Vozilo na daljinsko upravljanje

Martinović, Helena

Undergraduate thesis / Završni rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Algebra University College / Visoko učilište Algebra**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:225:333773

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-07-27**

Repository / Repozitorij:

Algebra Univerity - Repository of Algebra Univerity

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

Vozilo na daljinsko upravljanje

Helena Martinović

Zagreb, veljača 2018.

Student vlastoručno potpisuje Završni rad na prvoj stranici ispred Predgovora s datumom i oznakom mjesta završetka rada te naznakom:

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada“.

U Zagrebu, datum.

Predgovor

Ova zahvala ide svim predavačima Algebre koji su nam kroz preddiplomski studij pružili znanja u sklopu kolegija, kao i vlastita iskustva te iskustvene savijete koja su obogatila naša shvaćanja i pružila ugodan ulazak u svijet programiranja. Posebno se zahvaljujem profesoru Andreju Lackoviću bez kojeg ne bi bilo ovog rada, kao i svim osobama koje su mi pružale inspiraciju i motivaciju.

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

Koncept Internet stvari (engl. *Internet of things*, skraćeno IoT) sve je češći suradnik u našim životima, ponekad i bez našeg saznanja. Ono se često prikrada u velikome broju novih proizvoda bijelih tehnologija, televizora, pa čak i grijanja ili rasvjete. Kako bi se na jednostavan način prikazao spomenuti koncept, koristit će se otvorena platforma Arduino u kombinaciji sa Android mobilnim uređajem. Tj. ti uređaji će međusobno komunicirati preko Internet konekcije kako bi razmijenili potrebne podatke za funkcioniranje ili prikaz prikupljenih podataka.

Ključne riječi: IoT, Arduino, Android, Vozilo.

IoT concept is becoming more frequent segment in our life, even though it is overlooked sometimes. It is hidden inside house appliances, television sets, and even within heating devices or is used to control house lights. With intention to simplify that concept and experience it better, Arduino open platform and Android mobile device are used. Devices will communicate with each other though Internet connection and will exchange needed instructions or gathered data.

Key words: IoT, Arduino, Android, Car.

Sadržaj

1. Uvod 1

2. Arduino 2

2.1. Arduino Uno 2

2.1.1. Mikrokontroleri 3

2.1.2. Pinovi 3

2.2. WiFi Shield 5

2.2.1. Instaliranje nove verzije ugrađenog programa 5

2.3. Arduino softver 7

2.3.1. WiFi.h biblioteka 8

3. Izvedba vozila i klijentske strane 9

3.1. Hardver vozila 9

3.1.1. H Bridge 9

3.1.2. US senzor 10

3.2. Implementacija Arduino poslužitelja 11

3.2.1. MyWiFi.h biblioteka 12

3.2.2. MyUS biblioteka 14

3.2.3. Izvedba 14

3.3. Klijentska strana 17

3.3.1. Izvedba 18

Zaključak 24

Popis kratica 25

Popis slika 26

Popis kôdova 27

Literatura 28

1. Uvod

Tema ovog rada je izrada hardverskog i softverskog rješenja te povezivanje tih dijelova preko mreže. Internet stvari kao koncept je sve veći te je ideja ovog rada pokazati kako se dva generalno nepovezana sustava mogu spojiti preko mreže te time ostvariti komunikaciju. Kod Interneta stvari, sustavi se mogu spojiti bilo kojom vrstom mreže, bila to Bluetooth, lokalna mreža (engl. *local area network*, skraćeno LAN) ili bilo koja druga. U ovom radu će se prikazati kako se stvoreni sustavi spajaju preko Internet mreže, te neke od mogućnosti koje nam se pružaju umrežavanjem.

Sustavi koji će se kreirati su Arduino vozilo na daljinsko upravljanje te Android aplikacija koja će se izvršavati na pametnom telefonu, koji ima mogućnost spajanja na Internet mrežu. Arduino sustav se sastoji od niza komponenata koji zajedno čine cjelinu te ju Android aplikacija može nadzirati, upravljati njome. Cilj rada je izraditi Android aplikaciju koja može komunicirati s Arduinom bez žičanim putem te tako upravljati vozilom. Ta aplikacija ima mogućnost reguliranja brzine vozila kao i odabiranja smjera vožnje. Arduino platforma vraća aplikaciji podatke koliko je neki objekt blizu vozila te tako aplikacija može grafički prikazati podatke korisniku. U radu će se opisati kako radi komunikacija, te kako se rješavaju neki problemi koji mogu nastati u njoj.

Kroz naredna poglavlja prikazat će se i objasniti način izvedbe i funkcioniranje cjelokupnog rada, kao i pojedinih komponenata koji igraju veću ulogu u komunikaciji. Pošto je glavno težište na Arduino uređaju, prvo poglavlje biti će posvećeno njemu. Objasnit će se ideja iza platforme, a zatim i korišteni hardverski dijelovi kao i softverski dio. Drugo poglavlje posvećeno je samome radu. Tu će se objasnit dodatne hardverske komponente koje nisu dio Arduino obitelji a igraju važnu ulogu. U sklopu tog poglavlja objasnit će se logika iza svake strane u komunikaciji, na način da se objasni generalno logičko ponašanje svake strane, korištene biblioteke (samo kod Arduina) te kod koji se izvršava i što on radi.

Zbog veće koncentriranosti na Arduino većina izvora dolaze sa njihove službene stranice, dok su za detaljniji uvid, u neke nedovoljno objašnjene tematike, uzeti preporučeni stručno napisani blogovi ili članci.

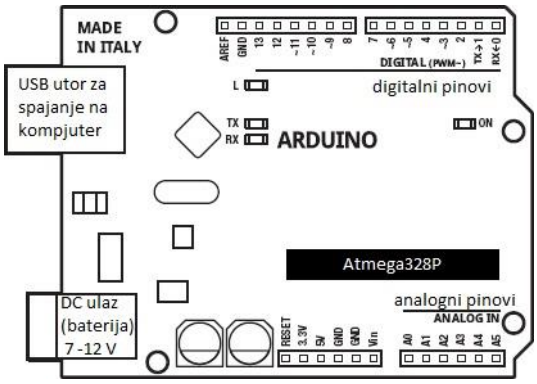
2. Arduino

Arduino je otvorena platforma koja je nastala kao rješenje na problem razvoja elektroničkih prototipa. Pomoću Arduina, osobama je olakšan razvoj aplikacija najviše zbog toga sto je lagano početi raditi na platformi. Platforma obuhvaća hardverske komponente i softver tj. IDE[1] potreban za kreiranje i prebacivanje programskog koda. Ono gdje se Arduino ističe jest u kombiniranom korištenju različitih izlaznim i ulaznim uređajima poput senzora, motora, monitora, led lampica, i sl. Arduino nije ništa drugo nego li jednostavan mikrokontroler koji s lakoćom izvršava jednostavne i repetitivne radnje. Također, zbog principa otvorenosti, daje se prostora entuzijastima da slažu vlastite uređaje ili pak IDE po uzoru na Arduino. To omogućuje korisnicima da nabave alternativne pločice koje funkcioniraju istom logikom ili pak da se izbjegne pisanje u c++ jeziku, što pruža službeni IDE, te programira u C#, pythonu ili u nekom vizualno prikazanom jeziku.

2.1. Arduino Uno

Jedan od najpoznatijih i najčešće korištenih Arduino pločica, Uno je izašao kao pratnja za IDE 1.0 verziju pa je popratno tome i dobio naziv. Ono sadržava 32 pina različitih namjena, utor za napajanje, USB[2] konekciju 2.0 tipa B, ICSP[3] zaglavlje (engl. *header*) i tipku za ponovno pokretanje. Za komunikaciju između IDE i Arduina potrebno je uspostaviti konekciju putem USB kabela. Uz prebacivanje koda na shield odnosno glavni mikrokontroler ono može dati i napajanje potrebno za izvršavanje zadatka. Kako bi mikrokontoler bio portabilan napajanje se može dobiti i preko baterije koja se spaja ili na prikladne pinove ili preko konektora za baterije

[1] Programsko okruženje (engl. *Integrated development environment*, skraćeno IDE)
[2] Univerzalna serijska sabirnica (engl. *Universal serial bus*, skraćeno USB)
[3] Ugrađeni programabilni sustav (engl. *In-circuit serial programming*, skraćeno ICSP)



Slika 2.1 Jednostavan prikaz Arduino Uno pločice

2.1.1. Mikrokontroleri

Uno model baziran je na Atmelovom mikrokontroleru ATmega328P, koji dolazi predprogramiran s ciljem lakšeg prebacivanja vlastitog koda na vlastiti uređaj. Također to znači da korisnici ne moraju biti upoznati s procesom instalacije ili imati ikakvih razumijevanja o čipu kako bi koristili Arduino. Ukoliko korisnik ne želi koristiti instalirani program za učitavanje (engl. *bootloader*) ono se može zaobići putem ICSP zaglavlja, koristeći Arduino ISP[4] ili slične programere. Tim procesom dobiva se na performansama te se oslobađa 0.5KB koju je prethodno koristio bootloader. Ukoliko se Uno koristi u kombinaciji s dodatkom (engl. *shield*) ICSP zaglavlje će se preklopiti, te će se koristiti bootloader dodatka.

Drugi mikrokontroler jest ATmega16U2 za rev3 ili 8u2 na starijim rev1 ili rev2 verzijama. Ono jest zapravo most između glavnog procesora, tj. ATmega328P i USB porta na računalu. Na njemu je instaliran Atmelov DFU[5] bootloader koji je kompatibilan sa više operacijskih sustava. Njega je također moguće zamijeniti vlastitim.

2.1.2. Pinovi

Kao i kod svih Arduina, pinovi kod verzije Uno spojeni su na priključke (engl. *port*) glavnog mikrokontrolera, u ovom slučaju to je ATmega328P. Pinovi rade na 5V te mogu pružiti ili opskrbiti 20mA što je u konačnici i preporučena jačina struje. U slučajevima davanja prejakog napajanja, tj. ako se pređe 40mA, Arduinu se može nanijeti trajna oštećenja.

[4] Drugi naziv za programabilni sustav (engl. *In-system programming*, skraćeno ISP)
[5] Pretprogramirani uređaj (engl. *Device firmware upgrade*, skraćeno DFU)

Postoji 14 digitalnih pinova koji su popraćeni oznakama od 0 do 13. Oni se mogu koristiti i kao ulazi i kao izlazi, s toga ih još možemo naći pod nazivom GPIO[6]. Digitalni signali funkcioniraju na jednostavan način – ima ili nema struje. Tj. prilikom slanja signala, na pin postavljamo vrijednost HIGH te se generira signal od 5V (najčešće signal za pokretanje uređaja), a ukoliko se pin postavi na LOW generirat će se 0V (najčešće signal za obustavljanje rada uređaja). Na pinove također mogu doći signali od 5 ili 0V šaljući time instrukcije za daljnju obradu.

Uz to, pinovi 3, 5, 6, 9, 10 i 11 korištenjem analogne funkcije za slanje mogu imitirati analogni signal. To su i dalje digitalni signali koji šalju vrijednost 0 ili 5V, no privid analognog signala se postiže promjenom vremena utrošenog na slanje određene vrijednosti tj. promjenom širine (engl. *width*) signala. Što znači da će pinovi i dalje slati krajnje vrijednosti, što je tipično za digitalne signale, no promjenom širine omogućit će veću kontrolu frekvencije imitirajući time postepeni pad vrijednosti, što je tipično za analogni signal. Takvu vrstu pinova nazivamo PWM[7] pinovima. Pinovima se postavlja vrijednost u rasponu od 0 do 255 korištenjem `analogWrite()` funkcije. U pozadini se svakih 2 milisekunde šalje novi impuls kojem je širina proporcionalna vrijednosti koju pokušava predočiti. Ako se treba na pin poslati vrijednost od 191 (75 posto od 255), Arduino šalje impulse koji će biti 1.5 milisekunde na HIGH, a 0.5 na LOW. Taj proces još nazivamo radni ciklus (engl. *duty cycle*). Pošto Arduino nema pretvarač digitalnih signala u analogni, PWM pinovi su jedino rješenje da se do izlaznih uređaja pošalje analogni signal. Pa se tako na PWM pinovima ne može koristiti `analogRead()`.

Kako bi se mogli zaprimati analogni signali, tu se nalaze 6 analognih pinova pod oznakama A0 – A6. Za razliku od digitalnih pinova koji funkcioniraju na principu ima ili nema napona, analogni pin može pročitati i vrijednosti između 0 i 5V. Što znači da umjesto krajnjih napona od 0 ili 5V oni mogu dobiti bilo koju vrijednost što je praktičnije za rad sa senzorima. Tu se koristi funkcija `analogRead()` koja prima vrijednosti od 0 do 1023. Očitana vrijednost napona pridaje se pripadajućoj vrijednosti funkcije, tj. učitani napon od 0V će dati vrijednost 0 dok 5V daje vrijednost 1023. Kako ATmega328P ima 6 kanala s mogućnošću pretvorbe analognih signala u digitalni, to svojstvo ima i Uno pločica pa prilikom nestašica digitalnih pinova, oni se mogu koristiti kao alternativa. Ukoliko se pinovi A0 – A6 koriste kao digitalni

[6] Pinovi za ulazno i izlazno korištenje (engl. *General-purpose input/output*, skraćeno GPIO)
[7] Modulacija širine impulsa (engl. *Pulse width modulation*, skraćeno PWM)

pinovi, brzina naspram ostalih digitalnih pinova jest manja zbog potrebnog vremena pretvorbe.

2.2. WiFi Shield

WiFi dodatak (engl. s*hield*), poznata još pod šifrom A000058, jest dodatak za Arduino pločicu koji omogućava bez žičanu komunikaciju sa drugim umreženim uređajima putem Interneta. Kako bi pločica mogla funkcionirati ona se postavlja na vrh Arduino uređaja preklopivši time sve pinove, kao i one za ICSP. Osim što dobiva napajanje od mikrokontrolera na koji je spojen, dodatak zauzima određeni broj pinova koji se ne smiju koristiti jer će u suprotnom funkcioniranje biti neispravno. Rezervirani pinovi su slijedeći:

- 13 – SCK[8]
- 12 – MISO[9]
- 11 – MOSI[10]
- 10 – SS[11] za WiFi
- 7 - Arduino-WiFi shield handshake
- 4 – SS za SD[12] karticu (samo ako se ona koristi)

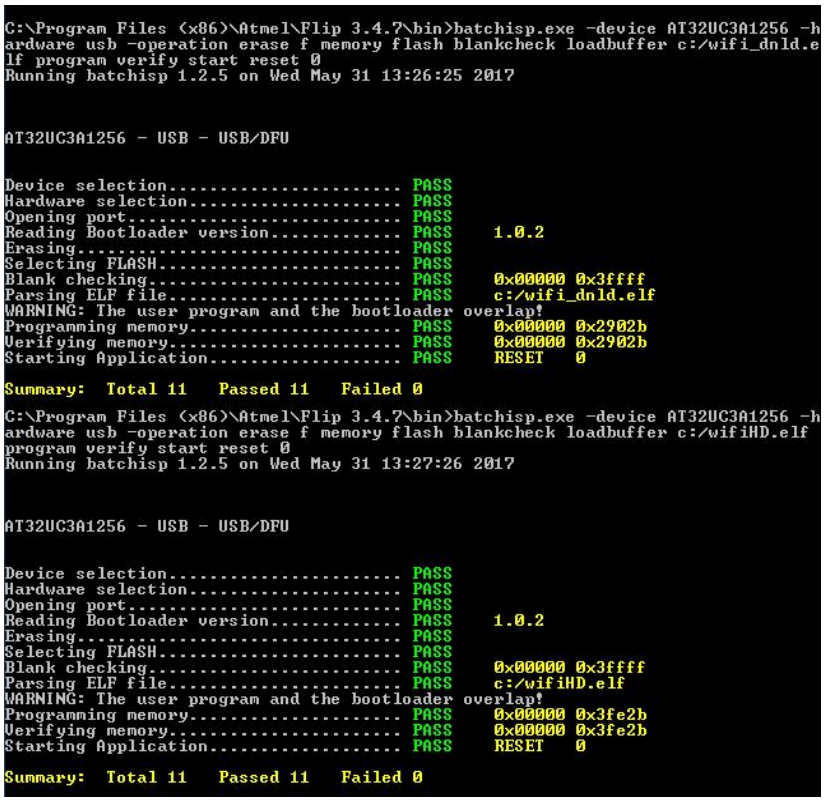
Ostali, ne korišteni pinovi, mogu se koristiti nesmetano pošto se signali propuštaju do mikrokontrolera na kojeg su spojeni. Funkcionalnosti shielda dostupne su putem WiFi i SD biblioteka. Dok se SD biblioteka koristi samo kod specifičnih namjena kao što je izlaganje datoteka putem Interneta, WiFi je neophodna biblioteka koja obavlja cijelu komunikacijsku logiku.

2.2.1. Instaliranje nove verzije ugrađenog programa

Priprema za uporabu WiFi pločice malo je kompliciranija, zbog dodatnog koraka u kojoj se obavlja obnova ugrađenog programa (engl. *firmware*). Po uputama sa službene stranice, potrebno je preuzeti novi firmware, te ga locirati u datoteku Arduinovog softvera na putanji `/hardware/avr/arduino/firmwares/wifishield`. Također, potrebno je imati instaliran AVR32 upravljački program (engl. *programmer*) i Flip program koji se može dobiti sa Atmelovih stranica. Zatim je potrebno spojiti WiFi pločicu putem USB utora te

[8] Serijski sat (engl. *Serial clock*, skraćeno SCK)
[9] Nadređeni ulaz podređeni izlaz (engl. *Master in slave out*, skraćeno MISO)
[10] Nadređeni izlaz podređeni ulaz (engl. *Master out slve in*, skraćeno MOSI)
[11] Podređeni okidač (engl. *Slave select*, skraćeno SS)
[12] Memorijska kartica (engl. *Secure digital*, skraćeno SD)

postaviti J3 spojnik (engl. *jumper*) u položaj za programiranje. Ukoliko WiFi pločica nije maknuta sa Arduina ili je spojnik u krivome položaju, proces obnove neće biti moguć. Nakon što su sve prethodni koraci zadovoljeni otvara se komandna linija (engl. *command prompt*), u ovom primjeru na Windows operacijskom sustavu. Direktorij se postavlja na `\Atmel\Flip 3.4.7\bin` , te se upisuju naredbe preuzete sa Arduinove službene stranice. Proces uspješne instalacije vidljiv je na slici (Slika 2.2).



Slika 2.2 Obnova ugrađenog programa

Kada proces instalacije završi, potrebno je maknuti USB i jumper, te postaviti pločicu na Arduino. WiFi pločica je spremna za rad te će pomoću LED[13] lampica, davati različite informacije poput:

- L9 (žuta boja) – vezan za digitalni pin 9
- Konekcija (zelena boja) – konekcija na mrežu je ostvarena
- Pogreška (crvena boja) – pogreška u komunikaciji
- Podaci (plava boja) – prilikom slanja ili primanja podataka

[13] Dioda (engl. *Light emitting diode*, skraćeno LED)

2.3. Arduino softver

Kada se kaže Arduino softver misli se prvenstveno na Arduinov službeni IDE koji dolazi s najčešće korištenim bibliotekama i jednostavnim primjerima koda. Unatoč jednostavnošću kod korištenja ono pruža osnovne funkcionalnosti te minimalnu pomoć korisniku prilikom pisanja koda. Komunikacija između hardverske i softverske komponente odvija se putem USB kabla no pošto sam IDE ne može automatski prepoznati o kojem se hardveru radi, potrebno je to postaviti u Bord kao i Serial Port koji se nalaze pod izbornikom Alati (engl. *tools*).

Arduinova skripte najčešće prate logičku formu u slijedećem nizu:

- Popis korištenih biblioteka
  - Nalaze se na samom vrhu skripte
  - Ne moraju biti prisutni
  - Npr. `#include <WiFi.h>`
- Popis varijabli
  - Sadržava tip varijable, naziv te vrijednost
  - U varijablama su također mapirani korišteni pinovi
  - Npr. `byte usrConnLed = 16`
- Funkcija setup
  - Izvršava se prva te se u njoj zadržava dok cijeli posao ne bude obavljen
  - Najčešće se koristi za pripremne radnje poput postavljanja načina rada pina, poput postavljanja na ulazni ili izlazni načina rada
  - `void setup(){ … }`
- Funkcija loop
  - Beskonačna petlja koja obavlja logiku programa, iz nje se izlazi jedino gašenjem ili ponovnim pokretanjem pločice
  - `void loop() { … }`

Skriptu je moguće provjeriti u Arduino IDE-u pritiskom na tipku Provjeri (engl. *verify*). Ukoliko se ne ispiše pogreška, postupak prebacivanja programa započinje se pritiskom tipke Upload. Ako prilikom procesa prebacivanja ili provjere programa završi neuspjehom, greška se ispisuje u konzoli IDE-a te kod neće biti prebačen.

Arduino ne podržava koncept višedretvenosti, s toga sve što se izvršava se mora nalaziti u `loop()` te se kod izvršava sekvencijalno.

2.3.1. WiFi.h biblioteka

WiFi pločica samo je jedna od komponenta koja omogućava spajanje na Internet mrežu. U kombinaciji s WiFi pločicom, ova biblioteka omogućava bez žičanu komunikaciju putem Interneta koristeći WEP ili WPA2 enkripciju. Korištenje biblioteke olakšava spajanje do te mjere da je ponekad za ispravno spajanje, potrebno jedino upisati naziv mreže i lozinku. Bibliotekom se omogućava da Arduino predstavlja ili korisničku ili klijentsku stranu, te da se u kombinaciji sa SD daje mogućnost dijeljenja datoteka. Komunikacija se odvija pomoću socketa što znači da se komunicira TCP/IP[14] protokolom. U ovom radu ne treba neki viši protokol poput HTTP ili HTTPS, već su dovoljni samo TCP/IP paketi. Iako poslani paketi ne sadržavaju neku posebnu formu (šalje se samo broj) to je dovoljno da se komunikacija uspostavi i da se poruka ispravno protumači. Pošto su zaprimljene instrukcije kompliciranije, vlastiti protokol se zaprima te tumači.

Biblioteku sačinjavaju više klasa koje se dodaju pomoću WiFi.h zaglavlja:

- WiFi
  - Inicijalizacija ethernet biblioteke i mrežne postavke
- IPAdress
  - Pribavlja informacije o mrežnim postavkama
- Server
  - Kreira server stranu
- Client
  - Kreira klijenta
- UDP
  - Omogućava primanje i slanje UDP poruka

Konkretnija objašnjenja korištenih funkcionalnosti opisana su u poglavlju 3.2.1.

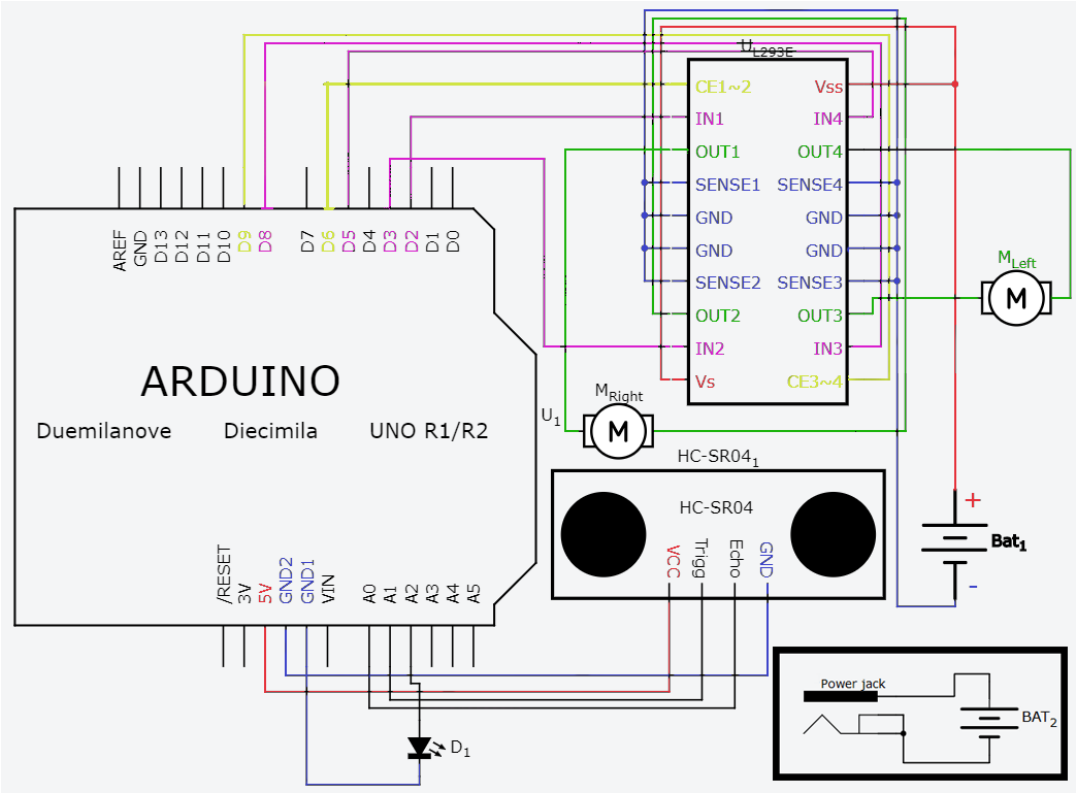
[14] Prijenosni protokol(engl. *Transmission control protocol/Internet protocol*, skraćeno TCP/IP)

3. Izvedba vozila i klijentske strane

3.1. Hardver vozila

Kako bi vozilo mogao obavljati funkcije, potrebno je pravilno postaviti sve potrebne komponente i ukomponirati sa instrukcijama na mikrokontroleru. Hardver vozila se sastoji od šasije (engl. *chassis*), Arduino Uno mikrokontrolera, WiFi pločice, dvije 9V baterije, US senzor, H most (engl. *bridge*), dva motora s kotačima, eksperimentalna pločica (engl. *breadboard*), LED lampica te žice koje sve to spajaju.

Na slici (Slika 3.1) prikazana je shema vozila Da bi se pojednostavio prikaz, WiFi shield je izostavljen iz sheme da bi se moglo lakše prikazati kako su uređaji spojeni na Arduino pločicu. Također tu se ne nalazi breadboard pošto on ne doprinosi funkcionalnostima.

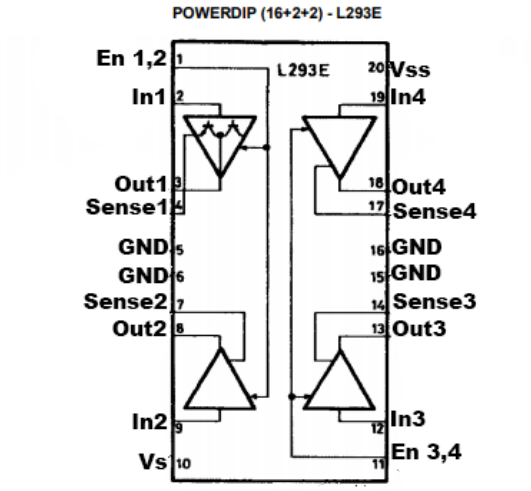


Slika 3.1 Arduino shema

3.1.1. H Bridge

Jedna od osnovnih funkcionalnosti vozila jest njegovo kretanje. Kako bi se moglo upravljati brzinom, kao i smjerom okretanja kotača, korišten je L293E čip. Ono je jedan od

predstavnika čipova koje nazivamo H mostovima. Ta skupina sklopova jest dobila ime po tome što sadržava 4 tranzistora koji podsjećaju na slovo H ukoliko ne teče struja. Motor se pokreće kombiniranim puštanjem struje, npr. ukoliko želimo da se vozilo kreće pravocrtno prednji tranzistori sa desne i lijeve strane moraju dobiti struju dok stražnji desni i lijevi ne. Ukoliko to obrnemo, vozilo će se kretati u drugome smjeru, a prilikom puštanja struje na samo jednu stranu vozilo će skrenuti u suprotnu.



Slika 3.2 L293E čip

L293E ima malo kompliciraniju shemu vidljivu na slici (Slika 3.2), no funkcionira na isti način. Ulazni napon se dobiva preko Vss i Vs pinova, gdje su u ovom slučaju opskrbljeni sa baterijom od 9V. Kontrolni pinovi (engl. e*nable*) služe kako bi propustili struju u čip te omogućili njen prolaz do ostalih sklopova. Tu se nalaze dva kontrolna pina (označena sa En) te se svaki brine za dva tranzistora sa svoje strane. Oni također omogućavaju regulaciju brzine spojenih motora, pa su tako spojeni na Arduinove PWN pinove(D9 i D6). Do tranzistora dolazi ulazni signal od Arduino uređaja putem In pinova. Ukoliko ima napona i kontrolni pin je postavljen na HIGH šalje se izlazni signal do motora putem izlaznih pinova. Senzor pinovi (engl. *sense*) služe za dodavanje praćenja otpora, no pošto se ta funkcionalnost ne koristi ti pinovi spojeni su u uzemljenje.

3.1.2. US senzor

Ultrazvučni senzor je uređaj koji omogućava praćenje udaljenosti objekta. Ono se sastoji od prijamnika zvučnih valova (engl. *reciever*), predajnika (engl. *transmitter*), pinovi za napon,

uzemljenje, okidač (engl. *trigger*) te osluškivač (engl. *echo*). Kada se želi saznati udaljenost nekog objekta od senzora, izvedba izgleda na slijedeći način:

```
digitalWrite(usTrig, LOW);
delayMicroseconds(2);
digitalWrite(usTrig, HIGH);
delayMicroseconds(10);
digitalWrite(usTrig, LOW);

duration = pulseIn(usEcho,HIGH);
distance = (duration / 2) * 0.0344;
```

Kôd 3.1 Snimanje udaljenih objekata US senzorom

Senzor snima kroz dvije glavne faze. Prva faza jest gdje se šalju zvučni valovi, tu se generira jedan impuls koji se pak sastoji od 3 djela. Na trigger pin šalje se LOW na dvije mikrosekunde a zatim na isti HIGH na 10 mikrosekundi. Kada se nanovo postavi na LOW to je znak da se pošalju zvučni signali, te senzor šalje 8 kratkih zvučnih signala koji putuju brzinom zvuka. Razlog slanja 8 signala jest da se oni odbiju o prepreke u nekom smjeru, te ne završe svi nazad do osluškivača. Druga faza započinje osluškivanjem povratnih signala. To izvodi funkcija `pulseIn`() koja očekuje povratni HIGH signal. No ukoliko se to ne ostvari, ona kratko blokira ostatak koda. Kada se vrati signal, funkcija će vratiti koje je vrijeme bilo potrebno da on stigne nazad te se sprema u `duration`. Dobiveni signal putovao je u dva smjera, od senzora do objekta i od objekta do senzora. Kako bi se što preciznije odredila udaljenost, dobiveno vrijeme dijelimo sa 2 te množimo sa konstantom 0.0344 što predstavlja količinu centimetara koju je signal prešao u jednom mikrosekundi.

3.2. Implementacija Arduino poslužitelja

Unatoč velikoj važnosti hardverskih komponenata, vozilo ne bi moglo odrađivati ikakav posao bez instrukcija koje daje kod. Logika instrukcija odvija se slijedećim koracima:

1. Postavljanje motor pinova
2. Pokušaj spajanja na WiFi konekciju pomoću dane šifre i naziva konekcije
3. Provjera da li postoji klijent koji je spreman na komunikaciju
4. Jednom nađen klijent pali se lampica, provjerava se postojanost komunikacije, šalju mu se informacije prikupljene US senzorom te se provjerava da li postoje dolazne poruke
5. Dolaskom poruke, ona se čita, preoblikuje u potreban format, instrukcije se postavljaju na motore

6. Odstupanje klijenta od konekcije, Arduino se vraća na korak 3
7. Gubljenjem Wifi konekcije Arduino se nastoji spojiti na istu mrežu do uspjeha

3.2.1. MyWiFi.h biblioteka

MyWiFi biblioteka omogućuje Arduinu da se spoji na mrežu te uspostavi komunikaciju s korisnikom. Za te potrebe koristi Arduinovu biblioteku WiFi.h te Arduino.h za upravljanje led lampicom. Kroz konstruktor se primaju `byte led` koja sadrži broj pina na koji je led spojen, polja karaktera `ssid` i `pwd` koji sadržavaju naziv konekcije i lozinku.

```
MyWiFi::MyWiFi(byte led, char* ssid, char* pwd){
      pinMode(led, OUTPUT);

      _led = led;
      _ssid = ssid;
      _pwd = pwd;
      _server = new WiFiServer(80);
}
```

Kôd 3.2 MyWiFi.h konstruktor

Način rada `led` pina postavlja se na `output` mod što signalizira da će se prema tom pinu slati signali, a ne primati. Svaka zaprimljena varijabla u toj metodi sprema se u odgovarajuću privatnu varijablu koja ih čuva za daljnju upotrebu. Također tu se instancira pokazivač (engl. *pointer*) na `WiFiServer* _server`. To je klasa koja dolazi od WiFi.h biblioteke te omogućava kreiranje servera na određenom, danom portu. Za potrebe spajanja u MyWiFi.h napravljene su slijedeće metode:

`connectToNetwork()` spaja uređaj na mrežu pomoću _ssid i pwd.

```
void MyWiFi::conntectToNetwork(){
      while (WiFi.status() != WL_CONNECTED){
            WiFi.begin(_ssid, _pwd);
      }

      _server->begin();

      digitalWrite(_led, HIGH);
      delay(100);
      digitalWrite(_led, LOW);
      delay(100);
      digitalWrite(_led, HIGH);
```

```
            delay(100);
            digitalWrite(_led, LOW);
        }
```

Kôd 3.3 Tijelo connectToNetwork()

Tu se započinje beskonačna petlja koja pomoću `WiFi.status()` provjerava da li je uređaj spojen na mrežu. Sve dok ono nije spojeno, pokušavat će se spojiti pomoću `WiFi.begin(_ssid, _pwd)` koja može vratiti jedan od dva statusa: `WL_CONNECTED` ili `WL_IDILE_STATUS` (nije spojen na mrežu). Uspješnim spajanjem izlazi se iz petlje, poziva se metoda `_server->begin()` na klasi `WiFiServer`, koja pokušava dohvatiti već postojeći socket ili kreirati novi ukoliko nije uspio naći otvoreni. Time se započinje slušanje dolazne komunikacije. Tim radnjama na Wifi shieldu zasvijetli lampica za ostvarenu konekciju te led lampica dva puta zasvijetli.

`network()` provjerava da li je uređaj spojen na željenu mrežu koristeći `WiFi.status()`.

`clientAvailable()` dohvaća dostupnog klijenta na serveru koji je spremna za komunikaciju putem `_server->available()` te vraćenog klijenta postavlja u `WiFiClient` varijablu `_client`. Ukoliko postoji klijent, pali se led lampica te se šalje klijent kao povratna vrijednost.

`clientConnected()` provjerava da li je komunikacija s klijentom postojana koristeći `_client.connected()`

`clientMsg()` provjerava da li je klijent poslao podatke za čitanje koristeći _client.available() koji vraća broj bajtova koji su spremni za čitanje.

`readMsg()` čita klijentsku poruku pomoću `_client.readStringUntil('%')`. Koju zapravo dobivamo u klasi `Stream`. koju `WiFiClient` preko Client klase. Ono čita niz (engl. *stream*) sve dok ne naiđe na znak % koji je u ovom slučaju korišten kao zadnji kontrolni znak u instrukciji postavljanja vrijednosti u motore koja je dobivena od klijentske strane. Primjer dobivenog `stringa` je vidljiv u poglavlju 3.3.1.

`sendDistance(long distance)` šalje klijentu poruku koja sadrži podatak o udaljenosti objekata. Pošto Arduino jedino šalje udaljenost dobivenog od strane US senzora, metoda je nazvana prikladno te prima `long` varijablu. Koristi se `_client.println()`.

`removeClient()` prekida komunikaciju s klijentom pomoću `_client.stop()`. Ta metoda čeka maksimalno 5 sekundi prije zatvaranja `socketa` povezanog na klijenta. Po zatvaranju komunikacije sa korisnikom, led lampica se gasi i time signalizira da je Arduino spreman za ostvarivanje nove komunikacije.

3.2.2. MyUS biblioteka

Biblioteka koja koristi ultrazvučni (engl. *ultrasonic*) senzor naziva se MyUS. Ono u svoj konstruktor prima pinove za trigger i echo.

```
MyUS::MyUS(byte pinEcho, byte pinTrig){
      pinMode(pinTrig, OUTPUT);
      pinMode(pinEcho, INPUT);

      _pinEcho = pinEcho;
      _pinTrigg = pinTrig;
      _minRange = 3;
      _maxRange = 200;
}
```

Kôd 3.4 MyUS konstruktor

Instanciranjem objekta pokreće se konstruktor u kojemu se pinovi spremaju u lokalne, privatne varijable `byte _pinEcho, _pinTrigg`, minimalna udaljenost se postavlja na `3`, dok maksimalna na `200`. Biblioteka sadržava samo jednu metodu `long pingPong()` koja izračunava udaljenost obližnjih objekata, njena izvedba je vidljiva u poglavlju 3.1.2. Ukoliko je izračunata _distance unutar okvira `_minRange` i `_maxRange`, ta vrijednost biti će vraćena, u suprotnome povratna vrijednost je `-1`.

3.2.3. Izvedba

Kao što je spomenuto u poglavlju 2.3, Arduinova skripta započinje nabrajanjem potrebnih biblioteka. Tu se dodaju `MyWiFi.h`, `MyUS.h` i `WiFi.h`. Unatoč tome što `MyWiFi.h` već ima dodanu Wifi biblioteku, ona se mora još jednom referencirati kako bi ju glavna skripta mogla naći. Zatim imamo deklaraciju varijabli:

- char ssid[]
  - polje karaktera koje sadržava naziv mreže na koju se spajamo
- char pwd[]
  - polje karaktera koje sadržava lozinku mreže

- byte led = 16
  - led lampica nalazi se na pinu A2
- byte echo = 14
  - US senzor echo pin na A0
- byte trigger = 15
  - US senzor trigger pin na A1
- byte eRight = 9
  - desni enable za motor
- byte eLeft = 6
  - lijevi enable za motor
- byte mRight1 = 8, mRight2 = 5
  - desni motor
- byte mLeft1 = 3, mLeft2 = 2
  - lijevi motor
- int usrRight, usrLeft
  - instrukcija dobivena od strane korisnika za desni i lijevi motor
- MyWiFi myWifi(led, ssid, pwd=
  - instanciranje MyWiFi klase
- MyUS myUS(echo, trigger)
  - instanciranje MyUS klase

Postavljanje načina rada na pinove za motore odvija se u funkciji `setup()` dok se za US senzor i led lampicu brinu prikladne biblioteke. Kako bi se osiguralo da se Arduino i pri gubljenju konekcije s Wifi može nanovo spojit, konekcija se odvija u `loop()` funkciji pomoću `myWifi.conntectToNetwork()`. Ona pak ima svoju petlju koja se neće prekinuti sve dok se ne izvrši zadaća funkcije pa tako se blokira ostatak funkcionalnosti. Uspješnim spajanjem slijedi cijeli proces provjera i petlji koje su vezane za uspostavu komunikacije s korisnikom.

```
while(myWifi.network()){
    if(myWifi.clientAvailable()){

    while(myWifi.clientConnected()){
        myWifi.sendDistance(myUs.pingPong());

        if(myWifi.clientMsg())
        /…
```

Kôd 3.5 poredak petlji i provjera

Prva petlja provjerava da li je konekcija sa mrežom održiva te konstantno provjerava da li postoji korisnik koji je spreman komunicirati sa serverom. Kada ga jednom dohvatimo, mora

se provjeravati da li je pak komunikacija s njim održiva. Ukoliko su svi prethodni uvjeti ostvareni, korisniku se šalju informacije dobivene od strane US senzora korištenjem `myUS.pingPong()`. Tako server cijelo vrijeme šalje informacije sve do trenutka kada `myWifi.clientMsg()` ne javi da je korisnik poslao poruku.

```
String cInput = myWifi.readMsg();
parseString(cInput);
```

Poruka se sprema u varijablu te ju pošalje dalje na formatiranje kako bi se lakše ovladalo dolaznim instrukcijama.

```
for(int i = 0; i < cInput.length(); i++){
  if(isDigit(cInput[i]) || cInput[i] == '-'){
    helper += cInput[i];
  }
  if(cInput[i] == '#' || cInput.length() == i+1)
  {
    helperArray[index] = helper.toInt();
    helper = "";
    index++;
  }
}
```

Kôd 3.6 Petlja za formatiranje korisničke instrukcije

Pošto dobivena instrukcija ima dva kontrolna znaka, kod `stringa` je potrebno pregledavati svaki karakter. Primjer instrukcije vidljiv je u poglavlju 3.3.1 te će demonstrirati prolaz pomoću instrukcije za kretanje unazad.

Prvim prolazom gleda se prvi znak `cInput` varijable; pošto je poslana instrukcija za kretanje unazad, prvi znak će biti znak minus. Pošto minus ne sugerira na prestanak čitanja ono se sprema u `helper` varijablu. Petlja se ponavlja tako dok ne naiđe na znak ljestvice (engl. *hash*). Ono signalizira da će ostatak instrukcije biti vezan za lijevi motor pa se tako `helper` vrijednost sprema u `usrRight` i postavlja na prazan `string`. Čitanje se nastavlja istom logikom, spremajući učitane vrijednosti uključujući i znak minus. Dolaskom do zadnjeg znaka u `stringu` dolazimo do zadnjeg kontrolnog znaka pa se tako njega ignorira, a `helper` se sprema u `usrLeft`. Time varijable `usrLeft` i `usrRight` sada imaju vrijednost `-150`.

Završenim formatiranjem započinje postavljanje prikladnih pinova na prikladno stanje. Tu postoje tri moguća događaja za svaki motor:

- instrukcija je veća od nule
  - usrLeft i usrRight nemaju predznak
  - mLeft1 i mRight1 postavljaju se na HIGH, dok mLeft2 i mRight2 na LOW
- instrukcija je manja od nule
  - usrLeft i usrRight imaju minus za predznak
  - mLeft1 i mRight1 postavljaju se na LOW, dok mLeft2 i mRight2 na HIGH
- instrukcija je jednaka nuli
  - usrLeft i usrRight su jednake nuli
  - na sve motore postavlja se LOW

Kako se odredio smjer kretanja potrebno je maknuti minus predznak ukoliko on postoji, zato što brzina ne smije imati negativan predznak. Također, zbog nesavršenosti fizičkih komponenta, jedan motor uvijek će potezati ječe te će uzrokovati neželjeno skretanje vozila. To se postiže povećanjem vrijednosti brzine za jedan motor (+40 na lijevi) i smanjenje na drugi (-10 na desni) te se također mora prekontrolirati da se ne prijeđe maksimalna vrijednost od `255`.

```
analogWrite(eLeft, usrLeft);

analogWrite(eRight, usrRight);
```

Poslavši analogni signal na aktivatore (engl. *enabler*) motora završavano obrađivanje korisničke instrukcije te se nastavlja sa provjerom održivosti konekcije s korisnikom i slanjem informacija prikupljenih US senzorom. Ukoliko korisnik u bilo kojem trenutku odstupi od konekcije poziva se `myWifi.removeClient()` koja miče korisnikov `socket`, gasi lampicu a motori se gase.

3.3. Klijentska strana

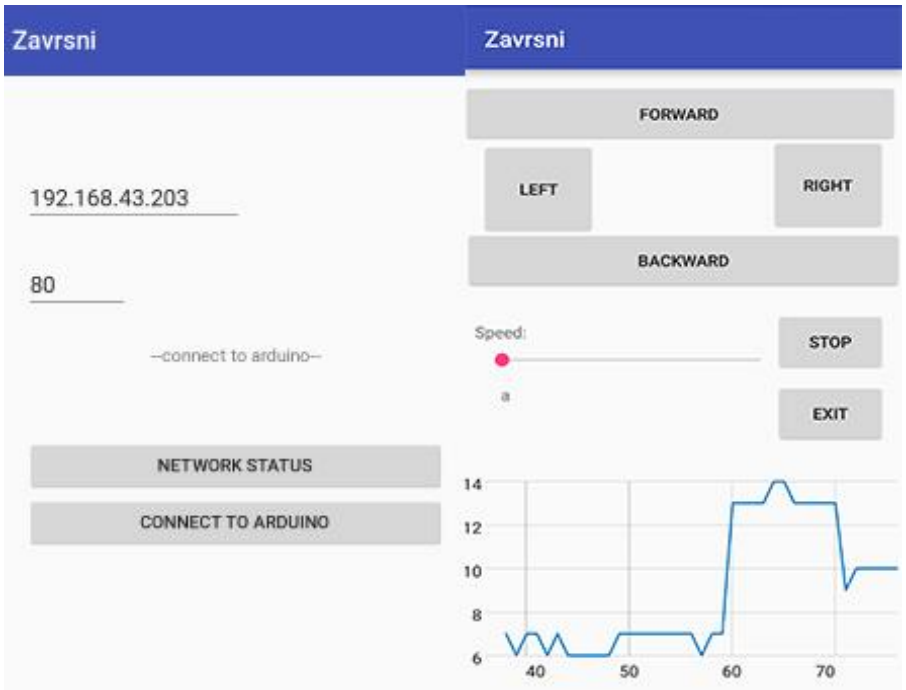
Klijent u ovom slučaju predstavlja korisnička aplikacija koja se spaja na Arduinov server pomoću IP adrese i porta te daje instrukcije za kretanje vozila dok za to vrijeme se ucrtavaju snimke US senzora u graf. Logika klijentske strane odvija se na slijedeći način:

- Korisnik se spoji na istu mrežu kao Arduino
- Otvorivši aplikaciju, korisnika se šalje na početnu aktivnost (engl. a*ctivity*)
- Korisnik unosi IP adresu i port
- Otvara se nova aktivnost
  - Ukoliko je konekcija neispravna, korisnika dolazi do druge aktivnosti ali ga se vrati na prvu te se ispiše poruka

- Nakon uspješnog spajanja na Arduino, korisnik ostaje na aktivnosti sa upravljačkim kontrolama
- Aplikacija cijelo vrijeme prima podatke snimljene US senzorom, te se oni ocrtavaju na grafu
- Odabirom željene brzine i pritiskom tipke smjera šalju se instrukcije vozilu
- Korisnik u bilo kojem trenutku može zaustaviti konekcijom pritiskom tipke „Exit“ čime ga se vrati na prvu aktivnost

3.3.1. Izvedba

Aplikacija se sastoji od dvije aktivnosti, a prva koja se pokreće jest `MainActivity`.



Slika 3.3 MainActivity i ControllsActivity

To je aktivnost koja prikuplja potrebne informacije za spajanje na Arduino vozilo. Nakon što su ulazni parametri sa informacijama o IP adresi i portu upisani, pritiskom na `CONNECT TO ARDUINO` gumb. Pokreće se metoda `openControllsActivity()` koja kupi korisniče unose te ih šalje u drugi, novootvoreni `Intent`

```
private void openControllsActivity(){
    String ip = etIP.getText().toString();
    String port = etPort.getText().toString();

    if(!checkIpAdress(ip)){
        twConnStatus.setText(WRONG_IP_FORMAT);
        return;
```

```
        }
        Intent intent = new Intent(this,
    ControllsActivity.class);
        intent.putExtra("ip", ip);
        intent.putExtra("port", port);
        MainActivity.this.startActivityForResult(intent,
    ARDUINO_INTERACTION);
    }
```

Kôd 3.7 Otvaranje nove aktivnosti

Prebacivši se time na aktivnost `ControllsActivity`, napravit će se instanca klase `Client` te će se pokušati spojiti na Arduino uređaj.

Klasa `Client` jest privatna klasa koja je smještena unutar same aktivnosti kako bi se lakše pristupalo UI elementima te nasljeđuje `AsyncTask`. Time se osigurava da će se komunikacija odvijati na drugoj dretvi (engl. *thread*) te omogućiti neometano korištenje drugih dijelova aplikacije. Pozivom metode `execute()` na instanci klijenta pokreče se dretva.

```
Client client = new Client();
client.execute(ip, port);
```

Za komunikaciju između klijentske i serverske strane koriste se `Socketi`. To su krajnje točke u komunikaciji koje u ovome slučaju imaju ulogu klijenta. Kako bi se uspostavila komunikacija sa serverskom stranom, potrebno je `Socketu` dati IP adresu te ulaz (engl. *port*) koji se osluškuje. Te informacije su prethodno unesene i poslane pomoću `Intenta`, pa ih tako dohvaćamo pomoću `getIntent().getStringExtra()`. Ukoliko se server ne uspije spojiti na Arduino, korisnika će nakon par sekundi prebaciti nazad na glavnu aktivnost. Uspostavom veze između Android `Socketa` i poslužitelja instanciraju se `DataInputStream` i `DataOutputStream`. To su `stream` klase kojima se pošalje kreirani `socket` te oni omoguće primanje i slanje paketa putem TCP/IP protokola.

U kodu to prati slijedeću logiku:

```
private class Client extends AsyncTask<String, String, Void>
{
    Socket socket = null;
    DataInputStream dataInputStream = null;
    DataOutputStream dataOutputStream = null;
```

```
    @Override
    protected Void doInBackground(String... params) {
        String ip = params[0];
        int port = Integer.valueOf(params[1]);

        try {
            socket = new Socket(ip, port);
            dataInputStream = new
DataInputStream(socket.getInputStream());
            dataOutputStream = new
DataOutputStream(socket.getOutputStream());

            while (socket.isConnected()) {
                if (isCancelled()) {
                    socket.close();
                    returnOk = true;
                    break;
                }

                String dataReturn =
dataInputStream.readLine();
                publishProgress("//", dataReturn);
            }
        }  catch (Exception e) {
            e.printStackTrace();
            returnOk = false;
        }

        returnResult();
        return null;
    }
```

Kôd 3.8 Klasa Client

Sve dok je `socket` spojen i ne zahtjeva se zatvaranje dretve, `Client` će se nalaziti u beskonačnoj petlji čitajući podatke. Primanje ulaznih podataka obavlja se pomoću `readLine()` koju pozivamo sa `DataInputStreama`. Ona je korištena zbog toga što čita dolazne parametre sve dok ne naiđe na oznaku za novi red, pa se tako osigurava da nikada neće zapeti u beskonačnoj petlji kao što bi bilo u slučaju da se koristi preporučena `readUTF()`. Da bi se omogućilo korištenje podataka od strane UI elemenata, poziva se funkcija `publishProgress()`. Kao parametar potrebno je predati rezultate dretve koja

ju poziva. U ovom slučaju šalju se dva `stringa`, od čega je prva oznaka kontrolna koja će dati do znanja da se radi o podatku koju smo dobili sa servera. Drugi podatak je dobivena vrijednost sa servera. Pozivom te metode pokreće se `onProgressUpdate()` koji se poziva na glavnoj tj. UI dretvi. `Client` nastavlja sa svojim učitavanjem, dok se paralelno poslani podaci obrađuju u `onProgressUpdate()`. Unatoč što su se metodi poslala dva `stringa`, `onPublishUpdate()` ih dobiva u obliku polja. Kako je prvi parametar bio kontrolni znak, njega vadimo sa prvog mjesta te ono prolazi `if` uvjet.

```
        @Override
        protected void onProgressUpdate(String... values) {
            try {
                if(values[0].equals("//")){
                    String dataReturn = values[1];
                    fifo.add(Integer.parseInt(dataReturn));
                } else {
                    String motorInstruction = values[0];
                    dataOutputStream.writeUTF(motorInstruction);
                }
            } catch (Exception e) {
                /…
            }

        }
    }
```

Kôd 3.9 Metoda onProgressUpdate()

Parametri dobiveni sa servera trebaju se prikazati po slijedu dolaska kako bi graf prikazivao realnu snimku. Za te potrebe koristit će se FIFO algoritam kroz `Queue` interface. Ono sadrži funkcije poput `remove()` koja vraća vrijednost sa početka liste te ju briše i funkciju `add()` koja umeće vrijednosti na kraj.

```
Queue<Integer> fifo = new LinkedList<>();
```

Za ocrtavanje vrijednost grafa korištena je biblioteka treće strane GraphView API[15]. To je besplatan API pod Apache licencom koji omogućava korištenje biblioteke u ne komercijalne svrhe. Graf je potrebno referencirati iz resursa te ga se priprema pomoću metode `prepareGraph()`.

[15] Aplikacijsko programsko sučelje (engl. *Application programming interface*, skraćeno API)

Iscrtavanje grafa se također mora odvijati u zasebnoj dretvi kako ne bi ometao ostalim radnjama.

```
private Runnable timer = new Runnable() {
@Override
public void run() {
    if(!fifo.isEmpty()){
    graphLastXvalue += 1d;
    series.appendData(new DataPoint(graphLastXvalue,
fifo.remove()), true, 40);
    }
    handler.postDelayed(this, 200);
}
}
```

Kôd 3.10 Iscrtavanje grafa

Sve prethodno objašnjene radnje nisu zahtijevale interakciju sa korisnikom. Ono što je njemu omogućeno jest upravljanje vozilom koja se odvija gumbima za naprijed, nazad, lijevo, desno, traku za odabir brzine, tipku za zaustavljanje ili pak prekid komunikacije. Sve navedene radnje ostavljene su na glavnoj dretvi pošto su one u konačnici glavne funkcionalnosti aplikacije. Kako bi se vozilo pokrenuo potrebno je odabrati početnu brzinu i odabrati smjer kretanja. Vrijednost brzine posprema se u varijablu `speed` dok se za pritisak gumba poziva `setOnClickListener()`. Ovisno o tome koji smo gumb odabrali, kreirat će se prikladni `String` koji propisan po slijedećem protokolu:

- Na početak se stavlja predznak za desni motor, stavit će se – ukoliko će se vozilo kretati unazad, u suprotnome predznak ne postoji
- Prvi broj predstavlja brzinu koja će biti postavljena na desni motor
- # znakom odvaja se naredba za desni i lijevi motor
- Postavlja se predznak za lijevi motor (ukoliko je desni motor imao negativan predznak, imat će ga i lijevi)
- Drugi broj predstavlja brzinu koja će biti postavljena na lijevi motor
- % znakom označava se kraj instrukcije

Primjer prikaza:

- 0#0%
    - Naredba za zaustavljanje vozila
- 150#150%
    - Vozilo će se kretati naprijed brzinom 150 naprijed
- -150#-150%
    - Vozilo će se kretati brzinom 150 unazad

- 150#0%
    - Vozilo će skretati brzinom 150 ulijevo

Kako bismo sa UI dretve pristupili `Clientu`, a time pristupili i `socketu`, na njegovoj instanci se ponovo poziva `onProgressUpdate()`. Ovog puta pokreće se isključivo pozvana metoda kojoj predajemo samo jedan `string`, što znači da nema kontrolnog znaka. Pošto njega nema, kod provjere prvog znaka završit će se u `else` bloku. Kako je to metoda koju koristi dretva `Client` preko nje možemo direktno pristupiti `dataOutputStreamu` te time riješiti problem komunikacije više dretvi. Na `streamu` se poziva metoda `writeUTF()` koja šalje serveru instrukcije.

```
if(values[0].equals("//")){//…}
else {
    String motorInstruction = values[0];
    dataOutputStream.writeUTF(motorInstruction);
}
```

Navedeni procesi ponavljat će se sve do trenutka kada korisnik odlučit da više ne želi upravljati vozilom. U tom trenutku pritiskom na tipku `Exit` pozivaju se metode za zatvaranje konekcije, dretve i vraćanje rezultata prvoj aktivnosti.

```
btnExit.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        closeConnection();
        returnResult();
    }
});
}
```

Kôd 3.11 Gumb za povratak na MainActivity

Kako vozilo ne bi ostao u pokretu kada se korisnik napusti komunikaciju, `closeConnection()` će poslati zadnju instrukciju za gašenje motora. Također će pozvati `cancel(true)` na `clientu`. Time će dretva koja još uvijek zaprima informacije od Arduina ući u `if` provjeru, zatvoriti `socket`, te postaviti varijablu `returnOK` na `true` pošto je konekcija dobrovoljno zatvorena. Slijedeća na red za izvođenje dolazi `returnResult()` koja postavlja rezultat `intenta` i završava `ControllsActivity`. Dolaskom nazad na `MainActivity` u labelu se ispisuje rezultat, te se daje korisniku ponovna mogućnost da uspostavi konekciju s Arduinom.

Zaključak

Internet stvari se u današnje vrijeme sve češće koristi te će se s vremenom samo rasti njegova prisutnost i upotreba. Već danas se koristi u brojnim mjestima i industrijama. Ovim radom se opisuje rad i komunikacija dva različita sustava bez žičanim putem. To su Android pametni telefon i vozilo bazirano na Arduino platformi s dodatnim senzorom. Unatoč tome što se na jednostavan način prikazuje način komuniciranja i funkcioniranja, ono se može primijeniti na veće i kompliciranije sustave i uređaje. Osmišljeni komunikacijski protokol je fleksibilan, čovjeku čitljiv te se može adaptirati na razne situacije.

Arduino platforma spaja se na Internet korištenjem biblioteke omotača koja koristi funkcionalnosti Arduinove biblioteke. Stvara se poslužitelj koji konstantno održava komunikaciju s korisnikom sve dok je mreža na koju je spojen održiva. Arduino prvo očita stanje na senzoru, pošalje ga klijentu, a zatim i primi instrukciju te ju izvrši, tj. svaku akciju izvršava jednu po jednu zbog nedostatka svojstva višedretvenosti. Unatoč tome sve funkcije odvijaju se dovoljno brzo da se stvori primjer paralelnog izvršavanja.

U ovom primjeru, Android aplikacija predstavlja klijenta koji se spaja preko Internet mreže na Arduino server. Kod korisničke strane koristi se mogućnost višedretvenosti, kako bi se paralelno moglo slati instrukcije i zaprimati snimke te ih vizualno prikazati. Problem nastaje kada više dretvi trebaju koristiti jedan dijeljeni resurs za čitanje i pisanje. To je riješeno implementacijom FIFO algoritma, gdje dretva koja dobiva podatke sa servera postavlja vrijednosti na kraj, a dretva koja ispisuje te vrijednosti ih vadi sa početka. Spajanje na Internet odvija se pomoću ugrađenih klasa, te se stvorio protokol prilagođen potrebama slanja instrukcija na Arduino. Za vizualni prikaz snimaka senzora, korišten je graf biblioteke treće strane GraphView koji svakih par milisekundi osvježava graf novim podacima.

U IT svijetu, komunikacija više različitih uređaja postaje sve češća, jednostavnija, ali u suštini svi moraju imati nekakav protokol koji sudionici moraju razumjeti. Ovdje su platforme komunicirale putem Interneta, no jednako tako se ista radnja mogla izvršiti preko drugog medija kao npr. Bluetooth, serijskom žicom ili pak radio signalima.

Popis kratica

IDE	*Integrated development environment*	programsko okruženje
USB	*Universal serial bus*	univerzalna serijska sabirnica
ICSP	*In-circuit serial programming*	ugrađeni programabilni sustav
ISP	*In-system programming*	drugi naziv za ICSP
DFU	*Device firmware upgrade*	pretprogramirani uređaj
GPIO	*General-purpose input/output*	pinovi za ulazno i izlazno korištenje
PWM	*Pulse width modulation*	modulacija širine impulsa
SCK	*Serial clock*	serijski sat
MISO	*Master in slave out*	Nadređeni ulaz podređeni izlaz
MOSI	*Master out slave in*	Nadređeni izlaz podređeni izlaz
SS	*Slave select*	podređeni okidač
SD	*Secure digital*	memorijska kartica
LED	*Emitting diode*	dioda
TCP/IP	*Transmission control protocol/Internet protocol*	prijenosni protokol
API	*Application programming interface*	Aplikacijsko programsko sučelje

Popis slika

Slika 2.1 Jednostavan prikaz Arduino Uno pločice 3

Slika 2.2 Obnova ugrađenog programa 6

Slika 3.1 Arduino shema 9

Slika 3.2 L293E čip 10

Slika 3.3 MainActivity i ControllsActivity 18

Popis kôdova

Kôd 3.1 Snimanje udaljenih objekata US senzorom 11

Kôd 3.2 MyWiFi.h konstruktor 12

Kôd 3.3 Tijelo connectToNetwork() 13

Kôd 3.4 MyUS konstruktor 14

Kôd 3.5 poredak petlji i provjera 15

Kôd 3.6 Petlja za formatiranje korisničke instrukcije 16

Kôd 3.7 Otvaranje nove aktivnosti 19

Kôd 3.8 Klasa Client 20

Kôd 3.9 Metoda onProgressUpdate() 21

Kôd 3.10 Iscrtavanje grafa 22

Kôd 3.11 Gumb za povratak na MainActivity 23

Literatura

[1] Banzi, Massimo *Getting started with Arduino, 2nd edition*. O'Reilly Media, 2009

[2] Arduino Guide, https://www.arduino.cc/en/Guide/HomePage, listopad. 2017.

[3] Arduino Language Reference, https://www.arduino.cc/reference/en, siječanj 2018

[4] Arduino Glossary, https://www.arduino.cc/glossary/en, studeni 2017

[5] Arduino Uno, https://store.arduino.cc/arduino-uno-rev3, veljača 2018

[6] Arduino WiFi.h library, https://www.arduino.cc/en/Reference/WiFi, siječanj 2018

[7] Arduino PWM https://www.arduino.cc/en/Tutorial/PWM, siječanj 2018

[8] Secrets of Arduino PWM https://www.arduino.cc/en/Tutorial/SecretsOfArduinoPWM, veljača 2018

[9] US senzor, https://www.robot-electronics.co.uk/htm/srf05tech.htm, listopad 2015

[10] H bridge, http://www.modularcircuits.com/blog/articles/h-bridge-secrets/h-bridges-the-basics, veljača 2018

[11] L293E Datasheet, https://www.mouser.com/ds/2/389/l293b-954809.pdf, srpanj 2003

[12] AsyncTask, https://developer.android.com/reference/android/os/AsyncTask.html, prosinac 2017

Algebra
visoka škola za
primijenjeno računarstvo

VOZILO NA DALJINSKO UPRAVLJANJE

Pristupnik: Helena Martinović, 0321003268

Mentor: Prof. Andrej Lacković