SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1300

INTERAKTIVNO GRAFIČKO SUČELJE ZA ELEKTRIČNO VOZILO

Ana Smolić

Zagreb, srpanj 2016.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1300

INTERAKTIVNO GRAFIČKO SUČELJE ZA ELEKTRIČNO VOZILO

Ana Smolić

Zagreb, srpanj 2016.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 10. ožujka 2016.

DIPLOMSKI ZADATAK br. 1300

Pristupnik: **Ana Smolić (0036459031)**
Studij: Elektrotehnika i informacijska tehnologija
Profil: Elektroničko i računalno inženjerstvo

Zadatak: **Interaktivno grafičko sučelje za električno vozilo**

Opis zadatka:

U okviru diplomskog rada potrebno je razviti interaktivno grafičko sučelje za kontrolnu ploču električnog bicikla. Kontrolna ploča je zasnovana na Arduino platformi, a osim OLED displeja sadrži i kapacitivni senzor osjetljiv na dodir. Sučelje je potrebno razviti u programskom jeziku C/C++, a treba sadržavati odgovarajuće indikatore i izbornike za prezentaciju i unos osnovnih parametara vožnje. Arduino računalo će korištenjem I2C međusklopa komunicirati s ARM računalom vozila. Dodatno, korištenjem Bluetooth Low Energy (BLE, Bluetooth 4.x) međusklopa ostvarit će se komunikacija s mobilnim uređajem (pametni telefon, tablet, ...).
Potrebno je razviti programsku potporu za mobilni uređaj na Android operacijskom sustavu. Za razvoj sučelja koristiti Adobe PhoneGap razvojnu okolinu ili Android SDK. Interaktivno sučelje će prikazivati, pohranjivati i odabirati sve parametre vožnje, kao i odabrane podatke s vlastitih senzora mobilnog uređaja.
Za detaljne informacije obratiti se mentoru.

Zadatak uručen pristupniku: 18. ožujka 2016.
Rok za predaju rada: 1. srpnja 2016.

Mentor:

Prof. dr. sc. Damir Seršić

Djelovođa:

Izv. prof. dr. sc. Dražen Jurišić

Predsjednik odbora za diplomski rad profila:

Prof. dr. sc. Mladen Vučić

Zahvaljujem svima koji su pomogli u izradi ovog rada,

posebno mentoru prof.dr.sc Damiru Seršiću

Sadržaj

Popis slika

1. Uvod

U današnje vrijeme kada su gradovi zatrpani automobilima, sve popularniji način prijevoza među ljudima je električni bicikl. Takav bicikl nudi ljudima mogućnost jednostavnijeg putovanja na manjim i većim udaljenostima, pritom također nudeći mogućnost korištenja bicikla u rekreacijske svrhe. Pošto su takva vozila napajana baterijskim izvorima energije, javlja se potreba za sučeljem na kojem će korisnik moći saznati stanje baterije kao i svoju trenutnu brzinu.

Upravo iz tog razloga izrađeno je interaktivno grafičko sučelje za električno vozilo sastavljeno od dva OLED ekrana, koja neprestano omogućuju korisniku uvid u stanje baterije kao i brzine vozila, i kapacitivne tipkovnice kojom je moguće upravljati samim ekranima i vozilom. Kako se ciljano električno vozilo promjenom opterećenja može iz električnog pretvoriti u sobni bicikl, korisniku je također dostupno trenutno stanje tog opterećenja, kao i mogućnost njegove promjene korištenjem tipkovnice.

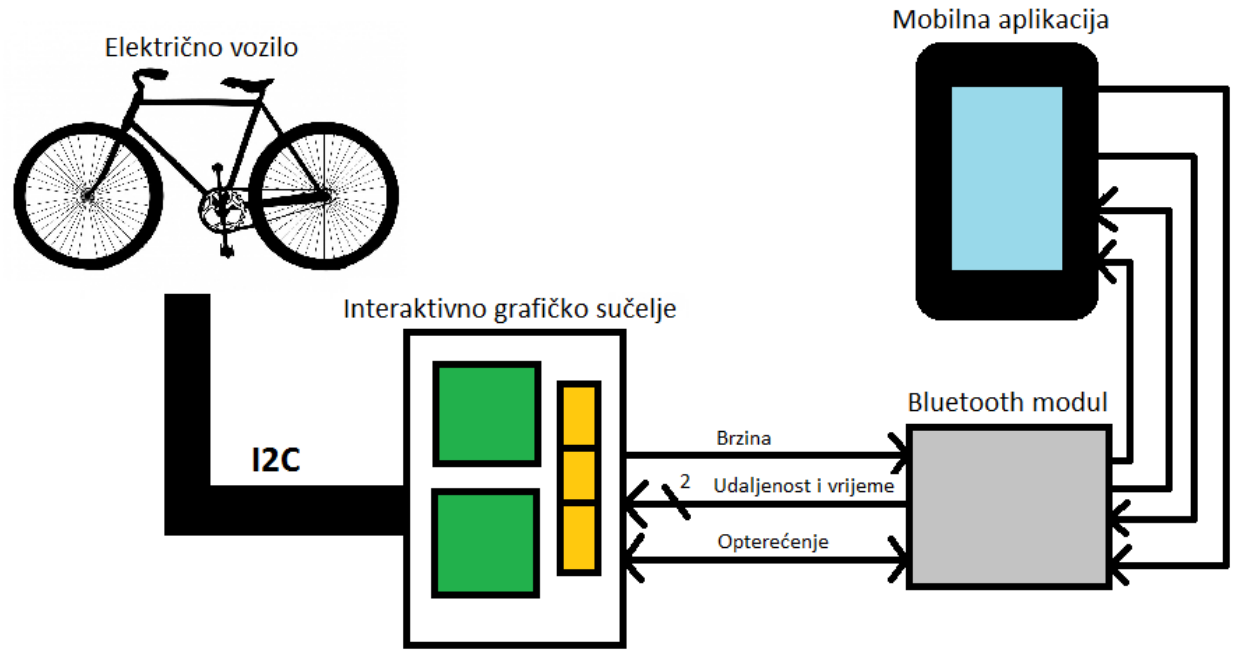
Kako bi korištenje električnog vozila bilo što ugodnije, napravljena je i mobilna aplikacija koja je povezana s električnim vozilom i interaktivnim grafičkim sučeljem preko Bluetooth veze. Aplikacija nudi mogućnost prikupljanja i pohrane podataka prikupljenih za vrijeme pojedinačnih vožnji te njihov pregled, kako bi korisnik imao bolji uvid u opseg svog korištenja električnog vozila.

U prvom dijelu ovog rada opisano je korišteno sklopovlje u izradi interaktivnog grafičkog sučelja, kao i pripadna programska podrška i način rada sučelja. Nakon toga navedeni su alati koji su bili potrebni za izradu mobilne aplikacije te na samom kraju opisana je izvedba mobilne aplikacije i način na koji se koristi.

2. Princip rada cijelog sustava

Interaktivno grafičko sučelje električnog vozila je sustav koji, kao što mu i samo ime kaže, omogućuje korisniku interakciju sa električnim vozilom koristeći različite grafičke i vizualne indikatore, te tako daje korisniku na uvid stanja različitih postavki vozila među kojima su stanje baterije i trenutna brzina.

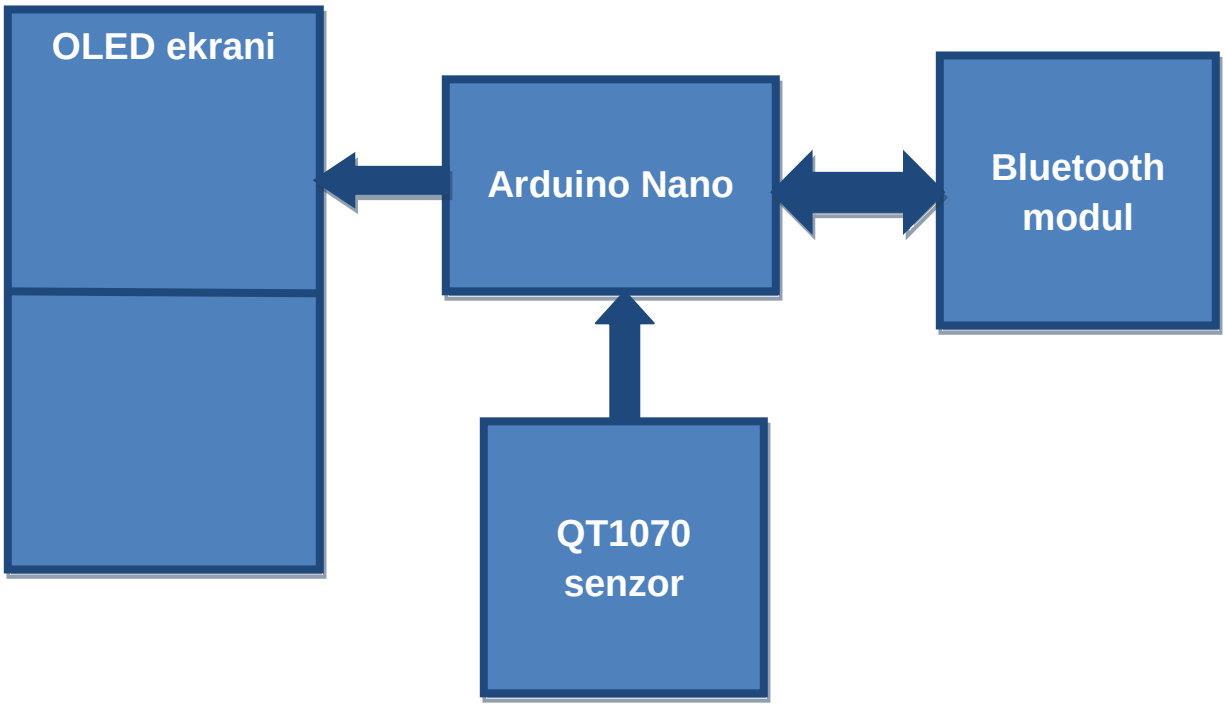
Sučelje se sastoji od ekrana i tipkovnice, a sa električnim vozilom komunicira pomoću I^2C veze. Osim na bicikl, sučelje je preko Bluetooth modula spojeno na mobilnu aplikaciju preko koje je također moguće upravljati električnim vozilom. Mobilna aplikacija od vozila, preko sučelja i Bluetooth modula, prima podatke o brzini i trenutnom opterećenju bicikla, a prema sučelju, a samim time i prema biciklu, šalje podatke koje prikupi tijekom svog rada (ukupna prijeđena udaljenost te vrijeme provedeno u vožnji). Mobilnom aplikacijom je također moguće mijenjati opterećenje vozila, pa je to još jedan podatak koji aplikacija mora vratiti vozilu.



Slika 2.1 Pregled sustava električnog vozila, grafičkog sučelja i mobilne aplikacije

3. Sklopovlje interaktivnog grafičkog sučelja

Interaktivno grafičko sučelje sastoji se od dva OLED ekrana na kojima je moguće prikazati različite stavke električnog bicikla kao što su: stanje baterije, brzine i opterećenje vozila, te kapacitivne tipkovnice koja omogućuje korisniku promjenu načina rada i upravljanje onim što se prikazuje na ekranu. Ekrani i tipkovnica su spojeni na Arduino čiji mikrokontroler upravlja cijelim sustavom. Na slici 3.1 prikazana je blok shema pločice sa njezinim glavnim dijelovima. Kao što je na slici vidljivo, središnji dio sustava je Arduino Nano pločica, koja s jedne strane upravlja prikazom na ekranima i obradom podataka pristiglih sa QT1070 senzora [10] kapacitivne tipkovnice, a s druge razmjenjuje podatke s mobilnom aplikacijom preko Bluetooth modula. U skolop u ovog rada dizajnirana je i izrađena tiskana pločica grafičkog sučelja. U privitku se nalazi električna shema i nacrti tiskanih veza izrađene pločice, a u nastavku poglavlja opisano je korišteno sklopovlje i njegov način rada.



Slika 3.1 Blok shema pločice grafičkog sučelja

3.1. Arduino

Kao što je rečeno, upravljački dio interaktivnog grafičkog sučelja izveden je pomoću Arduino sustava. Arduino je *open-source* platforma sastavljena od hardvera i softvera kojima se jednostavno služiti. Arduino sustavi u sebi imaju ugrađen mikrokontroler za čije se programiranje koristi razvojna okolina Arduino IDE [1] koja podržava C/C++ programski jezik uz poštivanje posebnih pravila u organizaciji koda. Pa tako svaki Arduino program mora sadržavati najmanje dvije funkcije:*setup()* i *loop()*. Funkcija *setup()* se izvrši samo jednom, prilikom pokretanja programa te se koristi za inicijalizaciju različitih dijelova sustava, dok se funkcija *loop()*izvršava neprekidno i ciklički te tako kontrolira izvođenje programa i promjene koje na taj način nastaju.

U ovom radu se koristi Arduino Nano pločica temeljena na Atmel ATmega328 mikrokontroleru. Pločica se može napajati na dva načina. Jedan je preko Mini-B USB napajanja, a drugi dovođenjem vanjskog izvora napajanja. Kada se pločica napaja preko vanjskog izvora, maksimalan dozvoljen raspon ulaznog napona je 6-20V. ATmega328 ima 32kB flash memorije, 2kB SRAM i 1kB EEPROM.

Arduino Nano ima ukupno 14 digitalnih pinova od kojih se svaki može koristiti kao ulazni ili izlazni. To se postiže korištenjem posebnih funkcija, pa se tako željeni način rada, ulazni ili izlazni, postavlja preko `pinMode(pin, mode)` funckije koja prima dva parametra, jedan predstavlja pin kojem se postavlja način rada, a drugi parametar definira sam način rada. Nakon što se odabere način rada pina, s njim se može upravljati pomoću funkcija`digitalWrite(pin, state)` i `digitalRead(pin)`. Funkcija `digitalWrite(pin, state)` također prima dva parametra od kojih je jedan pin s kojim se upravlja, a drugi je stanje u koje se pin želi postaviti (visoko ili nisko). Funkcija`digitalRead(pin)` prima samo jedan parametar i to pin kojem se želi pročitati stanje. Pa tako, naprimjer, ako želimo čitati stanje jednog digitalnog pina i nakon toga u to isto stanje postaviti neki drugi pin, to bi mogli učiniti na sljedeći način:

```
int stanjePin=6; //definiranje ulaznog pina
int izlazniPin=7; //definiranje izlaznog pina
int temp=0;
```

```
void setup() {
     pinMode(izlazniPin, OUTPUT);
     //unutar setup funkcije inicijaliziramo digitalni pin 7
     //kao izlazni i pin 6 kao ulazni
     pinMode(stanjePin, INPUT);
}
void loop () {
     //unutar loop funkcije dolazi odsječak koda koji se
     //neprestano izvodi
     temp=digitalRead(stanjePin); //čitanje stanja ulaznog pina
     digitalWrite(izlazniPin, temp);
     //postavljanje izlaznog pina
}
```

Neki od digitalnih pinova imaju posebnu namjenu pa se tako pin 0 (RX) i pin 1 (TX) koriste za primanje i slanje serijskih podataka, pinovi 2 i 3 se mogu konfigurirati tako da se ponašaju kao vanjski prekidi. Pinovi 3, 5, 6, 9, 10 i 11 se mogu koristiti za generiranje 8 bitnog PWM signala pomoću funkcije `analogWrite()`. Na pin 13 je spojen LED, a pinovi 10, 11, 12,13 hardverski podržavaju SPI komunikaciju. Radni napon digitalnih pinova je 5V, a maksimalna struja koju podržavaju je 40mA.

Nano pločica također sadrži 8 analognih pinova koji mogu imati 1024 različite vrijednosti. Po početnim postavkama na njima se nalazi napon od 5V koji je moguće softverski promijeniti funkcijom`analogReference(`. Analogni pinovi 4 i 5 podržavaju I^2C komunikaciju korištenjem *Wire library* [2], a svi analogni pinovi, s iznimkom pinova 6 i 7, mogu biti korišteni kao digitalni.

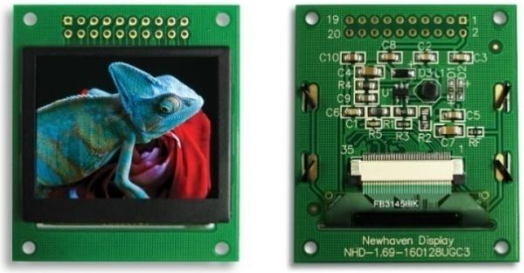
3.2. OLED ekrani

OLED (*Organic light-emitting diode*)je *solid-state* uređaj sastavljen od tankog sloja organskih molekula koje u doticaju sa električnom strujom proizvode svjetlost. Jednostavan OLED je najčešće sastavljen od šest slojeva. S gornje i donje strane se nalaze zaštitni slojevi od stakla ili plastike, a između njih negativna elektroda (katoda) i pozitivna elektroda (anoda). Između katode i anode su smještene organske

molekule koje sačinjavaju preostala dva sloja; emitirajući sloj koji se nalazi uz katodu i zaslužan je za proizvodnju svjetla te provodni sloj koji se nalazi uz anodu. Da bi OLED uređaj proizveo svjetlo, potrebno je primijeniti napon između anode i katode. Nakon što se primijeni napon i struja poteče, elektroni se počnu skupljati oko katode, dok ih u okolini anode ima sve manje, to jest pretvaraju se u šupljine. Kao posljedica toga, emitirajući sloj postaje jako negativno nabijen, a provodni sloj suprotno odnosno pozitivno. Pošto su pozitivne šupljine pokretljivije od elektrona, one će lakše preskočiti granicu između emitirajućeg i provodnog sloja i tako doći u doticaj sa elektronom na drugoj strani. Na mjestu gdje se šupljina i elektron spoje, odnosno dođu u doticaj, stvori se foton, a samim time i kratak svjetlosni bljesak. Taj proces se nastavlja sve dok postoji tok struje te se ponavlja puno puta u sekundi, što dovodi do kontinuirane proizvodnje svjetla. Ako se želi dobiti obojano svjetlo, potrebno je dodati obojani filter ispod gornjeg zaštitnog sloja ili iznad donjeg zaštitnog sloja [3].

OLED uređaji za svoj rad ne zahtijevaju pozadinsko ovjetljenje što ih čini superiornijima LCD uređajima. OLED je također fleksibilniji, tanji i lakši te ima puno manje vrijeme odziva. OLED također prikazuje čišće boje, čija je kvaliteta neovisna o kutu pod kojim se ekran promatra. Iako generalno troše puno manje energije od običnih LCD uređaja (do 60%), u slučaju prikazivanja slika s bijelom pozadinom potrošnja energije se drastično poveća. No zasigurno najveći nedostatak OLED uređaja je njihov,relativno kratak, životni vijek koji je posljedica degradacije organskih molekula od kojih su napravljeni. Drugi veliki problem je osjetljivost tih istih molekula na vodu, što može biti problematično u slučaju korištenja OLED ekrana u prijenosnim uređajima.

U ovom radu korištena su dva Newhaven OLED Display [4] sa 160 x 128 piksela. Uređaji imaju mogućnost serijske i paralelne komunikacije s kontrolerskom jedinicom te ugrađenim SEPS525 [5] upravljačkim programom.

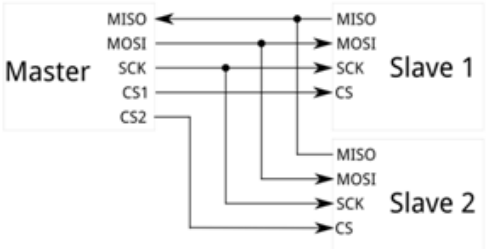


Slika 3.2 Prednja i stražnja strana OLED ekrana [4]

Uređaji se koriste u serijskom (SPI) načinu rada. SPI (*Serial Peripheral Interface)*[6] je serijsko sinkrono sučelje za komunikaciju. Koristi se upravljač – izvršitelj (*master-slave)* arhitektura sa jednim upravljačkim (*master)* uređajem dok je izvršiteljskih *(slave)* jedinica moguće imati više. U ovom slučaju Arduino je upravljač, a dva OLED ekrana su izvršitelji.

SPI sabirnica definira četiri logička signala:

1. SCK (*Serial clock)* – Signal takta kojeg generira i postavlja Arduino
2. MOSI (*Master output, slave input)* – Podaci koje Arduino šalje ekranima
3. MISO (*Master input, slave output)* – Podaci koje ekrani šalju Arduinu
4. CS (*Chip select)* – Signal pomoću kojeg Arduino selektira i kontrolira željeni OLED. Ovaj signal je aktivan u niskoj razini.



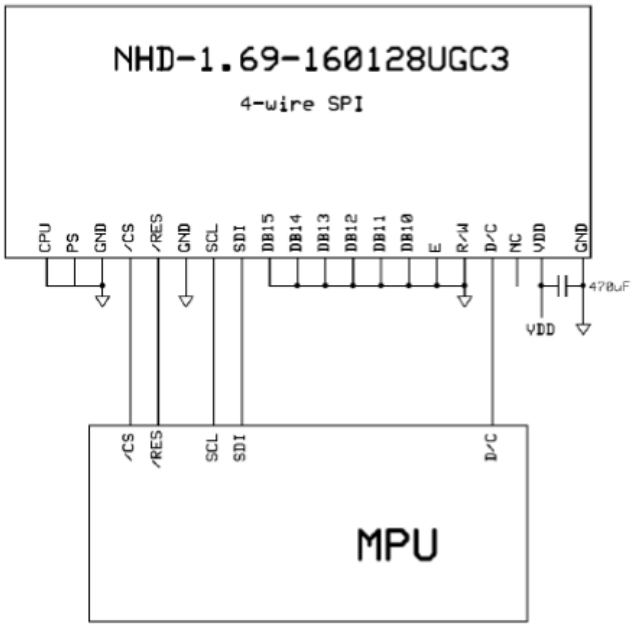
Slika 3.3 SPI sučelje s dva izvršitelja [6]

Slika 3.2 prikazuje konfiguraciju sa dvije *slave* jedinice koja je korištena u ovom radu. CS jednog ekrana spojen je na digitalni pin 10, a CS drugog ekrana na digitalni pin 9.

SPI prijenos započinje kada Arduino postavi signal takta (SCK) na frekvenciju koju podržava ciljani uređaj, kako bi bili usklađeni te kako bi na taj način bilo moguće započeti komunikaciju i prijenos podataka. Nakon što je signal takta postavljen i usklađen, Arduino selektira ekran s kojim želi razmjenjivati informacije postavljanjem

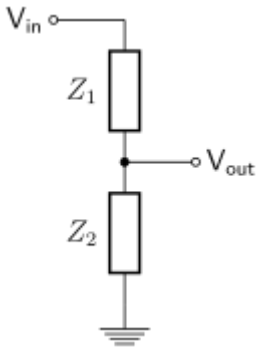
signala CS u nisku razinu. Kao što je vidljivo na slici, podatkovne linije i signal takta su zajednički i ekrani ih dijele, dok su signali kojima se obavlja selektiranje izvršitelja odvojeni. Na taj način je omogućeno istodobno izvođenje različitih radnji na ekranima.

Signali kojima su OLED ekrani i Arduino povezani i koji se koriste u njihovoj komunikaciji su $\overline{CS}$ kojim se odabire ekran, $\overline{RES}$ odnosno reset signal, SCL kojim se postavlja signal takta, SDI odnosno MOSI te D/C koji određuje je li primljena informacija podatak ili naredba.



Slika 3.4 Prikaz veze OLED ekrana s Arduinom [8]

Ekrani imaju napajanje od 3.3V, što je ujedno i maksimalan napon koji se smije pojaviti na njihovim pinovima. Iz tog razloga je potrebno smanjiti napon signala koji dolaze iz Arduina, pošto je radni napon pinova na Arduinu 5 V. To se postiže naponskim djelilom. Naponsko djelilo je pasivan linearni električni krug koji na svom izlazu daje napon koji predstavlja određeni dio ulaznog napona. Podjela napona je rezultat preraspodjele napona na otpornicima spojenima između ulaza i izlaza kao što je prikazano na slici 3.4.



Slika 3.5 Naponsko djelilo [7]

U ovom konkretnom slučaju ulazni napon (V_{in}) je napon koji dolazi iz Arduina i njegova vrijednost je 5 V. Izlazni napon je potreban da bude u granici od 2.6 do 3.3 V koja je prihvatljiva za ekrane, što znači da je potrebno spustiti ulazni napon na dvije trećine početne vrijednosti. Pod pretpostavkom da je struja na izlazu 0, formula za izlazni napon može se izvesti na sljedeći način:

$$V_{in} = I * (Z_1 + Z_2) \tag{3.1}$$

$$V_{out} = I * Z_2 \tag{3.2}$$

Uvrštavanjem formule 3.1 u 3.2 se dobije konačna formula za naponsko djelilo:

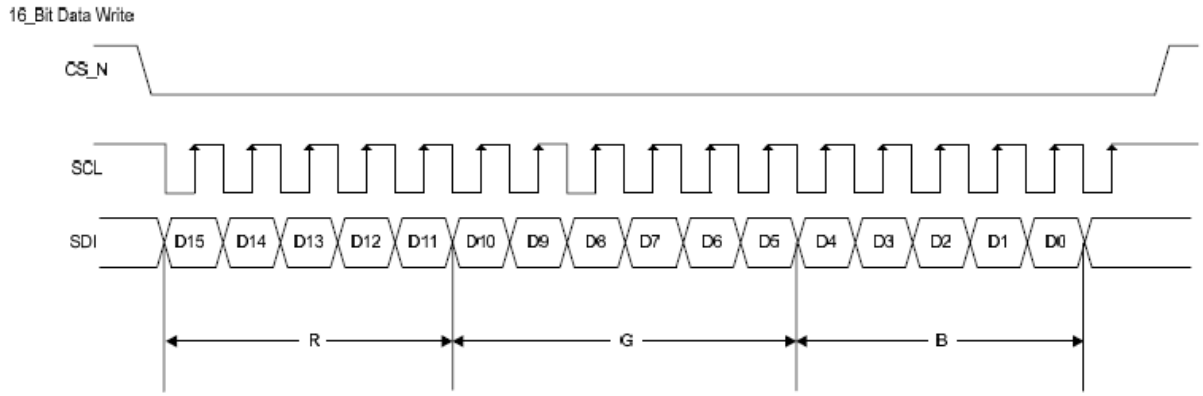
$$V_{out} = V_{in} * \frac{Z_2}{Z_1 + Z_2} \tag{3.3}$$

Uzimajući u obzir sve uvjete, Z2 je odabrano da bude 2kΩ a Z1 1kΩ, na taj način ograničen je maksimalan napon koji se može pojaviti na pinovima OLED-a na 3.3 V i tako je spriječeno pojavljivanje nedopuštene veličine napona.

OLED ekrani imaju ugrađen SEPS525 upravljački program koji sadrži tri različita sučelja za komunikaciju sa centralnom procesorskom jedinicom među kojima je i SPI koji se koristi u ovom radu. Sučelje se odabere postavljanjem pina PS u odgovarajuće stanje i pisanjem u MEMORY_WRITE_MODE registar koji se nalazi na adresi 16h. Serijsko sinkrono sučelje je tako moguće odabrati postavljanjem pina PS u nisku razinu. Na slici 3.5 je vidljivo kako se obavlja upis podataka. Nakon što upravljač postavljanjem CS pina u nisku razinu selektira željenog izvršitelja, na rastući brid SCL signala SDI pin prihvati podatak koji mu je poslan, nakon čega taj isti podatak pretvara u 16 ili 18-bitnu informaciju, te se na šesnaesti ili osamnaesti rastući brid

takta ta informacija prenese u memoriju. Kada je prijenos podataka završen, upravljač prestane s prebacivanjem signala takta i deselektira izvršitelja postavljanjem CS pina u visoku razinu. Ovisno o vrijednosti koja se nalazi na D/C pinu, informacija na SDI pinu se detektira kao podatak ili naredba [8].

SEPS525 upravljački program sadrži tri vrste registara: 8 bitni IR (*Index Register)*, WDR (*Write Data Register)* i RDR (*Read Data Register)*. IR sadrži informaciju o indeksu za kontrolne register i DDRAM (*Display Data RAM)*, a WDR i RDR služe za privremeno spremanje podataka koji se tek trebaju upisati u kontrole registre ili DDRAM i čitanje i privremeno spremanje tek pročitanih podataka iz DDRAM-a. Što znači da podaci, koji se spremaju u DDRAM, se prvo privremeno pohrane u WDR te se potom spremaju u DDRAM. Kada se podaci čitaju iz DDRAM, pohranjuju se u RDR s tim da je prvi pročitani podatak nevažeći.



Slika 3.6 Upis podataka u memoriju [8]

DDRAM je napravljen od 18 bitnog polja koje se sastoji od 160 stupaca i 120 redaka. Kada se podaci upisuju u DDRAM, oni se uzastopno upisuju u adresni prozor koji je definiran s horizontalnim adresnim registrom (MX1[7:0] i MX2[7:0]) i vertikalnim adresnim registrom (MY1[7:0] i MY2[7:0]). Podaci se upisuju u smjeru koji je definiran s HC,VC (uzastopno povećanje ili smanjenje u horizontalnom ili vertikalnom smjeru) i HV bitom koji definira smjer upisivanja (horizontalno ili vertikalno). Adresni prozor mora biti specificiran unutar DDRAM adresnog prostora i to na sljedeći način:

- horizontalan smjer – $00h \leq MX1[7:0] < MX2[7:0] \leq 9Fh$
- vertikalan smjer – $00h \leq MY1[7:0] < MY2[7:0] \leq 7Fh$

Inicijalizacija se obavlja nakon reseta, odnosno nakon što se uređaj upali. Prilikom pokretanja potrebno je dodati vremensko kašnjenje u intervalu kada dolazi do promjene stanja između visokog i niskog napona, kako bi OLED imao dovoljno vremena da se na ispravan način napuni i isprazni prije izvođenja bilo kakve operacije.

3.3. Kapacitivna tipkovnica

Interakcija korisnika s grafičkim sučeljem izvedena je pomoću kapacitivne tipkovnice. Tipkovnica se sastoji od 6 tipki koje kontrolira Atmelov AT42QT1070 [10] kapacitivni senzor.

Kapacitivni senzor QT1070 sam obavlja svu obradu signala potrebnu da se dobije stabilan izlaz,bez obzira na promjenu uvjeta okoline u kojoj se nalazi. Prikupljanje podataka se obavlja pomoću *dual pulse* metode, što eliminira potrebu korištenja vanjskih kondenzatora, pa je senziranje moguće korištenjem samo jednog pina.

Senzor se koristi u *Comms* načinu rada*. Comms* način rada omogućuje korištenje zaštitnog kanala (*Guard channel),* podešavanje praga dodira, korištenje tehnike potiskivanja susjednih tipki – AKS (*Adjacent Key Suppression)*, DI (*Detect Integrator)* filtra, LP (*Low Power)* Mode i podešavanje maksimalnog vremena u kojem tipka može biti dodirnuta (*max time on)* za svaku tipku [10].

Zaštitni kanal se može koristiti kako bi se spriječila pojava lažne detekcije signala. U*Comms* načinu rada bilo koja tipka može biti programirana da radi kao zaštitni kanal. Jedan od zahtjeva pri dizajnu tipke koja se koristi kao zaštitni kanal je potreba da bude fizički veća, a samim time i osjetljivija od ostalih tipki, kako bi u slučaju detektiranog dodira njezin signal nadjačao izlazni signal bilo koje druge tipke.Zaštitni kanal se definira postavljanjem najniža 4 bita registra na adresi 53. Prihvatljive vrijednosti su u rasponu od 0 do 6, a postavljanje bitova na veću vrijednost onesposobljava funkciju zaštitnog kanala.

Da bi senzor uspješno detektirao dodir potrebno je da signal prijeđe određeni prag dodira i ostane u istom stanju unaprijed definirani broj ciklusa. Ti ciklusi se mogu podesiti za svaku tipku posebno upisom 8 bitne vrijednosti na neku od adresa u

rasponu od 32 do 38 (tipke 0 do 6). U ovom radu broj ciklusa je postavljen na 10 za svaku tipku.

Uređaj također ima ugrađen DI filter koji je zadužen za filtriranje šuma koji se može pojaviti u signalima. DI zahtjeva da postoji određeni broj uzastopnih uzoraka, koji su potvrđeni kao detektirani dodiri prije nego se tipka može proglasiti dodirnutom. Minimalan broj takvih uzoraka je 2, a upis vrijednosti koja definira broj uzoraka se vrši na adresama od 46 do 52, također za svaku tipku posebno.

Senzor sadrži 255 različitih mogućih vremena akvizicije koja se kontroliraju pomoću LP bajta. Na taj način se može upravljati intervalima između uzimanja pojedinih uzoraka. Duži vremenski intervali troše manje energije, ali i povećavaju vremenski odziv senzora. Rad QT1070 senzora se temelji na fiksnom ciklusu koji traje 8ms, a LP bajt definira koliko takvih ciklusa treba postojati u jednom periodu mjerenja. LP se postavlja upisom 8 bitne vrijednosti na adresi 54, a po početnim postavkama se tu nalazi vrijednost 2 što znači da je interval između uzimanja uzoraka 16ms (2 uzorka * 8ms).

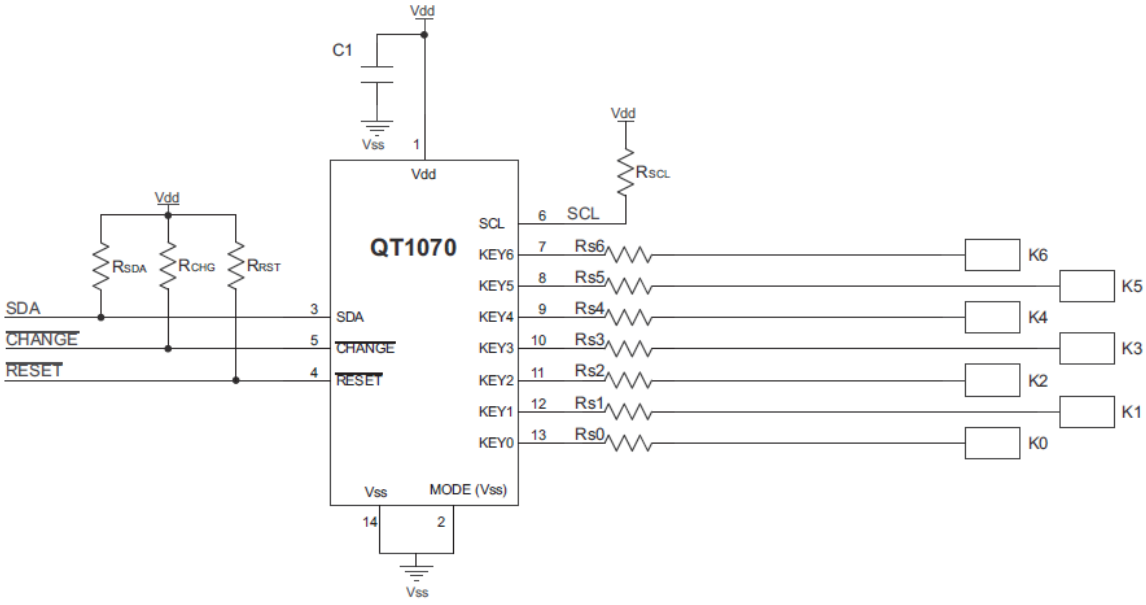
QT1070 također ima mogućnost definiranja koliko dugo tipka može biti detektirana kao pritisnuta prije nego se rekalibrira. Početna vrijednost ove veličine je 180 odnosno 28.8 sekundi, a može se promijeniti upisom 8 bitnog broja, s korakom promjene od 160ms na adresu 55. U ovom radu koristi se vrijednost 4, to jest ako je tipka dodirnuta duže od 640 milisekundi ona se rekalibrira.

Programski kod kojim se obavlja inicijalizacija se nalazi u privitku (AtTouch.cpp) [11].

Senzor komunicira s Arduinom putem I^2C (*Inter-integrated Circuit)*[11] sabirnice. I^2C sabirnica je *multi master* i *multi slave* serijska sabirnica koja za razmjenu podataka koristi samo dva dvosmjerna signala, SCL (*Serial Clock)* i SDA (*Serial Data)*. Čvorovi koji se nalaze u sabirnici mogu imati jednu od dvije uloge, upravljač ili izvršitelj. Upravljački čvor je onaj koji generira signal takta i inicira komunikaciju sa izvršiteljskom jedinicom.

SCL i SDA linije su *open drain* što znači da mogu postavljati odgovarajuće signale u nisku razinu ali ne i visoku. Komunikacija koja se odvija putem I^2C sabirnice se mora držati odgovarajućeg protokola. Poruke koje se šalju ovakvim putem su podijeljene u

dva okvira, adresni gdje upravljač adresira izvršitelja kojem šalje poruku i podatkovni okvir koji sadrži 8 bitni podatak koji se šalje. Svaka razmjena podataka, bilo slanje ili primanje, mora započeti sa START uvjetom i završiti sa STOP uvjetom. Prilikom slanja podataka uređaju,nakon START uvjeta,pošalje se adresa senzora sa jednim dodatnim bitom koji označava želju za pisanjem u uređaj (Write bit). Ako je senzor spreman, šalje bit potvrde što upravljaču daje signal da može poslati adresu memorijske lokacije unutar senzora kojoj želi pristupiti, te odmah potom podatak koji želi upisati na poslanu adresu. Postupak kod primanja podataka sa senzora je sličan onom kod slanja. Također je najprije potrebno poslati START uvjet, nakon čega slijedi adresa senzora sa bitom za pisanje i potom toga adrese memorijske lokacije s unutar uređaja. Nakon poslane adrese memorijske lokacije, upravljač mora poslati STOP i START uvjet, te odmah potom adresu senzora, ali ovaj put sa bitom koji signalizira želju za čitanjem podatka što omogućuje senzoru da šalje bajtove podataka prema upravljaču [10].



Slika 3.7 Spajanje senzora QT1070 u Comms načinu rada [10]

Na slici 3.6 je prikazano kako se uređaj spaja u *Comms* načinu rada. SDA i SCL signali se spajaju na odgovarajuće analogne pinove na Arduinu (pin 4 i pin 5).

Promjena stanja tipke detektira se pomoću signala koji generira pin $\overline{CHANGE}$ koji je spojen na digitalni pin 3 Arduina i ponaša se kao vanjski prekid. $\overline{CHANGE}$ je u

neaktivnom stanju preko *pull-up* otpornika spojen na napon napajanja senzora, a prilikom promjena stanja neke od tipki on prelazi u nisku razinu i postaje aktivan. Da bi $\overline{CHANGE}$ mogao generirati prekid, potrebno je prilikom inicijalizacije senzora postaviti odgovarajući digitalan pin Arduina (u ovom slučaju pin 3) kao ulazni i pridodati mu prekidnu rutinu:

```
//_changePin je digitalni pin koji želimo da generira
prekid//_interruptVal predstavlja digitalni pin pretvoren u
//odgovarajuću vrijednost internih prekidnih vrijednosti koje
//arduina; _interruptVal=_changePin-2 (vrijednost je 1 //kada
se //koristi digitalni pin 3)
     pinMode(_changePin, INPUT); //definiranje pina kao ulaznog
     attachInterrupt(_interruptVal,bttnPressISR,FALLING);
     //postavljanje prekida na padajući brid signala i
     //dodavanje prekidne rutine bttnPressISR
```

Jedini zadatak prekidne rutine `bttnPressISR()` je promjena varijable `keyHit` iz `false` u `true`. Pin $\overline{CHANGE}$ se vraća u neaktivno stanje nakon što se pročita stanje registra u kojem se nalazi podatak o detekciji dodira.

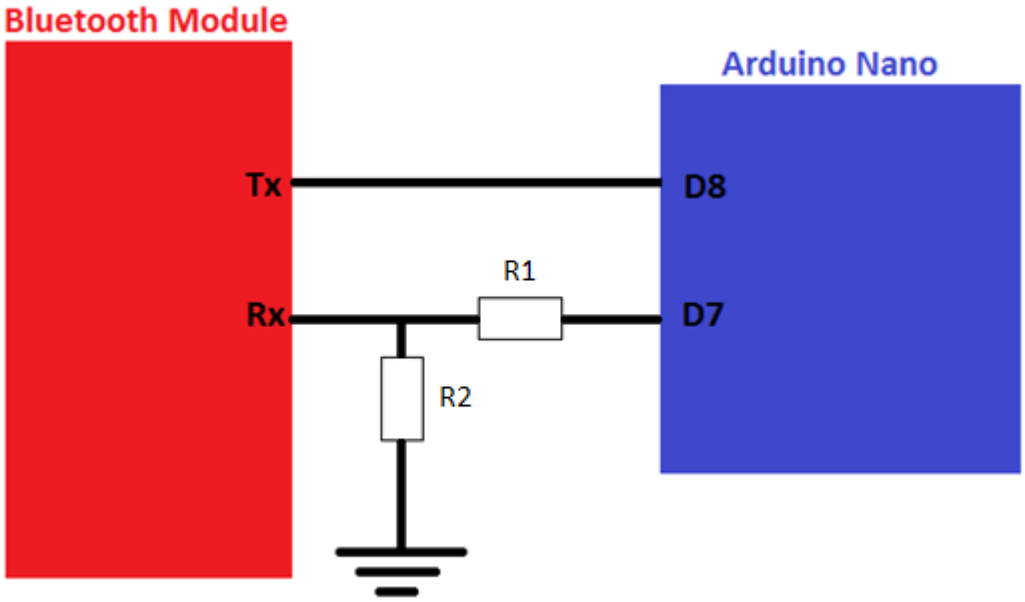
3.4. Bluetooth

Bluetooth je tehnologija koja se koristi za bežičnu razmjenu podataka između dvaju ili više uređaja na malim udaljenostima (najčešće do 10 metara). Izumila ga je 1994. godine švedska tvrtka Ericsson. Bluetooth za prijenos podataka koristi radio valove u ISM (*Industrial – Scientific - Medical)* frekvencijskom pojasu. Pa su tako frekvencije na kojima Bluetooth radi između 2402 i 2480 MHz ili 2400 i 2483.5 MHz, što uključuje zaštitne pojaseve od 2 i 3.5 MHz. Pri prijenosu podataka, Bluetooth ih dijeli u pakete i odašilje svaki paket preko jednog od 79 kanala širine 1MHz, koristeći metodu FHSS (*Frequency Hopping Spread Spectrum*). U samim počecima razvoja Bluetooth tehnologije, jedina dostupna modulacija je bila GFSK (*Gaussian frequency-shift key)*, što je značilo da je maksimalna brzina prijenosa bila 1Mbit/s. Dolaskom Bluetooth 2.0+EDR tehnologije, postalo je moguće koristiti π/4 – DQPSK (*Differential*

Quadrature Phase Shift Keying) i 8DPSK (*Differential Phase Shift Keying)* kojima je moguće postići brzine od 2 i 3Mbit/s [20].

Bluetooth u komunikaciji koristi upravljač – izvršitelj (*master-slave)* arhitekturu. Upravljač može u jednom trenutku biti povezan s maksimalno 7 izvršiteljskih jedinica. Po konvenciji, u bilo kojem trenutku, uređaji mogu zamijeniti mjesto, to jest dotadašnji upravljačmože postati izvršitelj i obrnuto.

Pošto je maksimalan dopušten napon na pinovima Bluetooth modula 3.3 V, a Arduino na svojim pinovima ima 5 V, potrebno je naponskim djelilom spustiti naponsku razinu signala koji ide iz Tx pina Arduina prema Rx pinu Bluetooth modula. Za to služe otpornici R1 od 1 kΩ i R2 od 2 kΩ čije vrijednosti se dobije pomoću formule 3.3.



Slika 3.8 Spajanje Bluetooth modula s Arduinom

Razmjena podataka između aplikacije i električnog vozila radi tako da aplikacija putem Bluetootha pošalje podatak Arduinu, koji ga potom proslijedi električnom vozilu preko I^2C veze. Komunikacija mora biti dvosmjerna, što znači da kada aplikacija zatraži podatak, električno vozilo, to jest Arduino ga mora poslati preko Bluetooth modula. Kako su mobilni uređaji na kojima se aplikacija izvodi u Bluetooth komunikaciji uvijek upravljačke jedinice, Bluetooth modul mora bitiizvršitelj.

Prije korištenja, Bluetooth modulu je potrebno promijeniti neke od tvorničkih postavki kako bi bio vidljiv mobilnom uređaju te kako bi bilo moguće uparivanje, a samim tim i komunikacija.

Prvo je potrebno postaviti modul u *slave* način rada pomoćnu naredbu `\r\n+STWMOD=0\r\n`. Nakon toga treba omogućiti slanje upita modulu naredbom `\r\n+INQ=1\r\n` i omogućiti spajanje na modul uparenim uređajima `\r\n+STOAUT=1\r\n`.

Poželjno je promijeniti tvornički zadano ime i lozinku, kao i saznati adresu modula, kako bi mobilna aplikacija znala na koji uređaj se treba spojiti. Naredbe koje to omogućuju su `\r\n+STNA=BIKEBT` za promjena imena, `\r\n +STPIN=1234\r\n` za promjena lozinke i `\r\n+RTADDR\r\n` za saznati adresu.

4. Interaktivno grafičko sučelje

Interaktivno grafičko sučelje sastoji se od dva OLED ekrana na kojima se prikazuju podaci primljeni od vozila, te od kapacitivne tipkovnice koja korisniku omogućuje pomicanje po različitim stanjima grafičkog sučelja i podešavanje određenih elementa električnog vozila.

U svakom trenutku na ekranima se brojčanim i slikovnim indikatorima prikazuje na jednom ekranu trenutna brzina vozila, a na drugom trenutno stanje baterije kao i opterećenje električnog vozila (Slika 4.1).

Kada su na ekranima prikazani stanje brzine, baterije i opterećenje, klizanjem po kapacitivnoj tipkovnici prema gore ili dolje, povećava se ili smanjuje opterećenje vozila ovisno o smjeru klizanja. Na taj način je moguće mijenjati opterećenje od nula (električni bicikl) do deset (sobni bicikl).

Dvostrukim dodirom tipkovnice ulazi se u izbornik (Slika 4.2), a vraćanje na prikaz baterije, brzine i opterećenja, moguće je dobiti ponovnim dvostrukim dodirom tipkovnice.

Kada je na ekranima prikazan izbornik, klizanjem po tipkovnici moguće je ući u neku od tri stavke izbornika. Prva stavka omogućuje prikaz stanja dvanaest baterijskih ćelija. (Slika 4.3) Drugom stavkom moguće je promijeniti neki od parametara vozila, a treća stavka prikazuje ukupnu kilometražu i vrijeme vožnje prikupljeno pomoćnu mobilne aplikacije (Slika 4.4). Iz svake od stavki dvostrukim dodirom tipkovnice moguće se vratiti do glavnog izbornika.

U slučaju da tipkovnica ne detektira nikakvu vrstu dodira više od deset sekundi, prikaz na ekranima se vraća u početno stanje, to jest na jednom će se prikazati indikator stanja brzine, a na drugom indikator stanja baterije i opterećenja.

4.1. Upravljanje ekranima

Ekranima se upravlja koristeći SEPS525_OLED biblioteku [9] koja je za potrebe ovog rada modificirana kako bi podržavala kontrolu oba OLED ekrana. Modifikacija se tako prvenstveno odnosi na dodavanje globalne varijable `slave` koja je zadužena za selektiranje željenog ekrana. Ovisno o njenoj vrijednosti, koja može biti 1 ili 2, obavlja

se prijenos podataka na ekran čiji je CS spojen na digitalni pin 10 ili digitalni pin 9, kako je objašnjeno u prethodnom poglavlju. Stanje varijable potrebno je postaviti u željenu vrijednost prije poziva funkcija koja su zadužene za osvježavanje ekrana. U nastavku je isječak programskog koda u kojem se može vidjeti opisani postupak:

```
slave = 1; //selektiranje ekrana, u ovom slučaju to je onaj
//spojen na digitalni pin 10
if (Battery) {
     Battery = false;
draw_battery(); //iscrtavanje baterije
}
//poziv funkcija iz SEPS525_OLED.cpp
disp.fillRect(7, 7, baterija_level, 52, BLACK);
baterija_level = map(bat2, 0, 100, 0, 130);
disp.fillRect(7, 7, baterija_level, 52, GREEN);
//isječak koda iz SEPS525_OLED.cpp
void SEPS525_OLED::fillRect(int16_t x, int16_t y, int16_t w,
int16_t h, uint16_t color){
     seps525_set_region(x, y, w, h); //odabir dijela ekrana
     koji //će se koristiti
     if(slave==1) {
          seps525_datastart1(); //postavi pin 10 u 0
          int n;
          //obojaj specificirani dio ekrana
          for(n = 0; n < h*w; n++) seps525_data(color);
          seps525_dataend1(); //vrati pin 10 u 1
     }
     ... //inače započni prijenos na ekran spojen na pin 9
}
```

4.2. Obrada podataka pristiglih s tipkovnice

U poglavlju o kapacitivnoj tipkovnici objašnjeno je na koji način senzor QT1070, preko $\overline{CHANGE}$ pina, detektira promjenu stanja neke od tipki spojenih na njega. U sklopu ovog rada razvijen je softver koji može detektirati nekoliko različitih vrsta dodira kapacitivne tipkovnice: dvostruki dodir te klizanje u oba smjera.

Prvi korak u detekciji dodira je poziv funkcije `hit()` iz [11] koja vraća stanje varijable `keyHit`. Varijablu `keyHit` u `true` postavi prekidna rutina pozvana nakon što $\overline{CHANGE}$ pin generira prekid, odnosno signalizira promjenu stanja neke od tipki. Ukoliko je senzor detektirao dodir i postavio varijablu `keyHit` u `true` program ulazi u if petlju i poziva `readActiveKey()` funkciju koja vraća vrijednost trenutno dodirnute tipke. Nakon toga, u polje `key_value` i `key_time`, se spremaju vrijednost dodirnute tipke i vrijeme kad se detektirao dodir. Ako je povratna vrijednost `hit()` funkcije `false`, te ukoliko je prošlo barem 400 milisekundi od prethodno detektiranog dodira, poziva se funkcija `check_event()` koja je zadužena za ispravno klasificiranje dodira.

Funckija `check_event()` prolazi kroz polja `key_value` i `key_time` te provjerava razliku susjednih vrijednosti unutar `key_time` polja. Ako su detektirana samo dva dodira po bilo kojim tipkama unutar kratkog vremenskog razdoblja, funkcija će vratiti vrijednost koja odgovara detekciji dvostrukog dodira. Ako je detektirano više od dva dodira, a vrijednosti tipki su ili uzastopno povećane ili smanjene, tada je riječ o klizanju gore ili dolje po tipkovnici, te funkcija vraća vrijednost koja odgovara toj akciji. Nakon što `check_event()` odradi svoj dio posla i vrati neku vrijednost, poziva se `update_dispaly(int state)` funkcija koja ovisno o poslanoj vrijednosti osvježi prikaz na ekranima. U nastavku je dan dio koda zadužen za opisani postupak, a u privitku se nalaze kompletne funckije.

```
//provjera je li se dogodila promjena stanje neke od tipki
if (touch.hit()) {
    key = touch.readActiveKey(); //čitanje aktivne tipke
    if (key != 1&& key != 0) {
```

```
      time_t = millis(); //trenutno vrijeme u milisekundama
      key_value[end_key] = key; //polje sa dodirnutim tipkama
      key_time[end_time] = time_t; //polje sa vremenima dodira
      check = true;
      end_key = (end_key + 1);
      end_time = (end_time + 1);
    }
  }
//ukoliko je prošlo 400 milisekundi od prethodnog dodira pozovi
//funkciju koja će se provjeriti o kakvoj vrsti dodira je riječ
  else if (millis() - time_t > 400) {
    if (check) {
      int m = check_event();  //klasifikacija dodira
      update_display(m); //funkcija za osvježavanje ekrana
    }
  }
```

Prilikom testiranja uređaja znalo se dogoditi da, u slučaju istodobnog dodira dvije različite tipke, senzor zablokira i dodirnuta tipka ostane u detektiranom stanju duži vremenski period, nakon čega bi se rekalibrirala. Problem se riješio postavljanjem *max on duration* opcije na već spomenutu vrijednost od 680 milisekundi.

4.3. Komunikacija s mobilnom aplikacijom

Komunikacija sa mobilnom aplikacijom odvija se preko Bluetooth modula spojenog na digitalne pinove 7 i 8. Sa strane grafičkog sučelja, to jest Arduina, ta komunikacija teče na sljedeći način; unutar `loop()` funkcije program stalno provjerava postoji li nešto na Bluetooth komunikacijskim linijama te ukoliko ima, čita bajt podatka i sprema ga u polje `buf`. Kada pročitani podatak odgovara slovu 'e' prestaje sa punjenjem polja,te kreće čitanje spremljenih podataka. Ukoliko je primljen podatak o promjeni opterećenja, osvježi se ekran sa novim opterećenjem te se pošalje prema vozilu, a ako se prime podaci o prijeđenoj udaljenosti i vremenu provedenom u vožnji, primljeni podaci se spreme u odgovarajuća polja, `dist` za prijeđenu udaljenost i `tim` za

vrijeme, te se prikažu na ekranu kada korisnik uđe u odgovarajuću stavku izbornika. Podaci o brzini i opterećenju se konstantno šalju prema mobilnoj aplikaciji. Trenutno se taj prijenos obavlja svaku sekundu. U nastavku je dio koda zadužen za slanje preko Bluetooth veze. Kako aplikacija šalje podatke prema Arduinu je opisano u poglavlju o mobilnoj aplikaciji.

```
//primanje podataka
while (BTSerial.available()) {
    char c = BTSerial.read();
    if (c =='e') {
      sendData();
       } else {
      bufferData(c);
    }
  }
//slanje podataka
unsigned long send_data = millis();
  if (send_data - time_0 >= 1000) {
    time_0 = send_data;
    BTSerial.print(spd);
    BTSerial.print('y'); //razdvaja brzinu i opterećenje
    BTSerial.println(mod);
  }
```

4.4. Komunikacija s električnim vozilom

Komunikacija između električnog vozila i grafičkog sučelja obavlja se putem I^2C sabirnice, gdje električno vozilo ima funkciju upravljača, a grafičko sučelje, odnosno Arduino, funkciju izvršitelja. Kako je u prethodnom poglavlju objašnjeno, u komunikaciji sa QT1070 senzorom, koja se također odvija preko I^2C sabirnice, Arduino ima funkciju upravljača što znači da netom prije početka prijenosa podataka sa senzora Arduino se mora prebaciti iz stanja izvršitelja u stanje upravljača, te nakon završetka komunikacije se vratiti u stanje izvršitelja.

Pošto I^2C protokol ne omogućuje izvršitelju pokretanje komunikacije, Arduino mora na neki način signalizirati vozilu da je potrebno započeti komunikaciju kada korisnik zatraži neki od podataka koji dolaze iz električnog vozila. To se postiže korištenjem dodatnog signala spojenog na digitalni pin 2. Kada Arduino, odnosno sučelje, želi primiti podatak sa vozila, digitalni pin 2 se postavi u stanje 1 i na taj način se signalizira električnom vozilu da je potrebno započeti komunikaciju. Nakon što vozilo započne prijenos podataka, taj isti pin se postavi u stanje 0.

Adresni prostor korišten u komunikaciji sa vozilom je organiziran na sljedeći način:

0x42 → adresa *slave* jedinice (Arduino odnosno ekrani)

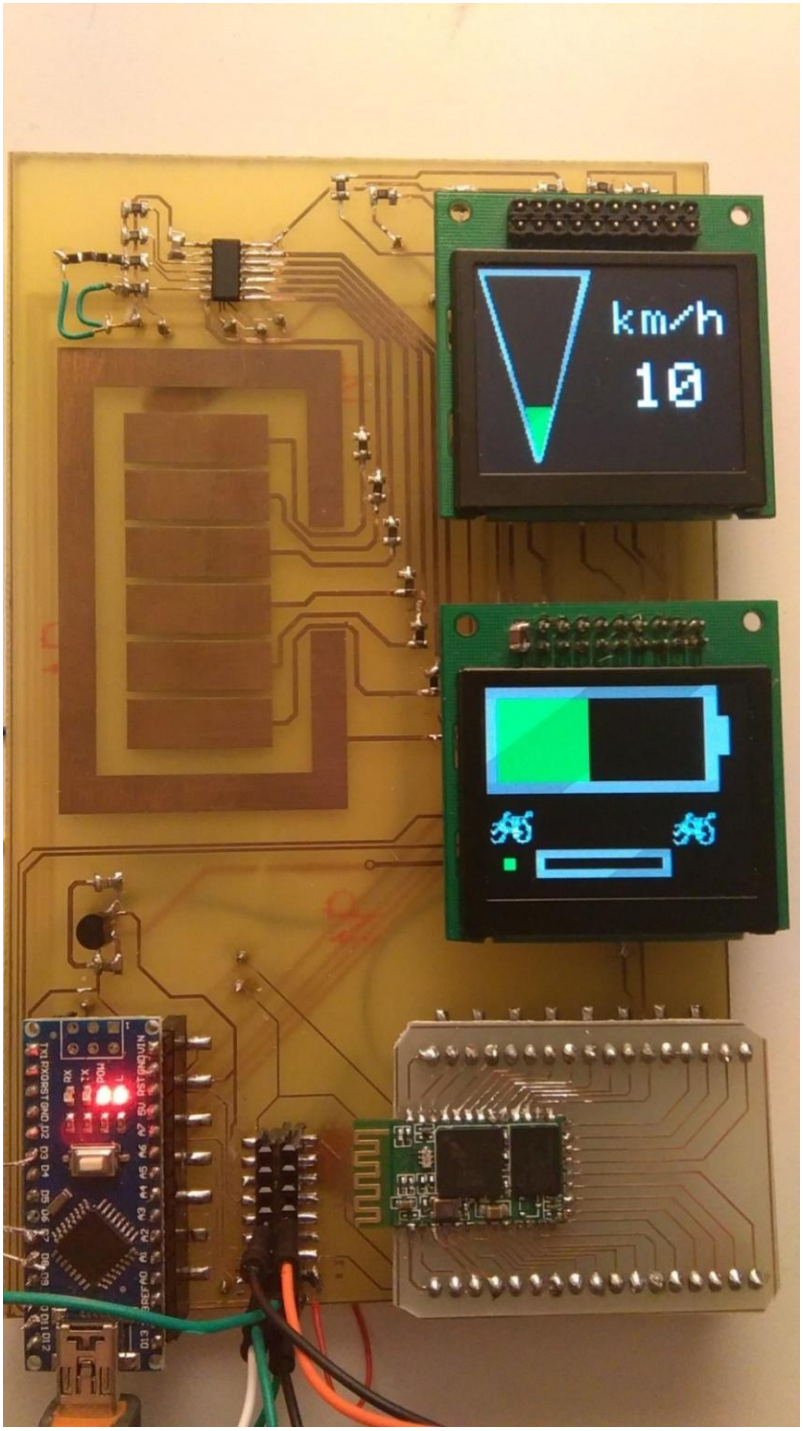
- raspodjela registara na adresi *slave* jedinice
  0x01 → ID registar
  0x02 → Status registar
  0x03 → Registar za dodatne parametre
  0x04 → Registar za stanje baterijskih ćelija

Nakon što komunikacija započne, vozilo i Arduino razmjene svoja trenutna stanja. Vozilo pošalje u status registar podatak o trenutnom stanju baterije, brzine i opterećenja te kontrolnu sumu koja nosi informaciju o ispravnosti poruke. Arduino kao odgovor vrati izmijenjeno ili neizmijenjeno opterećenje, dodatan bajt podataka koji signalizira vozilu je li potrebno slati dodatne informacije i kontrolnu sumu svoje poruke. Dodatan bajt koji Arduino vrati vozilu može biti jedna od sljedeće dvije vrijednosti:

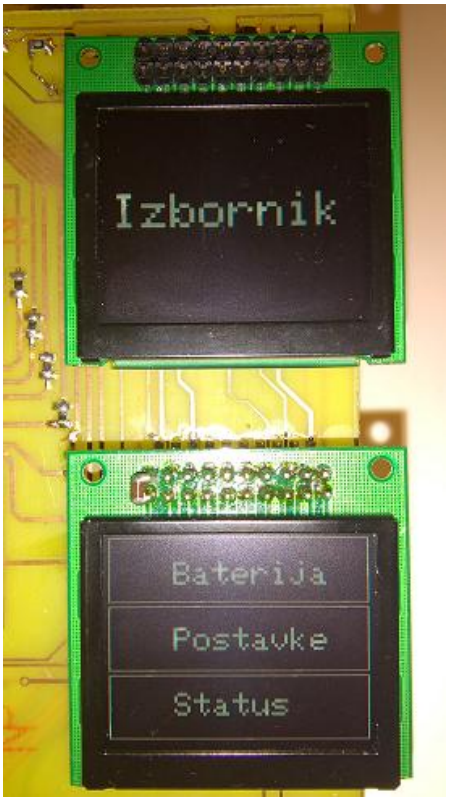
1. 0x01 – nakon čega vozilo u registar za stanje baterijskih ćelija upiše stanje svake ćelije
2. 0x02 – nakon čega se korisniku omogući promjena nekih od dodatnih parametara električnog vozila

Prilikom testiranja sustava dolazilo je do problema u I^2C komunikaciji. Naime, ponekad bi pinovi zaduženi za generiranje prekida, to jest signaliziranje promjene stanja tipkovnice i iniciranje komunikacije sa električnim vozilom, ostali na neodređeni vremenski period u aktivnom stanju i na taj način zamrznuli daljnju komunikaciju, ili sa

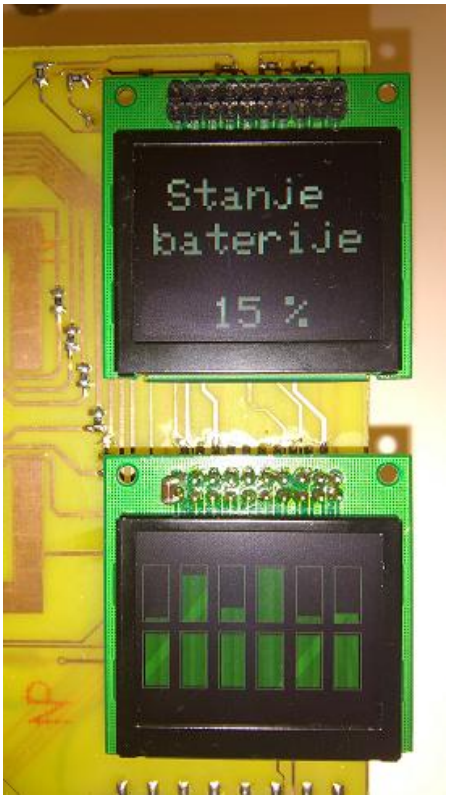
senzorom ili sa električnim vozilom. Problem se riješio postavljanjem timera koji provjerava stanje na digitalnim pinovima 2 i 3 te ukoliko su aktivni nakon određenog vremenskog perioda, u kojem nije bilo potrebe da budu, program ih vrati u neaktivno stanje. Na taj način komunikacija teče nesmetano.



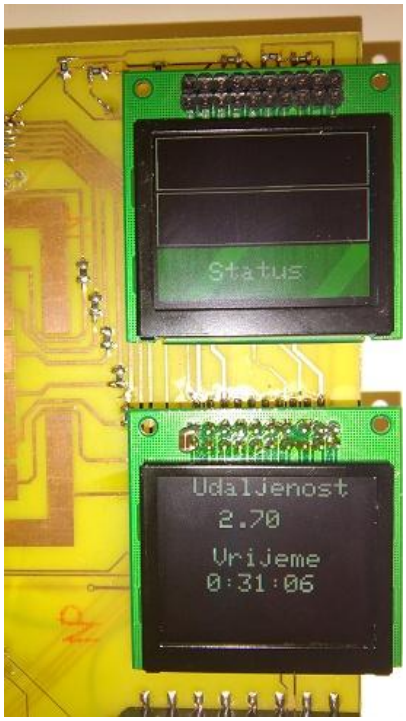
Slika 4.1 Pločica interaktivnog grafičkog sučelja



Slika 4.2 Izbornik grafičkog sučelja



Slika 4.3 Prikaz stanja baterijskih ćelija



Slika 4.4 Prikaz podataka prikupljenih pomoću mobilne aplikacije

5. Tehnologije i alati korišteni u razvoju Android aplikacije

U ovom poglavlju su opisane tehnologije i alati korišteni u razvoju Android aplikacije te način na koji je aplikacija povezana s električnim vozilom dok je sama aplikacija opisana u narednom poglavlju.

5.1. Adobe PhoneGap

Adobe PhoneGap [13] je jedna od distribucija i razvojnih okvira unutar Apache Cordova [14] sustava. PhoneGap omogućuje izradu aplikacija za mobilne uređaje različitih platformi koristeći HTML5, CSS3 i JavaScript, bez potrebe za radom u razvojnim okvirima i programskim jezicima specifičnima za pojedinu platformu. Apache Cordova, a samim time i PhoneGap, omogućuje pristup senzorima unutar mobilnog uređaja kao što su akcelerometar, baterija, kamera korištenjem dodataka nazvanih *Core Plugins.* Uz osnovne moguće je koristiti i dodatke napravljene samostalno ili do treće strane.

Aplikacije se mogu izgraditi koristeći PhoneGap Build ili PhoneGap CLI (*Command Line Interface)*. U ovom radu je korišten PhoneGap CLI, za čiji rad je potrebno imati instalirane *node.js* [15] i *git* [16]. U nastavku je dan primjer stvaranja aplikacije koristeći PhoneGap CLI:

```
phonegap create appFolder --id 'org.app.sample' -name 'app'
```

Gore navedena naredba će stvoriti mapu imena *appFolder* i definirati ime, *app*, i identifikator projekta, *org.app.sample*. Unutar mape *appFolder* stvorit će se nekoliko različitih datoteka i mapa uključujući datoteku *config.xml*, te mape *hooks, platforms, plugins* i *www.*

config.xml datoteka sadrži informacije o aplikaciji i dodacima koje ona koristi, dok se unutar *www* mape nalazi sav kod potreban za izgradnju aplikacije. *plugins* mapa se sastoji od dodataka koji su instalirani, a u *platforms* mapu potrebno je dodati platforme za koje se aplikacija želi izgraditi. Android platformu je moguće dodati sljedećom naredbom:

```
phonegap platform add android
```

Prije izvršenja prethodne naredbe potrebno se pozicionirati unutar projektne mape aplikacije. U ovom primjeru je to mapa *appFolder.*

Kada je sav potreban kod napisan i želi se izgraditi aplikaciju za Android platformu, potrebno je izvršiti sljedeću naredbu:

```
phonegap build android
```

Nakon što se aplikacija izgradi, unutar mape *platforms* se stvori odgovarajuća *.apk* datoteka.

5.2. HTML5

HTML5 je peta i najnovija revizija standarda HTML (*HyperText Markup Language)*, prezentacijskog jezika za izradu web stranica. HTML5 za zadaću ima uputiti web preglednik kako prikazati *hypertext* dokument uz nastojanje da taj isti dokument izgleda jednako, bez obzira o kojem je pregledniku se otvara.

Svaki HTML5 dokument se sastoji od HTML elementa, a svaki element od HTML oznaka (*tags)*. Postoje dvije vrste HTML oznaka, otvarajuće i zatvarajuće. Otvarajuće započinju znakom < i završavaju znakom >. Kod zatvarajućih oznaka se prije završnog znaka mora dodati kosa crta / [17].

Na početku HTML5 dokumenta se postavlja `<!DOCTYPE>` element kojim se definira standard koji se koristi, a nakon njega dolazi `<html>` element koji označava početak dokumenta. Unutar njega se nalaze `<head>` element, koji predstavlja zaglavlje i tu se dodaju jezične značajke dokumenta, naslov dokumenta, stilska obilježja stranice (CSS) te skripte napisane u JavaScript jeziku. Nakon `<head>` elementa dolazi `<body>` element u kojem se nalazi sadržaj samog dokumenta. U nastavku je primjer jednog HTML5 dokumenta:

```
<!DOCTYPE html>
<html>
     <head>
           <meta charset="UTF-8">
            <title>Naslov dokumenta</title>
     </head>
```

```
    <body>
        Sadržaj dokumenta
    </body>
</html>
```

5.3. CSS3

CSS3 je instanca CSS (*Cascading StyleSheets)* stilskog jezika. Koristi se za uređivanje izgleda dokumenta napisanog HTML jezikom. CSS je moguće napisati na nekoliko različitih načina; u zaglavlju HTML dokumenta unutar `<style>` elementa, unutar samih HTML oznaka ili ga je moguće napisati u posebnom dokumentu koji se onda poziva unutar zaglavlja [18].

CSS stilovi (*style sheet)* se sastoje od niza pravila, a svako pravilo se sastoji od jednog ili više selektora i deklaracijskog bloka. Selektori se koriste za deklaraciju na koji dio HTML dokumenta se stil odnosi. Selektori se mogu primjeniti na sve elemente određenog tipa ili elemente definirane istom *id* oznakom, kao i na sve elemente neke klase. Klase i id oznake su osjetljive na vrstu slova i uvijek moraju započeti slovom, a mogu sadržavati alfanumeričke oznake i podvlaku. Klasa se može primjeniti na više različitih elemenata dok *id* oznaka mora biti jedinstvena.

Deklaracijski blok se sastoji od niza deklaracija unutar uglatih zagrada. Svaka deklaracija ima svoje svojstvo i vrijednost koje su odvojene dvotočkom, a ako unutar jednog bloka postoji više deklaracija one moraju biti odvojene točka-zarezom. Svojstva kojima je moguće pridodati vrijednost su definirana u CSS standardu i svaki ima set vrijednosti koje su mu pridodane.

5.4. JavaScript

JavaScript [19] je jedan od najpopularnijih skriptnih jezika koji se danas koriste u svijetu. Podržan je na svim web preglednicima današnjice, a uz HTML i CSS predstavlja jezgru sadržaja napravljenih za WWW (*World Wide Web)*.

JavaScript se u HTML dokument može dodati izravno unutar `<script>` elementa ili uključivanjem vanjske datoteke u zaglavlju dokumenta. Jezik je po sintaksi sličan C jeziku s tim da se varijablama tip dodjeljuje dinamički nakon što im se pridjeli

vrijednost. Funkcije u JavaScriptu su objekti i svojstva im se također mogu dinamički mijenjati te ih se može slati kao argumente pri pozivu drugih funkcija. Kada se koristi u HTML dokumentima, komunicira sa elementima u dokumentu zahvaljujući interakciji s DOM (*Document Object Model)*. Na taj način je JavaScript skriptama omogućeno dohvaćanje i mijenjanje HTML elemenata kako bi se stranica napravila interaktivnijom i ugodnijom za korištenje.

U nastavku je naveden jednostavan primjer koji pokazuje na kakav način JavaScript može manipulirati HTML elementima:

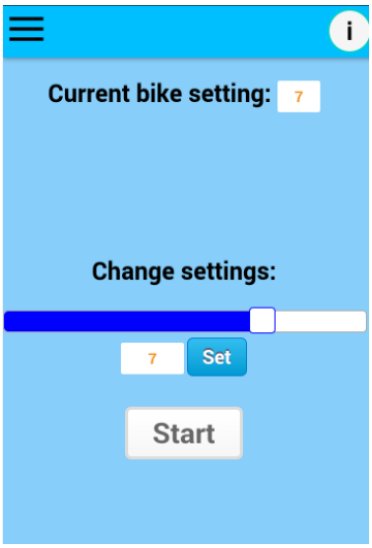
```
<!DOCTYPE html>
<html>
<body>
<p>Unesite broj:</p>
<input id="broj">
<button type="button" onclick="koji_broj()">Submit</button>
<p id="prikaz"></p>
<script>
function koji_broj() {
     var x;
     x = document.getElementById("broj").value;
// u varijabli x se nalazi vrijednost koja se unese u element s
// id="broj"
     document.getElementById("prikaz").innerHTML=x;
// dohvati se element s id="prikaz" i na njegovo mjesto napiše
//vrijednost koja se nalazi u varijabli x
}
</script>
</body>
</html
```

6. Android aplikacija

Android aplikacija je sastavljena od nekoliko funkcijskih cjelina koje razmjenjuju informacije s električnim vozilom pomoću Bluetooth veze. Aplikacija je preko Bluetooth modula povezana s Arduinom koji ima ulogu posrednika u komunikaciji vozila i mobilne aplikacije. U nastavku ovog poglavlja je opisana izvedba i način rada pojedinih dijelova aplikacije.

6.1. Promjena opterećenja bicikla

Prva velika funkcijska cjelina je odgovorna za promjenu opterećenja električnog vozila, a samim time i način rada vozila. Opterećenje je moguće mijenjati pomicanjem klizača u mobilnoj aplikaciji kako je prikazano na slici 6.1.



Slika 6.1 Promjena opterećenja vozila iz aplikacije

Moguće vrijednosti se kreću u razmaku od 0, kada je vozilo električni bicikl, do 10 kada je vozilo sobni bicikl. Odabir neke od međuvrijednosti mijenja opterećenje bicikla približavajući ga nekom od krajnjih stanja.

Postavljanjem željenog opterećenja aktivira se Bluetooth veza sa Arduinom. Za povezivanje i komunikaciju koristi se Bluetooth Serial plugin [22]. Veza se uspostavlja na sljedeći način:

```
document.addEventListener('deviceready', onDeviceReady, false);
```

```
function onDeviceReady() {
     //odspojiti za slučaj da je bilo spojeno kako ne bi došlo
     //do nenadanog ponašanja
     bluetoothSerial.disconnect();
     ... //kod za provjeru je li bluetooth omogućen
     //funkcija za spajanje na ciljani uređaj
     //nakon uspješnog spajanja poziva se funkcija onConnect
     bluetoothSerial.connect(macAddress, app.onConnect);
}
function onConnect() {
     //funkcija za pretplaćivanje odnosno čekanje na dolazeće
     //poruke sa delimiterom \n; nakon primljene poruke poziva
     //se funkcija onMessage a nakon neuspješnog pretplaćivanja
     //funkcija subscribe Failed
     bluetoothSerial.subscribe("\n", app.onMessage,
     app.subscribeFailed);
}
function onMessage(data) {
     //ide kroz primljenu poruku dok ne dođe do pozicije sa
     //znakom 'y' koji se koristi za razdvajanje podatka o
     //brzini i opterećenju poslanih sa bicikla
     for (var i=0; i<data.length; i++){
          if(data[i]=='y')
               var index=i;
     }
     //prvi dio poruke je brzina a nakon 'y' dolazi opterećenje
     //podatak o brzini se spremi u localStorage za kasnije
     //korištenje, a podatak o opterećenju se upiše u element s
     //id=setting
     //stanje unutar elementa se mijenja kako se promijeni
     //primljeni podatak o opterećenju
     var spd=data.substring(0,index);
```

```
    var md=data.substring(index+1, data.length);
    localStorage.setItem("speed", parseFloat(spd));
    $("#setting").val(parseFloat(md));
    localStorage.setItem("change_mod", parseFloat(md));
}
```

Da bi vezu bilo moguće uspostaviti, poziv `bluetoothSerial.connect()` funkcije je potrebno postaviti unutar 'deviceready' događaja. 'deviceready' je događaj koji se aktivira nakon što je Cordova u cijelosti pokrenuta, što signalizira aplikaciji da je moguće koristiti API funkcije pripadnog uređaja. Ukoliko bi se veza pokušala uspostaviti prije navedenog događaja aplikacija ne bi bila u mogućnosti koristiti senzore mobilnog uređaja.

Funkcija `bluetoothSerial.connect()` prima tri argumenta od kojih je samo jedan obavezan (MAC adresa uređaja s kojim se potrebno spojiti). Druga dva su funkcije koje se pozivaju nakon uspješnog spajanja ili odspajanja sa ciljanog uređaja.

Slanje postavljenog moda vožnje, to jest opterećenja, se šalje pristikom 'Set' tipke. Na taj način se aktivira funkcija koja putem Bluetooth veze šalje podatak o odabranom opterećenju na bicikl preko Arduino mikrokontrolera. U nastavku je prikazan dio koda zadužen za slanje podataka:

```
document.getElementById("mode_set").addEventListener('click',
send_mode); //čekanje na aktivaciju 'set' tipke
function send_mode(){
    var m=$("#amount").val(); //dohvaćanje vrijednosti
    //pridružene klizaču kojim se mijenja opterećenje
    bluetoothSerial.write("w"+m+"e"); //slanje dohvaćene
    //vrijednosti preko Bluetooth veze
}
```

6.2. Praćenje podataka tijekom vožnje

Druga velika funkcijska cjelina aplikacije se odnosi na dva različita korisnička sučelja koja je moguće koristiti ovisno o aktivnoj postavki vožnje (Slika 6.2 i 6.3). Do sučelja je moguće doći na dva načina; odabirom željenog opterećenja na prethodno opisani

način ili izravnim odabirom sobnog ili električnog bicikla u izborniku mobilne aplikacije. Ovisno o odabranom opterećenju učita se sučelje za korištenje sobnog ili električnog bicikla. Na početku oba `div` elementa (`id=outside` za električni bicikl i `id=free` za sobni bicikl) imaju postavljeno `display` svojstvo na `none` što znači da nisu vidljivi.

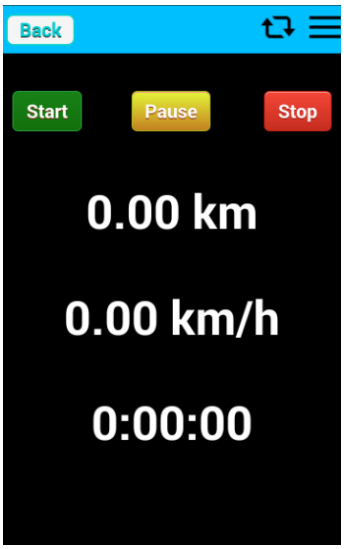
```
<div class="outside" id="outside" style="display:none;"></div>

<div id="free" style="display:none;"></div>
```

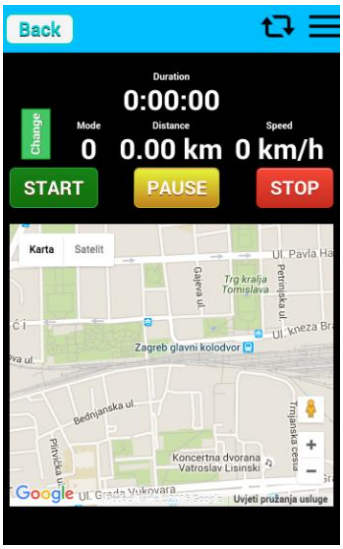
Svojstvo željenog elementa se promijeni nakon što se dohvati vrijednost opterećenja koja je prethodno spremljena te se ovisno o njoj postavi svojstvo `display` na `block` ili ostavi onakvo kakvo je.

```
document.addEventListener('deviceready', function(){
     bluetoothSerial.disconnect();
     var x=localStorage.getItem("change_mod");
     //ako je sobni bicikl
     if(x==10){
          document.getElementById("free").style.display="block"
     ;
          //... //provjera je li bluetooth omogućen
     else{
          //...//provjera je li omogućen bluetooth i korištenje
          //lokacije
          document.getElementById("outside").style.display="blo
          ck";
          //prikaz trenutnog opterećenja
          document.getElementById("current_load").innerHTML=x;
          //pozivanje funkcije za iscrtavanje karte s prikazom
          //lokacije
          start();
     }
```

```
    //spajanje na bluetooth i pozivanje funkcije connect koja
    //prima podatke o brzini i prikazuje ih na zaslonu
    bluetoothSerial.connect(macAddress, connect);
});
```



Slika 6.2 Sučelje za sobni bicikl



Slika 6.3 Sučelje za električni bicikl

Kada se učita sučelje u slučaju korištenja sobnog bicikla (Slika 6.2) postoje tri opcije: pokretanje ('Start' tipka), pauziranje ('Pause' tipka) i zaustavljanje ('Stop' tipka) mjerenja. Pritiskom 'Start' tipke započinje mjerenje vremena provedenog koristeći sobni bicikl što je ostvareno korištenjem `setInterval(function, duration)` funkcije koja prima dva argumenta, funkciju za izvođenje i vremenski interval ponavljanja njenog izvođenja u milisekundama. Funkcija koja se u ovom slučaju izvodi broji sekunde od početka mjerenja, prebacuje ih u format '0:00:00' i prikazuje na zaslonu. Istovremeno s tim započinje i primanje podataka o brzini na način kako je opisano u prethodnom poglavlju. Iz podataka o sekundama od početka mjerenja i prosječnoj brzini izračuna se prijeđena udaljenost.

Mjerenje je moguće u svakom trenutku pauzirati pritiskom tipke 'Pause' koja zaustavlja `setInterval(function, duration)` funkciju. Ponovnim pritiskom 'Start' tipke mjerenje se nastavlja. Pritsikom 'Stop' tipke otvara se prozor sa pregledom podataka prikupljenih tijekom vožnje i omogućuje se korisniku njihovo spremanje. Podaci se spremaju koristeći localforage [23] u kombinaciji sa

localForage-cordovaSQLiteDriver [24]. localforage omogućuje izvanmrežno askinkrono spremanje podataka na bazi ključ - vrijednost koristeći jednostavan API gotovo identičan onom od localStorage [25] bez potrebe za eksplicitnim pretvaranjem podataka koji se spremaju u stringove, a u kombinaciji sa localForage-cordovaSQLiteDriver omogućuje korištenje svih pogodnosti SQLite preko jednostavnih funkcija koje koristi.

localforage omogućuje stvaranje više različitih instanci pa spremanje podatka pod nekim ključem jedne instance se ne kosi sa podatkom spremljenim pod istim tim ključem u drugoj instanci. Na taj način moguće je odvojiti podatke prikupljene za vrijeme korištenja vozila kao sobnog ili kao električnog bicikla.

```
var stationaryb = localforage.createInstance({
    name: "Stationary"
});
var bike = localforage.createInstance({
    name: "ElectricBike"
});
```

Da bi se koristio localForage-cordovaSQLiteDriver potrebno je pozvati `defineDriver` i `setDriver` funkcije kako bi se prisililo localforage na korištenje SQLite upravljačkog programa.

```
stationaryb.defineDriver(window.cordovaSQLiteDriver).then(funct
ion(){
     return stationaryb.setDriver([
          //pokušaj postaviti cordovaSQLiteDriver
          window.cordovaSQLiteDriver._driver,
          //inače koristi nešto drugo
          stationaryb.INDEXEDDB,
          stationaryb.WEBSQL,
          stationaryb.LOCALSTORAGE
     ]);
});
```

Nakon postavljanja localforage u željeno stanje moguće je obaviti samo spremanje podataka. Prvo se dohvati vrijednost sa sljedećeg HTML elementa:

```
<input class="save_ride2" id="track_id_free" placeholder="Enter
name" />
```

te se spremi u `track_id_free` varijablu. Podaci prikupljeni za vrijeme korištenja vozila (ukupne sekunde, udaljenost, brzina i opterećenje) se spreme u varijablu `tracking_data_free`, a potom se ta dva podatka skupa sa datumom i vremenom spreme u stationaryb instancu.

```
stationaryb.getItem('ids', function(err, value){
     //provjera postoji li nešto spremljeno pod 'ids' ključem.
     //ako ima postavlja tu vrijednost u polje ids, inače
     stvara //prazno ids polje; nakon toga u ids polje
     postavlja id pod //kojim su spremljeni podaci, timestamp
     sa trenutnim //datumom i vremenom te tracking_data_free sa
     podacima
     if(!value){
          ids=[]; ids.push({id: track_id_free, timestamp: (new
          Date()).getTime(),
          tracking_data_2:tracking_data_free});
     }
     else{
          ids=value; ids.push({id: track_id_free, timestamp:
          (new Date()).getTime(),
          tracking_data_2:tracking_data_free});
     }
     //sprema se u stationaryb instancu
     stationaryb.setItem('ids', ids, function(err, value){
          ... //resetiranje korištenih varijabli
          //otvaranje history.html prozora nakon spremanja
          javascript:window.location.href="history.html";
     });
```

```
});
```

Drugo sučelje (Slika 6.3) se koristi kada je vozilo u stanju električnog bicikla. Jednako kao i u slučaju sobnog bicikla, ovdje je također moguće pokrenuti, pauzirati i zaustaviti praćenje pritiskom odgovarajuće tipke te je također konstatno dostupno vrijeme od početka praćenja, prijeđena udaljenost i trenutna brzina.

Ovo sučelje ima dodatak u vidu prikaza karte sa trenutnom lokacijom vozila. Sa svakom promjenom lokacije pomiče se i marker koji označava trenutni položaj vozila te se također iscrtava i linija povezujući sve prethodno detektirane lokacije. Prikaz karte omogućen je korištenjem Google Maps Javascript API [26]. Karta mora biti smještena unutar `div` elementa sa specificiranom dužinom i širinom. Prije korištenja API-a potrebno ga je učitati na sljedeći način:

```
<script
src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY
</script>
```

gdje API_KEY predstavlja jedinstveni identifikator aplikacije.

Nakon što se API učita, kartu je moguće stvoriti koristeći Map konstruktor:

```
Map(mapDiv:Node, opts?:MapOptions)
```

Na taj način unutar specificiranog `div` elementa iscrta se karta. Jednom kad se karta učita na nju je moguće dodavati različite dodatne opcije poput markera koristeći Marker konstruktor, linije koristeći Polyline konstruktor, različite simbole, prozorčiće sa informacijama i ostalo.

Da bi se uspješno pratila promjena lokacije mobilnog uređaja, a samim time i električnog vozila, koristi se Geolocation API [27]. Ovaj API daje podatak o geografskom položaju mobilnog uređaja koji je predstavljen geografskom širinom i dužinom koristeći podatke iz GPS-a ili različitih mrežnih signala (IP adresa, WiFi, GSM/CDMA stanice). Lokacija se dobije korištenjem Geolocation objekta:

```
interface Geolocation {
     void getCurrentPosition(PositionCallback successCallback,
```

```
          optional PositionErrorCallback errorCallback,
optional PositionOptions options);
     long watchPosition(PositionCallback successCallback,
optional PositionErrorCallback errorCallback, optional
PositionOptions options); void clearWatch(long watchId);
};
callback PositionCallback = void (Position position);
callback PositionErrorCallback = void (PositionError
positionError);
```

`getCurrentPosition()` metoda može primiti jedan, dva ili tri argumenta. Nakon što se pozove asinkrono pokuša dohvatiti trenutnu lokaciju uređaja te proslijedi stvoreni Position objekt u `successCallback` funkciju. Position objekt sadrži dva atributa od kojih je jedan `coordinates` koji sadrži geografske dužinu i širinu i točnost s kojima su prikupljene. Ako metoda ne uspije doći do trenutne lokacije poziva se `errorCallback` sa odgovarajućim PositionError objektom. Kao treći argument metoda ima PositionOptions objekt:

```
dictionary PositionOptions {
     boolean enableHighAccuracy = false;
     [Clamp] unsigned long timeout = 0xFFFFFFFF;
     [Clamp] unsigned long maximumAge = 0;
};
```

`enableHighAccuracy` atribut služi kao signal da aplikacija želi primiti najbolji mogući rezultat. Postavljanje atributa u `true` može rezultirati sporijim odzivom i većom potrošnjom energije, ali je korisno u situacijama kada je bitno dobiti najtočniju raspoloživu lokaciju. `timeout` atribut označava maksimalno vrijeme koje smije proći između poziva metode do poziva pripadajućeg `successCallback`. `maximumAge` atribut pokazuje je li aplikacija spremna prihvatiti prethodno spremljenu lokaciju. Starost prihvatljive lokacije dana je u vremenu u milisekundama. Ako je ovaj atribut postavljen na 0 aplikacija zahtjeva da se dohvati novi Position objekt, a ako je

postavljen u beskonačno aplikacija signalizira da prihvaća bilo koji Position objek neovisno o njegovoj starosti.

`watchPosition()` metoda radi slično kao prethodno opisana metoda i također prihvaća jedan, dva ili tri argumenta koji su isti kao u slučaju `getCurrentPosition()` metode. Jedina razlika je što nakon poziva, ova metoda vraća `watchID` identifikator koji predstavlja *watch operation* te nakon toga asinkrono započinje s praćenjem lokacije koje traje sve dok se ne pozove `clearWatch()` metoda sa odgovarajućim identifikatorom.

Sučelje koje se koristi kada je vozilu električni bicikl na izgled je, osim prikazane karte, više manje jednako onom korištenom kod sobnog bicikla, uz dodatak mogućnosti podešavanja opterećenja vozila tijekom same vožnje. Pritiskom tipke 'Change' prikazuje se `div` element sa klizačem za promjenu opterećenja čija je funkcija jednaka onoj na prije opisanom zaslonu.

Karta se prikaže čim aplikacija prepozna odabrano opterećenje u granicama onog postavljenog za električni bicikl. Funkcija koja je zadužena za prikaz karte mora četiri puta detektirati promjenu lokacije (kako bi procjenjena lokacija bila što točnija). Nakon toga `div` element koji sadrži tipke 'Start', 'Pause' i 'Stop' promijeni `display` svojstvo sa `none` na `block`.

Pritiskom na 'Start' pokreće se funkcija za praćenje lokacije korištenjem `watchPosition()` metode kao i iscrtavanje linije koja predstavlja prijeđeni put (Polyline). U nastavku se nalazi isječak koda koji za zadaću ima gore navedeno:

```
watch_id = navigator.geolocation.watchPosition(
        function(position){
            //spremanje lokacije za kasnije korištenje
            tracking_data.push(position.coords.latitude);
            tracking_data.push(position.coords.longitude);
            //mjerenje udaljenosti između dvije točne
            distance_meters+=gps_distance(latitude_l, lng_l,
position.coords.latitude, position.coords.longitude);
            //prikaz prijeđenih kilometara
```

```
            document.getElementById('distance').innerHTML =
parseFloat(distance_meters).toFixed(2) +' km';
            //nova geografska širina i dužina za potrebe
            mjerenja //udaljenosti
            latitude_l=position.coords.latitude;
            lng_l=position.coords.longitude;
        },function(msg){alert("turn on gps");},
{maximumAge:10000, timeout:50000, enableHighAccuracy:true});
poly= navigator.geolocation.watchPosition(function(position) {
var newPoint = new google.maps.LatLng(position.coords.latitude,
                  position.coords.longitude);
//iscrtavanje linije koja predstavlja prijeđeni put
mypath = new google.maps.Polyline({
               path: pathLineArray,
               strokeColor: '#FF0000',
               strokeOpacity: 2.0,
               strokeWeight: 3,
               map: map,
            });
pathLineArray.push(newPoint);
}, function(){alert("turn on gps");}, {maximumAge:10000,
timeout:50000, enableHighAccuracy:true});
```

Prijeđena udaljenost se mjeri sa svakom promjenom trenutne lokacije pomoću funkcije `gps_distance(lat1, lon1, lat2, lon2)`. Funkcija prima četiri argumenta koji predstavljaju dva para geografske širine i dužine. Funkcija se prvi put pozove pritiskom na 'Start' i svakom daljnom promjenom lokacije vozila.

Trenutna brzina i trajanje vožnje se dobiju na isti način kao u prethodno opisanom sučelju sobnog bicikla.

Pritiskom na 'Stop' otvara se prozor sa pregledom vožnje i omogućuje se korisniku njeno spremanje. Podaci se spremaju kao kad je u pitanju sobni bicikl, s tim da se umjesto u stationaryb spremaju u bike instancu. Uz podatke koje je moguće pratiti

tijekom cijele vožnje u tom novom prozoru se prikažu i podaci o maksimalnoj i minimalnoj nadmorskoj visini na kojoj se vozilo našlo tijekom vožnje za čiji se izračun koristi se Elevation service unutar Google Maps Javascript API.

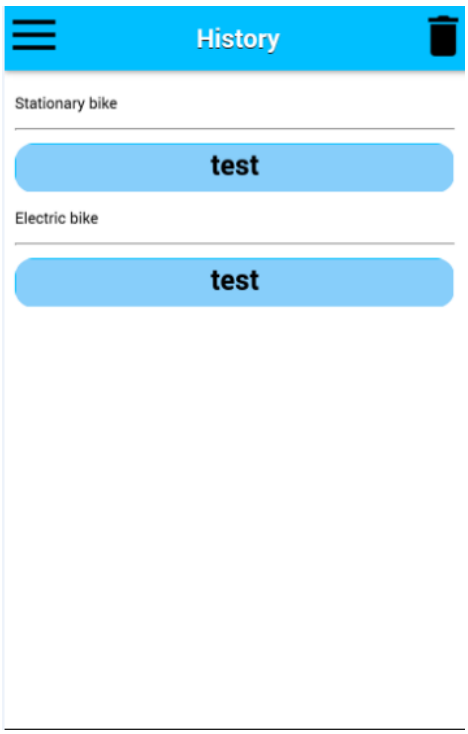
```
var newPoint = new google.maps.LatLng(position.coords.latitude,
                                      position.coords.longitude);
        elevator.getElevationForLocations({
            'locations':[newPoint]
            }, function (results, status) {
            if (status == google.maps.ElevationStatus.OK) {
                //dohvati prvi rezultat
                if (results[0]) {
                    text = results[0].elevation;
                    //dodaj  rezultat  u  elevation  array  za
                    //kasnije korištenje
                    if(!elevation) elevation=[];
                    elevation.push(results[0].elevation);
                } else {
                    alert('No results found');
                }
            }
        });
```

6.3. Ostale opcije dostupne u aplikaciji

Osim navedenog, aplikacija ima i nekoliko dodatnih mogućnosti. Na 'History' zaslonu (Slika 6.4) ispisuju se prethodno spremljene vožnje te je također omogućeno brisanje svih prikupljenih podataka selektiranjem odgovarajuće tipke. Vožnje su podijeljene na one spremljene dok je vozilo bilo sobni bicikl i one kada je vozilo bilo električni bicikl. To je omogućeno korištenjem prije spomenutih instanci stationaryb i bike pri spremanju prikupljenih podataka. Pohranjene vožnje se na zaslonu izlistaju unutar `table` elementa tako da se prolazi kroz cijelu stationaryb ili bike instancu te doda u

tablicu redak sa imenom pod kojim su podaci spremljeni. U nastavku slijedi isječak koda kojim se popunjava tablica.

```
//recs_2 predstavlja podatke o id-u podataka prikupljenih dok
je //vozilo bilo električni bicikl
for (var i=0; i<recs_2.length; i++) {
     $(".history_out").append("<tr><td id='"+i+"'
class='td_2'><a id='"+recs_2[i]['id_2']+i+"' class='object2'
href='track_info.html'>" + recs_2[i]['id_2'] +
"</a></td></tr>");
}
//recs predstavlja podatke o id-u podataka prikupljenih dok je
//vozilo bilo sobni bicikl
for (var i=0; i<recs.length; i++) {
     $(".history").append("<tr><td id='"+i+"' class='td_2'><a
id='"+recs[i]['id']+i+"' class='object3'
href='track_info_free.html'>" + recs[i]['id'] +
"</a></td></tr>");}
```



Slika 6.4 Prikaz zaslona na kojem su ispisane sve spremljene vožnje

Odabirom neke od spremljenih vožnji korisnik se vodi na jedan od zaslona gdje se prikažu podaci koji pripadaju odabranoj vožnji.

Nakon što se učita 'History' zaslon poziva se funkcija `calculate_data()` zadužena za slanje podataka o prijeđenoj kilometraži i vremenu provedenom u vožnji na interaktivno grafičko sučelje. Funkcija se poziva na takav način jer se aplikacija nakon spremanja podataka o vožnji preusmjeri upravo na 'History', pa se na taj način obrada i slanje podataka uvijek izvode na istom mjestu. `calculate_data()` ima zadatak pozbrajati podatke o pojedinačnim vožnjama i poslati ih Arduinu. Slanje se obavlja funkcijom `bluetoothSerial.write("q"+x+"z"+ t +"e")` gdje 'q' predstavlja početak slanja podatka o udaljenosti (x), 'z' o ukupnom vremenu (t), a 'e' signalizira Arduinu da je prijenos završen.

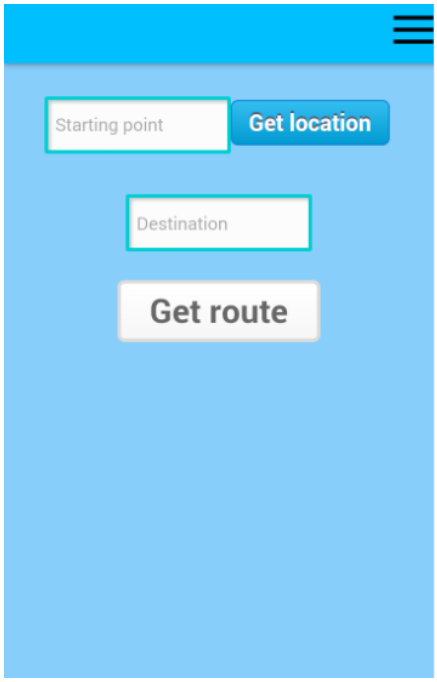
Uz 'History' postoji i 'Challenge' zaslon, koji mu je identičan, gdje odabir neke od izlistanih vožnji vodi na posebne zaslone na kojima je moguće popraviti rezultate zabilježene za vrijeme odabrane vožnje. Novi podaci se u tom slučaju prikupljaju na isti način kao što je opisano u poglavlju o praćenju podataka tijekom vožnje.

Omogućeno je također i korištenje zaslona koji iscrtava rutu definiranu s početnom i krajnjom točkom koju sam korisnik odabire (Slika 6.5 i 6.6). Pritiskom na 'Get route' tipku poziva se funkcija `calculateRoute(from, to, x, y)` koja prima četiri argumenta od kojih prva dva predstavljaju početnu i krajnju točku, a druga dva geografsku širinu i dužinu potrebnu za centriranje na karti. U nastavku je dio koda `calculateRoute` funkcije zadužen za iscrtavanje rute na karti:

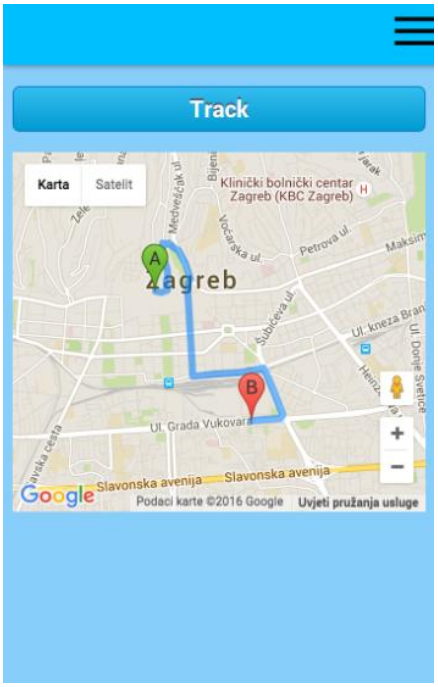
```
var directionsService = new google.maps.DirectionsService();
    var directionsRequest = {
        origin: from,
        destination: to,
        travelMode: google.maps.DirectionsTravelMode.DRIVING,
        unitSystem: google.maps.UnitSystem.METRIC
    };
    directionsService.route(
        directionsRequest,
```

```
function(response, status){
    if (status == google.maps.DirectionsStatus.OK){
        new google.maps.DirectionsRenderer({
        map: map,
        directions: response
        });
    }
});
```

Korisniku se daje na izbor da kao početnu točku ručno unese željenu adresu ili da pomoću 'Get location' tipke koristi svoju trenutnu lokaciju. Tada se pozove `geocodeLatLng2(geocoder,position)` funkcija kojoj je kao argumente potrebno poslati Geocoder [28] i Position objekt iz čega se dobije adresa na kojoj se uređaj nalazi. Korisnik ima mogućnost praćenja prije definiranih parametara vožnje (vrijeme trajanja, udaljenost, brzina) po zadanoj ruti. U tom slučaju se na ekranu iscrta zaslon identičan onome opisanom kada je vozilo električni bicikl te vrijede sve karakteristike opisane uz njegovo korištenje.



Slika 6.5 Zaslon za odabir rute



Slika 6.6 Prikaz iscrtane rute

7. Zaključak

Cilj ovog rada bio je izrada interaktivnog grafičkog sučelja i popratne mobilne aplikacije za postojeće električno vozilo.

Izrađeno je interaktivno grafičko sučelje sastavljeno od dva OLED ekrana i kapacitivne tipkovnice kojima se upravlja pomoću Arduino Nano pločice. Ekrani s Arduinom komuniciraju putem SPI sabirnice. Tipkovnicom se upravlja preko QT1070 senzora koji je na Arduino spojen preko I^2C sabirnice. Na ekranima je u svakom trenu moguće vidjeti stanje baterije, trenutnu brzinu vozila kao i opterećenje, a kapacitivnom tipkovnicom je moguće ući u dodatan izbornik. Izbornik trenutno sadrži tri stavke. Prva daje uvid u stanje svih baterijskih ćelija, druga omogućuje promjenu određenih parametara električnog vozila, a treća prikazuje ukupnu prijeđenu udaljenost u kilometrima kao i zbroj vremena trajanja svih vožnji. Posljednja dva podatka grafičko sučelje dobiva preko mobilne aplikacije sa kojom je povezan Bluetooth vezom.

Mobilna aplikacija je izrađena koristeći Adobe PhoneGap razvojnu okolinu koja omogućuje izradu aplikacija koristeći HTML5, CSS3 i JavaScript jezik. Aplikacija nudi mogućnosti praćenja, prikaza i pohrane podataka prikupljenih za vrijeme trajnja svake vožnje. Također nudi prikaz i praćenje lokacije električnog vozila na karti koristeći Google Maps JavaScript API i Geolocation API.

8. Literatura

[1] Arduino Software (IDE). URL= https://www.arduino.cc/en/Guide/Environment (datum pristupa: 26.6.2016)

[2] Wire Library. URL= https://www.arduino.cc/en/Reference/Wire (datum pristupa: 26.6.2016)

[3] Woodford, Chris. OLEDs (Organic LEDs) and LEPs (light-emitting polymers). 4. studeni 2015. URL= http://www.explainthatstuff.com/how-oleds-and-leps-work.html (datum pristupa: 26.6.2016)

[4] Newhaven Display International. URL= http://www.newhavendisplay.com/nhd169160128ugc3-p-5603.html (datum pristupa: 26.6.2016)

[5] SYNCOAM Co., Ltd. SEPS525. 14. travanj 2006. *160 RGB x 128 Dots, 262K Colors PM-OLED Display Driver and Controller.*

[6] Serial Peripheral Interface Bus. URL= https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus (datum pristupa: 26.6.2016)

[7] Voltage divider. URL= https://en.wikipedia.org/wiki/Voltage_divider (datum pristupa: 26.6.2016)

[8] Newhaven Display International. NHD-1.69-160128UGC3. *Graphic Color OLED Display Module.* 1. svibanj 2015.

[9] Solc, Tomaz. 2013. Software License Agreement (BSD License). Adafruit Industries. 2012. All rights reserved. URL= https://github.com/avian2/SEPS525-OLED-Library Kod modificiran prema uvjetima iz licence. (datum pristupa: 26.6.2016)

[10] Atmel. Atmel AT42QT1070. Svibanj 2013. *Seven-channel QTouch® Touch Sensor IC*

[11] Shibley, Noah. AtTouch. 18. srpnja 2011. GNU General Public License, version 3. Free Software Foundation. URL= https://github.com/nullboundary/AtTouch Kod modificiran prema uvjetima GNU General Public License. (datum pristupa: 26.6.2016)

[12] I2C URL= https://learn.sparkfun.com/tutorials/i2c (datum pristupa 26.6.2016)

[13] Adobe PhoneGap URL= http://phonegap.com/ (datum pristupa 26.6.2016)

[14] Apache Cordova URL= https://cordova.apache.org/ (datum pristupa 26.6.2016)
[15] Node.js URL= https://nodejs.org/en/ (datum pristupa 26.6.2016)
[16] git URL= https://git-scm.com/ (datum pristupa 26.6.2016)
[17] HTML5 URL= http://www.w3schools.com/html/html5_intro.asp (datum pristupa: 26.6.2016)
[18] CSS3 URL= http://www.w3schools.com/css/css3_intro.asp (datum pristupa 26.6.2016)
[19] JavaScript URL= http://www.w3schools.com/js/ (datum pristupa: 26.6.2016)
[20] Bluetooth URL= https://en.wikipedia.org/wiki/Bluetooth (datum pristupa: 26.6.2016)
[21] Serial port bluetooth module (Master/Slave) URL= http://www.seeedstudio.com/wiki/Serial_port_bluetooth_module_(Master/Slave) (datum pristupa: 26.6.2016)
[22] Coleman, Don. BluetoothSerial. Travanj 2013. Apache License, Version 2.0. URL= https://github.com/don/BluetoothSerial (datum pristupa: 26.6.2016)
[23] Mozilla. 2014. Apache License, Version 2.0. URL= https://github.com/mozilla/localForage (datum pristupa: 26.6.2016)
[24] Greasidis, Thodoris. 2015. Apache License, Version 2.0. URL= https://github.com/thgreasi/localForage-cordovaSQLiteDriver (datum pristupa: 26.6.2016)
[25] HTML5 Local Storage. URL= http://www.w3schools.com/html/html5_webstorage.asp (datum pristupa: 26.6.2016)
[26] Google Maps APIs. Google Maps JavaScript API. URL= https://developers.google.com/maps/documentation/javascript/ (datum pristupa: 26.6.2016)
[27] Geolocation API Specification. Copyright © 2008-2013 W3C® (MIT, ERCIM, Keio, Beihang), All Rights Reserved. URL= https://dev.w3.org/geo/api/spec-source.html (datum pristupa: 26.6.2016)
[28] Google Maps APIs. Google Maps Geocoding API. URL= https://developers.google.com/maps/documentation/geocoding/intro (datum pristupa: 26.6.2016)

9. Sažetak

Interaktivno grafičko sučelje za električno vozilo

Rezultat ovog rada je interaktivno grafičko sučelje sa pripadnom tiskanom pločicom i mobilna aplikacija za električno vozilo. Na ekranima se prikazuju grafički i brojčani indikatori stanja baterije, brzine, opterećenja te svih baterijskih ćelija. Tipkovnica omogućuje ulazak u izbornik, odabir jedne od stavki izbornika i povratak na početno stanje ekrana. Mobilna aplikacija omogućuje praćenje i pohranu nekoliko različitih informacija o vožnji električnog vozila. Neke od njih su ukupno trajanje vožnje, prijeđena udaljenost, prosječna i maksimalna postignuta brzina za vrijeme trajanja vožnje kao i praćenje promjene lokacije vozila uz prikaz na karti. Aplikacija je izgrađena koristeći PhoneGap CLI, a sve potrebne sheme su napravljene korištenjem programskog paketa Altium Designer 14.

Ključne riječi: interaktivno grafičko sučelje, električni bicikl, OLED ekran, kapacitivna tipkovnica, Arduino Nano, QT1070, SEPS525, Bluetooth, mobilna aplikacija, Adobe PhoneGap

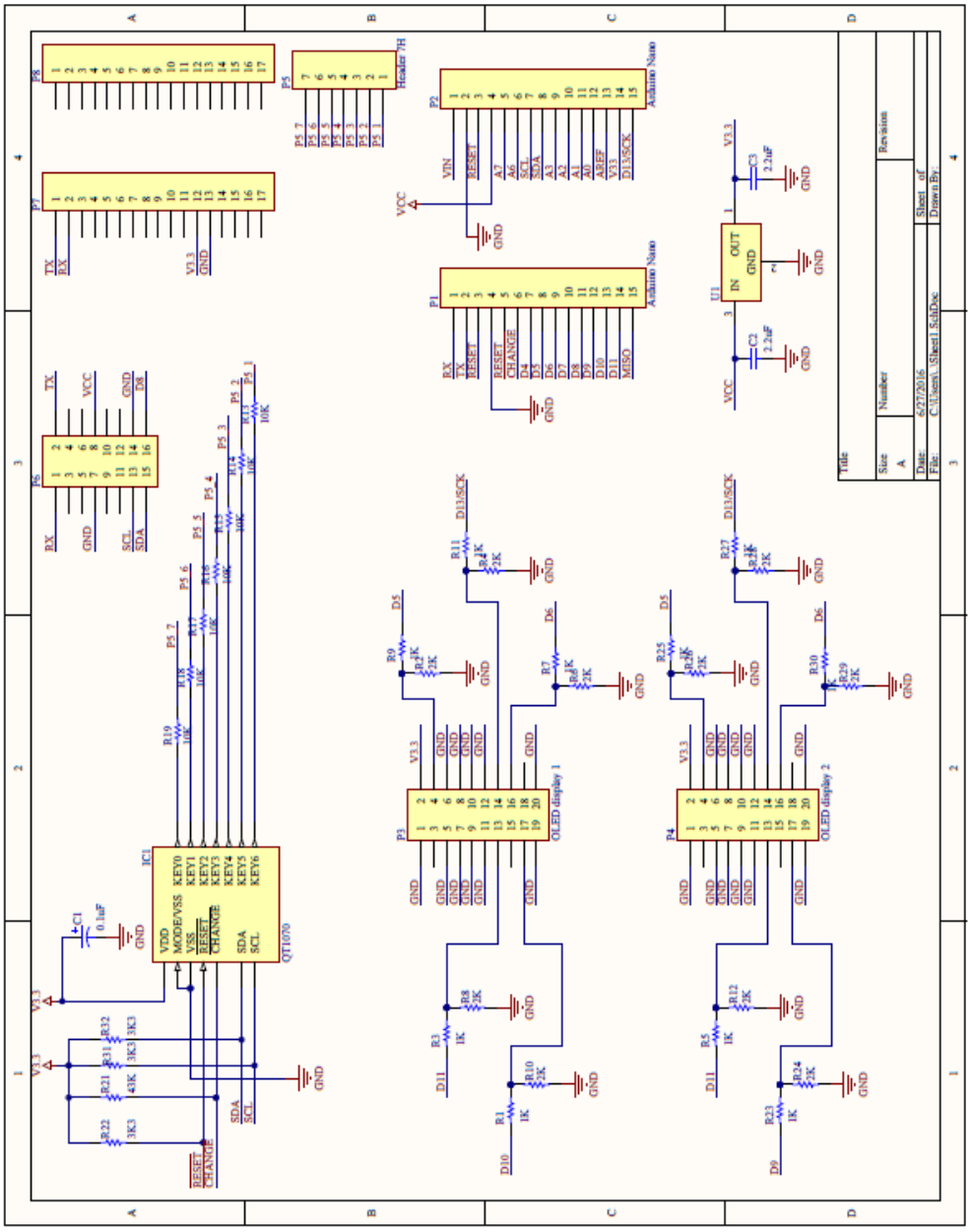
10. Summary

Interactive graphic interface for electric vehicle

The result of this thesis was interactive graphic interface with appropriate board and mobile app for electric vehicle. The displays of the graphic interface show graphic and numeric indicators of the battery state, speed, load and the state of all battery cells. Keyboard enables entrance to the menu, ability to select one of the menu entries and return to the default display state. Mobile app allows monitoring and storing of several different informations about the ride. Some of them are total duration of the rides, total distance covered, average and maximal achieved speed while driving as well as the tracking of vehicle's location and the change of it while displaying it on the map. Mobile app was built using PhoneGap CLI, and all electrical schemes were made in software package Altium Designer 2014.

Keywords: interactive graphic interface, electric vehicle, OLED display, capacitive touch keyboard, Arduino Nano, QT1070, SEPS525, Bluetooth, mobile app, Adobe PhoneGap

11. Privitak



Slika 11.1 Električna shema interaktivnog grafičkog sučelja

Isječak programskog koda iz gui.ino

```
int slave = 1; //selekcija ekrana

SEPS525_OLED disp;
AtTouch touch;
SoftwareSerial BTSerial(8, 7); // RX, TX

const uint8_t RFC = 2; //digitalni pin za komunikaciju s
vozilom
const uint8_t disp_addr = 0x42;
const uint8_t id_code[] = {0xDE, 0xAD, 0xBE, 0xEF};

//registri ekrana na adresi 0x42
typedef enum {
  NONE = 0,
  ID,
  STAT,
  PARAMS,
  CELLS,
  REG_END
} register_t;

//dodatni podaci
typedef enum {
  DETAIL_CELLS = 0x01,
  DETAIL_PARAMS = 0x02
} extra_t;

volatile uint32_t msg_last = 0;
volatile register_t i2c_reg = NONE;

//provjera kontrolne sume
uint8_t checksum(const uint8_t *data, const int n) {
  uint8_t cs = 0;
  for (int i = 0; i < n; i++) {
    cs ^= data[i];
  }
  return cs;
}

uint8_t bat = 0; //stanje baterije
uint8_t spd = 0; //stanje brzine
uint8_t extra = 0; //dodatni podaci
uint8_t cells[12] = {0}; //stanje baterijskih ćelija
```

```
//slanje podataka vozilu
void i2c_tx() {
  if (i2c_reg == NONE) {
    dbg("ERR: register not selected\n");
    return;
  }
  // return data
  dbg("tx reg: ");
  dbg((uint8_t)i2c_reg);
  dbg('\n');
  if (i2c_reg == ID) {
    uint8_t d[] = {0x04, id_code[0], id_code[1], id_code[2],
id_code[3], 0};
    d[5] = checksum(d, 5);
    Wire.write(d, 6);
    dbg("return ID: ");
    dbg(d, 6);
  }
  else if (i2c_reg == STAT) {
    uint8_t d[] = {0x02, mod, extra, 0};
    d[3] = checksum(d, 3);
    Wire.write(d, 4);
    dbg("return STATUS: ");
    dbg(d, 4);
    if (extra) {
      dbg("Extra");
      dbg("    (requesting extra info: ");
      if (extra & DETAIL_CELLS) dbg("DETAIL_CELLS)");
      if (extra & DETAIL_PARAMS) dbg("DETAIL_PARAMS)");
    }
  }
  else if (i2c_reg == PARAMS) {
    // TODO
    dbg("TODO: params");
  }
  dbg('\n');
}

//primanje podataka s vozila
void i2c_rx(int n) {
  digitalWrite(RFC, LOW);
  uint8_t data[n];
  int cnt = 0;
  dbg("received: ");
  while (cnt < n) {
    data[cnt] = Wire.read();
```

```
    cnt++;
  }
  dbg(data, n);
  dbg('\n');

  const uint8_t len = data[0];
  const uint8_t reg = data[1];

  if (checksum(data, n) == 0) {
    dbg("checksum ok\n");

    if (reg <= NONE || reg >= REG_END) {
      dbg("ERR: invalid register:");
      dbg(reg);
      dbg('\n');
      i2c_reg = NONE;
      return;
    }
    i2c_reg = (register_t)reg;
    if (reg == STAT) {
      spd = data[2];
      bat = data[3];
      mod = data[4];
    }
    else if (reg == CELLS) {
      for (int i = 0; i < 12; i++) {
        cells[i] = data[i + 1];
      }
      dbg("cells: ");
      dbg(cells, 12);
      dbg("\n");
    }
    msg_last = millis();
  }
  else {
    dbg("ERR: checksum fail\n");
    i2c_reg = NONE;
  }
}

//prikaz podataka primljenih s aplikacije
void stats() {
  sett = false;
  slave = 2;
  disp.fillScreen(BLACK);
  disp.drawRect(0, 42, 160, 40, WHITE);
  disp.fillRect(0, 84, 160, 40, GREEN);
```

```
  disp.drawRect(0, 0, 160, 40, WHITE);
  disp.setTextSize(2);
  disp.setCursor(40, 95);
  disp.print("Status");
  slave = 1;
  disp.fillScreen(BLACK);
  disp.setTextSize(2);
  disp.setCursor(25, 0);
  disp.print("Udaljenost");
  disp.setCursor(45, 25);
  for (int i = 0; i < 6; i++) {
    disp.print(dist[i]);
  }
  disp.setTextSize(2);
  disp.setCursor(40, 55);
  disp.print("Vrijeme");
  disp.setCursor(35, 75);
  for (int i = 0; i < 8; i++) {
    disp.print(tim[i]);
  }
}

//prikaz stanja baterijskih ćelija
void cell_state() {
  extra = 1; //signaliziranje vozilu da se žele vidjeti ćelije
  digitalWrite(RFC, HIGH);
  sett = false;
  slave = 2;
  disp.fillScreen(BLACK);
  disp.setCursor(20, 20);
  disp.setTextColor(WHITE);
  disp.setTextSize(3);
  disp.println("Stanje");
  disp.setCursor(10, 50);
  disp.println("baterije");
  disp.setCursor(50, 100);
  disp.setTextColor(WHITE);
  disp.setTextSize(3);
  disp.println(bat);
  disp.setCursor(100, 100);
  disp.setTextColor(WHITE);
  disp.setTextSize(3);
  disp.println("%");
  slave = 1;
  int k = 0;
  disp.fillScreen(BLACK);
//iscrtavanje grafičkih indikatora za svaku ćeliju
```

```
  disp.drawRect(1, 21, 19, 39, DARK_GRAY);
  disp.drawRect(27, 21, 19, 39, DARK_GRAY);
  disp.drawRect(53, 21, 19, 39, DARK_GRAY);
  disp.drawRect(79, 21, 19, 39, DARK_GRAY);
  disp.drawRect(106, 21, 19, 39, DARK_GRAY);
  disp.drawRect(132, 21, 19, 39, DARK_GRAY);
  disp.drawRect(1, 66, 19, 40, DARK_GRAY);
  disp.drawRect(27, 66, 19, 40, DARK_GRAY);
  disp.drawRect(53, 66, 19, 40, DARK_GRAY);
  disp.drawRect(79, 66, 19, 40, DARK_GRAY);
  disp.drawRect(106, 66, 19, 40, DARK_GRAY);
  disp.drawRect(132, 66, 19, 40, DARK_GRAY);
//popunjavanje iscrtanih indikatora s obzirom na primljene
//vrijednosti
  for (int i = 0; i < 12; i++) {
    cell_level = map(cells[i], 0, 100, 35, 0);
    if (i == 6) {
      k = 0;
      disp.setCursor(3, 50);
    }
    if (i >= 6) {
      disp.fillRect2(3 + k, 68, 16, 36, GREEN);
      disp.fillRect2(3 + k, 68, 16, cell_level, BLACK);
      k = k + 26;
    }
    else {
      disp.fillRect2(3 + k, 23, 16, 36, GREEN);
      disp.fillRect2(3 + k, 23, 16, cell_level, BLACK);
      k = k + 26;
    }
  }
//nije potrebno slati stanje ćelije osim kada se odabere
//odgovarajuća stavka izbornika
  extra = 0;
}

//prikaz opterećenja
void teret_test() {
  slave = 1;
  if (Battery) {
    Battery = false;
    draw_battery();
  }
//popunjavanje indikatora o opterećenju ovisno o njegovom
stanju
  if (mod == 0) {
    disp.fillRect(10, 170, 7, 7, GREEN);
```

```
    disp.fillRect(125, 170, 7, 7, BLACK);
    disp.fillRect(35, 170, teret_level, 7, BLACK);
  }
  else if (mod == 10) {
    disp.fillRect(10, 170, 7, 7, BLACK);
    disp.fillRect(125, 170, 7, 7, GREEN);
    disp.fillRect(35, 170, teret_level, 7, BLACK);
  }
  else if (mod > 0 && mod < 10) {
    disp.fillRect(10, 170, 7, 7, BLACK);
    disp.fillRect(125, 170, 7, 7, BLACK);
    disp.fillRect(35, 170, teret_level, 7, BLACK);
    teret_level = map(mod, 1, 9, 0, 75);
    disp.fillRect(35, 170, teret_level, 7, GREEN);
  }
}

//prikaz stanja baterije
void provjera_baterije() {
//resetiranje zastavica
  other = true;
  none = true;
  swipe_1 = false;
  swipe_2 = false;
  swipe_3 = false;
  sett = false;
  update = true;
//osvježavanje ekrana samo ako se nova i stara vrijednost
//razlikuju pa ih je potrebno izjednačiti
  bat2 = bat;
  check_other = millis(); //vrijeme zadnjeg ažuriranja
  slave = 1;
  if (Battery) {
    Battery = false;
    draw_battery();
  }
  disp.fillRect(7, 7, baterija_level, 52, BLACK);
  baterija_level = map(bat2, 0, 100, 0, 130);
  disp.fillRect(7, 7, baterija_level, 52, GREEN);
}

//prikaz trenutne brzine
void provjera_brzine() {
//resetiranje zastavica
  sett = false;
  slave = 2;
```

```
//osvježavanje ekrana samo ako se nova i stara vrijednost
//razlikuju pa ih je potrebno izjednačiti
  spd2 = spd;
  if (Speed) {
    Speed = false;
    draw_speed();
  }
  if (spd2 == 0)
    disp.fillTriangle(36, 118, 4, 4, 64, 4, BLACK);
  else {
    brzina_level = map(spd2, 0, 40, 36, 4); //tocka x0
    brzina_level2 = map(spd2, 0, 40, 36, 64); //tocka x1
    brzina_level3 = map(spd2, 0, 40, 116, 4); //tocka y0
    brzina_level4 = map(spd2, 0, 40, 116, 4); //tocka y1
    disp.fillTriangle(36, 118, 4, 4, 64, 4, BLACK);
    disp.fillTriangle(brzina_level, brzina_level3,
brzina_level2, brzina_level4, 36, 118, GREEN);
  }
  brzina_tekst();
}

//prikaz izbornika
void menu() {
//resetiranje zastavica
  other = false;
  Battery = true;
  Speed = true;
  sett = false;
  menu_draw = true;
  disp.fillScreen(BLACK);
  int i = 0;
  char* menuItems[] = {
    "Baterija",
    "Postavke",
    "Status",
    "",
  };
//ako se u izbornik ušlo s početnog ekrana aktivna zastavica je
//none; ako se u izbornik ušlo iz prve stavke aktivna je
swipe_1 //zastavica, ako iz druge swipe_2, a ako iz treće
swipe_3
//zastavice se resetiraju pri svakom povratku na početni prikaz
  if (none == true) {
    disp.drawRect(0, 42, 160, 40, WHITE);
    disp.drawRect(0, 84, 160, 40, WHITE);
    disp.drawRect(0, 0, 160, 40, WHITE);
  }
```

```
  else if (swipe_1 == true) {
    disp.drawRect(0, 42, 160, 40, WHITE);
    disp.drawRect(0, 84, 160, 40, WHITE);
    disp.fillRect(0, 0, 160, 40, GREEN);
  }
  else if (swipe_2 == true) {
    disp.fillRect(0, 42, 160, 40, GREEN);
    disp.drawRect(0, 84, 160, 40, WHITE);
    disp.drawRect(0, 0, 160, 40, WHITE);
  }
  else if (swipe_3 == true) {
    disp.drawRect(0, 42, 160, 40, WHITE);
    disp.fillRect(0, 84, 160, 40, GREEN);
    disp.drawRect(0, 0, 160, 40, WHITE);
  }
  for (i = 0; i < 3; i++) {
    disp.setTextSize(2);
    disp.setCursor(40, k);
    disp.print(menuItems[i]);
    k = k + 40;
  }
  i = 0;
  k = 15;
}

//inicijalizacija sustava
void setup() {
  touch.begin(3); //inicijalizacija ekrana
  delay(100);
  Wire.beginTransmission(0x1B);  //započni prijenos na QT1070
  Wire.write(0x37); //odabir registra 0x37
  Wire.write(4);
  Wire.endTransmission(); //završi prijenos
  delay(100);
  Wire.beginTransmission(0x1B);  //započni prijenos na QT1070
  Wire.write(0x21); //odabir registra 0x21
  Wire.write(10);
  Wire.endTransmission();//završi prijenos
  delay(100);
  Wire.beginTransmission(0x1B);  //započni prijenos na QT1070
  Wire.write(0x22); // odabir registra 0x22
  Wire.write(10);
  Wire.endTransmission();//završi prijenos
  delay(100);
  Wire.beginTransmission(0x1B);  //započni prijenos na QT1070
  Wire.write(0x23); // odabir registra 0x23
  Wire.write(10);
```

```
  Wire.endTransmission();//završi prijenos
  delay(100);
  Wire.beginTransmission(0x1B);  //započni prijenos na QT1070
  Wire.write(0x24); // odabir registra 0x24
  Wire.write(10);
  Wire.endTransmission();//završi prijenos
  delay(100);
  Wire.begin();
  Wire.beginTransmission(0x1B);  //započni prijenos na QT1070
  Wire.write(0x25); // odabir registra 0x25
  Wire.write(10);
  Wire.endTransmission();//završi prijenos
  delay(100);
  Wire.begin();
  Wire.beginTransmission(0x1B);  //započni prijenos na QT1070
  Wire.write(0x26); // odabir registra 0x26
  Wire.write(10);
  Wire.endTransmission();//završi prijenos
  delay(100);

  Wire.begin(disp_addr); //Arduino slave
  Wire.onRequest(i2c_tx);
  Wire.onReceive(i2c_rx);

  Serial.begin(115200);
  BTSerial.begin(9600);

  pinMode(RFC, OUTPUT);
  digitalWrite(RFC, LOW);

  randomSeed(analogRead(0));
  digitalWrite(RFC, HIGH);
//početno stanje ekrana
  slave = 1;
  disp.begin();
  disp.fillScreen(BLACK);
  provjera_baterije();
  teret_test();
  slave = 2;
  disp.fillScreen(BLACK);
  provjera_brzine();
}

void loop() {
//ako je CHANGE aktivan duže nego je potrebno resetiraj ga
  if (digitalRead(3) == 0) {
    if (millis() - change > 800) {
```

```
      change = millis();
      Wire.beginTransmission(0x1B); //započni prijenos na
QT1070
      Wire.write(0x02); //čitanje registra o statusu detekcije
      Wire.endTransmission();      //završi prijenos
      Wire.requestFrom(0x1B, 1);    //zatraži 1 bajt
      delay(1);
      if (Wire.available() == 1)
        int k = Wire.read();
      Wire.beginTransmission(0x1B); //započni prijenos na
QT1070
      Wire.write(0x03); //čitanje registra o stanju tipki
      Wire.endTransmission();      //završi prijenos
      Wire.requestFrom(0x1B, 1);    //zatraži 1 bajt
      delay(1);
      if (Wire.available() == 1)
        int l = Wire.read();
    }
  }
//ako je RFC signal aktivan duže od potrebnog resetiraj ga
  if (digitalRead(2) == 1) {
    if (millis() - rfc_t > 2000) {
      rfc_t = millis();
      digitalWrite(2, LOW);
    }
  }
//svake 2 sekunde zatraži podatke od vozila
  if (millis() - up > 2000) {
    up = millis();
    digitalWrite(RFC, HIGH);
  }
//čitanje podataka sa Bluetootha
  while (BTSerial.available()) {
    char c = BTSerial.read();
    if (c == 'e') { //kraj prijenosa
      sendData(); //obrada podataka
    } else {
      bufferData(c); //primaju se podaci
    }
  }
//osvježavanje ekrana ako su stara i nova vrijednost stanja
//baterije i brzine različite
  if (other == true) {
    if (bat2 != bat) {
      provjera_baterije();
    }
    if (spd2 != spd) {
```

```
      provjera_brzine();
    }
  }
//provjera je li se dogodila promjena stanja neke od tipki
  if (touch.hit()) {
    key = touch.readActiveKey(); //čitanje aktivne tipke
    if (key != 1 && key != 0) {
      time_t = millis(); //vrijeme dodira u milisekundama
      key_value[end_key] = key; //polje sa dodirnutim tipkama
      key_time[end_time] = time_t; //polje sa vremenima dodira
      check = true;
      end_key = (end_key + 1);
      end_time = (end_time + 1);
    }
  }
//ukoliko je prošlo 400 milisekundi od prethodnog dodira pozovi
//funkciju koja će se provjeriti o kakvoj vrsti dodira je riječ
  else if (millis() - time_t > 400) {
    if (check) {
      int m = check_event(); //klasifikacija dodira
      update_display(m); //osvježavanje ekrana
    }
  }
//ako prođe 10 sekundi bez detektirane promjene stanja prikaži
//bateriju, brzinu i opterećenje
  if (refresh == true) {
    unsigned long now = millis();
    if (now - check_menu >= 10000) {
      slave = 1;
      teret_test();
      provjera_brzine();
      slave = 2;
      provjera_baterije();
      refresh = false;
      menu_draw = false;
    }
  }
//slanje podataka mobilnoj aplikaciji
  unsigned long send_data = millis();
  if (send_data - time_0 >= 1000) {
    time_0 = send_data;
    BTSerial.print(spd);
    BTSerial.print('y');
    BTSerial.println(mod);
  }
}
```

```
//obrada podataka primljenih s aplikacije
void sendData() {
  int index = 0;
  int data = 0;
  for (int i = 0; i < len - 1; i++) {
    if (buf[i] == 'q' || buf[i] == 'w') {
      data = i;
    }
    if (buf[i] == 'z') {
      index = i;
    }
  }
//podaci primljeni nakon 'q' a prije 'z' označavaju prijeđenu
//udaljenost; nakon 'z' slijedi podatak o ukupnom vremenu
  if (buf[data] == 'q') {
    int j = 0;
    for (int i = data + 1; i < index; i++) {
      dist[j] = buf[i];
      j++;
    }
    int k = 0;
    for (int i = index + 1; i < index + 8; i++) {
      tim[k] = buf[i];
      k++;
    }
  }
//podatak poslan uz 'w' označava primljeno opterećenje
  if (buf[data] == 'w') {
    if (buf[data + 2] == '0') {
      mod = 10;
      if (other == true)
        teret_test();
    }
    else {
      mod = buf[data + 1] - '0';
      if (other == true) {
        teret_test();
      }
    }
    buf[2] = NULL;
  }
  len = 0;
}

//klasifikacija detektiranih dodira
int check_event() {
  int n = 0;
```

```
//dva dodira unutar 700 milisekundi označavaju dvostruki dodir
  if (end_time == 2) {
    if (key_time[(end_time - 1)] - key_time[(end_time - 2)] <
700) {
      n = 2;
      clear_array();
    }
  }
  else if (end_time >= 3) {
    if (key_time[(end_time - 1)] - key_time[(end_time - 3)] <
900) {
//klizanje prema većoj vrijednosti tipke
      if (key_value[(end_key - 1)] < key_value[(end_key - 2)])
{
        n=3;
        clear_array();
      }
//klizanje prema manjoj vrijednosti tipke
      else if (key_value[(end_key - 1)] > key_value[(end_key -
2)]) {
        n=4;
        clear_array();
      }
    }
  }
//vrati detektiranu vrstu dodira
  if (n != 0) {
    clear_array();
    check = false;
    return n;
  }
//ako nije prepoznata vrsta dodira vrati -1
  else {
    clear_array();
    check = false;
    return -1;
  }
}

//osvježavanje ekrana s obzirom na stanje tipkovnice
void update_display(int state) {
//dvostruki dodir
  if (state == 2) {
//ako je trenutno na ekranima izbornik prikaži bateriju i
brzinu
    if (menu_draw == true) {
      provjera_baterije();
```

```
      teret_test();
      provjera_brzine();
      menu_draw = false;
    }
    else {
      refresh = true;
      check_menu = millis();
      izbornik();
      slave = 1;
      menu();
    }
  }
//klizanje
  else if (state == 3) {
    refresh = false;
//ako je na ekranima izbornik ovisno o stanju zastavica
//none,swipe_1/2/3 prikaži neku od stavki
    if (menu_draw == true) {
      if (none == true) {
        cell_state();
        swipe_1 = true;
        none = false;
        extra = 0;
        menu_draw = false;
      }
      else if (swipe_1 == true) {
        settings();
        swipe_2 = true;
        swipe_1 = false;
        menu_draw = false;
      }
      else if (swipe_2 == true) {
        stats();
        swipe_3 = true;
        swipe_2 = false;
        menu_draw = false;
      }
      else if (swipe_3 == true) {
        cell_state();
        swipe_1 = true;
        swipe_3 = false;
        menu_draw = false;
      }
    }
//inače povećaj vrijednost opterećenja
    else if (menu_draw == false) {
      if (mod < 10)
```

```
        mod = mod + 1;
      else
        mod = mod;
        teret_test();
    }
  }
//klizanje u suprotnom smjeru
  else if (state == 4) {
    refresh = false;
    if (menu_draw == true) {
      if (none == true) {
        stats();
        swipe_3 = true;
        none = false;
        menu_draw = false;
      }
      else if (swipe_1 == true) {
        stats();
        swipe_3 = true;
        swipe_1 = false;
        menu_draw = false;
      }
      else if (swipe_2 == true) {
        cell_state();
        swipe_1 = true;
        swipe_2 = false;
        menu_draw = false;
      }
      else if (swipe_3 == true) {
        settings();
        swipe_2 = true;
        swipe_3 = false;
        menu_draw = false;
      }
    }
    else if (menu_draw == false) {
      if (mod > 0)
        mod = mod - 1;
      else
        mod = mod;
      teret_test();
    }
  }
}
```