona se pokreće pri pokretanju aplikacije. DeviceListActivity se pokreće prilikom odabira Bluetooth uređaja koji predstavlja senzor ili komunikaciju.

Komunikacija

Sva komunikacija odvija se putem Bluetooth protokola. Uređaj se spaja na komunikaciju sa površinom i različitim senzorima. Za svu komunikaciju zadužen je razred BluetoothChatService.

BluetoothChatService.java

Razred BluetoothChatService radi sav posao za postavljanje i upravljanje Bluetooth konekcijama sa ostalim uređajima. Sadrži dvije dretve

- Dretva za uspostavljanje konekcije
- Dretva za obavljanje prijenosa podataka sa spojenim uređajima

Bitno je primijetiti da ne postoji dretva koja sluša dolazne konekcije, to je zbog toga što ne postoji slučaj kada drugi uređaj inicira konekciju. DiverAssistant

aplikacija je *master* uređaj koji uvijek zahtjeva konekciju sa drugim uređajima. Kako bi se omogućilo spajanje na više uređaja potrebno je imati listu dretvi koje obavljaju prijenos podataka sa spojenim uređajima.

```
public synchronized void connect(BluetoothDevice device, boolean secure, DeviceType deviceType) {
    if (D) Log.d(TAG, "connect to: " + device);

    // Cancel any thread attempting to make a connection
    if (mState == STATE_CONNECTING) {
        if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}
    }

    // Start the thread to connect with the given device
    mConnectThread = new ConnectThread(device, secure, deviceType);
    mConnectThread.start();
    setState(STATE_CONNECTING);
}
```

Slika 23 Connect metoda

Kada se napravi zahtjev za spajanjem pozove se metoda connect (Slika 23). Metoda connect zaustavlja dretve koje pokušavaju uspostaviti konekciju i stvara novu dretvu sa potrebnim parametrima za spajanje.

```
public synchronized void connected(BluetoothSocket socket, BluetoothDevice
    device, final String socketType, DeviceType deviceType) {
    // Cancel the thread that completed the connection
    if (D) Log.d(TAG, "connected, Socket Type:" + socketType);

    if (mConnectThread != null) {mConnectThread.cancel(); mConnectThread = null;}

    // Start the thread to manage the connection and perform transmissions
    ConnectedThread mConnectedThread = new ConnectedThread(socket, socketType, deviceType);
    mConnectedThread.start();
    mConnectedThreads.add(mConnectedThread);

    setState(STATE_CONNECTED);
}
```

Slika 24 Connected metoda

Connected metodu (Slika 24) poziva dretva za uspostavljanje veze. Ona zaustavlja dretvu za uspostavljanje veze i započinje novu dretvu za razmjenu podataka sa potrebnim parametrima. Stvorenu dretvu dodaje u listu postojećih dretvi koje komuniciraju sa vanjskim uređajima.

Za zaustavljanje komunikacije služi metoda stop

```

public synchronized void stop() {
    if (D) Log.d(TAG, "stop");

    if (mConnectThread != null) {
        mConnectThread.cancel();
        mConnectThread = null;
    }
    for(ConnectedThread connectedThread : mConnectedThreads){
        if(connectedThread != null){
            connectedThread.cancel();
            connectedThread = null;
        }
    }

    setState(STATE_NONE);
}

```

Slika 25 Stop metoda

Metoda stop (Slika 25) zaustavlja dretvu za spajanje i sve dretve sa uspostavljenim konekcijama ukoliko te dretve postoje.

ConnectThread

Dretva za uspostavljanje konekcije je pokrenuta i pokušava stvoriti izlaznu konekciju sa uređajem.

```

private class ConnectThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final BluetoothDevice mmDevice;
    private String mSocketType;
    private DeviceType deviceType;

    public ConnectThread(BluetoothDevice device, boolean secure, DeviceType deviceType) {
        this.deviceType = deviceType;
        mmDevice = device;
        BluetoothSocket tmp = null;
        mSocketType = secure ? "Secure" : "Insecure";
        try {
            if (secure) {
                tmp = device.createRfcommSocketToServiceRecord(
                    MY_UUID_SECURE);
            } else {
                tmp = device.createInsecureRfcommSocketToServiceRecord(
                    MY_UUID_INSECURE);
            }
        } catch (IOException e) {
            Log.e(TAG, "Socket Type: " + mSocketType + "create() failed", e);
        }
        mmSocket = tmp;
    }
}

```

Slika 26 Konstruktor dretve za uspostavljanje konekcije

Kako je ConnectThread dretva, ona implementira Thread razred. Konstruktor (Slika 26) prima argumente potrebne za rad ConnectThread dretve a to su

- Bluetooth uređaj na koji se spaja (BluetoothDevice)
- Tip veze (Siguran/nesiguran)
- Tip uređaja (DeviceType)

BluetoothDevice je razred omotač za fizičku adresu Bluetooth uređaja i on je potreban za spajanje na sam uređaj. Tip veze je po pretpostavljenoj vrijednosti nesiguran. Odabran je nesiguran tip veze jer nema potrebe za sigurnosti i obavlja se manje posla, odnosno sigurno spajanje je skuplja operacija od nesigurnog. Razred DeviceType (Slika 27) služi za diferencijaciju spojenih uređaja, odnosno prilikom spajanja jedini identifikator uređaja je njegov naziv i fizička MAC adresa, kako bi se podaci prikazivali na željenim mjestima potrebno je razlikovati uređaje po tipu.

```
package com.fer.labust.diverassistant;

public enum DeviceType {
    COMMUNICATION(0), GPS(1), DEPTH_SENSOR(2);
    private int value;

    private DeviceType(int value){
        this.value = value;
    }

    public int getValue(){
        return this.value;
    }
}
```

Slika 27 DeviceType razred

DeviceType je tipa enum i ima konstruktor i metodu za dohvat vrijednosti.

```

public void run() {
    Log.i(TAG, "BEGIN mConnectThread SocketType:" + mSocketType);
    setName("ConnectThread" + mSocketType);
    mAdapter.cancelDiscovery();
    try {
        mmSocket.connect();
    } catch (IOException e) {
        // Close the socket
        try {
            mmSocket.close();
        } catch (IOException e2) {
            Log.e(TAG, "unable to close() " + mSocketType +
                " socket during connection failure", e2);
        }
        connectionFailed();
        return;
    }
    // Reset the ConnectThread because we're done
    synchronized (BluetoothChatService.this) {
        mConnectThread = null;
    }
    // Start the connected thread
    connected(mmSocket, mmDevice, mSocketType, deviceType);
}

```

Slika 28 Run metoda razreda ConnectThread

Nakon instanciranja razreda ConnectThread sa svim potrebnim parametrima, dretva se pokreće. Prilikom pokretanja dretve (Slika 28), zaustavlja se otkrivanje novih uređaja kako bi se ubrzalo uspostavljanje veze. Također dretvi se postavlja naziv i započinje se spajanje na utičnicu pozivanjem metode connected, za prekid konekcije koristi se metoda cancel (Slika 29).

```

public void cancel() {
    try {
        mmSocket.close();
    } catch (IOException e) {
        Log.e(TAG, "close() of connect " + mSocketType + " socket failed", e);
    }
}

```

Slika 29 Cancel metoda razreda ConnectThread

Cancel metoda zatvara uspostavljenu utičnicu.

ConnectedThread

ConnectedThread dretva se pokreće prilikom uspostavljanje konekcije i upravlja sa prijenosom podataka.

```

private class ConnectedThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final InputStream mmInStream;
    private final OutputStream mmOutStream;
    private DeviceType deviceType;

    public ConnectedThread(BluetoothSocket socket, String socketType, DeviceType deviceType){
        Log.d(TAG, "create ConnectedThread: " + socketType);
        mmSocket = socket;
        InputStream tmpIn = null;
        OutputStream tmpOut = null;

        // Get the BluetoothSocket input and output streams
        try {
            tmpIn = socket.getInputStream();
            tmpOut = socket.getOutputStream();
        } catch (IOException e) {
            Log.e(TAG, "temp sockets not created", e);
        }

        mmInStream = tmpIn;
        mmOutStream = tmpOut;
        this.deviceType = deviceType;
    }
}

```

Slika 30 ConnectedThread

Konstruktor dretve ConnectedThread (Slika 30) prima prethodno stvoren BluetoothSocket, tip utičnice i tip uređaja. Sučelje BluetoothSocket razred je slično kao i kod TCP utičnica. Ono služi za upravljanje konekcijom. Također stvaraju se ulazni i izlazni tokovi za prijenos podataka InputStream i OutputStream.

```

public void run() {
    Log.i(TAG, "BEGIN mConnectedThread");
    // Keep listening to the InputStream while connected
    while (true) {
        try {
            // Read from the InputStream
            //bytes = mmInStream.read(buffer);
            MmcMessage msg = sync(mmInStream);

            // Send the obtained bytes to the UI Activity
            mHandler.obtainMessage(CommunicationSectionFragment.MESSAGE_READ, -1, -1, msg)
                .sendToTarget();
        } catch (IOException e) {
            Log.e(TAG, "disconnected", e);
            connectionLost();
            // Start the service over to restart listening mode
            BluetoothChatService.this.start();
            break;
        }
    }
}

```

Slika 31 run metoda

Run metoda (Slika 31) izvršava beskonačnu petlju koja poziva metodu sync za čitanje ulaznog toka. Ukoliko dođe do greške prilikom čitanja, prekida se veza metodom connectionLost. Za upis podataka u izlazni tok koristi se metoda write.

```

public void write(byte[] buffer, String message) {
    try {
        mmOutputStream.write(buffer);

        // Share the sent message back to the UI Activity
        mHandler.obtainMessage(CommunicationSectionFragment.MESSAGE_WRITE, -1, -1, message.getBytes())
            .sendToTarget();
    } catch (IOException e) {
        Log.e(TAG, "Exception during write", e);
    }
}

```

Slika 32 write metoda

Write metoda (Slika 32) poziva odgovarajuću metodu na izlaznom toku i šalje podatak. Za zaustavljanje dretve koristi se metoda Cancel kao i kod ConnectThread dretve koja zatvara otvorenu utičnicu.

DeviceListActivity

Ova aktivnost pojavljuje se kao dijalog. Ona prikazuje sve uparene uređaje i uređaje koje su otkriveni nakon skeniranja područja. Kada se odabere uređaj, MAC adresa se šalje u pozivajuću aktivnost kao rezultat. U ovoj aktivnosti koristi se BroadcastReceiver (Slika 33), on služi za pronalazak uređaja i mijenja naslov nakon što je otkrivanje uređaja završilo.

```

private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();

        // When discovery finds a device
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {
            // Get the BluetoothDevice object from the Intent
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            // If it's already paired, skip it, because it's been listed already
            if (device.getBondState() != BluetoothDevice.BOND_BONDED) {
                mNewDevicesArrayAdapter.add(device.getName() + "\n" + device.getAddress());
            }
        }
        // When discovery is finished, change the Activity title
    } else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action)) {
        setProgressBarIndeterminateVisibility(false);
        setTitle(R.string.select_device);
        if (mNewDevicesArrayAdapter.getCount() == 0) {
            String noDevices = getResources().getText(R.string.none_found).toString();
            mNewDevicesArrayAdapter.add(noDevices);
        }
    }
};

```

Slika 33 BroadcastReceiver

Nakon dovršenog otkrivanja, čeka se korisnikov odabir uređaja (Slika 34).

```

private OnItemClickListener mDeviceClickListener = new OnItemClickListener() {
    public void onItemClick(AdapterView<?> av, View v, int arg2, long arg3) {
        // Cancel discovery because it's costly and we're about to connect
        mBtAdapter.cancelDiscovery();

        // Get the device MAC address, which is the last 17 chars in the View
        String info = ((TextView) v).getText().toString();
        String address = info.substring(info.length() - 17);

        // Create the result Intent and include the MAC address
        Intent intent = new Intent();
        intent.putExtra(EXTRA_DEVICE_ADDRESS, address);

        // Set result and finish this Activity
        setResult(Activity.RESULT_OK, intent);
        finish();
    }
};

```

Slika 34 Odabir uređaja

Nakon odabira uređaja uzima se zadnjih 17 znakova što predstavlja MAC adresu i postavlja se kao rezultat i šalje u pozivajuću aktivnost.

Parse

Za parsiranje dospjelih poruka i prikaz na korisničkom sučelju zadužen je Parse razred. Razred Parse u konstruktor (Slika 35) prima upravitelja lokacije i instancu kartice zaduženu za komunikaciju.

```

public Parse(LocationManager locationManager, String mocLocationProvider,
    CommunicationSectionFragment chat) throws IOException {
    bc = chat;
    this.locationManager = locationManager;
    this.mocLocationProvider = mocLocationProvider;
}

```

Slika 35 Parse konstruktor

Kako Parse razred proširuje razred Thread, on je dretva i implementira metodu run (Slika 36).


```

public void run() {
    while(run_flag) {
        synchronized(this) {
            try {
                this.wait();
            }
            catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        payc = payloadmsg;
        for(int i = 0; i < 6; i++) {
            Log.e("payc", Byte.toString(payc[i]));
            for(int j = 0; j < 8; j++) {
                if ((payc[i] & (1 << j)) != 0)
                    payb[(5-i)*8+j] = true;
                else
                    payb[(5-i)*8+j] = false;
            }
        }
        msg_type((short) msg_build(44, 4));
        bc.getActivity().runOnUiThread(new Runnable() {
            public void run() {
                if (!mssg.isEmpty())
                    bc.mConversationArrayAdapter.add(mssg);
                bc.getActivity().getActionBar().setTitle("Received Message: " + mssg);

                mssg = "";
            }
        });

        new_data_flag = false;
    }
}

```

Slika 36 Run metoda razreda Parse

Run metoda čeka na nove podatke, nakon primitka nove poruke ona se obrađuje sa metodama msg_build i msg_type. Nakon obrade poruka se dodaje u komunikacijsku karticu i prikazuje se na vrhu kako bi bila vidljiva u svakom dijelu aplikacije. Osim za postavljanje tekstualnih poruka, Parse razred zadužen je i za postavljanje lokacije.

```

private void new_pos(double latitude, double longitude) {
    Location location = new Location(mocLocationProvider);
    location.setLatitude(latitude);
    location.setLongitude(longitude);
    location.setAltitude(0.0);
    location.setProvider(mocLocationProvider);
    location.setTime(System.currentTimeMillis());
    Log.e(LOG_TAG, location.toString());

    locationManager.setTestProviderStatus(mocLocationProvider,
        LocationProvider.AVAILABLE, null, System.currentTimeMillis());
    locationManager.setTestProviderLocation(mocLocationProvider, location);
}

```

Slika 37 new_pos metoda Parse razreda

New_pos metoda (Slika 37) prima geografsku širinu i dužinu, stvara Location objekt i dodaje ga upravitelju lokacije koji koristi mapa za podatke o trenutnoj lokaciji. Lokacija se na karti prikazuje u obliku crvene oznake.

Predefinirane poruke

Jedna od funkcionalnosti kod komunikacije su predefinirane poruke. Predefinirane poruke šalju se na klik i nalaze se sa desne strane u korisničkom sučelju u djelu za komunikaciju. Model predefiniranih poruka je ostvaren razredima DefaultMessage i DefaultMessageList.

```
public class DefaultMessage {

    private int messageNumber;
    private String description;

    public DefaultMessage(int messageNumber, String description){
        this.messageNumber = messageNumber;
        this.description = description;
    }

    public int getMessageNumber() {
        return messageNumber;
    }

    public void setMessageNumber(int messageNumber) {
        this.messageNumber = messageNumber;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    @Override
    public String toString() {
        return this.description;
    }
}
```

Slika 38 DefaultMessage razred

Razred DefaultMessage (Slika 38) sadrži redni broj poruke i opis, i za potrebe prikaza u listi nadjačana je toString metoda koja vraća opis poruke.

```

public class DefaultMessageList {
    private List<DefaultMessage> defaultMessageList;

    public DefaultMessageList() {
        this.defaultMessageList = new ArrayList<DefaultMessage>();
        this.defaultMessageList.add(new DefaultMessage(0, "Alert"));
        this.defaultMessageList.add(new DefaultMessage(1, "Ok"));
        this.defaultMessageList.add(new DefaultMessage(2, "No"));
        this.defaultMessageList.add(new DefaultMessage(3, "SOS"));
        this.defaultMessageList.add(new DefaultMessage(4, "Yes"));
        this.defaultMessageList.add(new DefaultMessage(5, "Diving out"));
        this.defaultMessageList.add(new DefaultMessage(6, "Eagle has landed"));
    }

    public DefaultMessage getDefaultMessage(int messageNumber) {
        DefaultMessage defaultMessage = null;
        try {
            defaultMessage = this.defaultMessageList.get(messageNumber);
        }
        catch (IndexOutOfBoundsException ex) {
            defaultMessage = null;
        }
        return defaultMessage;
    }

    public List<DefaultMessage> getDefaultMessageList() {
        return defaultMessageList;
    }

    public void setDefaultMessageList(List<DefaultMessage> defaultMessageList) {
        this.defaultMessageList = defaultMessageList;
    }
}

```

Slika 39 DefaultMessageList

Razred DefaultMessageList (Slika 39) sadrži listu predefiniраних poruka koje se popunjavaju u samom konstruktoru te sadrži potrebne metode za dohvat i postavljanje pred definiraniх poruka. Slanje se odvija klikom na poruku (Slika 40).

```

mPredefinedMessagesView
    .setOnItemClickListener(new OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent,
            View view, int position, long id) {
            DefaultMessage message = mPredefinedMessagesArrayAdapter
                .getItem(position);
            sendMessage(String.valueOf(message
                .getMessageNumber()));
        }
    });

```

Slika 40 Slanje predefinirane poruke

Također kao što je napomenuto, ne šalje se cijela poruka nego samo njezin identifikator odnosno broj poruke.

Fotografija

Za slanje osim tekstualnih i predefiniranih poruka predviđeno je i spremanje fotografija ukoliko uređaj posjeduje potrebnu opremu. Fotografija nakon okidanja se sprema lokalno u punoj kvaliteti. Akcija fotografiranja pokreće se klikom na gumb „Take Picture“.

```
private void takePictureHandler() {  
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);  
    startActivityForResult(takePictureIntent, DeviceType.CAMERA.getValue());  
}
```

Slika 41 takePictureHandler

Nakon toga se poziva metoda takePictureHandler (Slika 41) koja odrađuje taj posao.

Mapa

Za prikaz karte, odnosno lokacije i staze za pretraživanje zadužena je prva kartica. U njoj se nalaze sve akcije potrebne za planiranje i pregled misije.

Dodavanje i brisanje oznaka

Dodavanje oznake na kartu je zamišljeno tako da se uključi način za dodavanje oznaka. To je ostvareno kao kućica.

```
map.setOnMapClickListener(new OnMapClickListener() {  
  
    @Override  
    public void onMapClick(LatLng arg0) {  
        CheckBox chkAddMode = (CheckBox) getView().findViewById(R.id.chkAddMode);  
        if(chkAddMode.isChecked()) {  
            MarkerOptions markerOptions = new MarkerOptions();  
            markerOptions.position(arg0);  
  
            Marker point = map.addMarker(markerOptions);  
            markers.add(point);  
        }  
    }  
});
```

Slika 42 Dodavanje oznaka na mapu

Nakon što je uključen način za dodavanje oznaka, oznaka se dodaje klikom na mapu. Prilikom svakog klika na mapu provjerava se označenost kućice (Slika 42).

Oznake se brišu odabirom same oznake koju se želi maknuti, nakon čega se otvara dijalog za potvrdu brisanja (Slika 43).

```

map.setOnMarkerClickListener(new OnMarkerClickListener() {

    @Override
    public boolean onMarkerClick(Marker arg0) {
        if(D) Log.i(TAG, "onMarkerClick called Marker: " + arg0.getPosition());
        MapSectionFragment.markerToDelete = arg0;

        DialogInterface.OnClickListener dialogClickListener = new DialogInterface.OnClickListener() {

            @Override
            public void onClick(DialogInterface dialog, int which) {
                switch (which) {
                    case DialogInterface.BUTTON_POSITIVE:
                        markers.remove(MapSectionFragment.markerToDelete);
                        if(D) Log.i(TAG, "OnClickMarker removed" + MapSectionFragment.markerToDelete.getPosition());
                        MapSectionFragment.markerToDelete.remove();
                        break;

                    case DialogInterface.BUTTON_NEGATIVE:
                        // No button clicked
                        break;
                }
            }
        };
    }
});

```

Slika 43 Brisanje oznake

Odabirom oznake ona se postavlja kao članska varijabla kako bi se mogla dohvatiti nakon što je dijalog zatvoren.

Generacija puta za pretraživanje

Kako bi ronilac pretražio područje potrebna mu je staza za pretraživanje na zadanom području. Algoritam za pretraživanje je prethodno implementiran i potrebno ga je pozvati nad točkama koje su dodane na kartu. Točke koje su dodane na kartu spremaju se kao članske varijable. Algoritam se poziva klikom na gumb „Draw Path“. Klikom na gumb poziva se metoda drawPath (Slika 44).

```

public void drawPath(){
    if(D) Log.i(TAG, "drawPath called ");
    if(MainActivity.markers.size() < 2){
        Toast.makeText(getActivity(), "Not enough markers", 12).show();
        return;
    }
    List<Point> pathPoints = new ArrayList<Point>();
    List<Point> points = new ArrayList<Point>();
    PolygonOptions polygonOptions = new PolygonOptions();
    for (Marker marker: MainActivity.markers){
        polygonOptions.add(new LatLng(marker.getPosition().latitude, marker.getPosition().longitude));
        points.add(new Point(marker.getPosition().latitude, marker.getPosition().longitude));
    }
    PlannerService plannerService = new PlannerService(new Polygon(points), 5);
    pathPoints = plannerService.odrediOptimalneTockePresjeka();

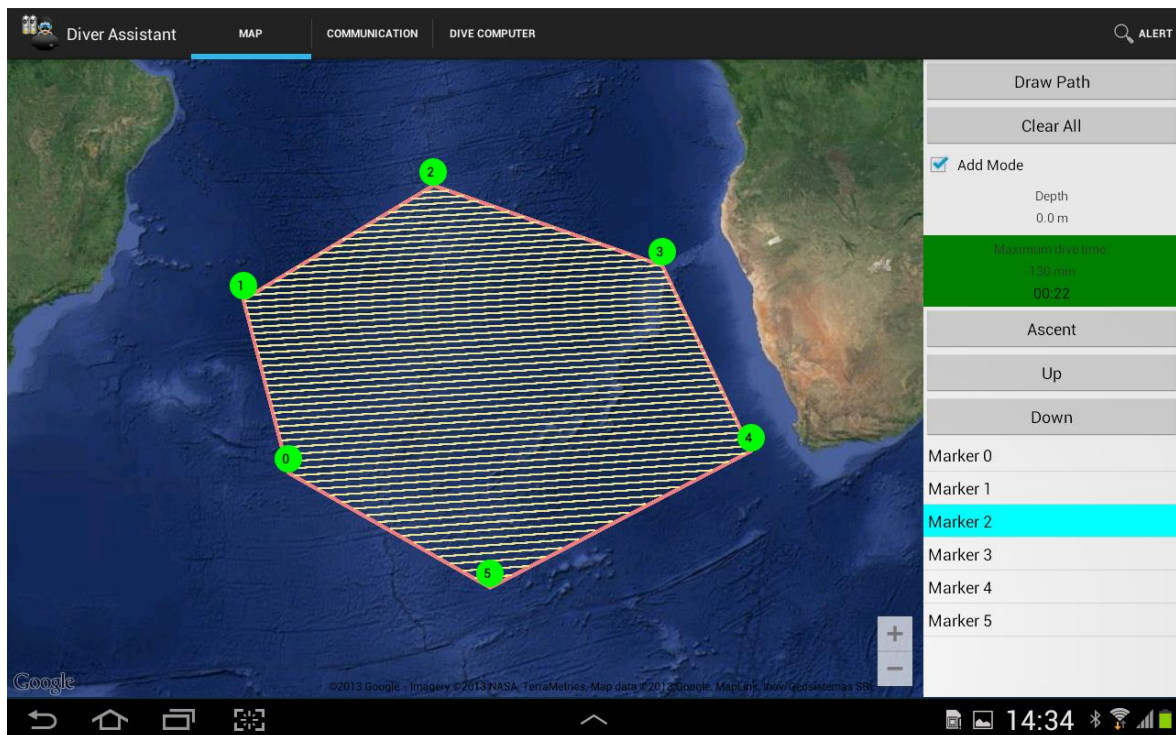
    polygonOptions.strokeColor(0xffff08080);
    polygonOptions.strokeWidth(4);
    if(MainActivity.polygon != null) {
        MainActivity.polygon.remove();
    }
    MainActivity.polygon = map.addPolygon(polygonOptions);

    for (int i=0; i<pathPoints.size();i+=2){
        Point firstPoint = pathPoints.get(i);
        Point secondPoint = pathPoints.get(i+1);
        PolylineOptions polylineOptions = new PolylineOptions();
        polylineOptions.add(new LatLng(firstPoint.getLatitude(), firstPoint.getLongitude()),
            new LatLng(secondPoint.getLatitude(), secondPoint.getLongitude()));
        polylineOptions.color(0xffff0e68c);
        polylineOptions.width(2);
        MainActivity.polyLines.add(map.addPolyline(polylineOptions));
    }
}

```

Slika 44 drawPath metoda

DrawPath metoda provjerava broj točaka dovoljan za generaciju staze i na temelju tih točaka je generira. Put i poligon se spremaju kao članske varijable. U samoj aplikaciji moguće je odabrati poredak po kojem se generira poligon. To je bitno zbog naknadnog dodavanje točaka. Svaka točka kada se doda, ona se numerira. Ta ista točka dodaje se i u listu sa strane. Odabirom točke i gumbima Up i Down moguće je pomicati redoslijed. Nakon odabira željenog redoslijeda, klikom na gumb Draw Path, generira se staza po poretку zadanom u listi sa strane (Slika 45).



Slika 45 Generirani poligon

Omogućeno je i brisanje svih elemenata sa karte naredbom „Clear All“. Tom naredbom se poziva clearAll metoda.

```
public void clearAll() {
    if (D) Log.i(TAG, "clearAll called ");
    if (MainActivity.polygon != null) {
        MainActivity.polygon.remove();
    }
    for (Polyline polyLine : MainActivity.polyLines) {
        polyLine.remove();
        if (D) Log.i(TAG, "polyline removed ");
    }
    for (Marker marker : MainActivity.markers) {
        if (D) Log.i(TAG, "marker removed " + marker.getPosition());
        marker.remove();
    }
    MainActivity.polygon = null;
    MainActivity.markers.clear();
    MainActivity.polyLines.clear();
}
```

Slika 46 clearAll metoda

ClearAll metoda (Slika 46) provjerava postoji li generirani put i poligon i ako postoji onda ih briše sa karte.

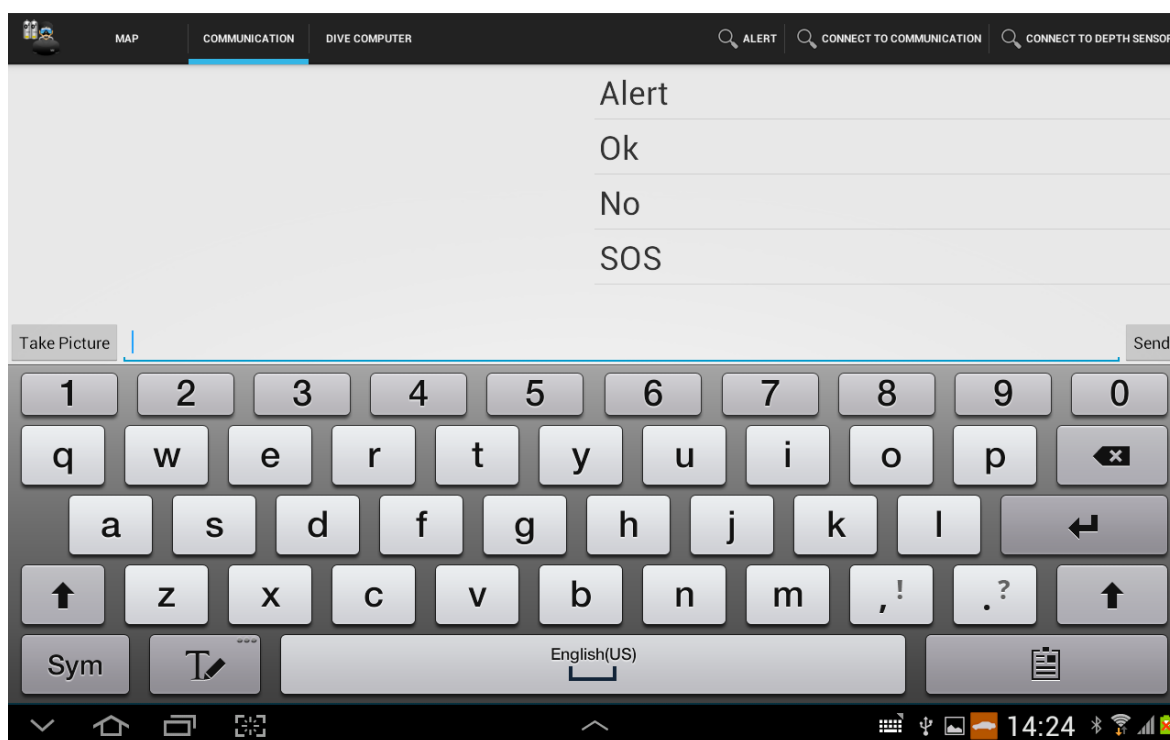
Testiranje i pokretanje aplikacije

Aplikacija se pokreće odabirom ikone Diver Assistant (Slika 47) u izborniku tablet računala.



Slika 47 Ikona aplikacije Diver Assistant

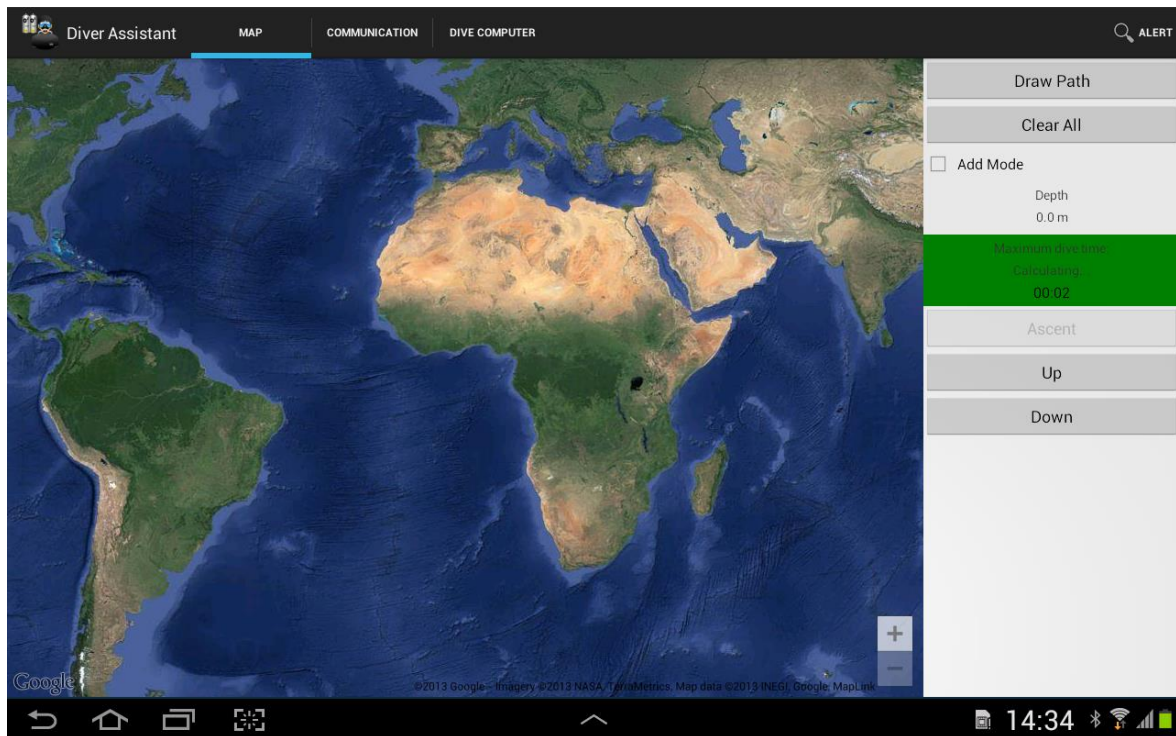
Nakon pokretanja aplikacije otvara se početna kartica sa kartom, kako bi se ostvarila komunikacija sa senzorima i površinom potrebno je izvršiti spajanje. Spajanje se odvija prelaskom na karticu za komunikaciju (Slika 48). Kada se otvori kartica za komunikaciju u izborniku sa karticama stvaraju se opcije za spajanje. Odabirom tih opcija i odgovarajućih uređaja, tablet računalo se spaja na željene senzore, odnosno komunikaciju.



Slika 48 Kartica za komunikaciju

Nakon spajanja dostupna je komunikacija sa površinom. Komunikacija sa površinom može se ostvariti na nekoliko načina. Slanjem predefiniраниh poruka, unosom teksta i primanjem trenutne lokacije sa površine. Također dostupna je u svim karticama opcija Alert koju ronilac odabire u slučaju problema sa ronjenjem ili

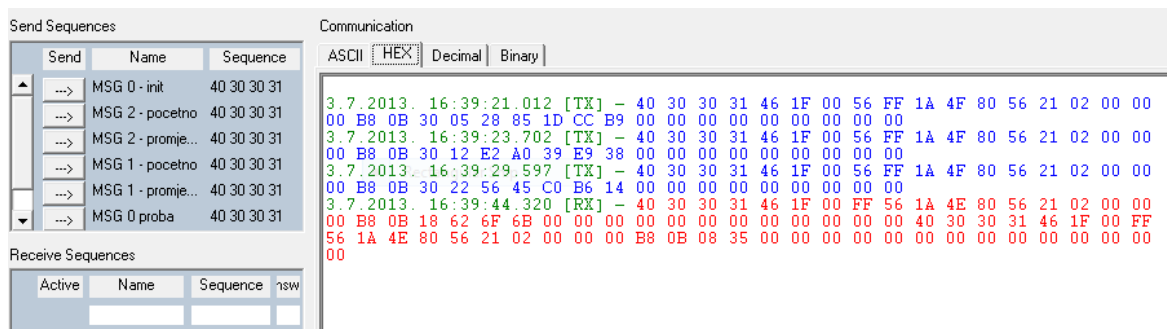
izranjanjem. U kartici sa komunikacijom moguće je i napraviti fotografiju koja se sprema lokalno za naknadno pregledavanje.



Slika 49 Kartica za planiranje i pregled misije

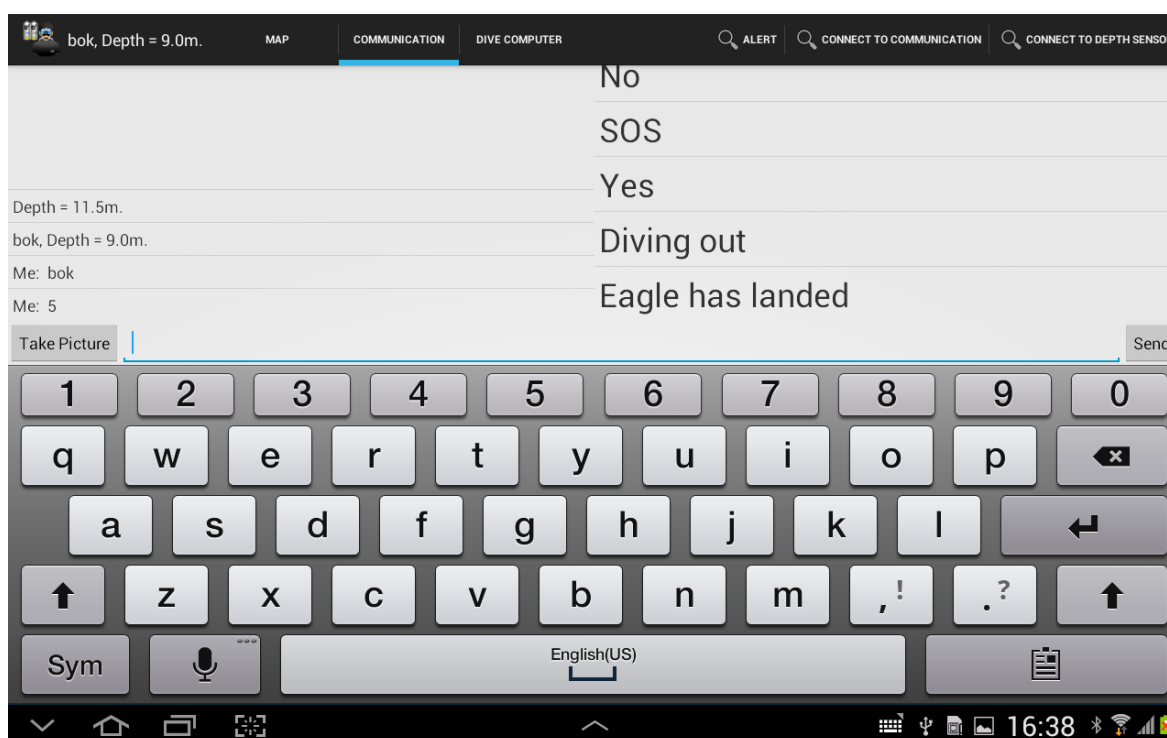
U kartici za planiranje i pregled misije (Slika 49) dostupne su opcije za dodavanje i brisanje točaka poligona te generacija i brisanje poligona. Osim toga, svaka dodana oznaka mjesta numerira se i dodaje se na listu sa strane u kojoj je moguće gumbima Up i Down kontrolirati poredak po kojima se generira poligon. Dodavanje točke ostvareno je tako da se uključi kućica Add Mode, nakon čega se pritiskom na kartu dodaje oznaka mjesta. Odabirom prikazane točke prikazuje se dijalog sa upitom o brisanju označene točke. Osim toga prikazano je vrijeme trajanja misije i opcija za izranjanje kojom se prebacuje u karticu za izranjanje i prikazuju se roniocu upute za sigurno izranjanje odnosno dekompresiju.

Za testiranje protokola korišten je alat Docklight. Alat Docklight može primati i slati poruke preko RS-232 protokola.



Slika 50 Poslane i primljene poruke u alatu Docklight

Na slikama su prikazane poslane i primljene poruke. Na slici 50 su te poruke prikazane u alatu Docklight, na slici 51 u kartici za komunikaciju.



Slika 51 Poslane i primljene poruke u kartici za komunikaciju

Zaključak

Rezultat diplomskog rada je aplikacija koja pomaže ronioniku prilikom podvodnog istraživanja. Aplikacija je ostvarena kao Android aplikacija za tablet računalo. Vođena je briga o svim funkcionalnostima koje su mu potrebne, kao i samoj ergonomiji sučelja. Kako bi korisnik mogao generirati i pregledavati stazu za pretraživanje ugrađen je prethodno razvijen algoritam za planiranje područja pretrage, te je moguća prilagodba parametara misije. Istražene su mogućnosti spajanja na više Bluetooth uređaja istovremeno. Različiti Bluetooth uređaji predstavljaju senzore koji su potrebni ronioniku za snalaženje tijekom obavljanja misije. Svi podaci su ronioniku lako dostupni i pregledni. Ronioniku su tijekom misije dostupne mogućnosti koje inače nisu dostupne u komercijalnim rješenjima sličnih aplikacija. Neke od tih mogućnosti su pregled i postavljanje misije te podaci sa senzora koji mu daju uvid u njegovu okolinu. Također ronilac može komunicirati sa površinom, čime dobiva dodatnu pomoć prilikom podvodne misije od tima koji se nalazi na kopnu.

Sažetak

Za izradu diplomskog rada bilo je potrebno korištenjem naprednih tehnologija poput podvodnih tableta ostvariti aplikaciju za pomoć ronionicima prilikom izvođenja misije. Također trebalo je proučiti trenutno dostupne tehnologije i njihove mogućnosti. Neke od tih mogućnosti su implementacija prethodno razvijenog algoritma za planiranje područja pretraživanja korištenjem Google Maps karti i prilagoditi ga za ronioce, istražiti mogućnosti spajanja i čitanja podataka s više Bluetooth uređaja istovremeno i razviti sučelje s predefiniranim skupom tekstualnih poruka koje ronilac može odabrati i poslati Bluetooth vezom na vanjski uređaj.

Rezultat diplomskog rada je ostvarena Android aplikacija za podvodni tablet koja omogućuje planiranje i pregled misije. Također istražena je i ostvarena komunikacija sa više Bluetooth uređaja istovremeno, na uređaj je spojeno više Bluetooth uređaja i uspješno je obavljena komunikacija sa svim uređajima. Dio aplikacije koji se odnosi na komunikaciju nudi mogućnost komunikacije sa površinom, slanje predefiniranih poruka i primanje podataka sa senzora.

Ključne riječi: Planiranje misije, komunikacija, Android, Bluetooth, predefinirane poruke, senzori.

Abstract

To create a thesis it was necessary to use advanced technologies such as underwater tablet to make an application to help divers during the execution of the mission. It was necessary to study the currently available technologies and their capabilities. Some of these features are implementation of the previously developed algorithm for area planning using Google Maps and customize it for divers, explore the possibilities of connecting and reading from multiple Bluetooth devices simultaneously and develop an interface with pre-defined set of text messages that a diver can select and send with Bluetooth connection to an external device.

Result of the thesis is realized Android application for underwater tablet that allows planning and mission review. Also communication with multiple Bluetooth devices was researched and realized and has been successfully communicating with all devices. Part of the application that relates to communication offers the ability to communicate with the surface, sending pre-defined messages and receive data from the sensor.

Key words: mission planning, communication, Android, Bluetooth, pre-defined messages, sensors.

Literatura

- [1] Android (operating system), Datum pristupanja: svibanj 2013.
[http://en.wikipedia.org/wiki/Android_\(operating_system\)](http://en.wikipedia.org/wiki/Android_(operating_system))
- [2] Bluetooth, Datum pristupanja: travanja 2013.
<http://en.wikipedia.org/wiki/Bluetooth>
- [3] Android package index, Datum pristupanja: travanj 2013.
<http://developer.android.com/reference/packages.html>
- [4] Android App Components, Datum pristupanja: travanj 2013.
<http://developer.android.com/guide/components/index.html>
- [5] Android design, Datum pristupanja: travanj 2013.
<http://developer.android.com/design/index.html>
- [6] Maras, T. – Naslov: Planer zadaća bespilotnih ronilica koje se sastoje od konveksnih mnogokutnih segmenata od kojih svaki sadrži uzorke naizmjeničnog pretraživanja, Diplomski seminar, Fakultet elektrotehnike i računarstva, Zagreb, 2012.
- [7] Guštin, M. – Naslov: Planer misije temeljen na Google Earth, Diplomski seminar, Fakultet elektrotehnike i računarstva, Zagreb, 2012.
- [8] Docklight, Datum pristupanja, svibanj 2013. <http://www.docklight.de/>