Integracija i primjena interneta stvari, ugrađenih sustava i mobilnih aplikacija

Jančić, Mislav

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mislav Jančić

Integracija i primjena interneta stvari, ugrađenih sustava i mobilnih aplikacija

DIPLOMSKI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mislav Jančić

JMBAG: 0016136166

Studij: ***Informacijsko i programsko inženjerstvo***

Integracija i primjena interneta stvari, ugrađenih sustava i mobilnih aplikacija

DIPLOMSKI RAD

Mentor:

Izv. prof. dr. sc. Zlatko Stapić

Varaždin, rujan 2023.

Mislav Jančić

Izjava o izvornosti

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

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Rad se bavi istraživanjem načina na koji se pametni telefoni povezuju ili mogu povezati s uređajima interneta stvari i ugrađenim sustavima, te usko vezano uz to – koja je uloga mobilnih aplikacija u IoT sustavima. Rad također komentira pojam ugrađenih sustava i odnos istog prema pojmu interneta stvari. Zbog boljeg razumijevanja domene IoT, ukratko su obrađene najznačajnije mrežne tehnologije za povezivanje IoT i ugrađenih uređaja. Opisane su i značajke pametnih telefona koje im omogućavaju da i sami preuzmu ulogu IoT uređaja u nekim slučajevima, a kao primjer toga obrađena je i jedna Android aplikacija. U nekim poglavljima komentirana je i sigurnost i privatnost popularnih IoT rješenja za unaprjeđenje domova. Kao kulminacija teoretskog dijela obrađene su tri referentne arhitekture sustava IoT u odnosu na ulogu pametnog telefona i pripadajuće mobilne aplikacije. U praktičnom dijelu opisan je stvarni problem koji je arhitekturalno riješen korištenjem koncepata IoT.

Ključne riječi: internet stvari, ugrađeni sustavi, pametni telefon, uloga mobilnih aplikacija; Arduino; bežična povezivost; plovilo; kormilo; autopilot

Sadržaj

1. Uvod

Tema ovog rada odnosi se na interakciju pametnih telefona, odnosno mobilnih aplikacija i ugrađenih sustava, uključujući internet stvari. Cilj je objasniti pojam ugrađenih sustava, njegov široki opseg, kao i pojam interneta stvari i njegov odnos sa pojmom ugrađenih sustava.

Pojam interneta stvari i njegova kratica IoT diže veliku buku u današnjem svijetu tehnologije. Pojavom 5G mreža u nedalekoj prošlosti i napretkom ostalih ključnih bežičnih tehnologija, ponajviše WiFi i Bluetooth, svijetu interneta stvari pridružuje se sve više raznolikih uređaja namijenjenih krajnjim korisnicima, najčešće u domeni pametnog doma.

Rad istražuje ulogu pametnih telefona bez kojih bi naši životi bili nezamislivi zbog njihove praktičnosti i svestranosti u mnogobrojnim aplikacijama, pa tako i u kontekstu interneta stvari, odnosno ugrađenih sustava. Kao dio istraživanja uloge mobilnih aplikacija objašnjeni su neki primjeri korištenja pametnih telefona gdje pametni telefon poprima razne uloge.

Osim praktičnog pogleda na uloge mobilnih aplikacija, rad se bavi i pogledom na sigurnost i privatnost korisničkih podataka, pogotovo u kontekstu kućnih IoT uređaja. Za poboljšanje kućnih IoT sustava opisane su neke dobre prakse, kao što je korištenje VPN-a i vlastitog poslužitelja umjesto oslanjanja na oblak proizvođača ili neke druge velike kompanije.

U trećem poglavlju objašnjena je uloga interneta stvari i (općenitije) ugrađenih sustava pri čemu je ključno razumijevanje domene djelovanja takvih sustava, a iz čega slijede hardverske značajke uređaja IoT (ugrađenih) sustava.

Četvrto poglavlje objašnjava što omogućava ulogu pametnih telefona i mobilnih aplikacija u IoT sustavima objašnjavajući hardverske i softverske značajke.

Peto poglavlje bavi se integracijom pametnih telefona s IoT sustavima putem mobilnih aplikacija. Zbog razumijevanja mogućnosti povezivanja u ovom poglavlju opisane su mrežne tehnologije, sigurnosni aspekti, načini na koje se najčešće postiže interakcija korisnika s IoT uređajima korištenjem pametnih telefona i na kraju jedan primjer kako se sam pametni telefon može iskoristiti kao IoT uređaj.

Iz istraživanja o primjenama pametnih telefona u IoT sustavima prepoznate su tri različite referentne arhitekture u odnosu na ulogu pametnog telefona, a za svaku je dan primjer i dijagram primjera kako bi se ideja iza arhitekture mogla lakše shvatiti i samim time prepoznati u drugim stvarnim primjerima. Time se bavi šesto poglavlje.

Praktični dio rada obrađuje rješenje stvarnog problema održavanja smjera plovidbe na manjem motornom plovilu. To je rješenje implementirano i korišteno na stvarnom plovilu, a u

ovom radu je opisano razmišljanje o razlozima za odabir tehnologija i prikazani su glavni dijelovi implementiranog rješenja. Praktični dio rada bio je koncipiran, dizajniran i razvijen prije početka pisanja ovog rada te je stoga i motivacija za pisanje ovog rada.

2. Metode i tehnike rada

U ovom dijelu rada objašnjene su metode prikupljanja informacija i tehnike kojima su informacije predstavljene. Osim toga, objašnjen je i proces pisanja rada.

Kao što je navedeno u prethodnom poglavlju, praktični dio rada bio je napravljen prvi. Na temelju njega dogovorena je tema rada i ostali detalji. Za izradu praktičnog dijela prvo je analiziran problem i istraženi su načini na koji bi se mogao riješiti. Zatim je odabran najbolji način uzevši u obzir potrebnu investiciju i jednostavnost implementacije. Slijedilo je detaljnije koncipiranje buduće implementacije i nabavljanje dijelova potrebnih za koncipiranu implementaciju. Dolaskom dijelova i nabavkom pomoćnih alata započeta je implementacija. Kad je implementacija dosegla dovoljno stanje dovršenosti, stavljena je na testiranje u stvarnim uvjetima i u ciljanom plovilu. Tijekom testiranja koje je trajalo više dana prepoznati su nedostatci i implementacija, a primarno algoritam u mobilnoj aplikaciji, bila je iterativno poboljšavana do zadovoljavajuće razine.

Razvoj praktičnog rješenja uključivao je električne poslove spajanja žica lemljenjem, pripreme nužnih dijelova u plovilu (oslobađanje prostora, lemljenje potrebnih vodova) i smještanja osjetljivih dijelova u odabrano kućište lijepljenjem. Kućište je trebalo prilagoditi bušenjem i oblikovanjem rupe za konektor. Dio rješenja koji se odnosi na pametni telefon uključivao je razvoj aplikacije za Android i razvoj algoritma autopilota.

Alati koji su korišteni u praktičnom dijelu uključuju, ali nisu ograničeni na: lemilicu, škare, kliješta za skidanje izolacije, izolacijsku traku, obična kliješta, digitalni multimetar. Za razvoj mobilne aplikacije korišten je Android Studio u Flatpak[1] distribuciji te (primarno) Android uređaj Honor 8 s verzijom sustava 8.1. Za repozitorij koda korišten je GitHub i samim time alat Git, odnosno njegova integracija u Android Studio IDE-u.

Alati koji su korišteni za pisanje rada su draw.io za kreiranje dijagrama, za upravljanje literaturom korišten je Mendeley.

Istraživanje literature u svrhu tematske analize vršeno je pomoću raznih internetskih pretraživača. Akademska literatura tražena je i pronalažena velikim dijelom korištenjem Bing Chat[2] AI aplikacije u Microsoft Edge pregledniku. Nijedan dio teksta u ovom radu nije generiran od strane tog ili nekog drugog AI alata.

[1] https://flatpak.org/

[2] https://www.bing.com/new

3. Uloga interneta stvari i ugrađenih sustava

Cilj ovog poglavlja je objasniti neke od glavnih značajki sustava koji se mogu kategorizirati u internet stvari ili pak ugrađene sustave. Time se stvara podloga za razumijevanje korisnosti takvih sustava, kao i osobina hardverskih sklopova koji čine takve sustave. Također će biti objašnjena razlika između interneta stvari i ugrađenih sustava, te razmišljanja autora o osobinama po kojima se može vršiti kategorizacija.

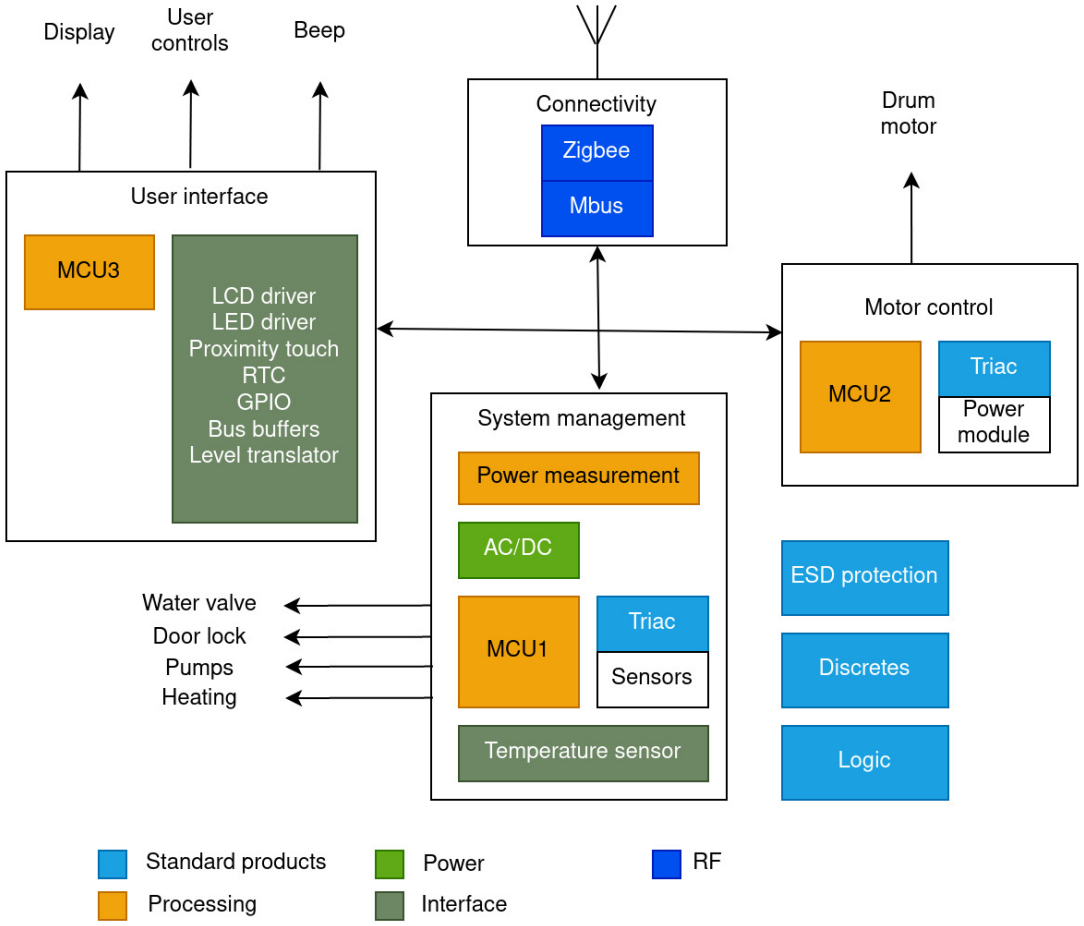
3.1. Razumijevanje domene djelovanja ugrađenih sustava i interneta stvari

Kad se govori o ugrađenim sustavima, treba definirati kategoriju i/ili domenu sustava o kojima je riječ iz jednostavnog razloga – pojam ugrađenih sustava iznimno je širok. Sam naziv *ugrađeni sustavi* (eng. *embedded systems*) kaže da se radi o sustavima koji su negdje ugrađeni i time ograničeni okolinom. Ugrađeni sustav je inženjerski artefakt koji uključuje računanje, a koji je podložan fizičkim ograničenjima (Henzinger et al, 2007). No, ta je definicija vrlo općenita pa tako u ugrađene sustave možemo svrstati veće i manje sustave, pa čak i njihove podsustave čime ugrađeni sustav može imati više ugrađenih (pod)sustava čija je kompleksnost niža, a specijaliziranost veća u odnosu na ugrađeni sustav kojem pripadaju. Osobna računala (eng. *personal computers*) mogla bi se nategnuto svrstati u tu definiciju, stoga je bolje nadopuniti definiciju ugrađenih sustava tvrdnjom da, uz to da je ugrađeni sustav elektronička oprema i da sadržava nekakav procesor, ugrađeni sustav je i dizajniran za specifičnu namjenu (De Micco et al, 2020). Tom tvrdnjom mogu se isključiti osobna računala jer su ona dizajnirana da budu za opću namjenu. Mogućnost obavljanja opće namjene dolazi od kombinacije hardvera i softvera.

Kratko razmišljanje o opširnosti pojma ugrađenih sustava dovoljno je da se vidi koliko su ugrađeni sustavi rašireni i svugdje oko nas, praktički u svakom imalo kompleksnom elektroničkom uređaju. Time je jasno da je internet stvari podskup ugrađenih sustava sa dodatnim minimalnim zahtjevima u njihovoj definiciji: povezivost. Prema nekim definicijama (Yang et al, 2017), internet stvari (IoT) je skraćeni naziv dobiven od izjave „Internet koji povezuje stvari“ pri čemu je internet ključna stvar koja povezuje bilo koji objekt u svrhu dijeljenja informacija. Kao i kod definicija ugrađenih sustava, vrlo je teško dobro definirati što je ustvari internet stvari, ali to ne znači da je teško i shvatiti. Uređaji koji se koriste u internetu stvari u velikom broju aplikacija izgrađeni su na namjenskom hardveru koji ima značajna ograničenja u pogledu performansi za obradu podataka. Razlog za to je kombinacija više

faktora: uloga koju sustav obavlja ne treba velike računalne resurse, cijena hardvera treba biti što niža, uređaj mora stati u ograničeni prostor, dostupnost i snaga električne energije za napajanje, itd.

Kao primjer ugrađenog sustava možemo uzeti perilicu rublja. Svaka moderna perilica rublja ima više modova operacije za različite kombinacije rublja i vrsta pranja. Radom perilice upravlja njezin pripadajući ugrađeni sustav koji sadržava softver dizajniran za modove operacije te perilice. Na dijagramu (slika 1) možemo vidjeti primjer ugrađenog sustava neke perilice rublja. Sam ugrađeni sustav sastoji se od više podsustava (na ovom primjeru tri) od kojih svaki ima svoj mikrokontroler (*MCU*). Zaduženja svakog podsustava su različita i međusobno komuniciraju. Ova perilica može se svrstati u internet stvari jer sadržava mrežnu povezivost i samim time vrlo vjerojatno različite značajke *smart home* uređaja.



Slika 1 Generički dijagram ugrađenog sustava jedne IoT perilice rublja[3]

[3] https://www.electronicproducts.com/nxps-white-goods-washing-machine-block-diagram/

3.2. Osobine i ograničenja namjenskog hardvera

Softver je vjerojatno najbitnija komponenta ugrađenih sustava jer definira ponašanje sustava i time obavlja njegovu svrhu. Osnovna komponenta bez koje softver ne služi ničemu je, naravno, hardver. Razlog zašto kažem da je softver vjerojatno najbitnija komponenta je to što se za hardverski dio može koristiti široki spektar rješenja. Staro stolno računalo uz dodatak (primjerice) određenog senzora i bežične ili čak obične žične mrežne povezivosti može u potpunosti obavljati brojne zadaće u domeni interneta stvari, pogotovo zbog velike modularnosti podržane standardnim sučeljima kao što su USB, PCIe i sl. Ipak, postoji par očitih razloga zašto takav pristup najčešće nije održiv. Prvi razlog je potreban prostor. Stolna računala, a čak i male verzije (Mini PC) zauzimaju dosta fizičkog prostora koji jednostavno nije dostupan u većini slučajeva. Hardverski zahtjevi operacija koje se procesiraju na ugrađenim sustavima obično nisu visoki nego su vrlo specifični i tijekom životnog ciklusa uređaja se ne mijenjaju uopće ili bar ne znatno. Stoga se može dobro odrediti kakav je hardver potreban za određeni ugrađeni sustav, a to je pogotovo bitno u masovnoj proizvodnji gdje se optimizacijom ukupno ostvari velika ušteda.

4. Značajke pametnih telefona u kontekstu interneta stvari

Pametni telefoni (eng. smartphones) već su duže vrijeme mnogima od nas važni kao odjeća - nikamo bez mobitela. Općenite sposobnosti prosječnog uređaja nije potrebno puno naglašavati. U ovom poglavlju bit će opisane hardverske značajke modernog pametnog telefona i neki načini na koje se te značajke mogu iskoristiti u kontekstu interneta stvari i ugrađenih uređaja.

4.1. Hardverske značajke pametnih telefona

Kod iskoristivosti pojedinog uređaja prvo je potrebno pregledati njegov hardver, jer ako je hardver nezadovoljavajuć, softver to najčešće ne može promijeniti. Ista stvar vrijedi i za pametne telefone. Hardverske značajke mogu se podijeliti na nekoliko kategorija:

- Dimenzije – postoje manji i veći uređaji, ali svima je zajednički plosnati četvrtasti oblik pogodan za stavljanje u male prostore kao što je džep hlača
- Autonomija – baterija je integralni dio svakog pametnog telefona, ovisno o korištenju, na jednom punjenju telefon može trajati od nekoliko sati do više dana, punjenje samo u vrijeme kad je električna energija dostupna (npr. solarno po danu) je u redu
- Povezivost – svaki moderni pametni telefon sadržava hardver za WiFi, Bluetooth, 4G/LTE, a čak i 5G je sve češći
- Senzori – svaki uređaj ima bar jedan mikrofon, par kamera, senzor blizine i osvjetljenja, senzore pokreta i mnoge druge
- Performanse – uređaji više klase imaju performanse procesora, memorije, grafike, pohrane... usporedive sa prijenosnim i stolnim računalima niže (nekad čak i srednje) klase, a sadržavaju i više specijalnih hardverskih akceleratora za različite funkcije kao što je zaključivanje kod umjetne inteligencije ili procesiranje signala kamere u visokim rezolucijama

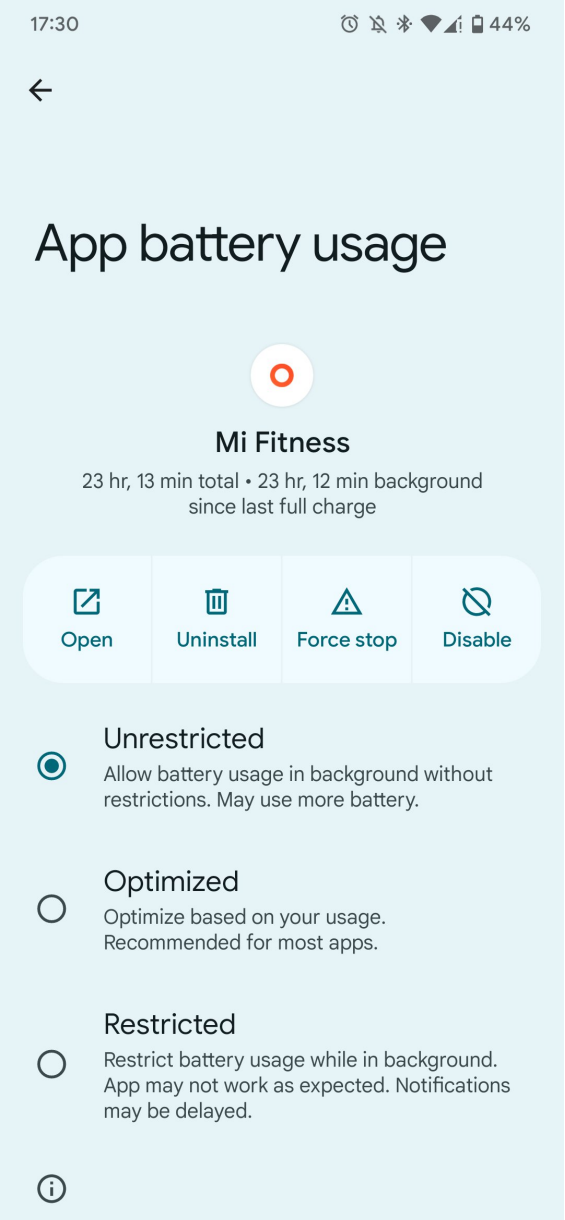
Ako gledamo na pametni telefon samo kao na sučelje između korisnika i IoT/ugrađenog sustava s kojim želi upravljati ili pregledavati podatke, obično je dovoljna samo internetska povezivost i određena mobilna aplikacija. Ipak, moderni pametni telefoni sposobni su za puno više od toga, pogotovo ako se uzme u obzir mogućnost korištenja izvan njihove primarne namjene kao osobnog uređaja kojeg nosimo svugdje gdje idemo. Kad zamijenimo naš pametni telefon s novim, stari vrlo često skuplja prašinu. S obzirom na navedene značajke

pametnih telefona, možemo zaključiti da su vrlo svestrani i kao takvi mogu dobro poslužiti u različitim ulogama vezanim za IoT – bilo to samo kao sučelje između korisnika i nekog ugrađenog sustava ili kao značajni dio rada ugrađenog sustava posuđujući senzorske i procesorske moći puno manje sposobnim ugrađenim uređajima.

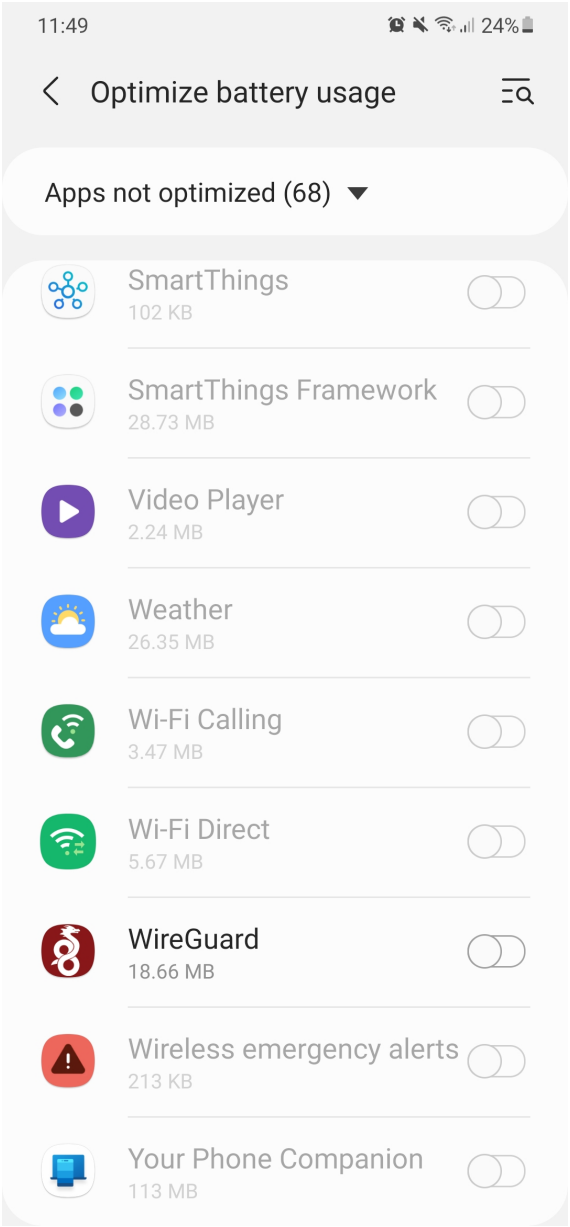
4.2. Softverske značajke pametnih telefona

Kako bi se hardver mogao iskoristiti potreban je zadovoljavajuć softver. Osnovna sotverska komponenta u kontekstu softverskih značajki zasigurno je operacijski sustav. On mora pružati podršku za korištenje dostupnog hardvera. Ta podrška u mobilnim operacijskim sustavima Android-u i iOS-u aplikacijama i njihovim razvojnim programerima pružena je putem definiranih API-ja za interakciju sa hardverskim i softverskim komponentama. Na taj način sučelje je standardizirano što olakšava razvoj i smanjuje mogućnost pogreške, ali i poboljšava sigurnost pružajući zaštitu od širenja malicioznih aktivnosti aplikacija tako što aplikacija ima vrlo ograničeni pristup koji joj može biti oduzet u bilo kojem trenutku bilo to automatski ili željom korisnika.

Ovisno o slučaju korištenja, mogu biti potrebne razne softverske značajke operacijskog sustava. Mrežna povezivost je često vrlo bitna. To je bitno u slučajevima korištenja gdje se pametni telefon koristi za upravljanje i/ili nadzor IoT uređaja, osobito u *Smart Home* domeni. Drugi slučaj korištenja je u tandemu s pametnim satovima/narukvicama gdje se takvi uređaji spajaju na pametni telefon putem Bluetooth veze i aplikacije koja stalno radi u pozadini. Rad u pozadini osigurava i veću potrošnju energije što uzrokuje i kraću autonomiju pametnog telefona. Zadatak razvojnih programera aplikacije je minimizirati potrošnju i što više koristiti API-je operacijskog sustava jer su oni tipično optimizirani za potrošnju energije, a pri tome omogućiti očekivano funkcioniranje aplikacije i samim time IoT uređaja (npr. pametni sat).



Slika 2 Postavke ograničenja pozadinske aktivnosti, Android 13



Slika 3 Postavke ograničenja pozadinske aktivnosti, Android 11+OneUI 3.1

Kod naprednijih korištenja pametnog telefona, obično za svrhe koje nisu tipično predviđene – kao što je korištenje samog pametnog telefona kao aktivnog IoT uređaja, bitne su softverske značajke koje podržavaju takve radnje. Primjer toga može biti dozvoljavanje aplikaciji da zadrži uređaj u aktivnom stanju. Sustav Android omogućava korisniku da takvo što dozvoli aplikaciji (slike 2 i 3). Osim toga, neke napredne funkcije mogu se postići samo s *root* ovlastima. Većina pametnih telefona sa sustavom Android dozvoljavaju korisniku da svojevoljno otključa *bootloader* čime ima opciju instalirati *root* pristup. U tom pogledu Android je puno fleksibilniji nego iOS, a iz tog razloga za Android postoje korisne aplikacije (kao što je aplikacija SensorServer) koje ga mogu pretvoriti u IoT uređaj.

Sustav Android je baziran na Linux jezgri pa stoga ima neke zanimljive značajke koje se mogu upotrijebiti i u kontekstu IoT – *chroot* i *proot.* U oba slučaja radi se o svojevrsnom kontejneru u kojem se na pametnom telefonu sa Android OS-om može instalirati i pokrenuti ugniježđeni operacijski sustav također baziran na Linux jezgri. Ovdje se ne radi o virtualizaciji, već o korištenju zajedničke jezgre na sličan način kao što to čine Docker kontejneri. Tako se na telefone s Android-om može instalirati vrlo funkcionalan Ubuntu sustav s dostupnim Ubuntu paketima i aplikacijama.

5. Integracija mobilnih aplikacija i IoT uređaja

Povezivost ugrađenih uređaja koji čine internet stvari jedno je od njihovih osnovnih obilježja. Ta povezivost postiže se različitim kombinacijama softverskih i hardverskih protokola, a odabir se temelji na svrhi uređaja, potrebama za mrežnim kapacitetom i dometom i naravno, dostupnim energetskim budžetom. Osim toga, bitna je i jednostavnost integracije u ekosustav zajedno s ostalim uređajima.

5.1. Mrežne tehnologije interneta stvari

Hardverski gledano, povezivost se dijeli na žičnu (eng. Wired) i bežičnu (eng. Wireless). Iako se ovdje odnosim na hardver, svako hardversko mrežno sučelje sadržava određeni, vrlo specifični softver poznat pod engleskim nazivom *firmware* koji je tvornički zapisan u čip i obično se ne mijenja. Naravno, mrežna sučelja su i sama ugrađeni sustavi dizajnirani za obavljanje jednog posla – pružanja mrežne povezivosti.

5.1.1. Bluetooth

Jedna od bežičnih tehnologija koja se koristi na najvećem broju uređaja za spajanje raznolikih drugih uređaja. Osnovne odlike Bluetooth tehnologije su niska potrošnja energije u stanju povezanosti i mnoštvo uređaja koji ga podržavaju. Niska potrošnja energije velikim dijelom postignuta je kratkim dometom i niskom propusnosti (eng. Bandwidth) u odnosu na druge tehnologije (npr. WiFi ili LTE). Tipični domet (prije Bluetooth verzije 5) iznosio je 10 metara, a maksimalni domet iznosi stotinjak metara, ali takvi uređaji su u manjini. Bluetooth se prvo pojavio davne 1994. godine kao rana specifikacija nadolazeće bežične revolucije. Do verzije 5, jedna od glavnih mana Bluetooth tehnologije bila je mogućnost povezivanja samo dva uređaja istovremeno. Dolaskom verzije 5, broj spojenih uređaja postaje virtualno neograničen (iako često pametni telefoni i drugi uređaji imaju ograničenje na pet istovremeno spojenih uređaja). U slijedećoj tablici nalazi se usporedba Bluetooth verzija i drugih sličnih bežičnih tehnologija:

Značajka	Bluetooth Classic	Bluetooth 4.x	Bluetooth 5	IEEE 802.15.4 — ZigBee	IEEE 802.11ah — HaLow
Radio frekvencija	2400 - 2483.5	2400 - 2483.5	2400 - 2483.5	868.3, 902–928,	900

(MHz)				2400–2483.5	
Domet (m)	Do 100	Do 100	Do 200	Do 150	Do 1000
Tehnika srednjeg pristupa (medium access control)	Frequency hopping	Frequency hopping	Frequency hopping	CSMA/CA	Restricted access window
Nominalni protok podataka (Mb/s)	1-3	1	2	0.02-0.25	0.15-7.8
Latencija (ms)	<100	<6	<3	<4	~1000
Mrežna topologija	Piconet, scatternet	Star-bus, mesh	Star-bus, mesh	Mesh	Star-bus
Multihop rješenje	Scatternet	Da	Da	Da	Do 2 skoka
Koncept profila	Da	Da	Da	Da	ne
Broj čvorova (nodes)	7	Neograničeno	Neograničeno	Neograničeno	Neograničeno
Veličina poruke (bytes)	Do 358	31	255	100	100
Certifikacijsko tijelo	Bluetooth SIG	Bluetooth SIG	Bluetooth SIG	ZigBee Alliance	IEEE

Tablica 1 Usporedba Bluetooth značajki (Collotta et al, 2018, p. 126)

Bluetooth 5 podržava LE (*Low Energy*) tip veze s propusnosti od 500 kb/s i 125 kb/s sa hadrverskom podrškom za ispravljanje grešaka (FEC – *forward error correction*) što znači da

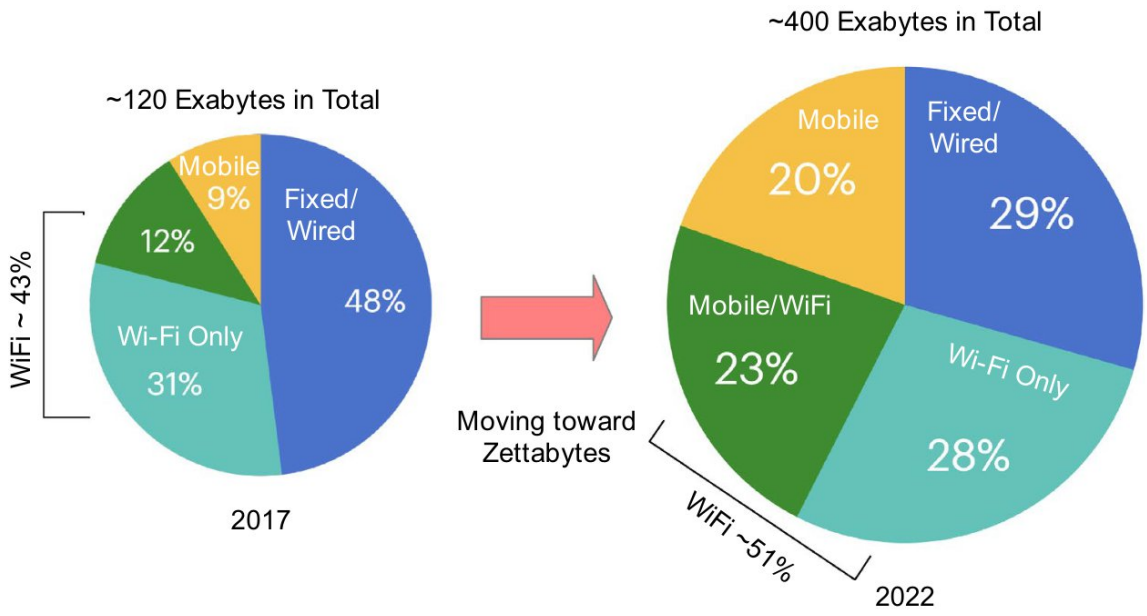
se razvojni programer koji koristi Bluetooth za svoju aplikaciju ne treba brinuti za stabilnost veze i greške koje se mogu dogoditi u prijenosu. Prednost LE moda operacije za primjenu u IoT aplikacijama je vrlo niska potrošnja energije postignuta niskom brzinom prijenosa podataka. Za IoT taj je kompromis tipično potpuno prihvatljiv jer najčeće nije potrebna velika brzina veze kakvu zahtjeva primjerice strujanje audio zapisa u stvarnom vremenu. Time Bluetooth LE nije namijenjen korištenju npr. bežičnih slušalica već uređaja kao što su pametni satovi i narukvice koji primaju tekstualne notifikacije, a trajanje baterije između punjenja im treba biti i više od 7 dana.

Još jedna značajka Bluetooth verzije 5 je podrška za *mesh* umreženost. Iako taj dio nije uključen u originalnu Bluetooth 5 specifikaciju (nego je došlo kasnije), ta značajka omogućava Bluetooth mreže slične drugim tehnologijama kao što su ZigBee, a povećavajući općenitu interoperabilnost s drugim uređajima zbog raširenosti Bluetooth prijemnika.

5.1.2. WiFi

WiFi nije potrebno puno predstavljati. Ova se tehnologija nalazi posvuda i postala je sinonim za pristup internetu. Malo je reći da je WiFi tehnologija već davno znatno promijenila način na koji povezujemo naše uređaje na internet, fleksibilnost koju to donosi i druge stvari omogućene korištenjem te bežične tehnologije.

Manje poznat, ali još uvijek često korišten alternativni naziv za WiFi je WLAN. To je kratica koja u slobodnom prijevodu znači *bežična lokalna meža* (Wireless Local Area Network). LAN se odnosi na lokalnu mrežu (najčešće) nekog prostora koja omogućava različitim kompatibilnim i spojenim uređajima da međusobno komuniciraju unutar te mreže, a da izvana budu nedostupni. Ta se umreženost može postići kablovima koji pružaju veću brzinu i stabilnost veze, ali se većina uređaja ipak spaja bežičnim putem zbog znatno veće fleksibilnosti i jednostavnosti korištenja u odnosu na žične veze. Na slijedećoj slici (Pahlavan et al, 2021) nalazi se usporedba korištenja razlčitih mrežnih tehnologija godine 2017. i godine 2022. Pri čemu je jasan trend povećanja korištenja bežičnih tehnologija u odnosu na žične čak i u vrlo nedavnoj prošlosti.



Slika 4 Usporedba korištenja bežičnih tehnologija (Pahlavan et al, 2021, p. 5)

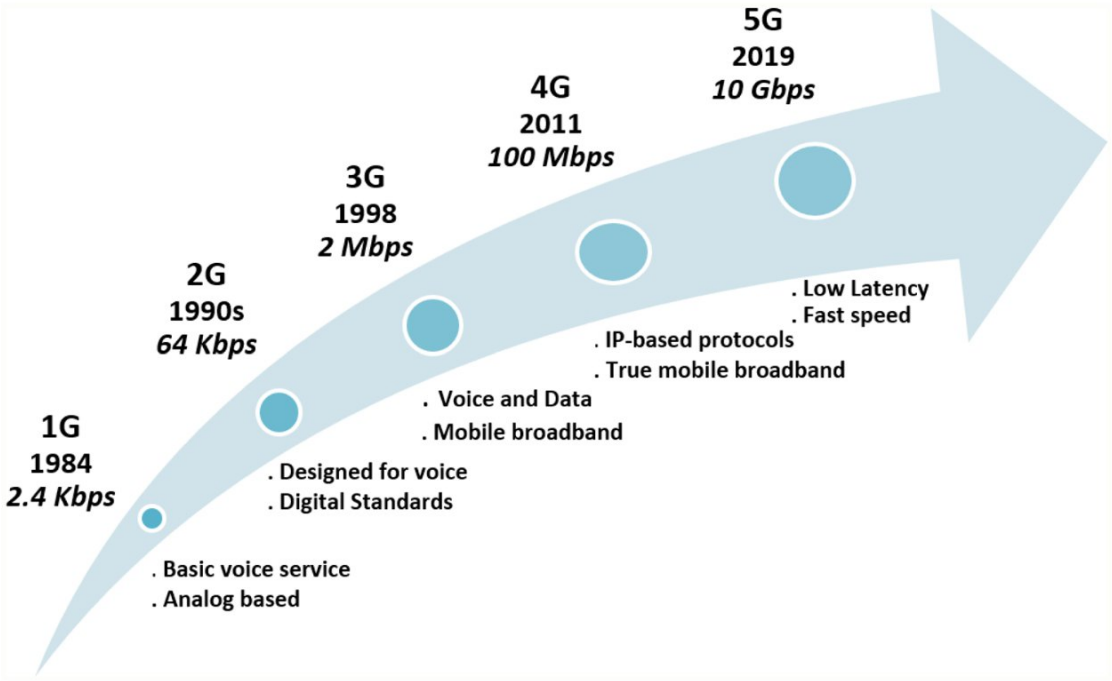
U kontekstu interneta stvari, WiFi je također vrlo često korištena tehnologija iz sličnih razloga kao i Bluetooth, ali s bitnim razlikama koje je bitno razmotriti. WiFi se nalazi u skoro svakom domu pa je u gradovima mnogo pristupnih točki, a većina uređaja koje koristimo (pametni telefoni, računala i sl.) podržavaju upravo WiFi tehnologiju kao primarni način povezivanja s internetom/mrežom. Kao i Bluetooth, WiFi se koristi za prijenos podataka, no taj je prijenos u većini slučajeva postignut primjenom IP, odnosno internetskog protokola koji uključuje korištenje IP adresa (IPv4 ili IPv6), TCP, UDP i ostalih mrežnih protokola ovisno o svrsi. Samim time, povezivanje s ostalim uređajima (npr. poslužiteljima) ne zahtijeva dodatan hardver ili softver koji bi u nekom obliku bio potreban ako bi se uređaji interneta stvari povezivali drugim tehnologijama.

Velika brzina prijenosa podataka koju nudi WiFi omogućava širok spektar IoT rješenja čiji su podatkovni zahtjevi veći nego što nisko-energetske tehnologije mogu pružiti. Primjer takvog IoT uređaja je bilo što s kamerom čije se fotografije ili videozapisi spremaju ili obrađuju negdje drugdje. Ta brzina prijenosa dolazi s cijenom – utrošak energije znatno je veći, a pogotovo kad se WiFi iskorištava do svojih granica.

Moderni WiFi prijemnici (standardi 6 i 6E) mogu doseći brzine iznad 1GBps što je iznad 500 puta više nego što Bluetooth 5 može postići. Udaljenost koju doseže WiFi s frekvecijama od 5GHz otprilike je 15 metara, pa je za veliko područje potrebno imati više pristupnih točaka koje su s ostatkom mreže dobro povezane (u idealnom slučaju kablom). U frekvenciji od 2.4GHz domet se može znatno povećati, ali pada brzina veze.

5.1.3. 4G (LTE)

Četvrta generacija mobilne mreže koja dovršena 2011. godine kao veliki korak naprijed naspram prethodne 3G tehnologije, a pogotovo u odnosu na 2G i 1G. Prva generacija (1G) bila je analogna, druga generacija (2G) je digitalna i donijela je SMS poruke. Treća generacija poboljšala je mnogo toga uključujući i podatkovni prijenos (*mobile broadband*) omogućujući mobilnim uređajima da pristupe internetu. 4G donosi veliku promjenu tako što uvodi IP (*internet protocol*). Uređaji se spajaju na pristupne točke koje su u pravilu u vlasništvu neke telekomunikacijske tvrtke. Te su pristupne točke poprilično velike u usporedbi s onima za WiFi i u gradovima se često nalaze na krovovima zgrada, a drugdje na tornjevima. U tome je jedna od glavnih praktičnih razlika u odnosu na WiFi, Bluetooth i druge, 4G (i samim time 5G) mreža ne može biti u vlasništvu korisnika (ne može biti lokalna) i zahtijeva SIM karticu i pretplatu na uslugu.



Slika 5 Generacije mobilnih mreža (Attaran, 2021, p. 5978)

Primjena mobilnih mreža u IoT je svejedno važna jer, ovisno o aplikaciji, pruža internetsku povezivost s relativno velike udaljenosti, bežično. Također je bitno naglasiti da se u pokretu prebacivanje od jedne do druge pristupne točke obično ne može ni primijetiti što otvara bezbroj mogućnosti za IoT aplikacije ugrađene u vozila.

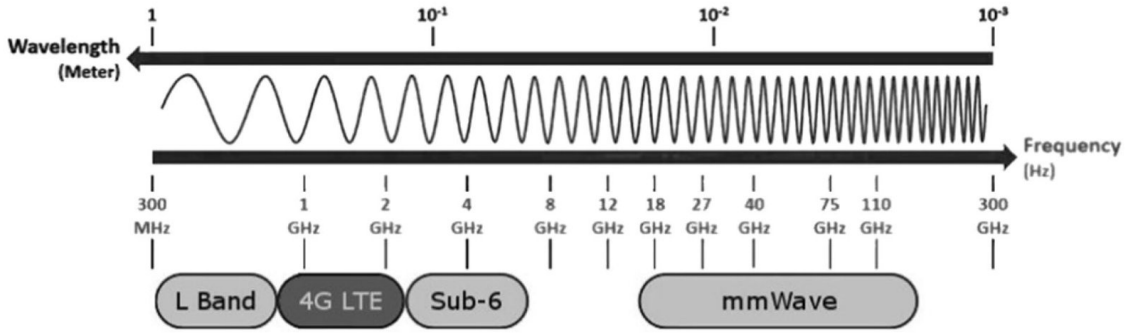
U kontekstu pametnih telefona, oni sadržavaju integrirani 4G modem zbog čega su sposobni služiti kao 4G modem raznim drugim uređajima koji nemaju taj hardver. Ta je funkcionalnost vrlo poznata pod nazivom WiFi Hotspot ili Tethering. Dijeljenje veze pametni

telefoni nude na tri načina – WiFI, Bluetooth i USB (pri čemu se u USB načinu rada predstavljaju kao USB Ethernet adapter).

Problem koji se može pojaviti, pogotovo u gušće naseljenim mjestima je zagušenje 4G mreže. Postoji određeni broj uređaja koji istovremeno mogu crpiti promet s nekog 4G tornja pa brzina veze u tim slučajevima bude poprilično loša. Taj broj istovremenih veza nije ni pretjerano velik, ovisno o kapacitetu specifičnog tornja to može biti u rasponu od stotinjak do tristotinjak istovremeno spojenih korisnika.

5.1.4. 5G

Najnovija generacija mobilnih mreža rješava ili smanjuje neke od glavnih nedostataka 4G tehnologije pri čemu je najveći napredak postignut u efikasnosti (značajno smanjenje potrošnje energije), brzini veze (propusnost i latencija značajno poboljšani) i broj uređaja koji mogu biti istovremeno spojeni. Zbog tih značajki, 5G se često spominje kao tehnologija interneta stvari jer time omogućava pristup internetu znatno većem broj uređaja. Na slici (slika 6) vidi se raspon frekvencija koji koriste različite verzije 5G mreža, kao i 4G:



Slika 6 Frekvencijski pojasi 5G i 4G mreža[4]

Neke od glavnih značajki 5G tehnologije su sljedeće (Attaran, 2021, p. 5981):

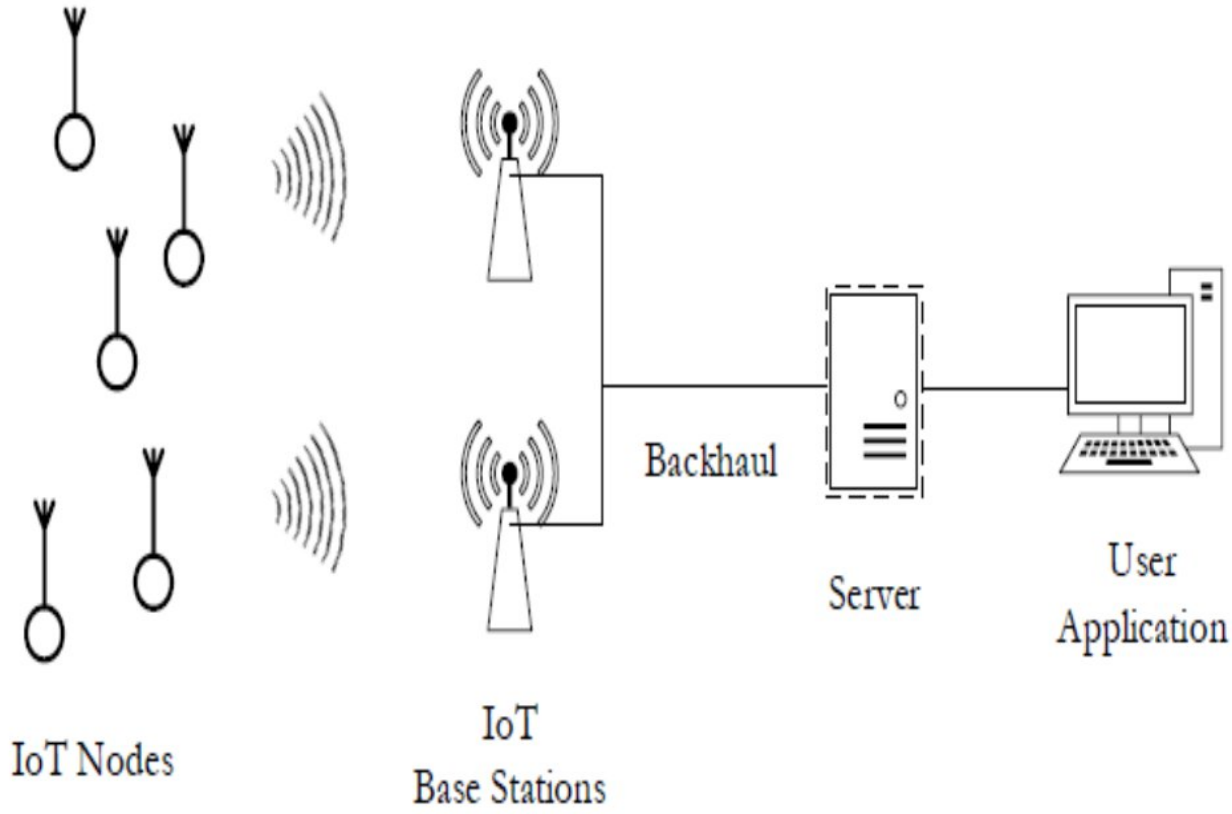
- *mmWave* – tehnologija kratkog dometa i velike brzine koja koristi visoki raspon frekvencija (17-110GHz)
- *Sub-6GHz* – očekuje se da će većina 5G uređaja koristiti frekvencije u WiFi rasponu, korisno za korištenje u zatvorenim prostorima
- *Low-band* – operacija u niskim frekvencijama manjim od 800MHz, namijenjeno velikim udaljenostima
- *Beamforming* – ključna tehnologija koja omogućuje usmjeravanje signala uređajima i samim time drastično efikasniju komunikaciju

[4] Robert Triggs, androidauthority.com

- *Massive MIMO* – podaci se prenose korištenjem više antena za efikasniju komunikaciju s velikim brojem spojenih uređaja

5.1.5. LoRa i LoRaWAN

Ova je tehnologija razvijena s ciljem vrlo niske potrošnje energije u svrhu komunikacije IoT uređaja. Kao takva ima i vrlo nizak protok podataka. LoRa je bežična mreža koja koristi CSS (*Chirp-Spread-Spectrum*) modulaciju za optimizaciju dometa i brzine u različitim uvjetima. LoRa koristi ISM frekvencije od 433MHz, 868MHz ili ovisno o legalnim uvjetima područja. Na slijedećoj slici (slika 7) nalazi se tipična arhitektura LoRa mreže:



Slika 7 Arhitektura LoRa mreže (Ntseane et al, 2019, p. 1)

Na slici je vidljivo nekoliko stvari. Prvo, IoT čvorovi na lijevoj strani moraju se povezati s baznim stanicama (sredina-lijevo) kako bi pristupili internetu, odnosno kako bi se ostvarila komunikacija s vanjskim svijetom.To je zato što je LoRa mreža kojom komuniciraju IoT čvorovi nekompatibilna sa standardnim načinima spajanja na internet ili druge mreže jer ne koristi IP protokol, a nije niti standardna za druge primjene kao npr. Bluetooth.

Primjena LoRa tehnologije ima smisla kod IoT aplikacija koje zahtijevaju vrlo nisku potrošnju energije i nemaju velike podatkovne zahtjeve, tipično su to senzorni IoT uređaji.

Ipak, problem s kojim se susreću korisnici LoRa mreže je njezina skalabilnost. Primjerice, dodavanjem novih mjernih uređaja za veću gustoću podataka može prouzročiti interferenciju na LoRa mreži čime se smanjuje njezina efikasnost (Ntseane et al, 2019).

5.1.6. Serijske veze - USB, RS-232, RS-485

Većina IoT aplikacija ne koristi USB ili serijske veze u svojem radu kao način komunikacije s vanjskim svijetom. Ipak, prilikom razvoja, postavljanja ili dijagnosticiranja ugrađenih sustava (u što spada IoT) često se koriste takve veze za pristup funkcijama niske razine, primjerice snimanje programa na memorijsku pohranu dotičnog uređaja. Arduino ploče koriste baš USB za instaliranje programa koji se instalira korištenjem kompatibilnog razvojnog okruženja. Uz to ide i čitanje ispisa programa (log u konzoli). Osim toga, USB pruža i napajanje za Arduino te se često koristi samo za napajanje. RS-232 i slični protokoli češći su u industrijskm ugrađenim sustavima, ali služe istoj svrsi (RS-232 se također može koristiti za napajanje). Kod serijskih konekcija su bitne dvije veze – Tx i Rx. Tx označava *Transmit* vezu za slanje, a Rx predstavlja *Receive* vezu za primanje podataka. Podaci koje jedna strana šalje putem Tx konektora druga strana prima na Rx konektoru, i obrnuto.

Iako se serijske veze prilikom rada IoT i ugrađenih sustava tipično ne koriste izravno za komunikaciju s vanjskim svijetom, one se ipak često koriste za komunikaciju između različitih komponenti. Primjer toga u praktičnom dijelu ovog rada nalazi se između Bluetooth modula i ploče s mikrokontrolerom *Arduino Uno* .

5.1.7. Ethernet

Kao i WiFi, ethernet ne treba posebno predstavljati. Radi se o žičnoj vezi za slanje i primanje paketa IP protokola. U načelu pruža isto što i WiFi – mrežnu povezivost. Osim očitih nedostataka žične veze, ona ima i važne prednosti koje vrijedi iskoristiti ako postoji mogućnost. Bez posebnog redoslijeda, prva prednost je stabilnost veze. Korištenjem kvalitetnih kablova može se postići vrlo stabilna hardverska veza koja nije sklona interferenciji kao bežične veze s kojima je bitno uzeti u obzir okolne bežične mreže i njihove frekvencijske pojase. Druga važna prednost je efikasnost prijenosa. Korištenje kablova, pogotovo optičkih, donosi puno veću energetsku učinkovitost u usporedbi sa bežičnim vezama, a razlika se samo povećava kako udaljenost raste. Treća prednost je brzina prijenosa. Žične veze imaju puno veću propusnost podataka koja ostaje vrlo konzistentna i na velikim udaljenostima.

U stvarnom svijetu, žične veze koriste se u tandemu s bežičnim vezama čime se postiže dobar kompromis, a količina kablova nije prevelika, kao ni udaljenost i interferencija za bežične veze.

5.1.8. ZigBee i Z-Wave

ZigBee i Z-Wave dva su otvorena standarda. Z-Wave je bio zatvoren, ali je izvorni kôd za implementacije na uređajima od nedavno otvoren javnosti. Oba protokola u širokoj su uporabi među *smart-home* IoT uređajima namijenjenima širokoj populaciji. Te uređaje proizvode velike tvrtke kao što su Google, Samsung, Xiaomi i mnoge druge.

ZigBee koristi tri frekvencijska pojasa - 868MHz, 915MHz, 2.4GHz. Kao i LoRa, fokus je na niskoj potrošnji energije i velikom dometu (Danbatta et al, 2019). Ovisno o prostoru, ZigBee uređaji imaju domet od 10 do 100 metara. Time je ovaj protokol primarno namijenjen korištenju u zatvorenim prostorima, specifično u domovima. Kako bi se pametni uređaji mogli koristiti, potrebno je imati ZigBee *hub* koji upravlja vezama, slično kao usmjernik (*router*) za WiFi mrežu.

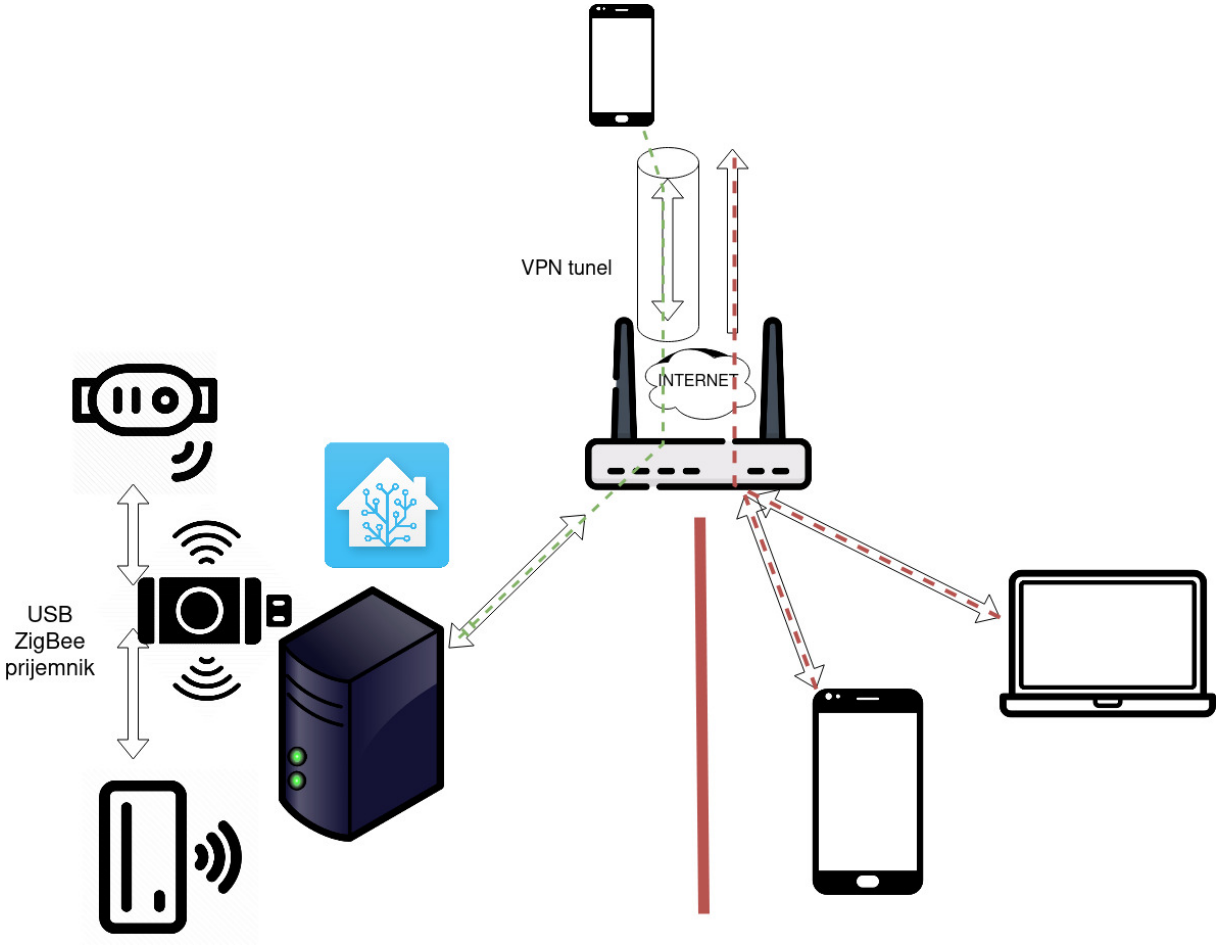
S druge strane se nalazi Z-Wave koji je također česta pojava u IoT svijetu. Da je prije bio otvoren, vjerojatno bi bio znatno popularniji jer ima neke vrlo dobre značajke. Svaki Z-Wave uređaj može biti i *repeater* čime se efektivni domet mreže može dobro skalirati. Samim time olakšva mrežnu redundantnost zbog čega je poželjniji u komercijalnim aplikacijama (Danbatta et al, 2019). Z-Wave radi na frekvenciji od 900MHz i ima domet od tridesetak metara.

5.2. Sigurnost i privatnost

Povezivanje bilo kojeg uređaja na internet značajno mijenja model prijetnji i potrebu za zaštitom u odnosu na striktno *offline* uređaje. Sigurnosne zakrpe i softverska podrška vrlo su bitni aspekti svakog sustava koji se spaja na internet. Česta pojava kod IoT uređaja za kućnu upotrebu je da se sami IoT uređaji spajaju na internet i pristupaju proizvođačevu oblaku – spajaju se na javne poslužitelje treće strane. Već je poznat veliki broj iskorištavanja sigurnosnih propusta kod takvih rješenja (Schuster et al, 2022). Svaki IoT uređaj prikuplja nekakve podatke ili ima neku funkciju koju izvršava. Jedna primjena IoT kod kuće je paljenje grijanja ili hlađenja unaprijed. Korisnici putem aplikacije mogu po zimi uključiti grijanje da ih dočeka topli dom. Na tom primjeru, maliciozna osoba s neovlaštenim pristupom dobivenim hakiranjem usluge proizvođača može uključiti grijanje i prouzročiti vlasnicima potencijalno veliki trošak i gubitak vremena. No, što ako hakeri dobiju pristup nadzornim kamerama ili sigurnosnom sustavu? Tada mogu imati detaljni uvid u sve što ukućani rade. Kod takvih *cloud* rješenja, čak i ako nisu hakirani, pružatelj usluge može imati isti uvid koji i hakeri mogu imati, a tko garantira da neki zaposlenik pružatelja usluge nema maliciozne namjere?

Korištenje gotovih rješenja od popularnih kompanija (Samsung, Xiaomi...) pruža jednostavnost zbog čega su takva rješenja vrlo popularna, osobito među korisnicima koji vole tehnologiju ali si ne žele „zaprljati ruke“ i složiti rješenje koje bi pružalo puno bolju sigurnost i privatnost. Kako bi se to postiglo, potrebno je ne koristiti *cloud* rješenje na koje se spajaju svi IoT uređaji, odnosno gdje se nalazi svo upravljanje njima. Rješenje je taj *cloud* imati odvojen i osiguran, u vlastitoj kontroli. Kad već imamo IoT uređaje u svojem domu, nije puno dodati i poslužitelj na kojem će biti softver za upravljanje tim IoT uređajima.

Dobra praksa je koristiti VPN (ili slična rješenja) za spajanje na lokalnu mrežu izvana. Alternativa je otvaranje administrativnog sučelja javnom internetu što je veliki sigurnosni rizik. Osim toga, dobro je napraviti i segregaciju mreže tako da su osobni uređaji ukućana na odvojenoj mreži koja nema izravni pristup IoT i poslužiteljskoj mreži, i obrnuto. To se može postići korištenjem vatroštitnog (*firewall*) softvera kao što je *OpenWRT* ili *pfSense*. Pojednostavljeni dijagram mreže kako bi ona mogla izgledati može se vidjeti na slijedećoj slici (slika 8):



Slika 8 Primjer segregacije IoT mreže u svrhu sigurnosti

Na dijagramu se s lijeve strane može vidjeti mreža na kojoj se nalazi *Home Assistant* poslužitelj kojem se izvana može može pristupiti samo putem VPN-a (bez VPN-a djeluje firewall). Mreža

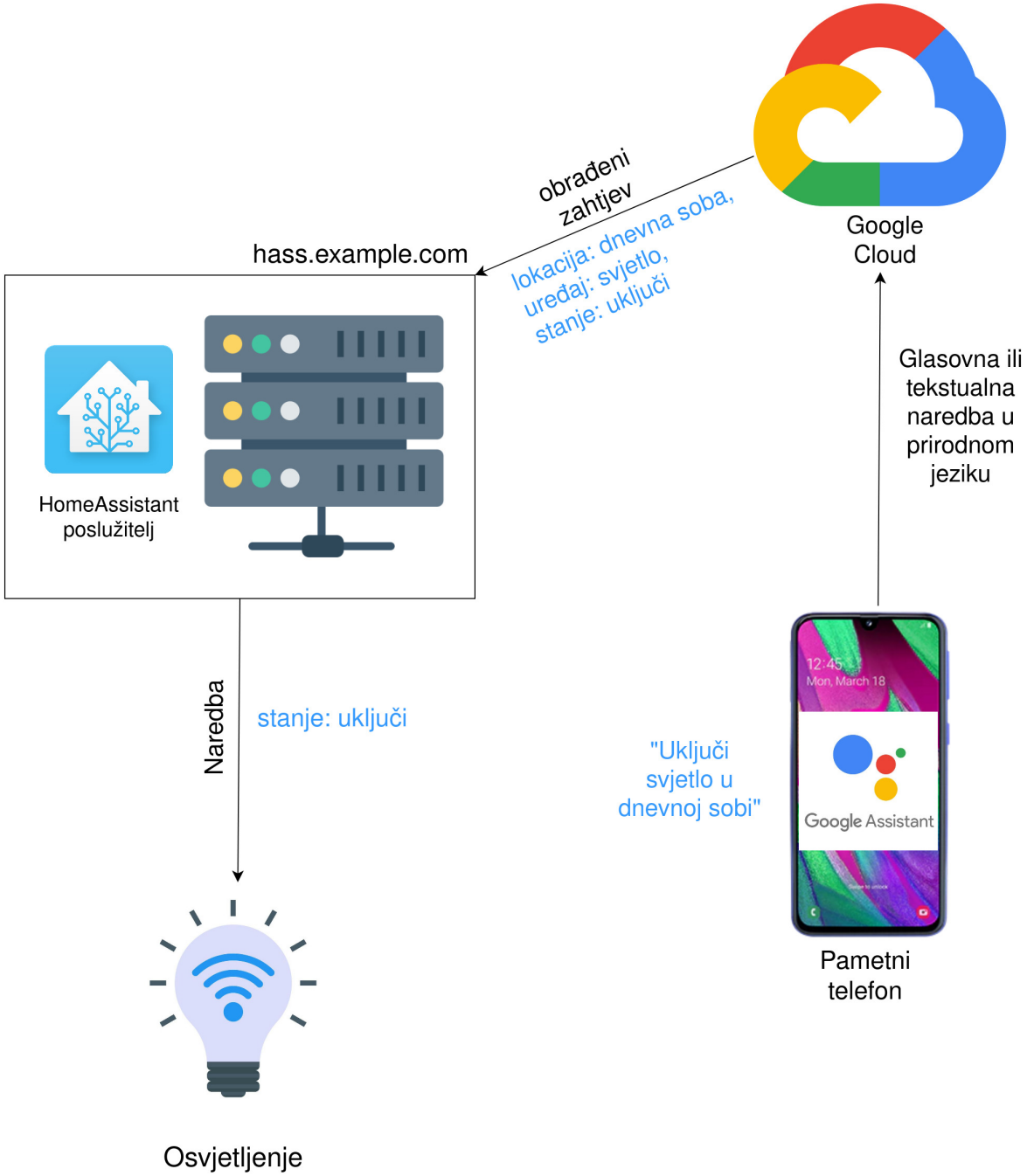
na kojoj je poslužitelj je odvojena od mreže na kojoj su ostali uređaji, a poslužitelj sa IoT uređajima komunicira putem USB adaptera koji pretvara npr. ZigBee komunikaciju u nešto s čime Home Assistant može raditi. Na taj način se postiže i znatno bolja interoperabilnost jer Home Assistant podržava puno više različitih protokola i implementacija nego što podržavaju proizvođači kada se koristi njihov oblak. Time se izbjegava i *vendor lock-in*, odnosno izbjegava se ovisnost o jednom i samo jednom proizvođaču koji je svoje proizvode napravio nekompatibilnima sa proizdovima drugih proizvođača kako kupci ne bi kupovali tuđe proizvode. U ovoj konfiguraciji svi podaci mogu ostati na lokalnom poslužitelju što je puno bolje za sigurnost i privatnost.

5.3. Postizanje interakcije između korisnika i IoT

S obzirom na sveprisutnost pametnih telefona, najlakši način za povezati korisnike s njihovim uređajima je putem mobilnih aplikacija i usluga kojima su te aplikacije sučelje. Ovisno o slučaju korištenja i svrhi specifičnog IoT ili ugrađenog uređaja, poželjno je odabrati odgovarajuću metodu komunikacije. Mnogobrojne primjene interneta stvari ovise o kvalitetnoj korisničkoj interakciji s uređajima koji čine internet stvari. Interakcija je potrebna i za upravljanje radom sustava, pregled zdravlja sustava, ažuriranje softvera (firmware), ali i za pregled podataka koje prikuplja neki uređaj. Uređaji interneta stvari često su *headless*, što znači da nemaju ugrađeno korisničko sučelje za upravljanje nego ovise o uređajima/sustavima treće strane, a to mogu biti i pametni telefoni, tj. pripadajuće aplikacije.

U arhitekturi IoT sustava gdje se IoT uređaj spaja na internet i korisnik njime upravlja aplikacijom koja za funkcionalnost upravljanja IoT uređajem/uređajima također zahtijeva pristup internetu postoji više vrlo popularnih rješenja. Razlog popularnosti je to što većina velikih proizvođača prodaje uređaje koji se spajaju na njihov *cloud*, a korisnicima je jednostavnije za osposobiti i započeti s korištenjem. Nedostaci takvih rješenja spomenuti su drugdje (*vendor lock-in*[5], sigurnost, privatnost), no jedna od prednosti je podrška velikih kompanija, a u nekim slučajevima i podrška za ekstenziju funkcionalnosti proizvodima drugih velikih tvrtki. Najveći primjer toga je korištenje AI asistenata kao što su *Google Assistant* i *Amazon Alexa*. Povezivanjem IoT uređaja s tim uslugama može se postići glasovno upravljanje radom IoT uređaja. Pri tome je jedan od najjednostavnijih načina za korištenje te metode upravo putem pametnog telefona. Na sreću, *Google Assistant* može se koristiti i sa lokalnom (ali javno dostupnom) instancom *Home Assistant*-a, i to uz ne puno truda. Na dijagramu (Slika 9) prikazan je tok komunikacije od naredbe Google asistentu u prirodnom jeziku do promjene stanja IoT kućnog uređaja:

[5] https://en.wikipedia.org/wiki/Vendor_lock-in



Slika 9 Primjer korištenja Google asistenta

Prvi korak je iznošenje zahtjeva u prirodnom jeziku korištenjem mobilne aplikcaije Google Assistant[6] koja je predinstalirana na većini uređaja s Android-om. Uz neke iznimke, procesiranje prirodnog jezika (bilo iz glasovne ili tekstualne naredbe) obavlja se na Google-ovim poslužiteljima, a ne na mobilnom uređaju korisnika. Google poslužitelji kao rezultat procesiranja imaju konkretne determinističke podatke o akciji koja se želi izvršiti. Instanca Home Assistant poslužitelja koja se u primjeru nalazi na javnoj internetskoj adresi

[6] https://assistant.google.com/

hass.example.com registrirana je na Google Assistant i od njega prima rezultat procesiranja, a iz tog rezultata Home Assistant dalje upravlja ponašanjem IoT uređaja.

Primjer na dijagramu nije jedini način za postići takvu interakciju. Google Assistant moguće je spojiti i lokalno[7]. Takva postava vrlo je vjerojatno privatnija i sigurnija, ali i potencijalno brža nego kad se koriste Google poslužitelji za procesiranje unosa.

Osim Google Asistenta tu su i Amazon Alexa i IFTTT[8] potonji od kojih je servis koji može agregirati funckionalnosti više različitih servisa i omogućava automatizaciju drugih cloud servisa. IFTT se pojednostavljeno može opisati kao Home Assistant za cloud servise. Ovisno o tipu IoT uređaja i njegove primjene, korisnička interakcija može biti vrlo različita – ekrana i gumbića na samom IoT uređaju, preko serijskih kablova i blizinskih bežičnih veza (Bluetooth) do upravljanja preko IP mreža čak vrlo često uključujući internet i cloud servise.

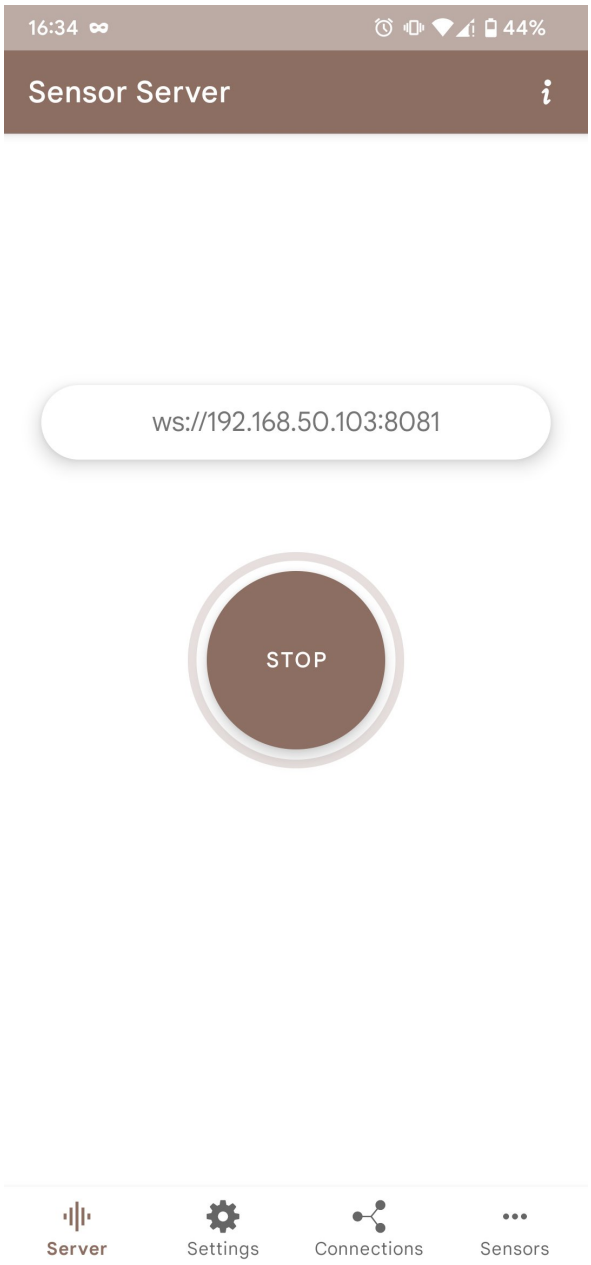
5.4. Pametni telefon kao IoT uređaj

Pametni telefoni imaju raznolike senzore, sposoban hardver i (ovisno o slučaju) proširiv softver zbog čega pametni telefoni u odgovarajućoj situaciji i sami mogu biti IoT uređaji. Osim toga, dostupni su u velikom broju i lako je nabaviti ili iskoristiti stariji uređaj za neku IoT svrhu. Sposoban hardver omogućuje i zahtjevnije operacije kao što je prepoznavanje objekata na slikama i videozapisima koji mogu doći s kamere samog uređaja, ili pak prepoznavanje govora/ostalih zvukova korištenjem ugrađenih mikrofona. Praktički svi novi pametni telefoni već više godina imaju namjenski hardver dizajniran za zaključivanje temeljem modela umjetne inteligencije, stoga se najzahtjevnije procesiranje može obaviti izravno na uređaju što smanjuje potrebu za mrežnim performansama. Brojni senzori pametnih telefona mogu se iskoristiti individualno ili kombinirano, s lokalnom obradom na uređaju ili slanjem podataka na neki centralni poslužitelj. Naravno, pametni telefon ne može služiti kao IoT uređaj za bilo koju svrhu, ali svestranost današnjih pametnih telefona omogućava im da budu korišteni za drugačiju svrhu (primjerice nakon što se više ne koriste za primarnu svrhu).

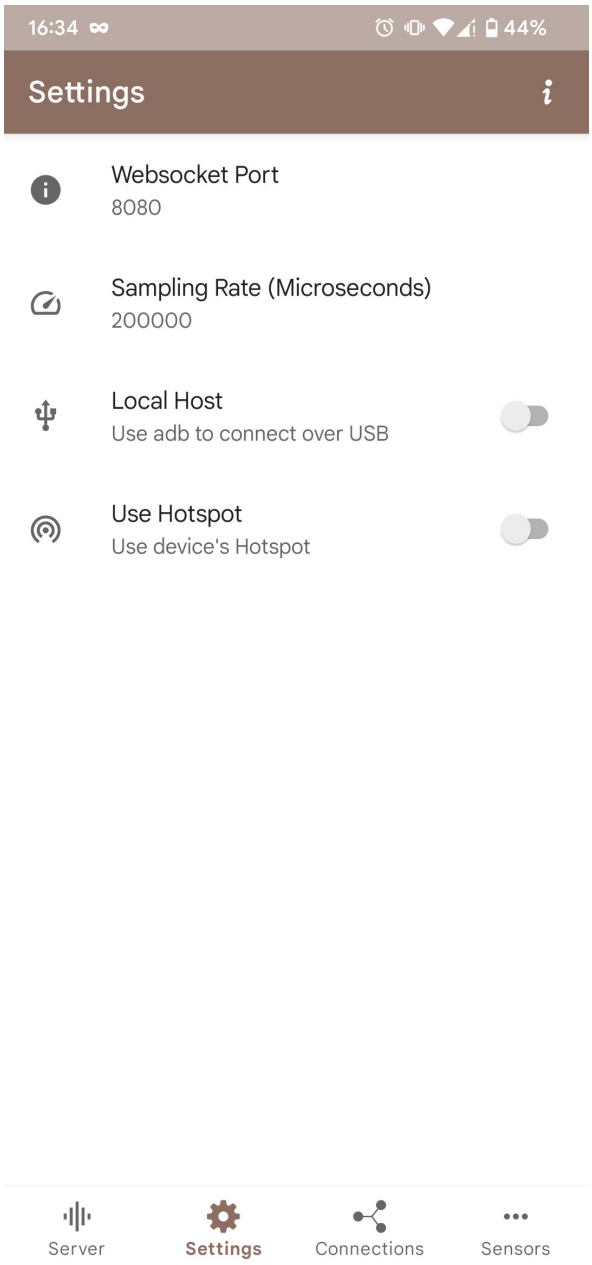
[7] https://www.home-assistant.io/integrations/google_assistant/
[8] https://ifttt.com/

5.4.1. Aplikacija SensorServer za Android

Kao primjer načina na koji se IoT funkcionalnost može postići kod pametnog telefona temeljenog na sustavu Android može se uzeti aplikacija otvorenog kôda *SensorServer*[9] koja očitanja sa senzora pametnog telefona može prenositi drugim uređajima putem *websocket* veze, u JSON formatu. Aplikacija se u vrijeme pisanja ovog rada može preuzeti sa *GitHub releases* stranice ili sa *F-droid*[10] repozitorija aplikacija.

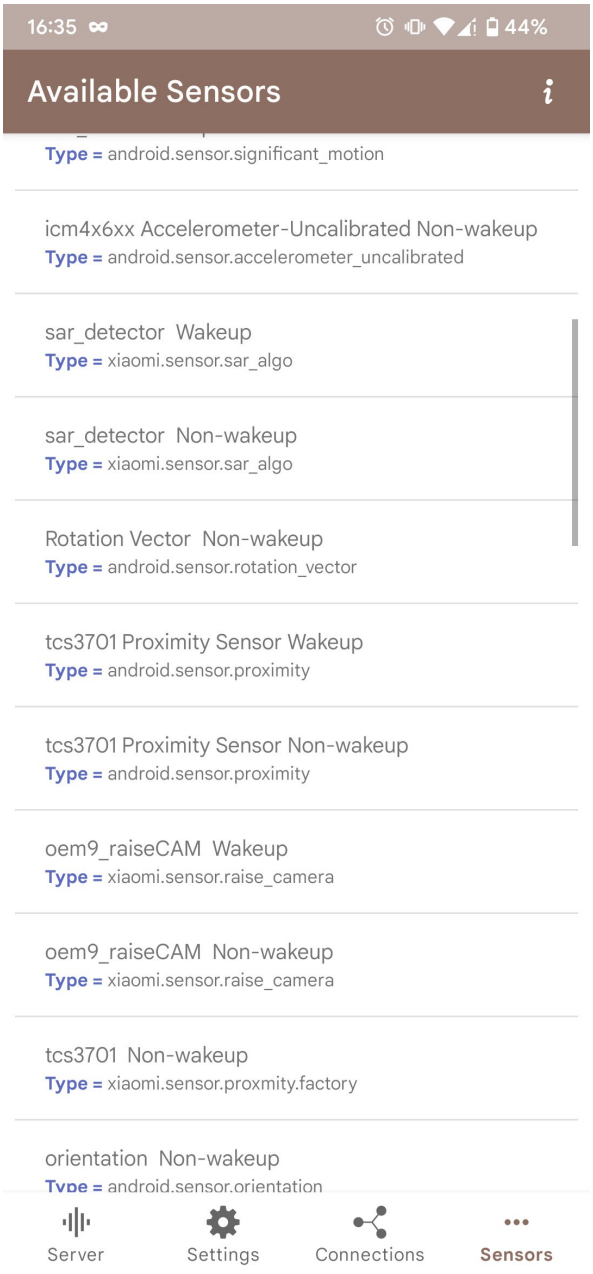


Slika 10 Početni zaslon i pokrenut websocket server

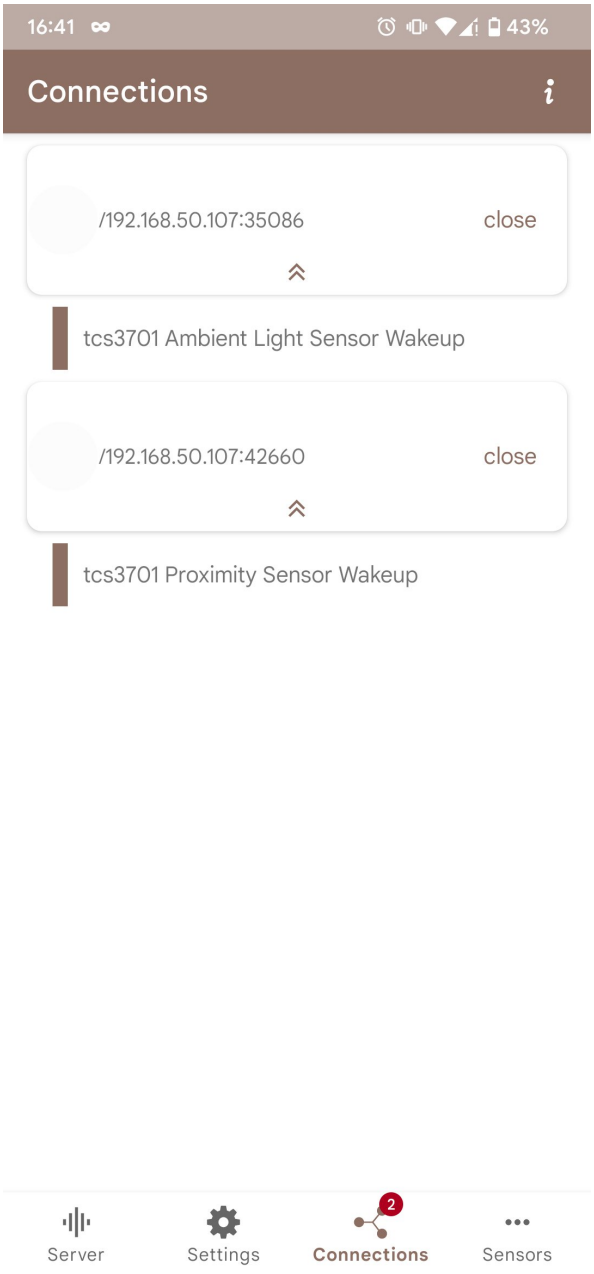


Slika 11 Postavke aplikacije

[9] https://github.com/umer0586/SensorServer
[10] https://f-droid.org/en/



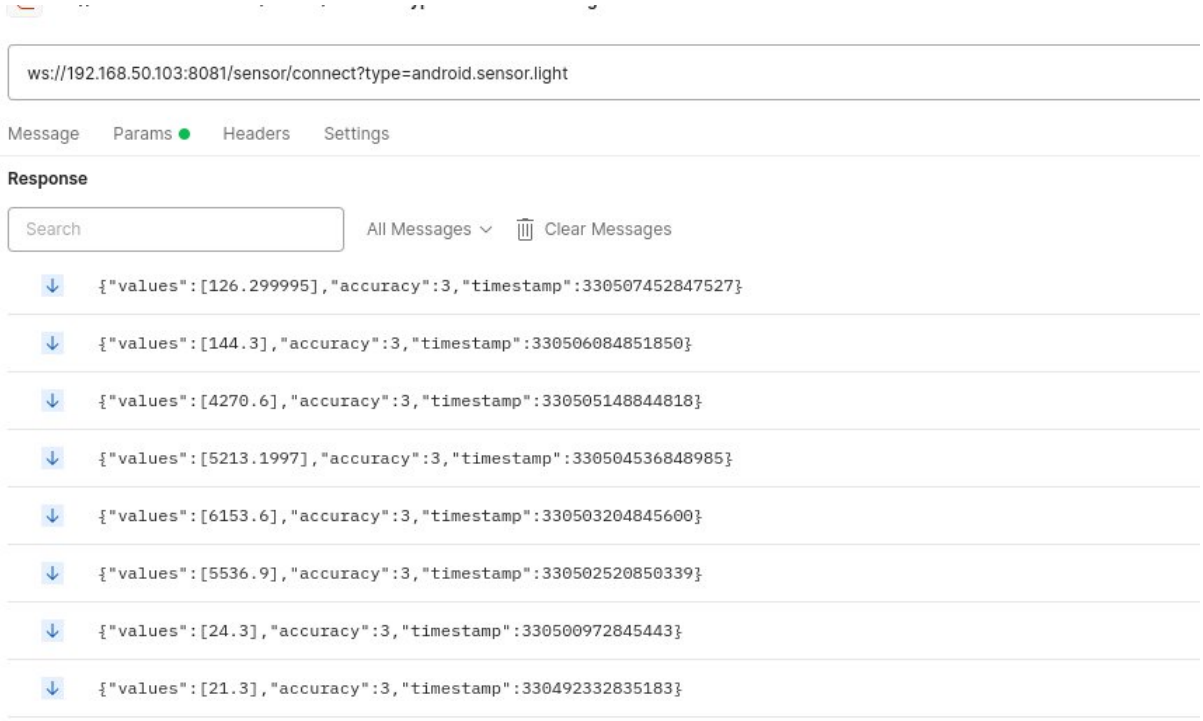
Slika 12 Lista dostupnih senzora s identifikatorima



Slika 13 Popis aktivnih websocket veza

Na slici (slika 10) može se vidjeti adresa na koju se klijenti s iste mreže (za pristup s druge mreže može se koristiti *VPN*) mogu spojiti na *websocket* poslužitelj koji pruža aplikacija. Na slici 11 vidljiv je zaslon s postavkama aplikacije. Moguće je promijeniti port websocket poslužitelja i interval čitanja senzora (i slanja očitanja na *websocket*). Osim toga, dostupne su dvije postavke koje se odnose na načine na koje se klijenti mogu spojiti na *websocket*. Osim običnog načina gdje je pametni telefon spojen na WiFi i ima svoju IP adresu na lokalnoj (i/ili *VPN* mreži), veza se može ostvariti i spajanjem na *adb* (*Android Debug Bridge*) USB kablom ili spajanjem na *WiFi hotspot* samog uređaja. Na slici 12 nalazi se zaslon s popisom svih dostupnih senzora koje aplikacija nalazi. Obično se mogu očitavati i senzori temperature, no

oni se odnose na temperaturu hardverskih komponenti uređaja pa su od ograničene relevantnosti ako se pokušava utvrditi temperatura zraka u okolini uređaja. Svaki senzor ima svoju identifikaciju, npr. *android.sensor.light*, što se koristi kod spajanja na websocket za taj senzor pri čemu bi potpuna adresa (IP adreu i port treba prilagoditi) mogla biti: *ws://192.168.50.103:8081/sensor/connect?type=android.sensor.light.* Ta se adresa za potrebe testiranja može unijeti u alat *Postman,* kao na primjeru na slici 14:



Slika 14 Spajanje na websocket u alatu Postman[11]

Poruke klijentu spojenom na određeni websocket pristižu poruke u definiranom intervalu. Kao što je vidljivo na slici 14, poruke su u JSON formatu i sadržavaju vrijednost (ili vrijednosti) očitane na senzoru, preciznost očitanja i vremensku oznaku svakog očitanja. Te su informacije dostatne kako bi se konstruirala slika vrijednosti kroz vrijeme. Ako se promotre vrijednosti iz svake poruke, mogu se vidjeti velike razlike. Jačina svjetlosti simulirana je korištenjem svjetiljke usmjerene prema mjestu senzora svjetla na uređaju.

Senzori dostupni za korištenje variraju od uređaja do uređaja, no evo nekih standardnih senzora:

- *Accelerometer* – mjeri akceleraciju u prostoru, vrijednosti za X, Y i Z
- *Gyroscope* – mjeri rotaciju, koristi se za automatsku orijentaciju zaslona
- *Magnetic field* – detektor metala, u slobodno vrijeme kompas

[11] https://www.postman.com/

- *Light* – senzor osvjetljenja prostora, koristi se za adaptivno osvjetljenje zaslona
- *Orientation* – obično virtualni senzor baziran na žiroskopu, za orijentaciju uređaja
- *Gravity* – virtualni senzor baziran na senzoru ubrzanja (*accelerometer*)
- *Linear acceleration* – virtualni senzor koji fltrira očitanja senzora ubrzanja za linearno ubrzanje
- *Rotation vector* – još jedan virtualni senzor koji filtrira očitanja sa žiroskopa
- Significant motion – virtualni senzor koji se aktivira kad *accelerometer* zabilježi jaki pokret
- *Step counter/Pedometer* – virtualni senzor koji filtrira očitanja sa senzora ubrzanja i bilježi pokret ako odgovara koraku.
- *Proximity* – udaljenost, koristi se za određivanje je li mobitel na uhu tijekom poziva ili ne

Razni uređaji mogu imati neke specifične senzore (npr. senzor pritiska zraka, pogodan i za mjerenje nadmorske visine i sl.), no iz navedenih senzora može se vidjeti da su neki od njih samo virtualni senzori koji su bazirani na manje fizičkih senzora (ili čak kombinaciji fizičkih senzora). Takvi senzori smanjuju kompleksnost razvoja aplikacija jer procesiranje i filtriranje nekih senzora ostavljaju sustavu.

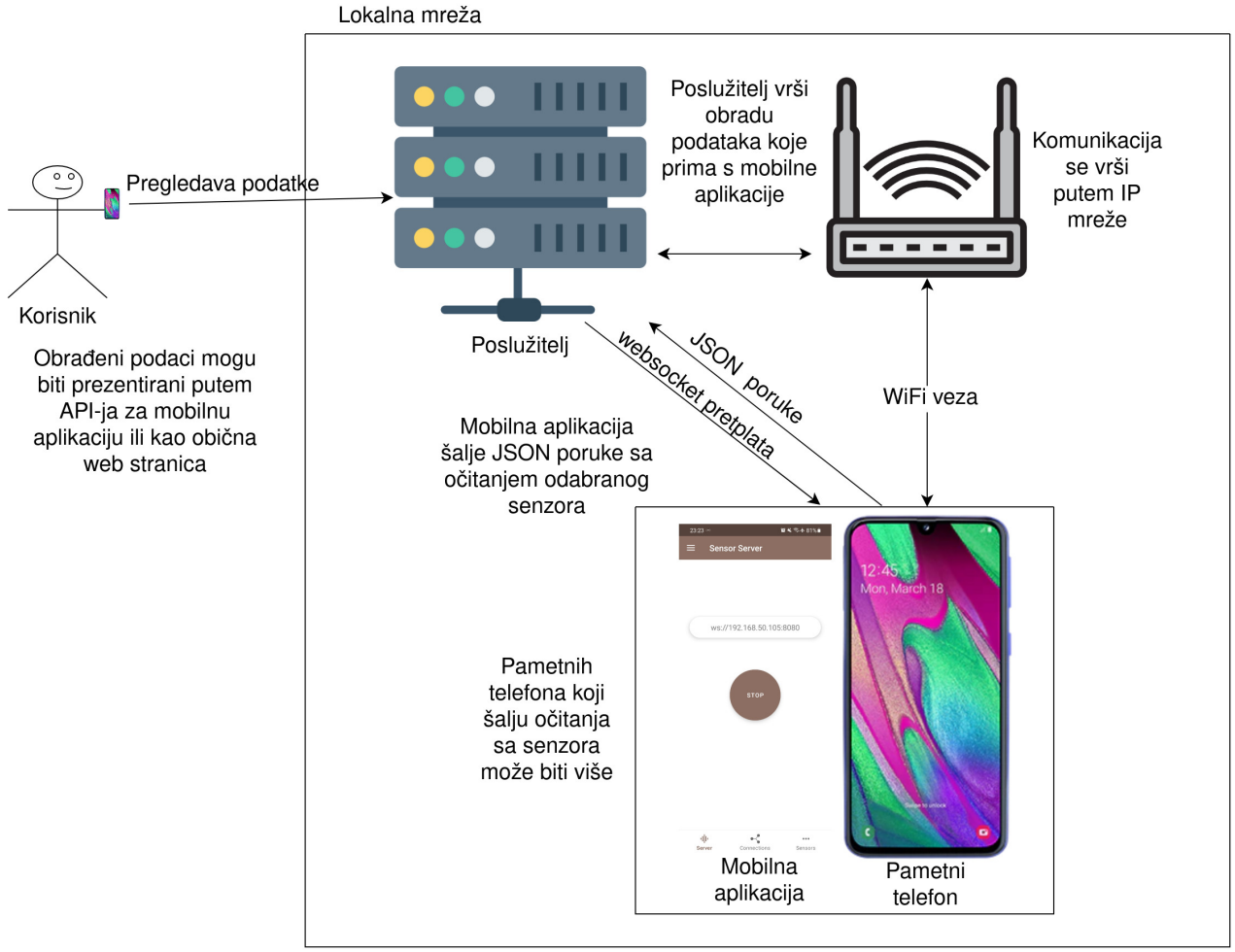
Senzor ubrzanja može se koristiti za detekciju otvaranja vrata ili za detekciju vibracija. Senzor osvjetljenja može se koristiti za automatsko paljenje/gašenje osvjetljenja, senzor udaljenosti upotrebljiv je za akcije koje se mogu ostvariti mahom ruke i može se koristiti zajedno sa senzorom osvjetljenja za naprednije geste. *Significant motion* virtualni senzor može se primjerice koristiti kao alarm za prenaglo otvaranje ili zatvaranje nečega. Za svaku od ovih ideja mogu se koristiti namjenski IoT uređaji, ali za kućne potrebe i za prototipiranje, jedan ili više pametnih telefona svakako mogu dobro poslužiti.

6. Referentne arhitekture M&IoT sustava

Postoje brojne primjene tehnologija IoT, a većina primjena IoT na neki način je vezana za pametne telefone. Ta vezanost za pametne telefone (i samim time vrlo često za dedicirane mobilne aplikacije) nije iznenađujuća, ali ju svakako treba prepoznati i uključiti u dizajn IoT sustava. Stoga, možemo definirati termin *M&IoT* koji označava ulogu pametnih telefona i mobilnih aplikacija u radu IoT sustava. U terminu „M“ se odnosi na korisnički dio koji korisniku ili korisnicima omogućava prikladnu komunikaciju s IoT stranom sustava. U ovom poglavlju bit će objašnjene neke arhitekture IoT sustava u odnosu na njihovu ovisnost i interakciju s mobilnim aplikacijama, odnosno pametnim telefonima. Arhitekture navedene i opisane u nastavku objašnjene su na primjerima, ali opisuju općenitu ideju iza arhitekture svakog primjera. Moguće su različite implementacije koje se drže iste pozadinske ideje arhitekture. Svaku varijaciju implementacije nemoguće je općenito opisati pa su korišteni konkretni primjeri zbog lakšeg razumijevanja.

6.1. Pametni telefon je IoT uređaj, M=IoT

Ova arhitektura koristi mobilnu aplikaciju i pametni telefon koji ju izvršava kao IoT uređaj čime je uloga „M“ ovdje neuobičajena. Takva primjena pametnog telefona i mobilnih aplikacija u kontekstu interneta stvari pogodna je za niskobudžetna kućna rješenja i za privremena rješenja kao što je prototipiranje. Ipak, ta uloga nije zanemariva i dobro je znati da je ostvariva u određenim uvjetima.



Slika 15 Dijagram - primjer arhitekture M=IoT

Na slici (slika 15) nalazi se dijagram s primjerom jednog sustava gdje se jedan (ili više) pametni telefon uz pomoć senzora i mobilne aplikacije koristi kao IoT uređaj. U primjeru arhitekture korištena je prethodno obrađena aplikacija *SensorServer* koja kreira *websocket* poslužitelj na uređaju na kojem je pokrenuta i klijenti na mreži mogu se pretplatiti na poruke za senzor s kojeg žele dobivati očitavanja. U primjeru na slici uređaj je spojen na lokalnu mrežu putem WiFi veze, ali su dostupne i druge opcije (USB kabel, mobilna mreža + VPN, hotspot) koje nisu prikazane na dijagramu. Očitanja senzora konzumira poslužitelj. Poslužiteljska aplikacija pretplaćena je na jedan ili više *websocket*-a putem kojih *SensorServer* aplikacija šalje JSON poruke. Poslužitelj može vršiti nekakvu obradu dobivenih podataka i pohranjivati podatke i rezultate, te ih pripremiti za prezentaciju krajnjem korisniku. Krajnji korisnik u tom primjeru može pregledavati obrađene podatke s poslužitelja (prezentacija podataka u mobilnoj aplikaciji ili web aplikacija/stranica). U proširenju ovog primjera mogu se automatizirati razne akcije i korisniku dati ručna kontrola, a odluke može bazirati na predstavljenim podacima koji su prikupljeni sa senzora jednog ili više pametnih telefona.

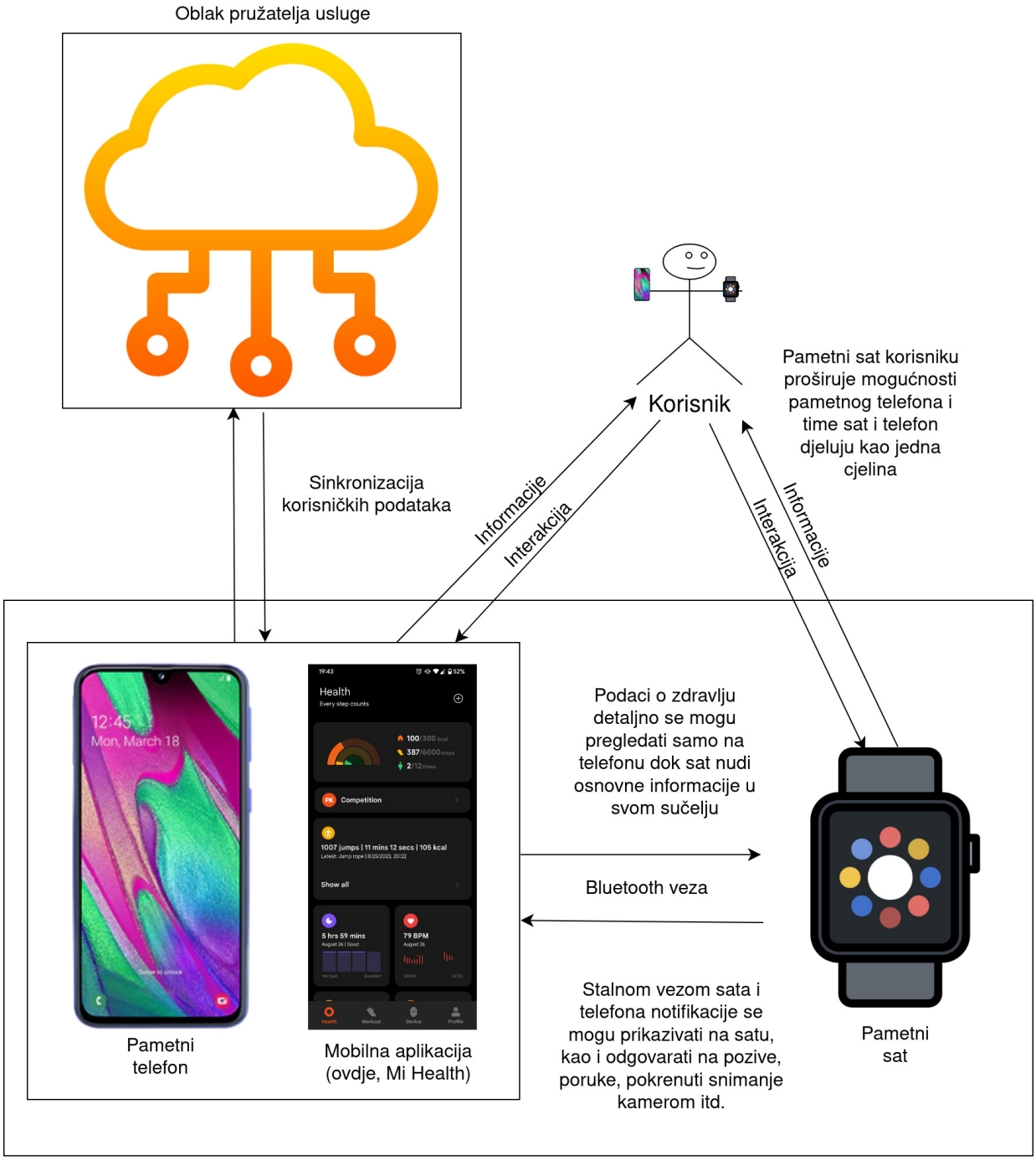
U primjeni M=IoT arhitekture nije nužno imati poslužitelj kao što je navedeno u primjeru. Za neke aplikacije može biti dovoljan samo pametni telefon. Također, može se koristiti više pametnih telefona koji međusobno komuniciraju, a ako situacija dozvoljava, nema razloga ni da se sam pametni telefon ne koristi još i kao poslužitelj koji vrši obradu podataka i pristup istima. Na Android-om pogonjenim telefonima korištenje *chroot* ili *proot*[12] kontejnera omogućava pokretanje brojnih alata i programa dostupnih na raznim distribucijama Linux-a (npr. Ubuntu Server). Mogućnosti se mogu još znatno povećati ako se omoguće *root* ovlasti.

6.2. Simbioza pametnog telefona i IoT uređaja, M+IoT

Simbioza znači suživot u kojem dva ili više živih bića žive zajedno i korist je obostrana, u kontrastu sa parazitima gdje samo jedna strana ima korist, a druga ima gubitak. U kontekstu mobilnih aplikacija i interneta stvari, simbioza se može interpretirati kao svojevrsna ovisnost IoT uređaja o pametnom telefonu, odnosno relevantnoj aplikaciji i vezi između njih dvoje. Takva interakcija korisniku nudi proširenje funkcionalnosti pametnog telefona, a IoT uređaj je bez njega značajno manje koristan ili čak beskoristan. Također, podrazumijeva se i fizička blizina oba uređaja – korištenje Bluetooth tehnologije za komunikaciju.

Na dijagramu (slika 16) nalazi se primjer vrlo široko korištene primjene IoT uređaja koji se može uzeti kao glavna primjena M+IoT arhitekturalnog koncepta. Ovaj primjer predstavlja cijelu kategoriju *wearable* uređaja koji s nekim varijacijama imaju osnovni skup značajki kako je prikazano na dijagramu. U ovom slučaju, pametni sat koristi Bluetooth tehnologiju za svoju vezu s pametnim telefonom. Pametni telefon za povezivanje s većinom pametnih satova/narukvica mora imati instaliranu pripadajuću aplikaciju za taj *wearable* uređaj. Početno uparivanje i spajanje se također vrši unutar aplikacije, kao i izmjena raznih postavki pametnog sata. Tipičan glavni razlog za korištenje pametnog sata je nadgledanje zdravlja i fizičke aktivnosti. To obično uključuje mjerenje otkucaja srca, kisika u krvi; prepoznavanje tipa fizičke aktivnosti (npr. brojanje koraka) i slično. Pametni sat samostalno procesira podatke sa svojih senzora i privremeno ih pohranjuje u svoju lokalnu memoriju. Kad se nakupi dovoljno podataka ili ako korisnik to zatraži, ti se podaci sinkroniziraju s pametnim telefonom nakon čega se u aplikaciji mogu vidjeti detaljni izvještaji o aktivnosti dok se na tipičnom pametnom satu mogu pogledati izvještaji s najbitnijim informacijama.

[12] https://wiki.termux.com/wiki/PRoot



Slika 16 Primjer arhitekture M+IoT

Što se tiče proširenja funkcionalnosti pametnog telefona, pametni sat može se koristiti kao dodatni način za prikazivanje važnih informacija tako da korisniku bude jednostavnije ili da ranije bude obaviješten. Vibracija uređaja na ruci vrlo se lako primijeti, dok vibracija pametnog telefona može biti propuštena, a korištenje zvuka nije prigodno u mnogim

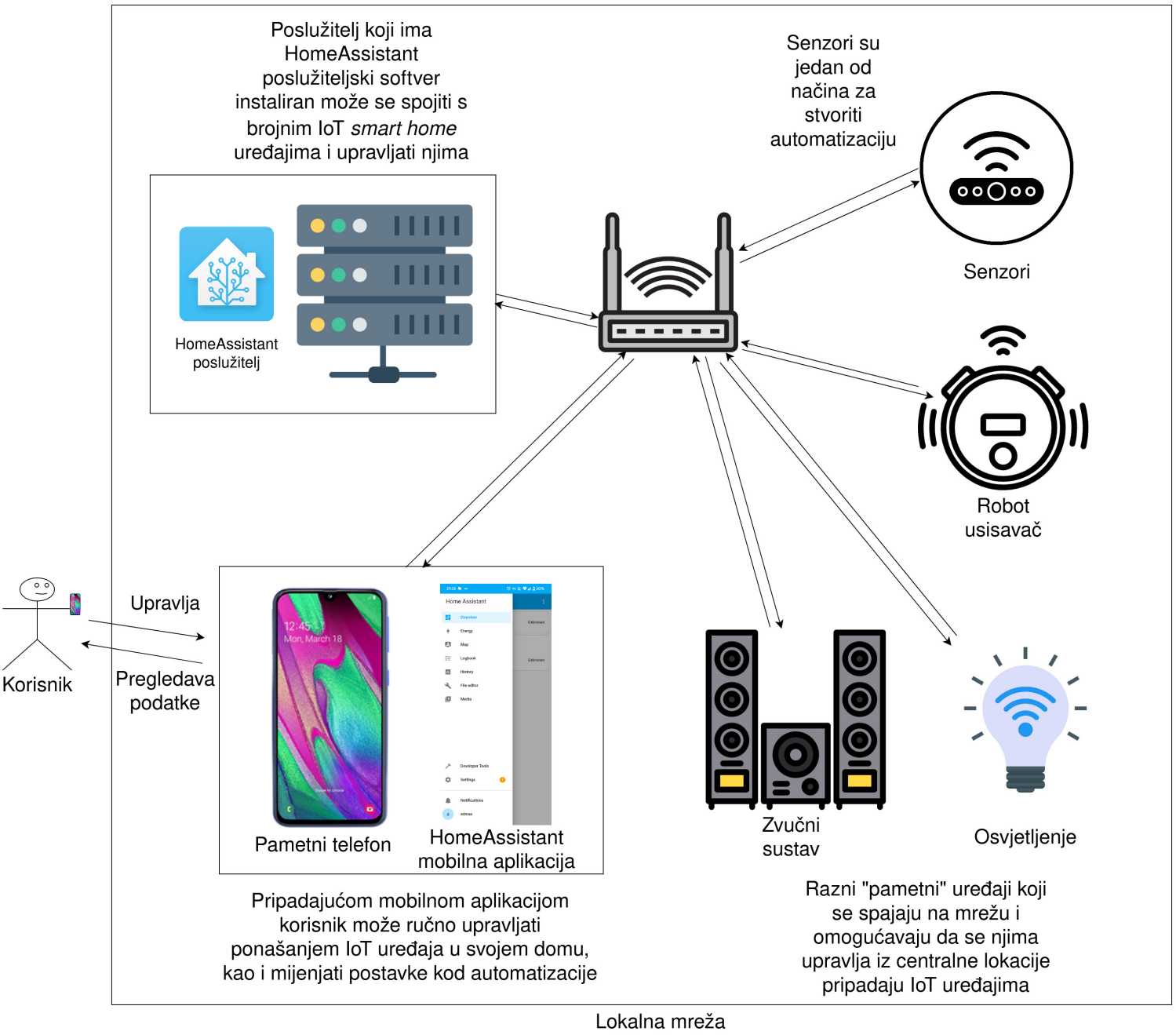
situacijama. Tako je korisno sve (ili selektivno odabrane) notifikacije proslijediti na pametni sat. Ovisno o modelu, na neke poruke može se odgovoriti i s pametnog sata. Također, neki modeli imaju i mikrofon i zvučnik pa se pozivi mogu odgovarati bez diranja telefona.

Pametni telefon i pametni sat dvije su odvojene fizičke cjeline povezane bežično. To znači da je gubljenje veze u dnevnom radu oba uređaja sasvim neizbježno kod normalnog korištenja. Oba se uređaja međusobno upotpunjuju ako su povezani, no pametni sat mora biti u mogućnosti nastaviti obavljati sve funkcije za koje mu veza s telefonom nije nužna. Spajanje s telefonom kad se uvjeti ponovno ostvare mora biti brzo i transparentno – sva funkcionalnost koja je omogućena a zahtijeva vezu s telefonom mora nastaviti raditi bez kašnjenja ili drugih problema. Suprotno bi konstituiralo loš proizvod koji je nepraktičan za korištenje.

Manje relevantna funkcionalnost za ovaj slučaj je sinkronizacija podataka o zdravlju i aktivnosti. Kao što je standard za mnoge aplikacije, tako je i u ovom slučaju skoro pa neizbježno slanje podataka na oblak proizvođača. Postoje neka rješenja koja to ne rade, ali je potrebno imati kompatibilan *wearable* uređaj.

6.3. Pametni telefon kao upravljač i nadziratelj IoT uređaja, M>IoT

U ovom slučaju mobilna aplikacija služi za upravljanje rada IoT uređaja, a uz to i nadziranje njihova rada. Pametni telefon time služi kao korisniku prikladno rješenje za obaviti određene akcije ili provjeriti trenutno stanje. Jedna česta primjena takve arhitekture IoT sustava je kod *pametnih* uređaja za unapređenje domova – IoT osvjetljenje, roboti usisivači, mjerni uređaji za potrošnju energije, pametno grijanje/hlađenje prostora i još mnogo toga.



Slika 17 Dijagram primjera M>IoT referentne arhitekture

Na dijagramu (slika 17) nalazi se primjer u kontekstu automatizacije doma i upravljanja *smart home* uređajima. Spomenuto je nekoliko mogućih IoT uređaja – pametne lampe, robot usisivač, zvučni sustav i senzori. Svaki od tih uređaja spojen je na kućnu mrežu. Na dijagramu je povezivanje pojednostavljeno na WiFi, no u stvarnom sustavu s više *smart home* uređaja neki od njih koristili bi WiFi, a neki drugi pak *ZigBee* ili *Z-Wave* koji za spajanje na lokalnu kućnu mrežu trebaju posredni uređaj. Centralni dio ovog sustava je svakako poslužitelj *Home Assistant*. Na njega su spojeni svi IoT uređaji i preko njega se mogu konfigurirati procedure i automatizacije, a svim uređajima se može i ručno upravljati. U stvarnom sustavu *Home Assistant* ne mora biti na lokalnom poslužitelju (iako je to najsigurnija, najprivatnija i najrobusnija konfiguracija ako se pravilno postavi) već može biti i u oblaku. Također, korisnik može odabrati koristiti neko drugo rješenje. Proizvođači IoT uređaja obično nude vlastito

rješenje, no za korištenje IoT uređaja oni često moraju imati konstantan pristup internetu, a ako se koriste različiti proizvođači može postati nužno koristiti različite aplikacije za različite uređaje, a povezivanje njih u jednu cjelinu tada postaje kompliciranо. Bitno je napomenuti da kod kupovine *smart home* IoT uređaja treba prvo provjeriti mogu li se oni spojiti na Home Assistant.

Uloga mobilne aplikacije u ovom kontekstu je da korisniku omogući jednostavno upravljanje *smart home* uređajima. Na primjer, korisnik prije spavanja želi ugasiti svjetlo nakon što legne u krevet. Ako mu prekidač nije u dovoljnoj blizini, to neće moći učiniti. No, mobilnom aplikacijom ta akcija postaje trivijalna. *Home Assistant* mobilna aplikacija nudi bezbroj mogućnosti za automatizaciju. Jedan popularan način je korištenjem *NFC tag*-ova na različitim mjestima. Prislanjanjem pametnog telefona s konfiguriranom aplikacijom može pokrenuti ili zaustaviti odabrane akcije. Na primjer, prislanjanjem na *NFC tag* u dnevnoj sobi može poslati robot usisivač da očisti pod u toj sobi. Mogućnosti su mnogobrojne i ograničene najviše budžetom i kreativnošću korisnika.

7. Implementacija sustava za automatsko održavanje smjera na plovilu

Razvoj ugrađenih sustava s IoT značajkama ne mora biti kompliciran i nedostupan pojedincima sa željom da uče ili razviju rješenje za svoje potrebe. Popularna otvorena platforma Arduino pruža cjenovno prihvatljivo rješenje pomoću kojeg se može razviti sustav za (skoro) svaku potrebu. Osim cijene, primamljivost platforme Arduino je iznimno velik broj modula koji se mogu koristiti – od malih senzora do WiFi-ja i kontrolera za velike industrijske elektromotore. Ono što je vrlo bitno je da postoji i velika zajednica ljudi koji se bave razvojem u ekosustavu Arduino.

7.1. Uvod u problem

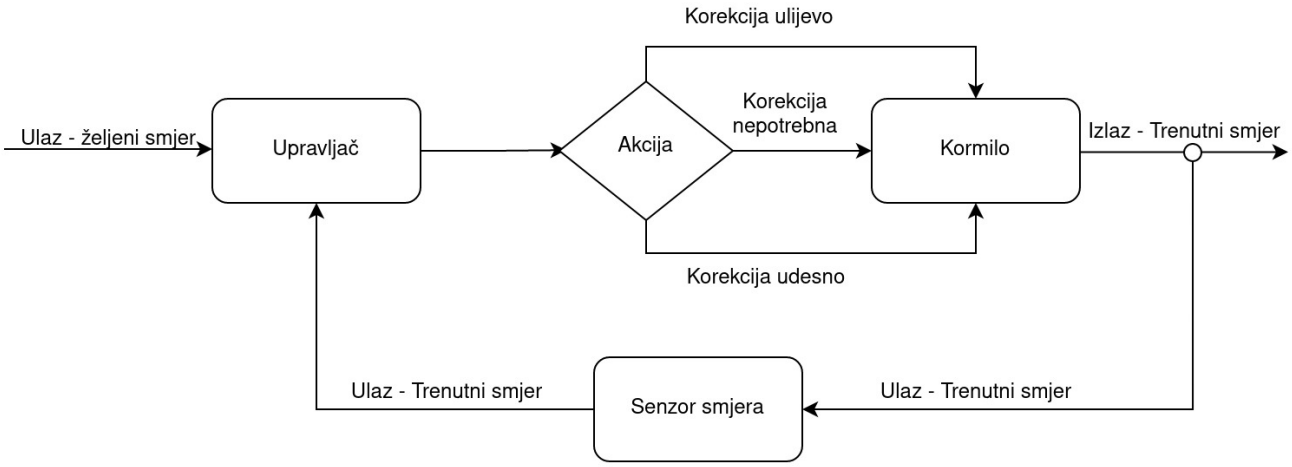
Problem koji će biti riješen razvojem ugrađenog sustava s IoT značajkama u suštini je jednostavan: kormilarenje motornim plovilom na veće udaljenosti je čovjeku zamorno i može se automatizirati.

Konkretno plovilo u pitanju dugačko je oko 7 metara i najčešće plovi brzinom od oko 5 čvorova (ili između 9 i 10 km/h). List kormila (dio u vodi) nema nikakve senzore koji bi ukazali na stupanj okrenutosti stoga se to obavlja vizualno pogledom iza krme, ili u vožnji osjećajem i kretnjama plovila. List kormila okreće se hidraulički s upravljačkog mjesta gdje se običan volan okreće lijevo ili desno. Hidraulički sustav nije u potpunosti bez propusta pa se tako mjerenjem okretaja volana ne može točno utvrditi stupanj okrenutosti lista, već se bilo kakvo mjerenje treba redovito kalibrirati. Automatizacija okretanja lista kormila najjednostavnije se može riješiti korištenjem relativno snažnog elektromotora na volanu kormila – umjesto ručnog okretanja volan će okretati elektromotor. Taj pristup ima i prednost da se aktivnost okretanja kormila vrlo lako vidi i u slučaju greške u radu može se odmah preuzeti ručna kontrola. Ručno upravljanje temelji se na vizualnoj navigaciji uz pregled smjera kretanja na kompasu u rasponu od 0 do 359 stupnjeva. Upravljanje brzinom nije potrebno automatizirati jer cilj nije napraviti potpuno autonomno plovilo, iako je i to moguće kao ekstenzija. Brzinom se upravlja ručicom za gas, dok se smjerom (naprijed/nazad/neutralno) upravlja drugom ručicom. Količina gasa se tijekom putovanja najčešće ne mijenja, već se održava konstantna brzina.

7.2. Moguća rješenja problema

Problem u pitanju nije od značajne kompleksnosti, ali se i dalje može riješiti na više različitih načina. Ignorirajući velik izbor komponenti s kojima se može postići isti primarni cilj, ostaju konceptualni pristupi rješenju koje treba razmotriti i odabrati onaj pristup koji ima najviše smisla u danoj situaciji – cjenovno, implementacijski (trud i znanje koje je potrebno uložiti) i mogućnost nadogradnje funkcionalnosti u budućnosti.

Proces održavanja smjera u načelu se sastoji od tri sekvencijalne operacije: očitavanje trenutnog smjera kretanja, usporedba sa željenim smjerom i određivanje akcije, okretanje kormila u određenom smjeru. Taj proces je u načelu isti i za čovjeka i robota. Razlike u obavljanju procesa su potencijalno vrlo različite i informacije dolaze iz različitih izvora. Izvora je vjerojatno više kod čovjeka koji u obzir uzima puno više stvari iz okoline nego čak i kompleksni autopiloti, no ovisno o iskustvu, koncentraciji i utreniranosti, rezultati toga nisu nužno bolji nego što bi odradio autopilot. Proces se može prikazati dijagramom na slijedeći način (slika 18):



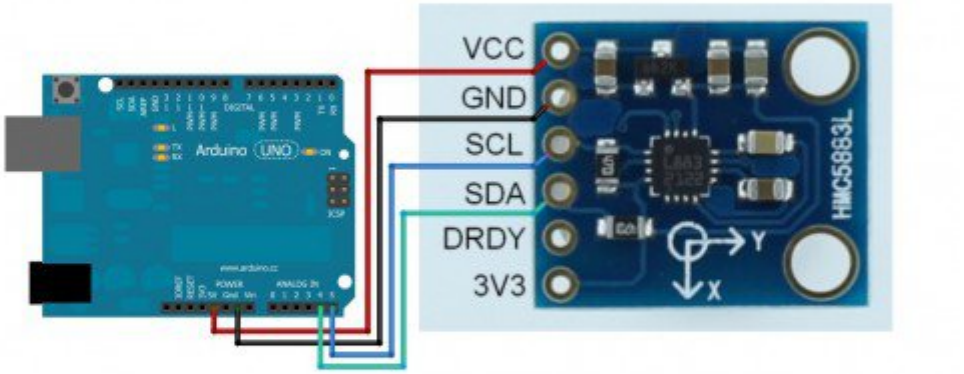
Slika 18 Dijagram procesa upravljanja održavanjem smjera

Željeni smjer definiran je u stupnjevima. Upravljač uspoređuje smjer kretanja sa željenim smjerom kretanja i određuje smjer u kojem treba okrenuti kormilo, ako ga treba okrenuti. U ovom dijelu dolazi do male problematike – koliko okrenuti kormilo? Naime, stupanj okretaja kormila izravno utječe na brzinu okretanja broda, a stalno isti okret neće biti dovoljan u svakoj situaciji ako je mali, a ako je veći biti će manje efikasan jer će brod više gubiti na brzini. Osim toga, kormilo treba vratiti na neutralnu poziciju u pravo vrijeme. Inercija okretanja broda

nastavit će ga okretati kratko vrijeme i nakon što se izravna kormilo. Iz tog razloga, kormilo treba izravnati i prije nego što se postigne željeni smjer tijekom korekcije smjera.

Što se tiče potrebnog hardvera, potreban je mehanizam za okretanje kormila, senzor za određivanje trenutnog smjera kretanja i kontroler koji će upravljati elektromotorom. Uz to, treba i upravljač za elektromotor jer se radi o snažnom elektromotoru za koji treba robustan sustav napajanja, kontroler ploča nije dovoljna.

Senzor smjera može biti kompas ili GPS. U slučaju GPS-a, potrebno je proći određenu udaljenost prije nego što se smjer može odrediti tim putem. Iz tog razloga, GPS nije dobro rješenje i neće biti korišten. Kompas kao modul može se nabaviti za Arduino, Raspberry Pi i slične uređaje sa podrškom za GPIO (slika 19).



Slika 19 Dijagram proizvoda, HMC5883L[13] digitalni magnetni kompas za Arduino

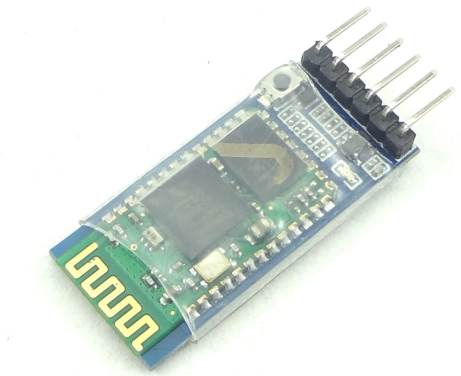
Taj pristup stavlja većinu ovisnosti i funkcionalnosti na sami ugrađeni sustav za upravljanje što ima za prednost veću rigidnost. Ipak, koliko je to prednost, još više je mana (naravno, ovisno o slučaju korištenja). Ako se sve komponente slože u jedan ugrađeni sustav, nadogradnju funkcionalnosti biti će puno teže obaviti nego ako se ide drugim pristupom.

Umjesto postavljanja svega u jedan ugrađeni sustav, moguće je iskoristiti pametni telefon koji u sebi već sadrži sve potrebne senzore i mrežna sučelja, a ono što preostaje u tom slučaju je omogućiti pametnom telefonu da upravlja radom elektromotora za okretanje kormila. Kao što je prethodno spomenuto, moderni pametni telefoni imaju integrirani veliki broj različitih senzora koji se mogu koristiti individualno ili kombinirano za bezbroj aplikacija. Jedan od tih senzora je magnetometar koji se tipično koristi uz GPS, WiFi i senzore pokreta u svrhu

[13] https://shop.wtihk.com/index.php?route=product/product&product_id=124

određivanja smjera kretanja prilikom navigacije ili sličnih aktivnosti. Baš taj senzor koristi se i za kompas za što postoje brojne aplikacije.

Osim kompasa, potrebno je imati upravljač koji će određivati slijedeću akciju temeljenu na željenom smjeru kretanja i trenutnom smjeru kretanja. Ako koristimo pametni telefon za određivanje trenutnog smjera kretanja, nema razloga da on ne odradi i sve ostalo što može. Zatim preostaje samo omogućiti da pametni telefon putem aplikacije okreće kormilo. Taj dio može se izvesti na razne načine, ali najjednostavnije i vjerojatno najjeftinije bilo bi koristiti Arduino Uno (ili neki drugi, ne treba biti *high-end* model), a komunikacija između Arduina i aplikacije može se vršiti Bluetooth vezom. Arduino u normalnoj konfiguraciji ne dolazi s Bluetooth radiom integriranim, već treba koristiti modul. Takvih modula ima više, no HC-05 modul niske je cijene i često se koristi pa ima dobru podršku i lako je pronaći primjere kako ga koristiti.



Slika 20 HC-05 Bluetooth modul[14]

Kako bi Arduino mogao upravljati elektromotorom, potrebno je koristiti hardver koji može podnijeti jako struju elektromotora, a Arduino to nije. Dva moguća rješenja su korištenje elektromagnetskih relay-a koji funkcioniraju na način da im se niskonaponska struja pusti na kontrolni sklop nakon čega se aktivira elektromagnet i zatvori strujni krug za gladno trošilo (u ovom slučaju elektromotor). Druga mogućnost je korištenje kontrolnog sklopa baziranog na MOSFET-ima napravljenim za jaku struju. To rješenje ima vrlo niski odaziv u odnosu na relej (eng. *relay*), no ta razlika za autopilot nije vrijedna spomena pa tako u načelu nije bitno koje od tih rješenja se odabere. Ono što jest bitno je da rješenje bude i više nego dovoljno za

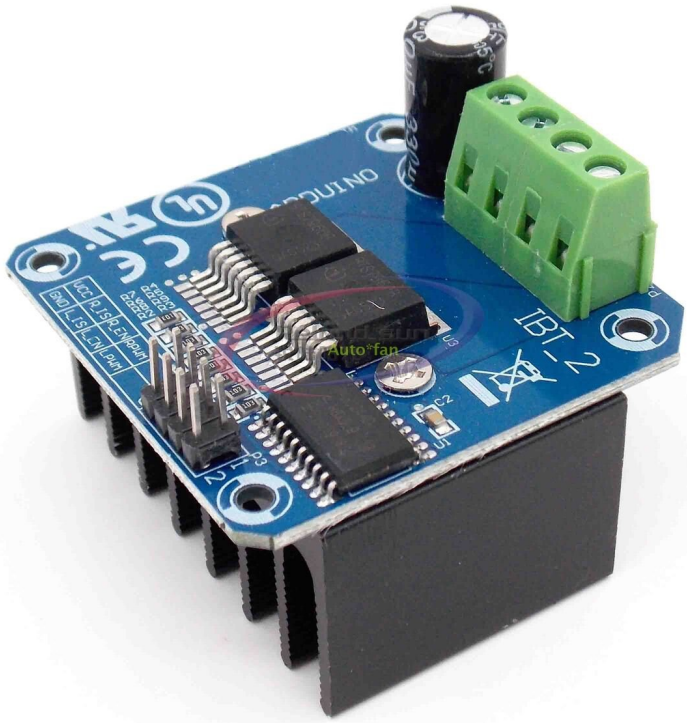
[14] http://exploreembedded.com/wiki/Setting_up_Bluetooth_HC-05_with_Arduino

električne potrebe trošila kako ne bi došlo do oštećenja, ometanja u radu sustava ili u najgorem slučaju – vatre.

Zadnja potrebna komponenta za cjelovito rješenje iz hardverskog dijela je napajanje za Arduino i module. Tipični Arduino ovisno o modelu treba ulaznu istosmjernu struju od 5V putem (micro) USB konektora. Neki modeli imaju posebni utor samo za napajanje koji prima i nešto veći napon. Ovaj dio najviše ovisi o okolini gdje će biti korišten, a egzaktna konfiguracija mojeg rješenja bit će navedena u nastavku.

7.3. Odabrana konfiguracija

Za vlastite potrebe zaključio sam da je najmanje ulaganja potrebno korištenjem Arduino Uno platforme, zajedno sa HC-05 Bluetooth modulom i IBT-2 upravljačem za elektromotor (slika 21) s podrškom za PWM (Pulse Width Modulation) što znači da ima podršku za mijenjanje brzine rotacije, značajka koja mi nije potrebna. Za napajanje nabavio sam jednostavnu pločicu za smanjenje napona istosmjerne struje koja se konfigurira mjerenjem i odvijačem. Arduino USB ulaz prima napone između 4.75 i 5.25V, a dostupni napon za napajanje na brodu je tijekom plovidbe između 13V i 14.7 volti. Elektromotor koristi taj napon bez regulacije što znači da IBT-2 upravljač mora biti spojen na izvorni napon.



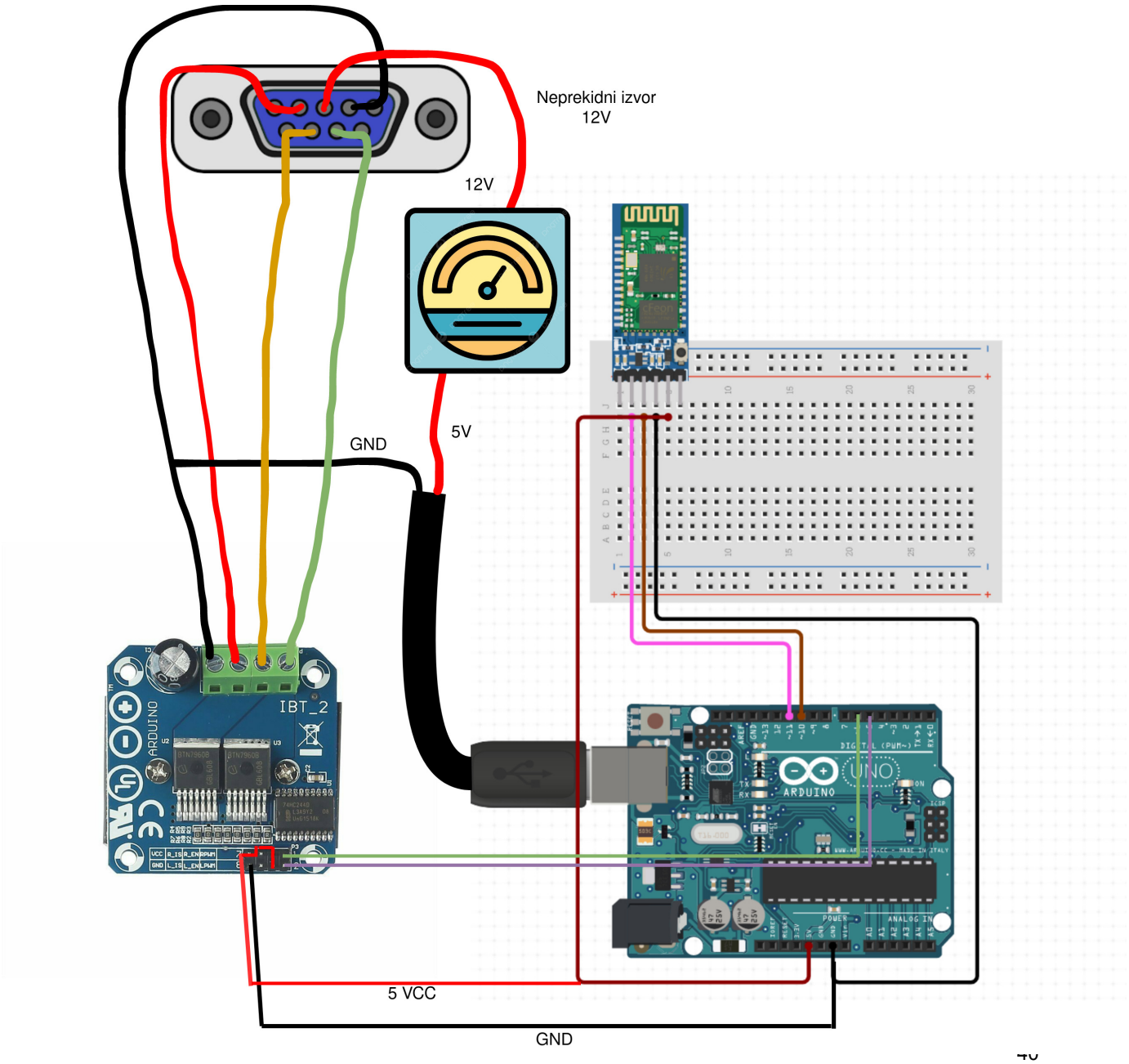
Slika 21 IBT-2 upravljač elektromotora[15]

[15] ebay.com

Srce operacije predstavlja pametni telefon. Nije bitno koji je model, ali je bitno da se može spojiti na odabrani Bluetooth modul i da ima potrebne senzore koji se u mogu koristiti unutar aplikacije. Za HC-05 modul je bitno da se koristi sustav Android jer iOS uređaji imaju znatno užu podršku za Bluetooth pa se zato ne mogu spojiti na taj modul. Elektromotor za okretanje kormila već je instaliran, a radi se o vrsti motora koji se koristi za automobilske brisače.

7.4. Složen hardver

Nakon što su svi dijelovi stigli, mogao sam početi slagati Arduino i potrebne module u jednu cjelinu koju je zatim potrebno testirati „na suho“. Električni dijagram izgleda ovako:



Slika 22 Skica-dijagram električnih spojeva Arduino podsustava

Kako bi taj hardver funkcionirao na način na koji je složen, a u zamišljenoj suradnji s Android aplikacijom koja šalje jednu od tri poruke: L (lijevo) , R (desno) ili N (neutralno, stop). Zadatak Arduina je da preko Bluetooth modula primi znak i ovisno o znaku otvori ili zatvori napon na kontrolnim pinovima IBT-2 upravljača. U nastavku se nalazi programski kôd učitan na Arduino:

```
#include <SoftwareSerial.h>

SoftwareSerial mySerial(10, 11); // RX receive, TX transmit
char inputByte = 'z';
uint8_t LEFT = 5; //broj pina na Arduino ploči
uint8_t RIGHT = 6;

void setup() {
  pinMode(LED_BUILTIN, OUTPUT); // LED svijetli ako ima dolaznih podataka
  pinMode(LEFT, OUTPUT); // +lijevo
  pinMode(RIGHT, OUTPUT); //+desno
  Serial.begin(9600);
  mySerial.begin(9600);
}

void loop() { // stalno radi
  while (mySerial.available()) {
    inputByte = mySerial.read();
    Serial.println(inputByte);
    if (inputByte == 'L') {
      leftTurn();
    } else if (inputByte == 'R') {
      rightTurn();
    } else if (inputByte == 'N') {
      stopTurn();
    }
    delay(100);
  }
}

void leftTurn() { //postavi pin za lijevo na +, pin za desno na -
```

```
  digitalWrite(RIGHT, LOW);
  digitalWrite(LEFT, HIGH);
  digitalWrite(LED_BUILTIN, HIGH);
}

void rightTurn() { //postavi pin za lijevo na -, pin za desno na +
  digitalWrite(LEFT, LOW);
  digitalWrite(RIGHT, HIGH);
  digitalWrite(LED_BUILTIN, HIGH);
}

void stopTurn() { //postavi oba pina na -, zaustavi okretanje
  digitalWrite(LEFT, LOW);
  digitalWrite(RIGHT, LOW);
  digitalWrite(LED_BUILTIN, LOW);
}
```

Programski kôd 1 Arduino kôd

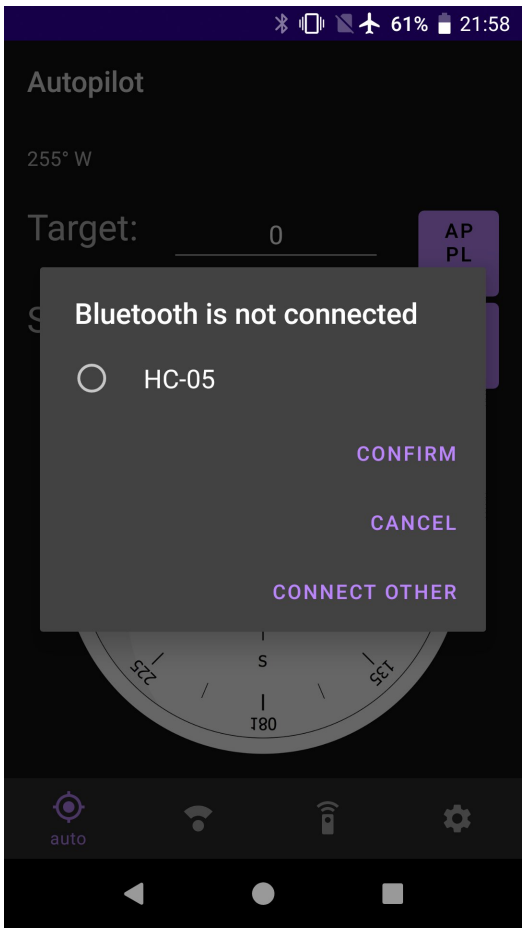
Svaka promjena Arduino kôda mora se nanovo snimiti (eng. *flash*) na *EEPROM* čip, na Arduino. Postupak snimanja je vrlo jednostavan. Najlakše je ako se koristi *Arduino IDE*[16] jer već ima ugrađenu funckionalnost kompiliranja kôda za specifičnu Arduino platformu (u ovom slučaju Uno) i snimanja programa putem USB kabla. Arduino obično ima *micro USB* ili *USB B*.

[16] https://docs.arduino.cc/software/ide-v2

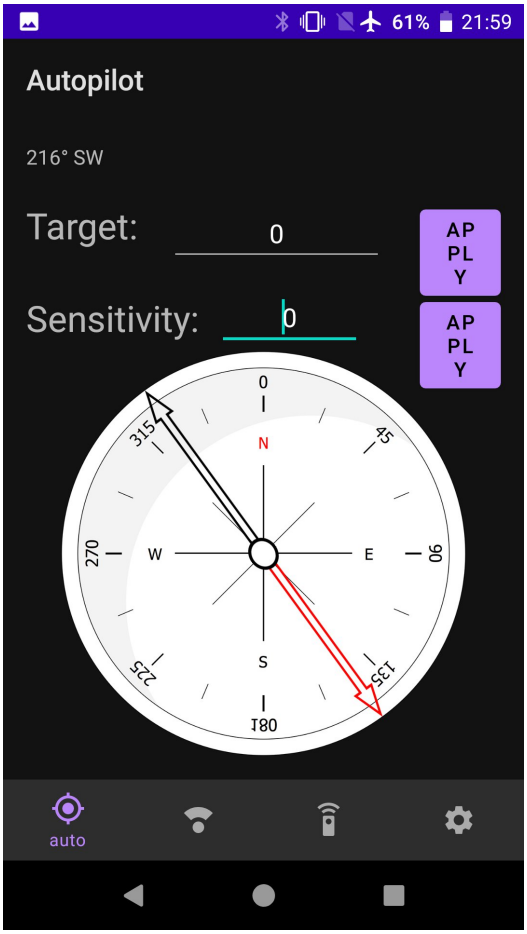
7.5. Mobilna aplikacija

Razvijena mobilna aplikacija u trenutku pisanja ovog rada u prototipnom je obliku. Sadržava puno oštrih rubova u pogledu dizajna, ali i rada s dopuštenjima za lokaciju. Aplikacija je razvijena korištenjem Android Studio IDE-a i koristi standardni Android SDK s ugrađenim elementima sučelja.

Cilj aplikacije je da uspostavi Bluetooth vezu s HC-05 modulom i samim time preuzme kontrolu nad kormilom plovila. Kod svakog pokretanja aplikacija korisnika traži da odabere Bluetooth uređaj s kojim će aplikacija (pokušati) uspostaviti vezu. Prije toga potrebno je upariti HC-05[17] i Android-om pogonjen telefon u Bluetooth postavkama. Nakon uparivanja prikazat će se na listi dostupnih Bluetooth uređaja (slika 24). Prisutnost uređaja na listi ne znači da je HC-05 aktivan i u blizini nego samo da je uparen.



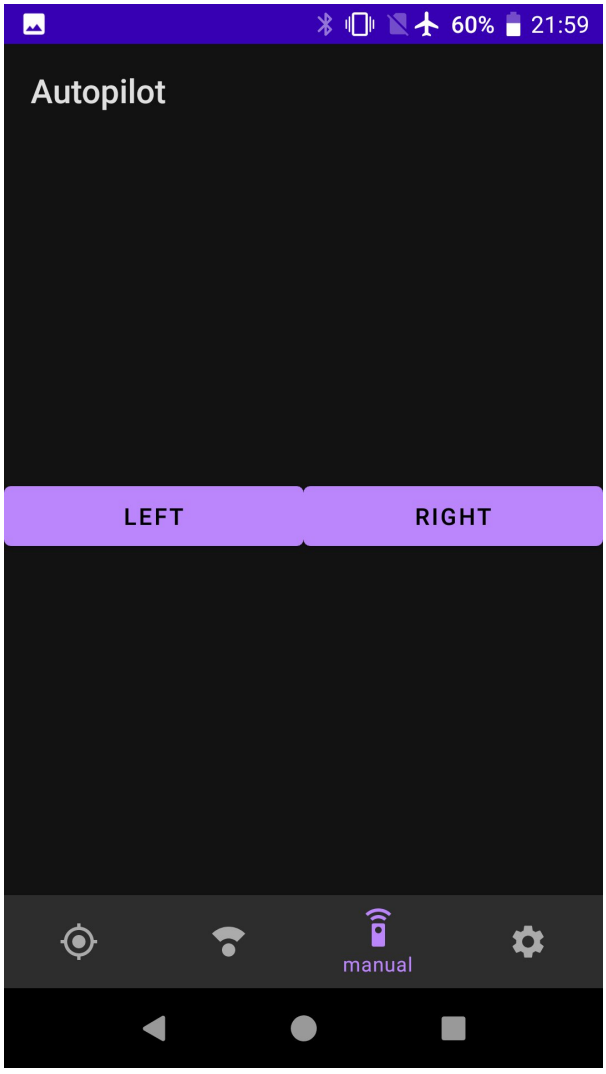
Slika 24 Lista Bluetooth uređaja



Slika 23 Autopilot sučelje s kompasom

[17] Zadani Bluetooth PIN je 1234

Spajanjem na HC-05 zatvara se dijaloški okvir i prikazuje se navigacijski *meni* i animirani kompas uz precizni iskaz trenutne orijentacije u stupnjevima (0-359) u gornjem lijevom kutu. U ovom fragmentu korisnik određuje smjer kretanja koji će sustav nastojati održavati (polje *Target*). U polju *Sensitivity* može se unijeti cjelobrojčana pozitivna vrijednost u rasponu od 1 do 10 gdje 1 uzrokuje najmanje okretanje kormila, ali u dobrim uvjetima može postići najveću preciznost, dok 10 uzrokuje najviše okretanja kormila i samim time manju preicznost, odnosno više vrludanja. Veća preciznost jednaka je manjem ukupnom prijeđenom putu i sukladno tome uštedi goriva i vremena. Uz automatsko okretanje kormila aplikacija nudi i ručno okretanje pritiskom i držanjem gumba za lijevo ili desno (slika 25):



Slika 25 Fragment za ručno upravljanje

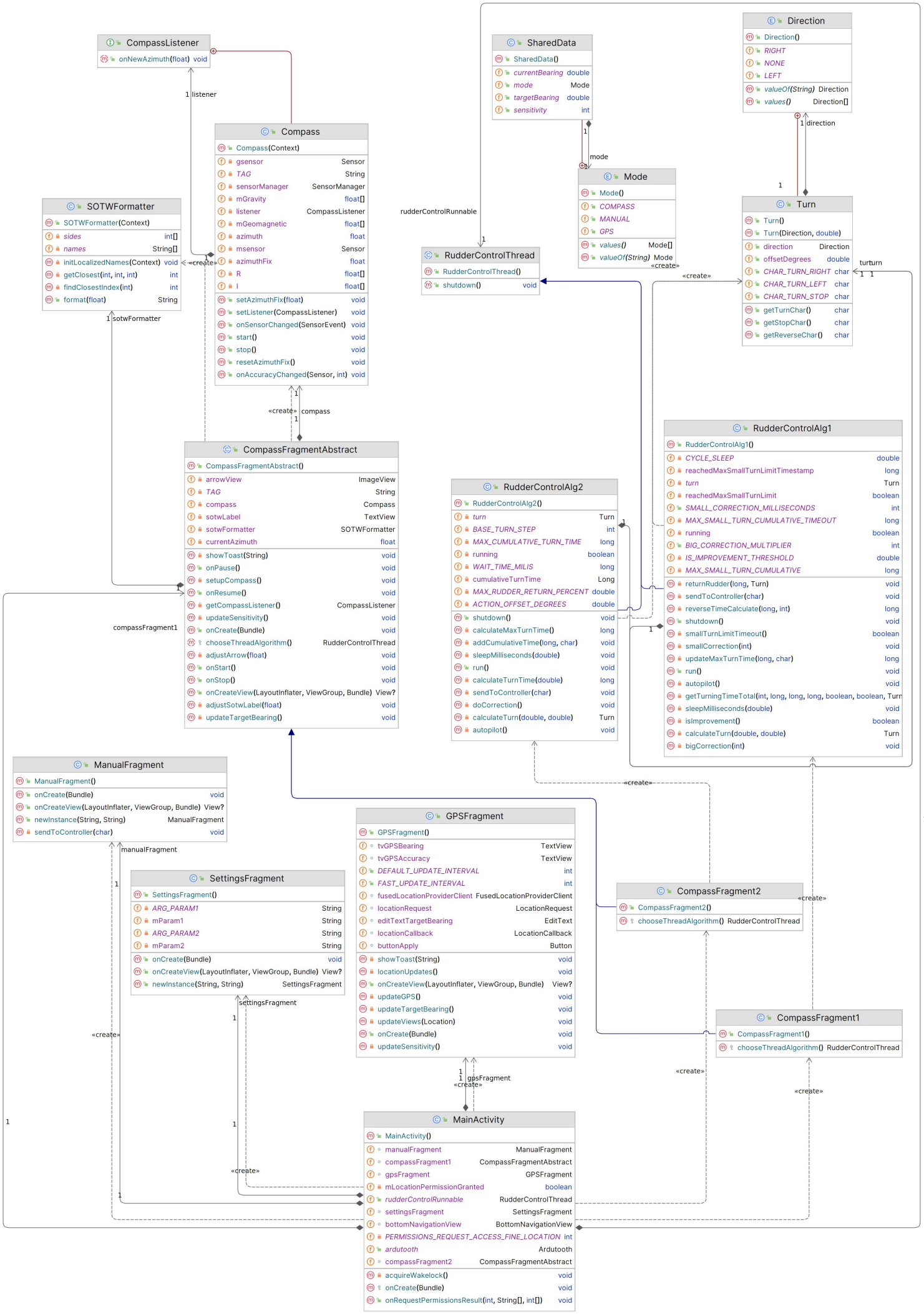
Ručno upravljanje posebno je korisno jer omogućava daljinsko upravljanje kormilom sa svake pozicije na plovilu.

7.5.1. Arhitektura mobilne aplikacije

Mobilna aplikacija napisana je u programskoj jeziku Java i koristi neke od koncepata objektno orijentiranog programiranja (OOP) uz dretvu samog autopilota koja radi u pozadini aplikacije, konzumira podatke iz ostalih dijelova aplikacije i šalje naredbe na *HC-05* i *Arduino* koji te naredbe pretvara u rotaciju elektromotora, a koji pak pokreće hidrauličku pumpu koja ulje tjera na lijevi ili desni kraj cilindra spojenog na list kormila koji snažnu struju vode iza propelera usmjerava lijevo, desno ili samo propušta ravno uzrokujući da par tona teško plovilo vrlo brzo čak i znatno promijeni smjer.

Na dijagramu klasa (slika 26) mogu se vidjeti sve ključne klase ove aplikacije. Na dnu dijagrama nalazi se klasa *MainActivity* koja je početna klasa kod pokretanja aplikacije. U njoj su definirani ostali osnovni objekti i navigacijska traka. U navigaciji su četiri opcije – Autopilot koji koristi kompas - *CompassFragment* (1 ili 2, uz neke razlike), autopilot koji koristi GPS - *GPSFragment* (zbog prevelikog kašnjenja nije pogodno za korištenje), ručno upravljanje - *ManualFragment* i fragment u kojem će se smjestiti razne postavke za koje su vrijednosti u prototipnoj fazi hardkodirane u konstante (*SettingsFragment*). Jezgru automatskog upravljanja kormilom čini dretva *RudderControlThread* – abstraktna klasa koja nasljeđuje Java klasu *Thread*. Tu klasu implementiraju dvije podklase: *RudderControlAlg1* i *RudderControlAlg2* prva od kojih je u trenutku pisanja funkcionalna implementacija, a druga je eksperimentalna namijenjena testiranju u svrhu poboljšanja. Dretva autopilota konstantno konzumira podatke s kompasa, a za to je iskorištena gotova implementacija[18] i integrirana u aplikaciju. Oba kompas fragmenta koriste tu biblioteku i populiraju statičke varijable u klasi *SharedData*: trenutni smjer (ažuriran u stvarnom vremenu mnogo puta u sekundi), željeni smjer (definiran od strane korisnika), postavka senzitivnosti i mod operacije (korišten kod prebacivanja s GPS, kompas ili ručnog načina rada kako bi dretva znala treba li slati naredbe na Arduino). U nastavku bit će prikazan i opisan kôd funkcionalnog algoritma za upravljanje kormilom bez podatka o trenutnom kutu okrenutosti lista kormila, informacija koja bi posao autopilota učinila vrlo jednostavnim, a bez koje se algoritam može vrlo zakomplicirati.

[18] https://github.com/iutinvg/compass



Slika 26 Dijagram klasa mobilne aplikacije

Kao centralni dio aplikacije, algoritam dretve autopilota valjalo bi detaljnije objasniti. Kao što je već navedeno, dretva je definirana u obliku apstraktne klase *RudderControlThread* kako bi korištenje različitih algoritama u svrhu razvoja, testiranja ili pak različitih vremenskih uvjeta bilo jednostavno za implementirati. Apstraktna je klasa definirana ovako:

```
package com.vaslim.autopilot.ruddercontrol;

public abstract class RudderControlThread extends Thread{
    public abstract void shutdown();
}
```

Programski kôd 2 Apstraktna klasa RudderControlThread

Jedina metoda koju treba implementirati svaka konkretna klasa je ona za gašenje/zaustavljanje dretve. Nadalje, jedna implementacija koja je funkcionalna izgleda ovako:

```
package com.vaslim.autopilot.ruddercontrol;

import com.vaslim.autopilot.MainActivity;
import com.vaslim.autopilot.SharedData;
import com.vaslim.autopilot.Turn;

public class RudderControlAlg1 extends RudderControlThread {

    private static final double CYCLE_SLEEP = 100;

    public static final int SMALL_CORRECTION_MILLISECONDS = 300;
    public static final int BIG_CORRECTION_MULTIPLIER = 2;
    private static final long MAX_SMALL_TURN_CUMULATIVE = 1000;
    private static final long MAX_SMALL_TURN_CUMULATIVE_TIMEOUT = 5000;
    private static final double IS_IMPROVEMENT_THRESHOLD = 0.7;

    private static Turn turn;
    private boolean running = true;
    private boolean reachedMaxSmallTurnLimit = false;
    private long reachedMaxSmallTurnLimitTimestamp;
```

```
    public RudderControlAlg1() {
        turn = new Turn();

    }

    @Override
    public void shutdown() {
        this.running = false;
    }
...
```

Programski kôd 3 Početak klase RudderControlAlg1

Ovdje su definirane konstante koje se koriste jednom ili više puta u ostatku kôda. U odnosu na pisanje tih vrijednosti u ostatku kôda, ovaj način okuplja sve „postavke“ na jedno mjesto i čini ukupno iskustvo preglednijim.

Klasa *Turn* sadržava podatke o skretanju, primarno na koju stranu se treba skrenuti i koliko je odstupanje od željenog smjera kretanja. Algoritam na temelju tih podataka određuje koliko, kada i na koju stranu će okretati kormilo.

```
package com.vaslim.autopilot;

public class Turn {
    public enum Direction {
        LEFT,
        RIGHT,
        NONE
    }

    public static final char CHAR_TURN_LEFT = 'L';
    public static final char CHAR_TURN_RIGHT = 'R';
    public static final char CHAR_TURN_STOP = 'N';

    public Direction direction;
    public double offsetDegrees;

    public Turn(Direction direction, double offsetDegrees) {
        this.direction = direction;
        this.offsetDegrees = offsetDegrees;
```

```
    }

    public Turn(){

    }
    public char getTurnChar(){
        if(direction == Direction.RIGHT) return CHAR_TURN_RIGHT;
        else if(direction == Direction.LEFT) return  CHAR_TURN_LEFT;
        return CHAR_TURN_STOP;
    }
    public char getStopChar(){
        return CHAR_TURN_STOP;
    }
    public char getReverseChar(){
        if(direction == Direction.RIGHT) return CHAR_TURN_LEFT;
        else if(direction == Direction.LEFT) return  CHAR_TURN_RIGHT;
        return CHAR_TURN_STOP;
    }
}
```

Programski kôd 4 Klasa Turn

Znakovi „L“, „R“ i „N“ se šalju na Arduino i označavaju okret nalijevo (L), nadesno (R) i prekid okretanja (N). Vrijednost stupnjeva odstupanja smjera kretanja od željenog smjera nalazi se u varijabli *offsetDegrees*.

Populiranje *Turn* objekta obavlja ova metoda:

```
private Turn calculateTurn(double destination, double origin){
        double LH = origin - destination;
        if (LH < 0) LH += 360;
        double RH = destination - origin;
        if (RH <0 ) RH += 360;

        Turn.Direction direction;
        double offsetDegrees;
        if (LH < RH) {
            direction = Turn.Direction.LEFT;
            offsetDegrees = LH;
        } else {
            direction = Turn.Direction.RIGHT;
            offsetDegrees = RH;
        }
        turn.direction = direction;
        turn.offsetDegrees = offsetDegrees;
        return new Turn(turn.direction,turn.offsetDegrees);
    }
```

Programski kôd 5 Metoda za populiranje Turn objekata

Glavna petlja dretve autopilota izgleda ovako:

```
  private void autopilot () {
        double targetBearing;
        double currentBearing;
        int sensitivity;

        while (running) {
            if (SharedData.targetBearing >= 0 &&
SharedData.mode!=SharedData.Mode.MANUAL) {
                targetBearing = SharedData.targetBearing;
                currentBearing = SharedData.currentBearing;
                sensitivity = SharedData.sensitivity;
                int smallCorrectionDeviation = sensitivity + 5;
                calculateTurn(targetBearing,currentBearing);
                if (turn.offsetDegrees <= smallCorrectionDeviation) {
```

```
                smallCorrection(smallCorrectionDeviation);
            } else {
                bigCorrection(smallCorrectionDeviation);
            }

        }
    }
    System.out.println("THREAD OUT");
    MainActivity.rudderControlRunnable = null;
}
```

Programski kôd 6 Glavna petlja dretve

Petlja traje sve dok je vrijednost varijable *running* jednaka *true*, a rad dretve može se i pauzirati ako se drugdje u aplikaciji mod operacije postavi na manualni. Podsustav aplikacije koji čita podatke s kompasa populira statičku klasu *SharedData* iz koje se ovdje iščitavaju vrijednosti. Grananje vezano za akciju skretanja ovdje se odnosi na jakost odstupanja. Ako je odstupanje manje od određene vrijednosti, pozvat će se metoda koja je znatno blaža u svojem djelovanju. Kôd te metode izgleda ovako:

```
private void smallCorrection (int smallCorrectionDeviation) {
    long maxTurnTime = 0;//negative for LEFT, positive for RIGHT
    while (turn.offsetDegrees <= smallCorrectionDeviation) {
        boolean improvement = isImprovement();
        if (!improvement) {
            if ((maxTurnTime*-1 < MAX_SMALL_TURN_CUMULATIVE && turn.getTurnChar()
                    == Turn.CHAR_TURN_LEFT) ||
                (maxTurnTime < MAX_SMALL_TURN_CUMULATIVE && turn.getTurnChar()
                    == Turn.CHAR_TURN_RIGHT)){
                System.out.println("SMALL TURN "+ turn.getTurnChar());
                maxTurnTime = updateMaxTurnTime(maxTurnTime, turn.getTurnChar());
                sendToController(turn.getTurnChar());
                sleepMilliseconds(SMALL_CORRECTION_MILLISECONDS);
                sendToController((turn.getStopChar()));
            }
            if (reachedMaxSmallTurnLimit && smallTurnLimitTimeout()) {
                System.out.println("SMALL TURN LIMIT TIMEOUT " + maxTurnTime);
                if (maxTurnTime > 0) maxTurnTime = maxTurnTime -
                    SMALL_CORRECTION_MILLISECONDS;
```

```
                else if (maxTurnTime < 0) maxTurnTime = maxTurnTime +
                        SMALL_CORRECTION_MILLISECONDS;
            }
        }
        sleepMilliseconds(CYCLE_SLEEP);
    }
}
```

Programski kôd 7 Metoda smallCorrection()

Kako se kormilo ne bi previše okrenulo u bilo kojem slučaju, postoje provjere koje uzimaju u obzir postavku maksimalnog vremena okretanja u jednu stranu. Kao što je vidljivo, ključni dio funckioniranja metode je računanje trenutnog vremena okretanja (veliki if uvjet). Kumulativno vrijeme okretanja predstavlja procjenu položaja kormila s obzirom na nedostatak točne informacije temeljem nekog senzora.

Kontrastno, metoda koja djeluje kod velikog odstupanja izgleda ovako:

```
private void bigCorrection(int smallCorrectionDeviation){
    long turnTimeLimit = BIG_CORRECTION_MULTIPLIER *
            SMALL_CORRECTION_MILLISECONDS;
    long startTime = 0;
    long turningTimeTotal = 0;
    boolean isTurning = false;
    boolean maxTurn = false;
    Turn committedTurn =
            calculateTurn(SharedData.targetBearing,SharedData.currentBearing);
    turningTimeTotal = getTurningTimeTotal(smallCorrectionDeviation, turnTimeLimit,
            startTime, turningTimeTotal, isTurning, maxTurn, committedTurn);

    returnRudder(turningTimeTotal, committedTurn);
}
```

Programski kôd 8 Metoda bigCorrection()

Ovaj je dio jednostavniji, no ako pogledamo metode koje se ovdje pozivaju, priča je drukčija:

```
private long getTurningTimeTotal(int smallCorrectionDeviation, long turnTimeLimit, long
startTime, long turningTimeTotal, boolean isTurning, boolean maxTurn, Turn committedTurn)
{
    while (turn.offsetDegrees>= smallCorrectionDeviation){
        boolean improvement = isImprovement();
        long currentTime = System.currentTimeMillis();
```

```
        if(isTurning){
            turningTimeTotal = currentTime-startTime;
        }
        if (!improvement && !maxTurn){
            sendToController(committedTurn.getTurnChar());
            startTime = System.currentTimeMillis();
            isTurning = true;
        }
        if (!improvement && maxTurn){
            sendToController(committedTurn.getTurnChar());
            sleepMilliseconds(SMALL_CORRECTION_MILLISECONDS);
            sendToController(turn.getStopChar());
            isTurning = false;
            turningTimeTotal += SMALL_CORRECTION_MILLISECONDS;
        }
        if (isTurning && currentTime- startTime >= turnTimeLimit){
            sendToController(turn.getStopChar());
            maxTurn = true;
            isTurning = false;
            turningTimeTotal = currentTime- startTime;
        }
    }
    sendToController(turn.getStopChar());
    return turningTimeTotal;
}
```

Programski kôd 9 Metoda getTurningTimeTotal()

Metoda *getTurningTimeTotal()* sastoji se od petlje koja okreće kormilo dok ne postoji poboljšanje i dok nije dostignut raspon operacije za metodu *smallCorrection()*. Metoda također prati vrijeme okretanja kormila kako bi se moglo vratiti na početnu, neutralnu poziciju za ravno kretanje. Metoda *isImprovement()* izgleda ovako:

```
private boolean isImprovement() {
    double previousOffset = turn.offsetDegrees;
    Turn.Direction previousDirection = turn.direction;
    sleepMilliseconds(SMALL_CORRECTION_MILLISECONDS);
    calculateTurn(SharedData.targetBearing, SharedData.currentBearing);
    return turn.offsetDegrees - previousOffset < IS_IMPROVEMENT_THRESHOLD ;
```

```
}
```

Programski kôd 10 Metoda isImprovement()

Ona uzima novu vrijednost odstupanja i uspoređuje ju sa prethodnim odstupanjem koje je izmjereno prije određenog vremena spavanja. Koristi globalnu varijablu *turn* za aktualno stanje.

Od ostalih pomoćnih metoda imamo *updateMaxTurnTime()*:

```
private long updateMaxTurnTime(long maxTurnTime, char turnChar) {
    if (turnChar == Turn.CHAR_TURN_LEFT){
        maxTurnTime = maxTurnTime - SMALL_CORRECTION_MILLISECONDS;
    }
    else if (turnChar == Turn.CHAR_TURN_RIGHT){
        maxTurnTime = maxTurnTime + SMALL_CORRECTION_MILLISECONDS;
    }
    if (Math.abs(maxTurnTime) >= MAX_SMALL_TURN_CUMULATIVE){
        reachedMaxSmallTurnLimit = true;
        reachedMaxSmallTurnLimitTimestamp = System.currentTimeMillis();
    } else {
        reachedMaxSmallTurnLimit = false;
    }
    return maxTurnTime;
}
```

Programski kôd 11 Metoda updateMaxTurnTime()

Tu se računa vrijednost kumulativnog okretanja u svrhu određivanja je li se dosegla definirana granica okretanja u jednu stranu. Dalje imamo ove metode:

```
private void sendToController(char command) {
    MainActivity.ardutooth.sendChar(command);
}

private long reverseTimeCalculate(long time, int sensitivity){
    long time2 = (long) (time*0.2*sensitivity);
    if(time2 > time) time2 = time;
    return time2;
}

private void sleepMilliseconds(double value) {
    try {
```

```
            Thread.sleep((long) value);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

private void returnRudder(long turningTimeTotal, Turn committedTurn) {
        long startTime;
        sendToController(committedTurn.getReverseChar());
        startTime = System.currentTimeMillis();
        long sleepTime = reverseTimeCalculate(turningTimeTotal,SharedData.sensitivity);
        sleepMilliseconds(sleepTime);
        sendToController(turn.getStopChar());
    }
```

Programski kôd 12 Pomoćne metode

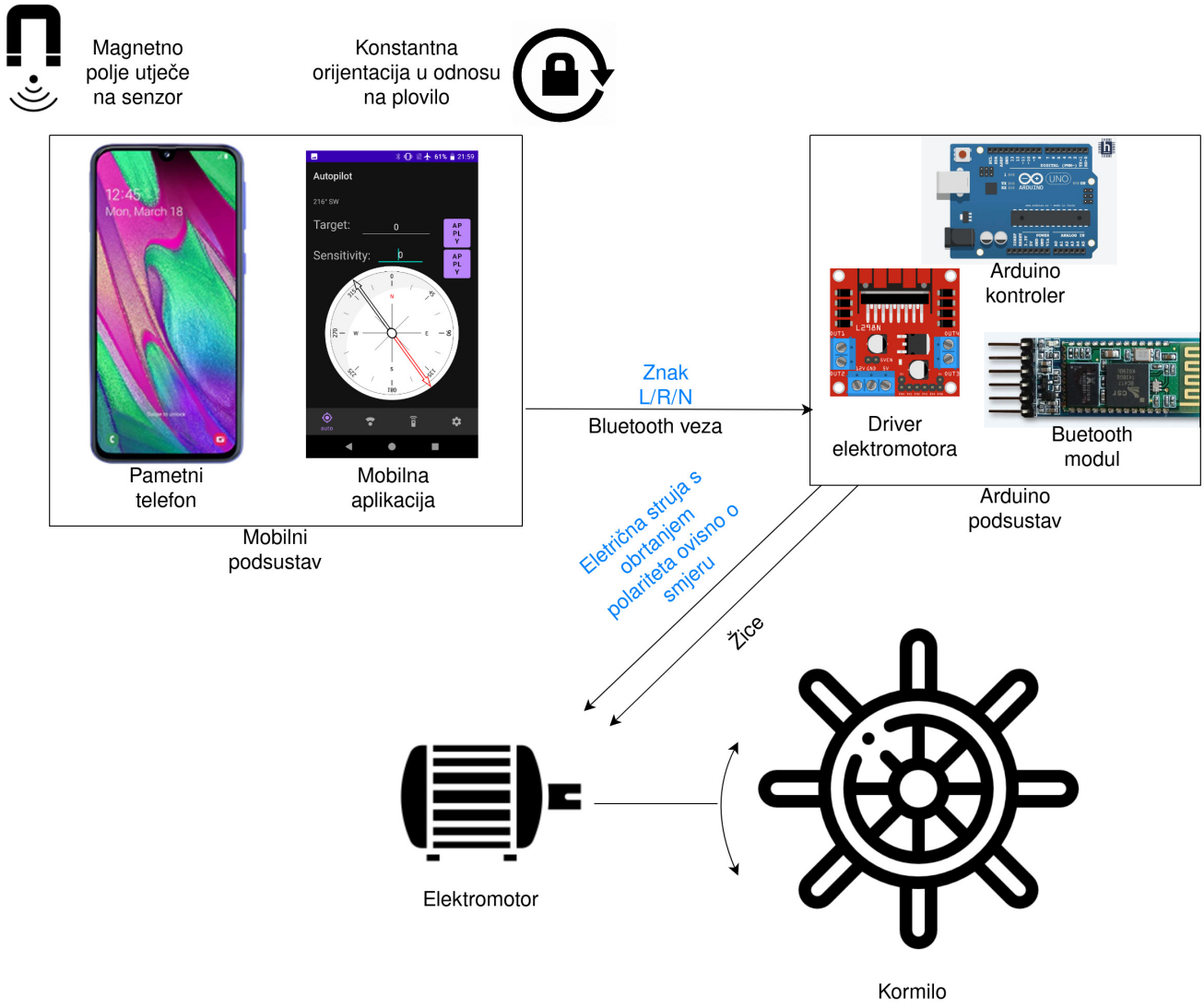
Metoda *reverseTimeCalculate()* računa vrijednost vremena koje se kormilo treba okretati u suprotnu stranu nakon obavljenog okretanja u stranu korekcije smjera. Metoda *returnRudder()* poziva prethodnu metodu za vraćanje kormila, a metoda *sendToController()* prima znak i šalje ga na Arduino. Time je prikazan kôd glavne logike upravljanja kormilom. Ostatak koda dostupan je na https://github.com/dinccey/autopilot.

7.6. Arhitektura sustava autopilota

Pogledom na šesto poglavlje može se dobiti ideja kojoj od referentnih arhitektura najbolje odgovara ovo rješenje. Uloga pametnog telefona u rješenju autopilota je senzorna i obradna. Mobilna aplikacija čita podatke sa senzora uređaja i obrađuje korisničke ulaze u skladu s trenutnim stanjem senzora. Pametni telefon je također i *internet gateway*[19] ako bi se implementirala dodatna funkcionalnost koja bi iskoristila tu značajku. S obzirom da je senzor koji se koristi ustvari kompas, telefon mora fizički biti statičan i ne mijenjati svoju orijentaciju u odnosu na plovilo, a u suprotnom će izgubiti kalibraciju i bez intervencije plovilo će se nastaviti kretati u krivom smjeru.

[19] https://en.wikipedia.org/wiki/Gateway_(telecommunications)

S druge strane nalazi se Arduino sustav za upravljanje radom elektromotora koji je s pametnim telefonom povezan Bluetooth vezom i pametni telefon bez tog dijela sustava ne može biti korišten u svrhu autopilota u kontekstu implementiranog rješenja. No, i suprotno vrijedi – Arduino sustav je sam po sebi beskoristan jer nema kako odrediti trenutni smjer kretanja (ako bi se to riješilo, morao bi se napisati i novi kôd za Arduino).

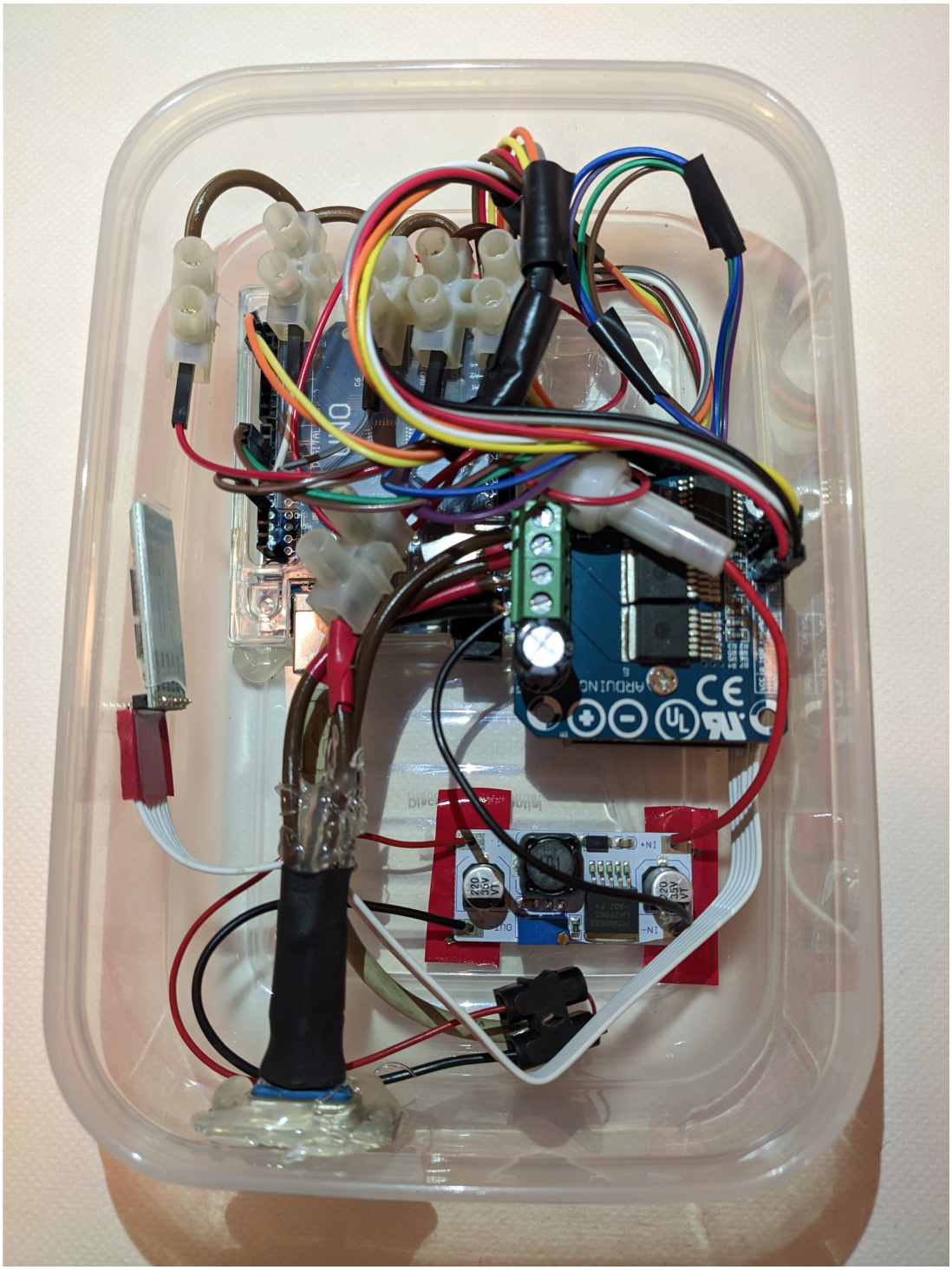


Slika 27 Arhitektura implementiranog rješenja autopilota

Kojoj referentnoj arhitekturi onda pripada implementirani autopilot? Uzevši u obzir navedeno, pripada kombinaciji M=IoT i M+IoT arhitektura. Pametni telefon i Arduino sustav djeluju zajedno i individualno su beskorisni pa se može reći da su u simbiozi (M+IoT). Ipak, uloga pametnog telefona je takva da mora biti statičan (osim u manualnom modu operacije) pa bi se moglo reći i da je sam pametni telefon u ovom slučaju IoT uređaj koji u svojem radu ima interakciju s drugim ugrađenim sustavom (M=IoT).

7.7. Galerija fotografija krajnjeg rješenja

Za kraj prigodno je prikazati ključne dijelove rješenja praktičnog rada. Na slici (slika 28) nalazi se pogled u podsustav baziran na platformi Arduino:



Slika 28 Unutrašnjost kućišta s komponentama i kablovima

Na slijedećoj slici (slika 29) vidljiv je konektor kojim se ova kutija spaja na plovilo. Radi se o standardnom konektoru DB-9 s devet pinova. Po dva pina su iskorištena za ulazni napon raspona 11-15 volti i pripadajuće uzemljenje (GND) te još dva para pinova za dvije žice koje idu na elektromotor. Parovi pinova su korišteni zbog potencijalno jake struje. Zadnji preostali pin koristi se da konstantno napaja Arduino i održava Bluetooth vezu s pametnim telefonom dok je autopilot onemogućen.

Slika 29 Kutija s Arduino-m i samotni konektor

Na zadnjoj slici (slika 30) vidljiv je i pametni telefon s otvorenom aplikacijom autopilota. U budućnosti funkcionalnost ovog sustava mogla bi se znatno poboljšati ugradnjom senzora kuta lista kormila u vodi čime bi se mogla postiči veća preciznost i pojednostaviti algoritam autopilota. Osim hardverskih nadogradnji, moguća su i softverska proširenja. Kako se radi o mobilnoj aplikaciji, može se postići potpuno autonomno upravljanje s automatskim promjenama smjera tokom izračunate rute. Korištenjem besplatnih pomorskih karata mogu se odrediti mjesta koja treba izbjeći, a pragovi blizine mogli bi se odrediti postavkama u aplikaciji.

Slika 30 Arduino i pametni telefon

8. Zaključak

Pametni telefoni, odnosno mobilne aplikacije jedan su od ključnih elemenata u posredovanju između krajnjih korisnika i namjenskih IoT uređaja. No, pojam interneta stvari vrlo je širok, kao i pojam ugrađenih uređaja. U ovom radu iznesena je opširnost tih pojmova, a pojam ugrađenih uređaja širi je od pojma interneta stvari. IoT je podskupina ugrađenih uređaja koji imaju obaveznu značajku mrežne povezivosti, tipično uključujući i internetsku povezivost na neki servis u oblaku.

Također je objašnjena svestranost pametnih telefona koja ide do razine da se i sami pametni telefon može koristiti kao IoT uređaj ako zadovoljava zahtjeve specifične aplikacije. Kao primjer toga ukratko je obrađena aplikacija *SensorServer* koja senzore pametnog telefona očitava i šalje drugim uređajima koji su na nju spojeni putem *websocket* veze. Brojni senzori modernog pametnog telefona omogućuju mu da se koristi za razne aplikacije, a ta mogućnost je pogotovo korisna za kućnu upotrebu i prototipiranje.

U kontekstu pametnog doma posebno je važno razmotriti i sigurnost i privatnost sustava jer su podaci potencijalno vrlo senzitivni. Kao primjer privatnog i sigurnog rješenja dana je segregirana kućna mreža s IoT uređajima odvojenima od ostalih uređaja. Također, predložen je kućni lokalni *HomeAssistant* poslužitelj kao privatna i sigurna alternativa servisima u oblaku gdje su brojni privatni podaci u vlasništvu velikih korporacija koje su nerijetko mete hakerskih napada. Takav pristup značajno bi poboljšao sigurnost i privatnost korisnika, ali zahtijeva i nešto više truda za postavljanje.

Kao rezultat istraživanja raznih uloga i primjena pametnih telefona (mobilnih aplikacija) u sustavima interneta stvari i, općenitije, ugrađenim sustavima, prepoznate su tri referentne arhitekture IoT sustava koji uključuju pametne telefone. To su M=IoT gdje pametni telefon preuzima ulogu IoT uređaja, zatim M+IoT gdje pametni telefon i namjenski IoT uređaji djeluju kao jedna cjelina i izravno su povezani bežičnim tehnologijama (primarno i najčešće Bluetooth), i zadnje M>IoT gdje pametni telefon s pripadajućom mobilnom aplikacijom djeluje kao upravljač ili nadglednik rada raznih IoT uređaja pružajući korisniku sučelje i interakciju.

U praktičnom dijelu opisana je primjena tehnologija važnih u svijetu interneta stvari. Primjenom tih tehnologija implementirano je rješenje za automatizaciju održavanja smjera plovidbe na manjem motornom plovilu – autopilot. Implementirano rješenje koristi mobilnu aplikaciju instaliranu na pametnom telefonu kako bi se odredio trenutni smjer kretanja i Bluetooth prijemnik za slanje naredbe o smjeru kretanja na podsustav Arduino. To je rješenje korištenu u stvarnom svijetu i uspješno je zamijenilo ljudski napor. Programski kôd nalazi se na javnom repozitoriju na poveznici https://github.com/dinccey/autopilot.

Popis literature

[1] *Thomas A. Henzinger, & Joseph Sifakis. (2007). The Discipline of Embedded Systems Design.*

[2] L. De Micco, & F. Vargas. (2020). A Literature Review on Embedded Systems; A Literature Review on Embedded Systems. In IEEE Latin America Transactions (Vol. 18, Issue 2).

[3] Yang, D., & Ren, H. (n.d.). The Research on The Technology of Internet of Things And Embedded System.

[4] Schuster, F., & Habibipour, A. (2022). Users' Privacy and Security Concerns that Affect IoT Adoption in the Home Domain. International Journal of Human-Computer Interaction. https://doi.org/10.1080/10447318.2022.21473025

[5] Collotta, M., Pau, G., Talty, T., & Tonguz, O. K. (2018). Bluetooth 5: A Concrete Step Forward toward the IoT. IEEE Communications Magazine, 56(7), 125–131. https://doi.org/10.1109/MCOM.2018.1700053

[6] Pahlavan, K., & Krishnamurthy, P. (2021). Evolution and Impact of Wi-Fi Technology and Applications: A Historical Perspective. In International Journal of Wireless Information Networks (Vol. 28, Issue 1, pp. 3–19). Springer. https://doi.org/10.1007/s10776-020-00501-8

[7] Attaran, M. (2021). The impact of 5G on the evolution of intelligent automation and industry digitization. Journal of Ambient Intelligence and Humanized Computing. https://doi.org/10.1007/s12652-020-02521-x

[8] Ntseane, L., & Isong, B. (n.d.). Analysis of LoRa/LoRaWAN Challenges: Review.

[9] S. J. Danbatta and A. Varol, "Comparison of Zigbee, Z-Wave, Wi-Fi, and Bluetooth Wireless Technologies Used in Home Automation," 2019 7th International Symposium on Digital Forensics and Security (ISDFS), Barcelos, Portugal, 2019, pp. 1-5, doi: 10.1109/ISDFS.2019.8757472.

Popis slika

Popis tablica

Popis programskih kodova

Dodaci

Programski kôd koji je razvijen za potrebe sustava autopilota, a referenciran je i djelomično objašnjen u ovom radu, nalazi se na GitHub repozitoriju na slijedećoj poveznici:

https://github.com/dinccey/autopilot