Robotska kolica za rješavanje labirinta temeljena na Arduino platformi

Ban, Arian

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Sveučilišni prijediplomski studij elektrotehnike

Završni rad

ROBOTSKA KOLICA ZA RJEŠAVANJE LABIRINTA TEMELJENA NA ARDUINO PLATFORMI

Rijeka, rujan 2024.

Arian Ban

0069092443

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Sveučilišni prijediplomski studij elektrotehnike

Završni rad

ROBOTSKA KOLICA ZA RJEŠAVANJE LABIRINTA TEMELJENA NA ARDUINO PLATFORMI

Mentor: izv. prof. dr. sc. Ivan Volarić

Rijeka, rujan 2024.

Arian Ban

0069092443

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 12.03.2024.

Zavod: Zavod za automatiku i elektroniku

Predmet: Digitalna elektronika

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Arian Ban (0069092443)**

Studij: Sveučilišni prijediplomski studij elektrotehnike (1030)

Zadatak: **Robotska kolica za rješavanje labirinta temeljena na Arduino platformi / Arduino based maze solving robotic vehicle**

Opis zadatka:

U sklopu završnog rada potrebno je izraditi robotska kolica s pogonom na sva četiri kotača, tj. s četiri DC motora. Robotska kolica moraju biti opremljena s tri ultrazvučna senzora udaljenosti orijentirana lijevo, desno, te ispred kolica. Tri senzora, te četiri H-mosta za upravljanje motorima potrebno je spojiti na Arduino razvojnu pločicu, te napisati programsku podršku koja će omogućiti kolicima samostalno rješavanje labirinta.

Rad mora biti napisan prema Uputama za pisanja diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 20.03.2024.

Mentor:
doc. dr. sc. Ivan Volarić

Predsjednik povjerenstva za
završni ispit:
prof. dr. sc. Dubravko Franković

IZJAVA

Sukladno članku 7. Pravilnika o završnom radu, završnom ispitu i završetku sveučilišnih prijediplomskih studija Tehničkog fakulteta, Sveučilišta u Rijeci, izjavljujem da sam samostalno izradio završni rad prema zadatku koji mi je uručen 20. ožujka 2024.

Rijeka, rujan 2024.

Arian Ban

0069092443

SADRŽAJ

1. UVOD

Roboti za rješavanje labirinta, čija je osnovna zadaća kretati se kroz složena okruženja te pronaći optimalnu putanju, predstavljaju ne samo fascinantan izazov u robotici, već i temelj za niz praktičnih primjena. Ovi roboti, opremljeni senzorima i algoritmima, mogu se prilagoditi nepoznatim okruženjima, što ih čini korisnima u raznim područjima. Jedna od glavnih primjena robota za rješavanje labirinata je u operacijama traganja i spašavanja. U scenarijima prirodnih nepogoda, kao što je potres ili urušavanje zgrada, ovi roboti mogu prolaziti kroz ljudima nedostupna područja kako bi locirali preživjele. Njihova sposobnost da autonomno istražuju nepoznata okruženja i izbjegavaju prepreke ključna je u situacijama gdje spasioci neće moći sigurno pristupiti određenim područjima. Mapiranjem terena i identifikacijom putanja, ovi roboti mogu spasiocima pružiti vitalne informacije o okruženju, čime se poboljšava učinkovitost i sigurnost spasilačkih misija. Još jedna značajna primjena je u industrijskoj automatizaciji. U skladištima i tvornicama, roboti za rješavanje labirinata mogu se koristiti za optimizaciju kretanja robe i materijala. Ovi roboti se mogu kretati kroz složene topologije skladišta ili tvornice kako bi transportirali predmete s jedne lokacije na drugu, izbjegavajući prepreke i pronalazeći najučinkovitije rute. Ovo ne samo da povećava produktivnost, već i smanjuje rizik od nezgoda, budući da roboti mogu djelovati u okruženjima u kojima bi ljudskim radnicima moglo biti teško ili opasno raditi. Roboti za rješavanje labirinata također igraju ulogu u praćenju okoliša i istraživanju. U opasnim ili nepristupačnim područjima, kao što su nuklearna postrojenja, kemijske tvornice ili dubokomorska okruženja, ovi roboti se koriste za prikupljanje podataka i provođenje raznih inspekcija. Njihova sposobnost kretanja kroz zamršena okruženja omogućuje im pristup područjima koja bi za ljude bila opasna ili nemoguća za pristup. Ova primjena je od posebne važnosti u održavanju sigurnosnih standarda i sprječavanju ekoloških katastrofa. Štoviše, roboti za rješavanje labirinata ključni su u napretku umjetne inteligencije i robotike. Algoritmi i tehnologije razvijeni za ove robote, poput pronalaženja putanja, integracije senzora i donošenja odluka, imaju šire implikacije za autonomne sustave. Ove inovacije doprinose razvoju samovozećih automobila, dronova i drugih autonomnih vozila, gdje je navigacija kroz složena i dinamična okruženja neophodna. U sklopu ovog završnog rada izrađen je jednostavan primjer robotskih kolica čiji je osnovni zadatak pronalazak izlaza iz labirinta. Odabir putanje kojom će se kolica kretati vrši se mjerenjem udaljenosti lijevo, desno i ispred samih kolica. Navedeno mjerenje udaljenosti ostvareno je s tri ultrazvučna senzora udaljenosti orijentirana lijevo, desno, te ispred kolica. Kretanje kolica kroz sam labirint omogućeno je korištenjem četiri istosmjerna elektromotora. Obrada podataka dobivenih iz senzora udaljenosti i upravljanje

elektromotorima ostvareno je korištenjem Arduino Uno razvojne pločice uzimajući u obzir da je za preciznu kontrolu elektromotora (reverziranje, brzina vrtnje, itd.) korišten *motor shield* s integriranim krugom L293D koji se sastoji od četiri H-mosta.

2. ARDUINO

Arduino je talijanska tvrtka (slika 2.1) koja se bavi *open-source* hardverom i softverom i korisničkom zajednicom koja dizajnira i proizvodi razvojne pločice s mikrokontrolerom za izradu digitalnih uređaja. Sam cilj tvrtke jest taj da bi proces stvaranja interaktivnih projekata učinio pristupačnijim umjetnicima, dizajnerima, hobistima i svima koji su zainteresirani za stvaranje interaktivnih objekata ili okruženja. Sam Arduino se sastoji i od fizičke razvojne pločice s mikrokontrolerom i softverskog dijela (sučelja) Arduino IDE (eng. *Integrated Development Environment*) koji se koristi za za pisanje i učitavanje računalnog koda na mikrokontroler.

Slika 2.1. Logo tvrtke Arduino [1].

Jedna od najvažnijih prednosti Arduina, kao što je već navedeno, jest upravo njihov pristup otvorenog koda što znači da su hardverski dizajni, softver i dokumentacija besplatno dostupni svima za korištenje, modificiranje i dijeljenje, što je potaknulo veliku, zajednicu korisnika koja kontinuirano pridonosi rastu i poboljšanju platforme. Također Arduino dizajnira i proizvodi niz razvojnih pločica, koje često služe kao mozak raznih digitalnih uređaja, sposobnih osjetiti i kontrolirati objekte u fizičkom svijetu. Pločice su opremljene ulazno/izlaznim *pin*-ovima koji se mogu spojiti na senzore, motore, LED i druge komponente, što ih čini vrlo svestranim za širok raspon projekata. Postoji nekolicina Arduino razvojnih pločica od kojih su neke najpopularnije UNO, Micro, Leonardo, Mega. Due, MKR, itd. Tvrtka također nudi vlastiti IDE koji korisnicima omogućuje pisanje, učitavanje i otklanjanje pogrešaka koda na Arduino pločicama. Programski jezik Arduino temelji se na C/C++, što ga čini dostupnim i početnicima i iskusnim programerima. Još

jedna bitna značajka Arduina po čemu su poznati je njihov fokus na obrazovanje. Tvrtka nudi opsežne resurse, upute i setove dizajnirane da pomognu korisnicima svih razina vještina u učenju o elektronici, kodiranju i robotici. Jedan od najjačih aspekata Arduina je njegova globalna zajednica korisnika i programera. Ova zajednica doprinosi bibliotekama, štitovima (dodatnim pločicama) i projektnim idejama, olakšavajući drugima izgradnju složenih sustava s Arduinom.

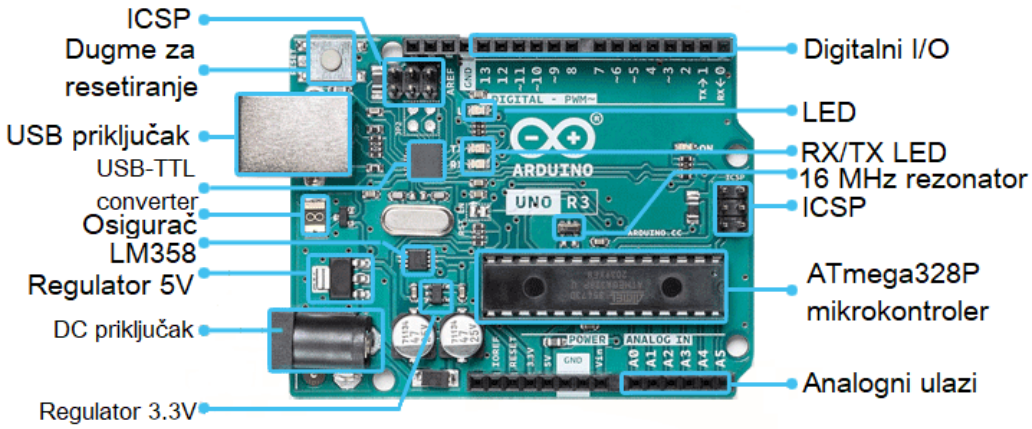
2.1. Arduino Uno

Arduino UNO, prikazan na slici 2.2, jedna je od najpopularnijih i najčešće korištenih razvojnih pločica u Arduino obitelji. Dizajniran je za početnike i iskusne programere, nudeći ravnotežu jednostavnosti, funkcionalnosti i pristupačnosti. Nazvan je kao UNO za oznaku prvog izdanja Arduino softvera. To je također bila prva USB pločica koju je proizveo Arduino. Arduino UNO temelji se na ATmega328P mikrokontroleru. Jednostavna je za korištenje u usporedbi s drugim pločicama, kao što je Arduino Mega, itd. Pločica se sastoji od 14 digitalnih ulazno/izlazno *pin*-ova, 6 analognih *pin*-ova, USB konektora, konektora za napajanje i ICSP (eng. *In-Circuit Serial Programming*) priključka. Može raditi i na online i offline platformama.

Slika 2.2. Izgled Arduino UNO pločice [2].

U ovom završnom radu korišten je upravo Arduino UNO zbog njegovih mnogih prednosti. Sam Arduino UNO sastoji se od nekoliko komponenti prikazanih na slici 2.3:

- Digitalni *pin*-ovi 2-13
- Digitalni *pin*-ovi 0-1 (Tx i Rx pinovi serijske UART komunikacije preko kojih se između ostalog može programirati mikrokontroler)
- Tipkalo za resetiranje
- ICSP konektor
- Analogni ulazi A0-A5
- *Pin*-ovi za napajanje i uzemljenje
- Ulaz za vanjsko napajanje (9-12 V DC)
- USB (univerzalna serijska sabirnica)
- Kristalni oscilator
- Regulator napona LM358



Slika 2.3. Komponente Arduino UNO pločice [3].

Mikrokontroler koji se koristi na Arduinu je, prije spomenuti, ATmega328P kojeg proizvodi tvrtka ATMEL. Mikrokontroler ATmega328P ima 14 digitalnih I/O *pin*-ova. Od 14 *pin*-ova, 6 daje PWM (eng. *Pulse Width Modulation*) izlaz tj. signale s pulsno širinskom modulacijom. Može imati 6 (DIP (eng. *Dual In-line Package*) kućište) ili 8 (SMD (eng. *Surface Mount Device*) kućište) analognih ulaznih *pin*-ova. Maksimalna izlazna struja svakog I/O *pin*-a je oko 40 mA. Ima flash memoriju od 32 KB, SRAM (eng. *Static Random-Access Memory*) od 2 KB te EEPROM (eng. *Electrically Erasable Programmable Read-Only Memory*) od 1 KB. Na razvojnoj pločici, u neposrednoj blizini mikrokontrolera nalazi se kristalni oscilator od 16 MHz koji mikrokontroleru daje signal radnog takta (eng. *clock*).

USB priključak omogućuje povezivanje pločice s računalom. Neophodan je za programiranje Arduino UNO pločice te se može koristiti i za napajanje same pločice.

ICSP priključci omogućuju korisniku programiranje dva mikrokontrolera na pločici. Na slici 2.3 gornji ICPS priključak služi za programiranje ATMega16U mikrokontrolera koji se nalazi malo ispod konektora, te služi za pretvaranje USB komunikacijskog protokola u UART protokol, a koji se koristi za programiranje glavnog mikrokontrolera. Glavni mikrokontroler, ATMega328P, osim preko USB priključka moguće je i direktno programirati preko desnog ICSP priključka. Uobičajeno se koristi za snimanje bootloadera na mikrokontroler, malog programa koji omogućuje programiranje mikrokontrolera preko USB priključka.

Reset tipkalo koristi se, kao što ime sugerira, za resetiranje ATMega328 mikrokontrolera. Spojen je na PC6/Reset *pin* s 10kΩ *pull-up* otpornikom. Kada se tipkalo pritisne, *pin* se spaja na uzemljenje što uzrokuje resetiranje mikrokontrolera.

Dva regulatora napona (5V i 3.3V) služe za regulaciju i stabilizaciju ulaznog napona. Arduino možemo napajati naponom od 7-12 V, a najveći dozvoljeni napon Atmega328P je 5 V te većina senzora i perifernih uređaja kojih spajamo na Arduino rade na ili 5 ili 3.3 V.

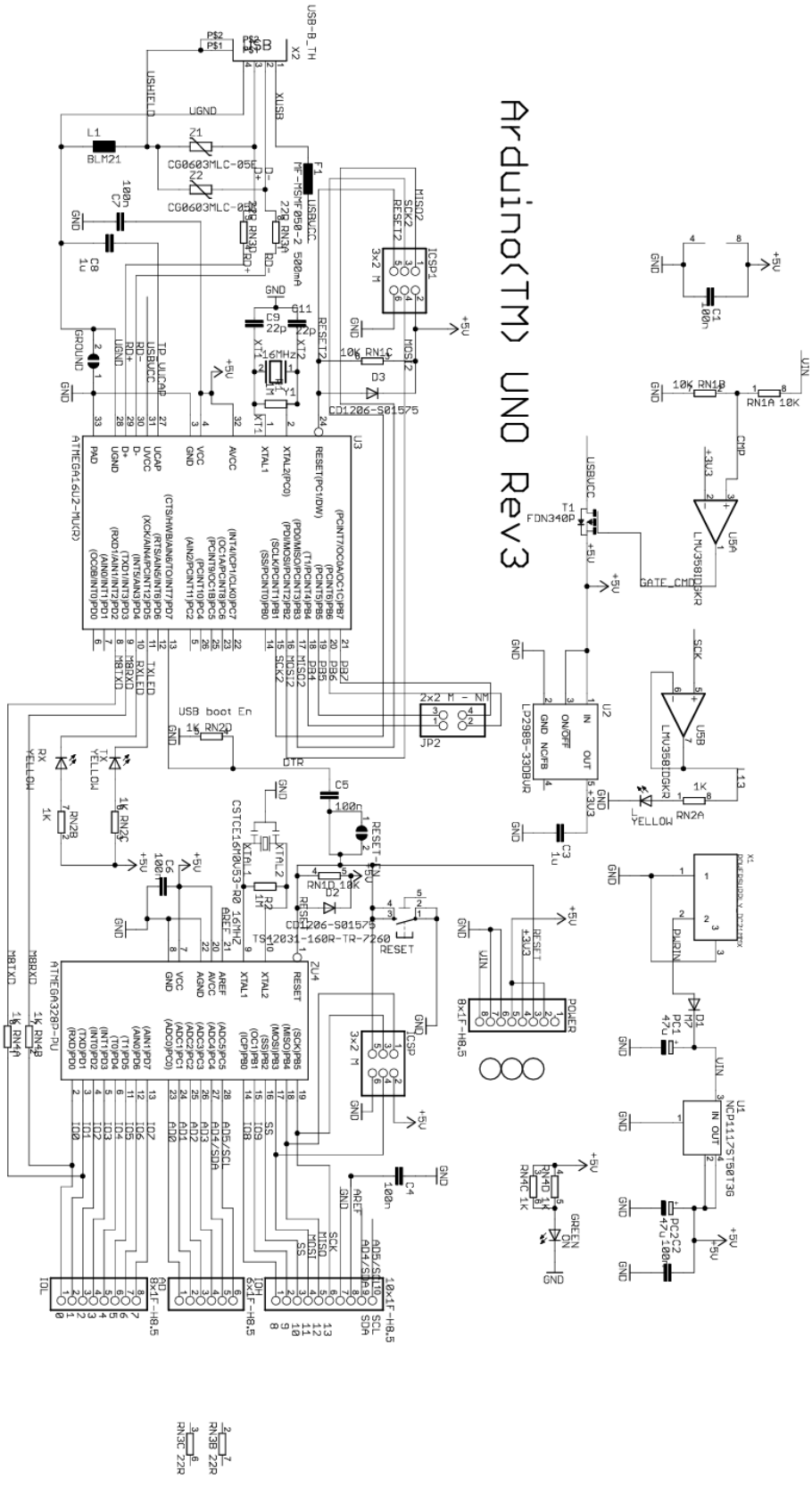
Pin-ovi 0-13 su digitalni. Njihova značajka jest ta da mogu poprimati vrijednost 1 (HIGH) i 0 (LOW). Kod UNO-a jedinicu predstavlja napon 5 V dok nulu predstavlja napon 0 V.

Arduino UNO također ima 6 ulaznih analognih *pin*-ova označenih A0-A5 spojenih na 10-bitni analogno-digitalni pretvornik.

LED indikator napajanja - kada svijetli pokazuje da je napajanje aktivirano. Kada je napajanje isključeno, LED neće svijetliti.

Operacijsko pojačalo LM358 koristi se kao komparator za kontrolu izvora ulaznog napajanja. Ukoliko je spojeni vanjski izvor napajanja preko konektora za napajanje, napajanje iz USB priključka se isključuje, čime se vrši njegova zaštita.

Na slici 2.4 prikazana je električna shema Arduino UNO pločice.

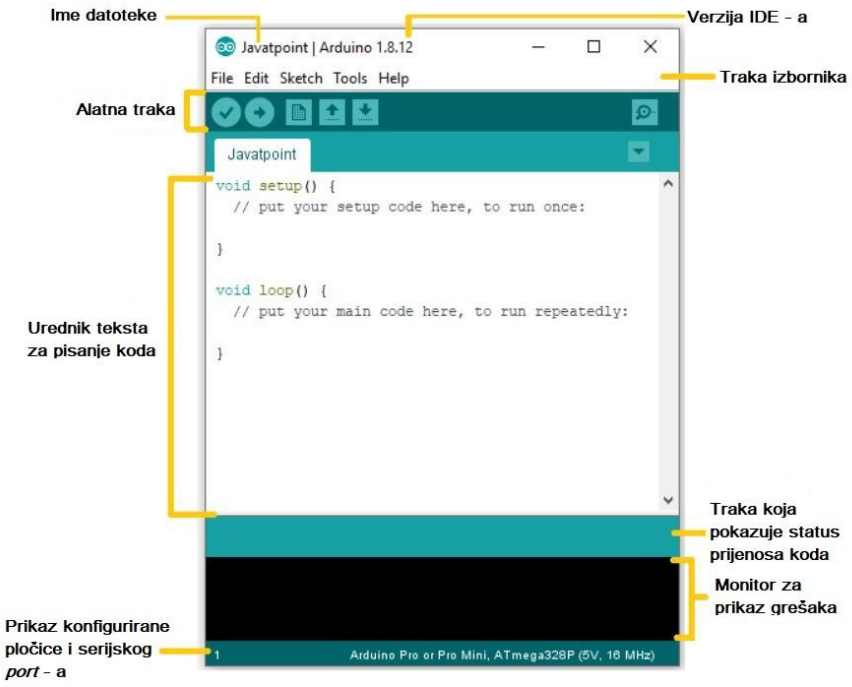


Slika 2.4. Električna shema Arduino UNO pločice [4].

2.2. Arduino IDE

Arduino IDE ključan alat je koji pruža jednostavnu platformu za pisanje, kompajliranje i učitavanje koda. U svojoj srži, Arduino IDE je pojednostavljeno programsko okruženje dizajnirano da olakša razvoj softvera za Arduino pločice. Podržava C i C++ programske jezike sa specifičnim bibliotekama koje su lako dostupne, omogućujući korisnicima kontrolu hardverskih komponenti poput LED, motora i senzora. Jednostavnost IDE-a jedna je od njegovih najvećih prednosti. Opremljen je s *code editor*-om koji ima osnovne funkcije poput označavanja sintakse, automatskog uvlačenja i provjere pogrešaka. To olakšava korisnicima pisanje i uklanjanje pogrešaka u kodu. Dodatno, *editor* uključuje serijski monitor, koristan alat za vizualizaciju podataka u stvarnom vremenu i komunikaciju s Arduino pločicom, što je bitno za testiranje i otklanjanje pogrešaka.

Na slici 2.5 možemo vidjeti izgled IDE sučelja kada otvorimo novi projekt. Novi projekt se sastoji od dvije prazne funkcije: *void setup()* i *void loop()*. Prilikom pokretanja programa, nakon deklaracije i incijalizacije globlanih varijabli, prvo će se pozvati *void setup()* funkcija, te će se kod napisan unutar nje izvršiti. Nakon toga, funkcija *void setup()* završava, poziva se *void loop()* funkcija, te se izvršava kod napisan unutar nje. Kada funkcija *void loop()* završi program se vraća na opet na vrh *void loop()* funkcije te ponovo izvršava naredbe. Ovo ponavljanje provodi se cijelo vrijeme dok je Arduino spojen na napajanje i u normalnom radu. U suštini kod unutar *void setup()* funkcije izvršit će se jednom, i samo jednom, na početku programa. Dok će se kod unutar *void loop()* funkcije izvršavati iznova i iznova, sve dok se ne isključi napajanje Arduino pločice ili ponovno pokrenemo Arduino program - pritiskom na tipku za resetiranje, učitavanjem nove skice ili ponovnim otvaranjem serijskog monitora.



Slika 2.5. Izgled Arduino IDE - a [5].

Još jedan vrlo bitan dio Arduino IDE-a su Arduino biblioteke (eng. *libraries*). Arduino biblioteke obično su mali paketi koda koje je netko drugi napisao za određenu svrhu, te omogućuju manipuliranje podacima i rad s hardverom. Jedan primjer je biblioteka koja dodaje podršku za određeni senzor, kao što je HY-SRF05 (korišten u ovom završnom radu). Korištenje biblioteka znači da se ne mora poznavati senzor ili modul iznutra kako bi se povezao i koristio s Arduinom. Također, biblioteke su korisne zbog toga što korisnik ne mora pisati neki složeni kod ako već postoji biblioteka koja implementira često korištenu značajku (npr. matematičke funkcije kao što su vektorski i matrični izračuni). Arduino IDE dolazi s nekoliko ugrađenih korisnih biblioteki te se lako mogu pronaći nove biblioteke koje dodaju podršku za razne module, senzore, *shield*-ove, itd. Kako je svatko slobodan pisati vlastite Arduino biblioteke, nove biblioteke se također mogu preuzeti s različitih web stranica. U ovom završnom radu korištene su biblioteke <*AFMotor.h*>, koja služi za pojednostavljanje koda vezanog uz upravljanje elektromotorima, te <*NewPing.h*>, koja služi za korištenje ultrazvučnih senzora pri mjerenju udaljenosti.

3. ISTOSMJERNI MOTOR SA PRIJENOSOM

U ovom završnom radu za pogon samih robotskih kolica korištena su četiri istosmjerna motora s prijenosom. Kao što ime govori sami pogonski element sastoji se od dva dijela: istosmjerni motor te prijenos. Korišteni motor s prijenosom, prikazan na slici 3.1, ima veliki raspon ulaznog napajanja i dvostrano vratilo koje ga čini dobrim izborom za konstrukciju modela i prototipova, posebno zato što uz njega dolazi i kotač s gumama i plastičnim naplatkom. Dvostrano vratilo je korisna značajka jer se strana na kojoj nije kotač može iskoristiti za ugradnju enkodera, što omogućuje upravljanje brzinom i položajem.

Slika 3.1. Istosmjeri motor s prijenosom [6].

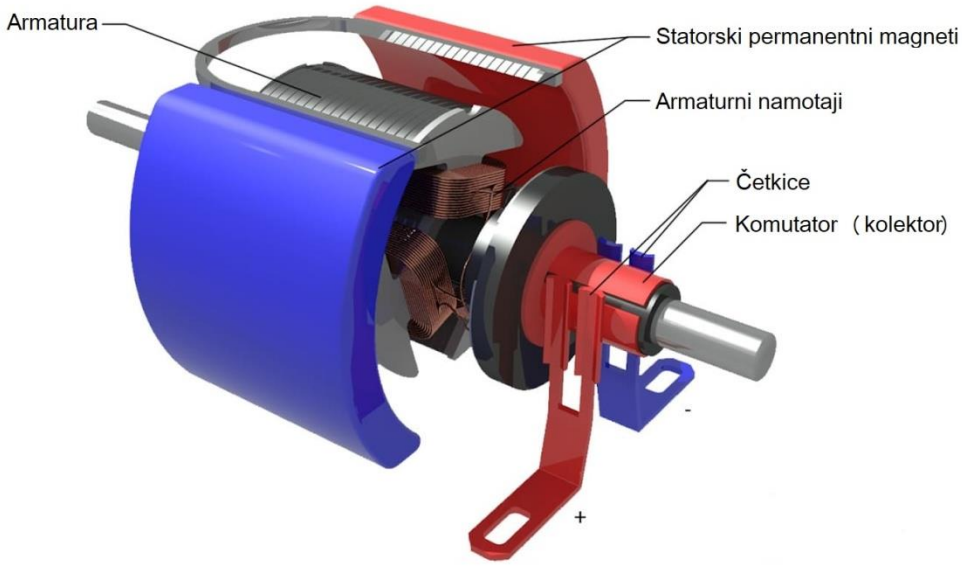
Tehnički podaci motora s prijenosom dani od proizvođača:

- vratilo 36 mm, dvostrano, s otvorom 1.9 mm
- napajanje: 3-9 V DC (preporučeno: 4.5 V)
- dimenzije (kotač): promjer = 65 mm; širina = 27 mm
- dimenzije (motor): 37.6 x 64.2 x 22.5 mm
- težina: 58 g

3.1. Istosmjerni motor

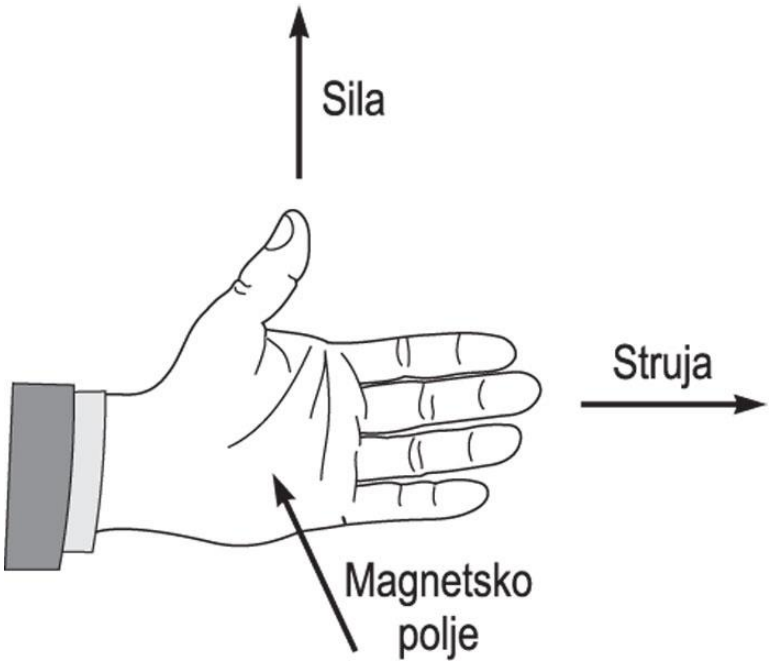
Istosmjerni motor je električni stroj koji pretvara istosmjernu električnu energiju u mehaničku. Gotovo sve vrste istosmjernih motora imaju neki unutarnji mehanizam, bilo elektromehanički ili elektronički, za povremenu promjenu smjera struje u rotoru motora. Kod klasičnih istosmjernih strojeva taj se sustav sastoji od četkica i kolektora.

Istosmjerni motor se sastoji od armature i armaturnih namotaja, statora i statorskih namotaja ili permanentnih magneta, četkica te komutatora odnosno kolektora. Raspored ovih dijelova vidljiv je na slici 3.2. Istosmjerni strojevi se najčešće izvode s istaknutim polovima na statoru i armaturnim namotajem na rotoru. Na polovima statora nalazi se uzbudni namot kroz koji teče istosmjerna struja i stvara uzbudni magnetski tok. Istosmjerni motori izrazito malenih snaga, kao što su korišteni u ovom završnom radu, umjesto uzbudnog namotaja na statoru imaju permanentne magnete. Na rotoru je armaturni namot - jednoliko raspoređen u utorima po obodu. Krajevi svakog svitka spojeni su na lamele kolektora po kojima klize fiksno montirane četkice.



Slika 3.2. Unutarnji prikaz istosmjernog stroja i njegovih dijelova [7].

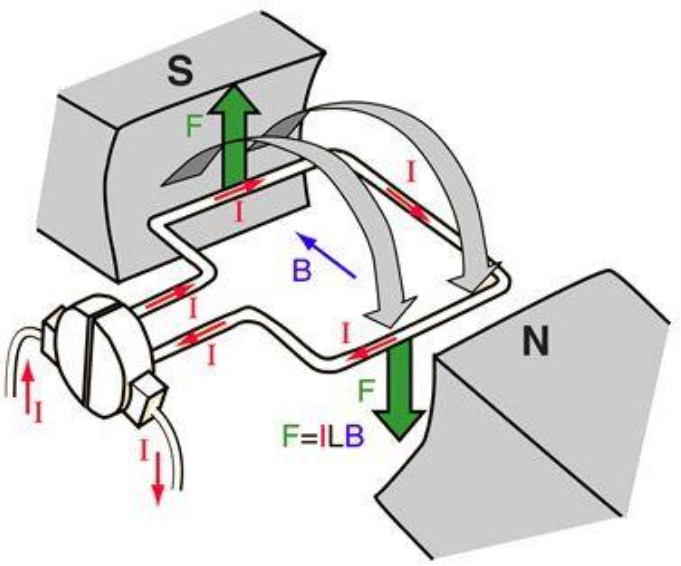
Princip rada ovih strojeva vrlo je sličan principu rada gotovo svih rotirajućih strojeva, a temelji se na osnovnim fizikalnim pojavama: da sila djeluje na električni vodič kada njime u magnetskom polju protječe električna struja. Uzbudni namotaji stvaraju glavno magnetsko polje koje je nepromjenljivo. Istosmjerna armaturna struja dovedena na motor se preko ugljenih četkica i kolektora komutira (pretvara u izmjeničnu) i prenosi na rotorske (armaturne) vodiče koji su smješteni u utorima rotorskog paketa dinamo limova. Na rotorske vodiče kojima teče izmjenična struja, a nalaze se u magnetskom polju statora djeluje sila koja zakreće rotor (pravilo lijeve ruke, slika 3.3).



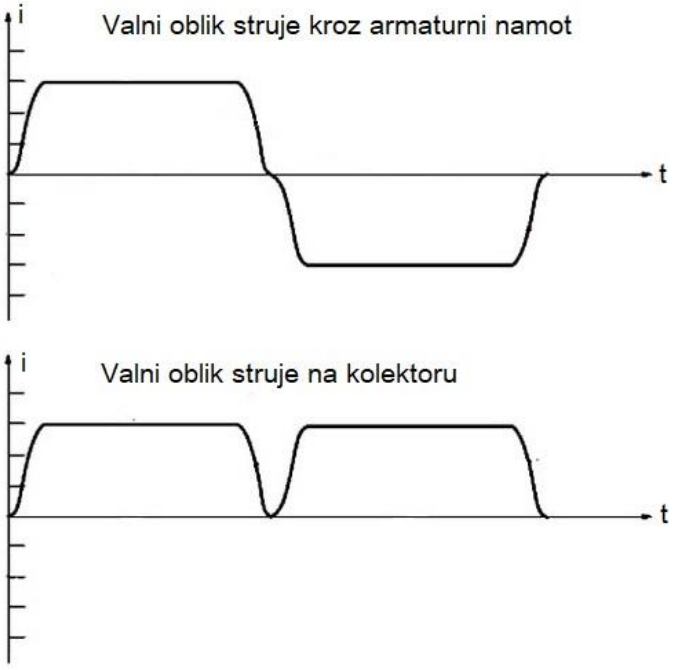
Slika 3.3. Pravilo lijeve ruke [8].

Kroz rotorske vodiče teče struja samo dok prolaze ispod glavnih polova jer se u tom trenutku i lamele na kolektoru preko kojih se napajaju nalaze ispod četkica. Rotorska struja mora biti izmjenična kako bi vodiči dok prolaze ispod sjevernog pola imali suprotan smjer struje nego dok prolaze ispod južnog pola jer se tako dobiva elektromagnetska sila suprotnog, a razvijeni moment istog smjera. Ta pojava izmjenične struje dobiva se upravo preko kolektora i četkica. Naime u jednom trenutku četkice će dodirivati svaka jedan kraj rotorskog namotaja te će struja kroz njega teći u jednom smjeru. Kako se pojavi sila rotor se rotira, a s njime i kolektor koji je spojen na krajeve armaturnog namota. Kako su četkice fiksne, odnosno ne miču se skupa sa rotorom u jednom

trenutku će doći do pojave da četkica koja je dodirivala jedan kraj namota počinje dodirivati drugi kraj, a četkica koja je dodirivala drugi kraj sada dodiruje prvi kraj. Zbog promijene polariteta napona spojenog na namotaj mijenja se i smjer struje. Fizikalni prikaz opisanih pojava prikazan je na slici 3.4 dok je na slici 3.5 prikazan valni oblik struje kroz armaturni namot.

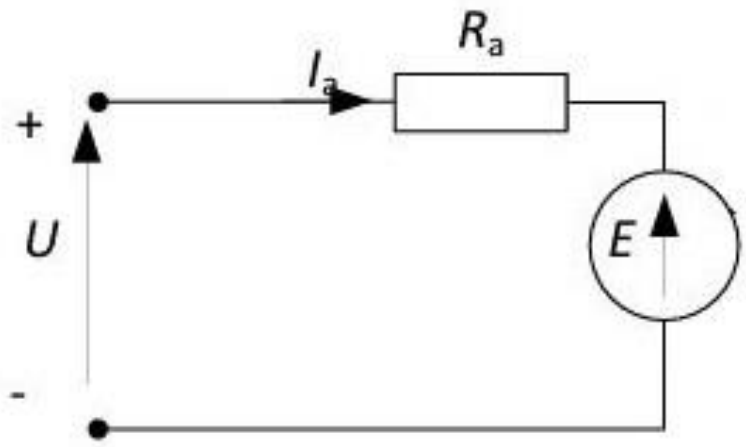


Slika 3.4. Princpi rada istosmjernog motora [9].



Slika 3.5. Valni oblici struje armature [10].

Svaki istosmjerni motor može se opisati nadomjesnom shemom pomoću koje se može izraditi matematički model stroja u različitim režimima. Kod istosmjernih strojeva nadomjesna shema veoma često uključuje samo rotorski krug s obzirom da je uzbudni magnetski tok često konstantan. Nadomjesna shema istosmjernog stroja prikazana je na slici 3.6.



Slika 3.6. Nadomjesna shema istosmjernog stroja [11].

Ova nadomjesna shema vrijedi samo za stacionarna stanja te na njoj zanemarujemo reaktivni otpor namotaja rotora. Napon *U* je narinuti napon odnosno napon napajanja stroja. Njemu ravnotežu drži elektromotorna sila *E*. Iz nadomjesne sheme možemo izraziti naponsku jednadžbu rotorskog kruga:

$$U = I_a \cdot R_a + E, \tag{3.1}$$

gdje je *U* narinuti napon, I_a struja armature, R_a otpor armaturnog namotaja, dok je *E* elektromotorna sila.

Korištenjem još dvije karakteristične jednadžbe istosmjernog motora: (1) jednadžbu induciranog napona *E*:

$$E = k_e \cdot \phi \cdot \omega, \tag{3.2}$$

gdje je *E* elektromotorna sila, k_e električna konstanta motora, ϕ magnetski tok stvoren od uzbude te ω brzina vrtnje motora.

; (2) te jednadžbu momenta:

$$M = k_m \cdot \phi \cdot I_a, \tag{3.3}$$

gdje je M razvijeni moment motora, k_m mehanička konstanta motora, ϕ magnetski tok stvoren od uzbude, dok je I_a struja armature; možemo dobiti jednadžbu ovisnosti brzine o fizikalnim veličinama istosmjernog stroja:

$$\omega = \frac{U}{k_e \cdot \phi} - \frac{R_a}{k_e \cdot k_m \cdot \phi^2} \cdot M. \tag{3.4}$$

Gdje je ω brzina vrtnje motora, U narinuti napon, k_e električna konstanta motora, k_m mehanička konstanta motora, ϕ magnetski tok stvoren od uzbude, R_a otpor armaturnog namotaja, dok je M razvijeni moment motora.

Izraz (3.4) nam je veoma bitan zbog toga što ukazuje na načine regulacije brzine vrtnje strojem koje je u ovom završnom radu vrlo bitno.

3.2. Mehanički prijenosnik snage

Prijenosni sustav, koji se također naziva i prigon, mehanički je sklop dizajniran za prijenos snage ili gibanja s pogonske komponente na gonjenu komponentu unutar stroja. Ovaj sustav se primarno sastoji od pogonskog elementa i gonjenog elementa koji se okreću, ostvarujući prijenos putem izravnog kontakta ili neizravno preko remena, užadi, lanaca i sličnih mehanizama. Različiti promjeri ovih komponenti rezultiraju različitim brzinama rotacije. Posljedično, ključna karakteristika prijenosnika je njegov prijenosni omjer, koji predstavlja odnos između brzine vrtnje pogonskih i gonjenih komponenti. Kada ovaj omjer premaši vrijednost jedan, brzina rotacije stroja se smanjuje dok se okretni moment povećava, te se takav element naziva reduktorom. Nasuprot tome, postoji i multiplikator kod kojeg je prijenosni omjer manji od jedan. Također postoje uređaji koji imaju promjenjivi prijenosni omjer te mogu raditi kao reduktor ili multiplikator.

Zajednička karakteristika svih mehaničkih prijenosnika snage jest ta da ulazna snaga mora biti jednaka izlaznoj, ako zanemarimo gubitke snage unutar samog mehanizma:

$$P_{ul} = P_{iz}. \tag{3.5}$$

Pojam prijenosnog omjera predstavlja omjer brzine vrtnje pogonskog ili ulaznog člana s brzinom vrtnje gonjenog ili izlaznog člana. Spomenute brzine su kutne brzine vrtnje, s druge strane obodne brzine vrtnje ovakvog sustava moraju biti jednake. Točnije rečeno obodna brzina pogonskog člana mora biti jednaka obodnoj brzini gonjenog člana. Obodna brzina rotirajućeg elementa može se izraziti kao:

$$v = r \cdot \omega. \tag{3.6}$$

Gdje je *v* obodna brzina vrtnje, *r* polumjer rotirajućeg elementa dok je ω kutna brzina vrtnje.

Korištenjem izraza za pogonski i gonjeni član dobivamo:

$$v_1 = v_2 = r_1 \cdot \omega_1 = r_2 \cdot \omega_2. \tag{3.7}$$

Gdje je v_1 obodna brzina vrtnje pogonskog člana, v_2 obodna brzina vrtnje gonjenog člana, r_1 polumjer pogonskog elementa, r_2 polumjer gonjenog elementa, ω_1 kutna brzina vrtnje pogonskog člana, dok je ω_2 kutna brzina vrtnje gonjenog člana.

Iz (3.7) možemo izraziti prijenosni omjer kao:

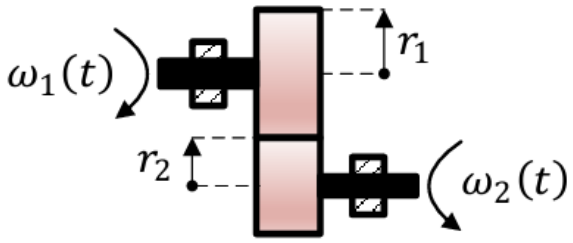
$$i_{1,2} = \frac{\omega_1}{\omega_2} = \frac{r_2}{r_1}. \tag{3.8}$$

Na sličan način možemo doći i do odnosa momenata:

$$i_{1,2} = \frac{M_2}{M_1}. \tag{3.9}$$

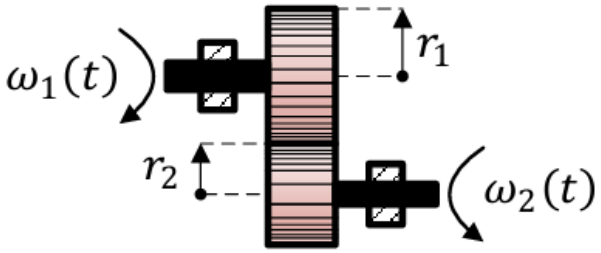
Gdje je M_1 moment sile pogonskog člana, a M_2 moment sile gonjenog člana.

Tarni prijenos ili prijenos trenjem (slika 3.7) predstavlja najosnovniji oblik izravnog kontaktnog prijenosa, gdje se snaga prenosi kroz interakcijom trenja između dviju komponenti, poznatih kao tarnice.



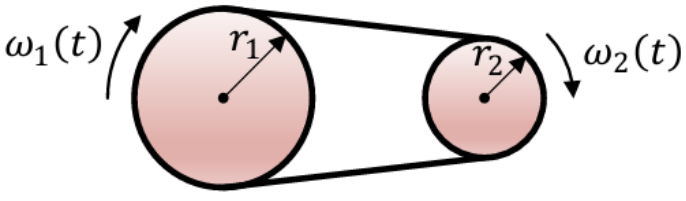
Slika 3.7. Tarni prijenos [12].

U današnjem svijetu prijenos je zupčanicima (slika 3.8) najraširenija metoda, koja koristi parove međusobno povezanih zupčanika čije osi mogu biti paralelne, okomite ili pod kutom. Ovaj sustav može prenijeti širok raspon razina snage i prijenosnih omjera, čak i pri vrlo velikim brzinama. Karakterizira ga visoka radna pouzdanost, produljeni radni vijek, sposobnost podnošenja kratkotrajnih preopterećenja i relativno jednostavni zahtjevi za održavanjem.



Slika 3.8. Prijenos zupčanicima [13].

Nasuprot tome, remenski prijenos (slika 3.9) djeluje neizravno između komponenti, preko savitljivih remena koji su na svojim krajevima spojeni. Ovi remeni mogu biti ravni, nazubljeni ili klinasti, a prijenos obodne sile s remenice na remen događa se trenjem. Ova vrsta prijenosa je slična sustavu koji koristi uže, bilo da je izrađeno od čelika ili prirodnih ili sintetičkih vlakana, umjesto remena.



Slika 3.9. Remenski prijenos [14].

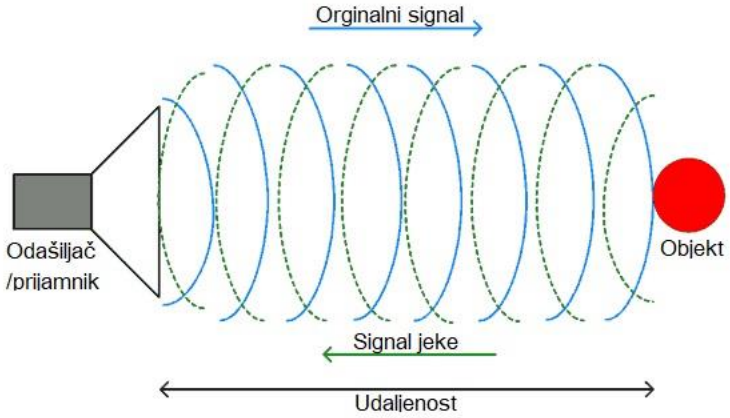
Kod zupčastih i tarnih prijenosa, pri računanju prijenosnih omjera, u (3.8) je potrebo staviti negativan predznak zbog činjenice da se pogonski i gonjeni član vrte u suprotne strane.

4. ULTRAZVUČNI SENZOR UDALJENOSTI

Ultrazvučni senzor za mjerenje udaljenosti je uređaj koji koristi zvučne valove za određivanje udaljenosti do objekta. Sastoji se od odašiljača, prijamnika, elektroničkog sklopa za obradu signala, te izlaznog kruga. Odašiljač emitira visokofrekventne zvučne valove (ultrazvučne valove) koji putuju kroz zrak dok ne udare u predmet i odbiju se natrag do prijamnika. Mjerenjem vremena koje je potrebno da se poslani val odbije te vrati, senzor računa udaljenost do objekta koristeći brzinu zvuka. Ovi se senzori obično koriste u robotici, automobilskim sustavima i industrijskim aplikacijama za precizno mjerenje udaljenosti, otkrivanje prepreka i detekciju razine. Beskontaktni su, pouzdani i dobro rade u različitim okruženjima, uključujući i ona s prašinom.

4.1. Princip rada

Ultrazvučni senzori rade na principu eholokacije, na principima sličinim onima koje možemo naći kod šišmiša i dupina. Senzor emitira niz ultrazvučnih zvučnih valova, obično u rasponu od 20 kHz do nekoliko MHz, koji putuju zrakom dok ne naiđu na objekt. Nakon udarca u predmet, zvučni valovi se reflektiraju natrag na senzor kao jeka. Senzor tada mjeri vrijeme potrebno da se jeka vrati. Opisani princip prikazan je na slici 4.1. Raspon frekvencija valova vrlo je bitan zbog toga što valovi moraju biti nečujni, a znamo da je gornja granica osjetljivosti ljudskog uha 20 kHz. Zvučni valovi kojima je frekvencija veća od 20 kHz nazivaju se ultrazvučni valovi.



Slika 4.1. Princip rada ultrazvučnog senzora [15].

Koristeći brzinu zvuka u zraku, koja je otprilike 343 m/s na sobnoj temperaturi udaljenost između senzora i predmeta možemo izraziti kao:

$$l = \frac{1}{2} \cdot v * t. \tag{4.1.}$$

Gdje je *l* udaljenost senzora od objekta, *v* brzina zvuka, dok je *t* vrijeme proteklo od odašiljanja vala do prijama vala.

U izmjerenom vremenu primljeni ultrazvučni val je prešao mjerenu udaljenost dva puta, što objašnjava faktor od 1/2.

Ultrazvučne valove, ovakvi senzori udaljenosti, generiraju pomoću piezoelektričnih elemenata izrađenih od kristala, a njihov princip rada temelji se na piezoelektričnom učinku. Piezoelektrični učinak je fenomen kod kojeg određeni materijali stvaraju razdvajanje električnih naboja kao reakciju na djelovanje silom. Ovaj se učinak javlja na atomskoj razini u materijalima koji imaju kristalnu strukturu bez središta simetrije, poput kvarca, određenih keramika i nekih polimera. Kada se na te materijale primijeni mehanička sila, atomi unutar kristalne rešetke lagano se pomaknu, uzrokujući neravnotežu električnih naboja. Ovaj učinak je reverzibilan, što znači da ako se na te materijale primijeni električno polje, oni mogu promijeniti oblik ili se deformirati. Piezoelektrični odašiljači generiraju ultrazvučne valove upravo putem obrnutog piezoelektričnog učinka. Na piezoelektrični odašiljač dovodi se izmjenični napon frekvencije koja odgovara prirodnoj rezonantnoj frekevnciji kristala koja se nalazi u ultrazvučnom području. Zbog obrnutog piezoelektričnog učinka, piezoelektrični materijal deformira se na isti način kao i ulazni napon, što uzrokuje vibracije u materijalu, koje se zatim šire u okolni medij.

4.2. Modul HY-SRF05

U ovom završnom radu korištena su tri ultrazvučna senzora udaljenosti tipa HY-SRF05 (slika 4.2) postavljenih na desno, na lijevo te ispred robotskih kolica. Ovaj model senzora je nadogradnja nad češće korištenim HC-SRO4 zbog svoje veće preciznosti. Senzor ima 5 *pin*-ova i može se koristiti u 1-*pin* modu gdje se jedan *pin* koristi za slanje signala te za dobivanje povratne

informacije o primitku reflektiranog vala, ili 2-*pin* modu, gdje se jedan *pin* koristi za slanje signala, a drugi *pin* za dobivanje povratne informacije. Domet mjerenja je u opsegu 2-400 cm, a točnost mjerenja može doseći i do 3 mm.

Slika 4.2. HY-SRF05 senzor [16].

Tehnički podatci dani od proizvođača:

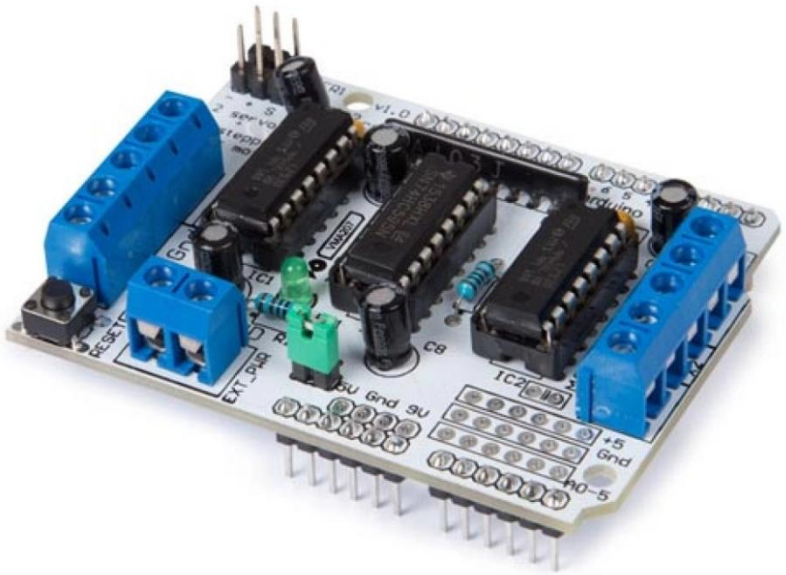
- format *trigger pin*-a: digitalni impuls od 10 µs
- frekvencija vala: 40 kHz
- *echo pin* izlaz: 0-Vcc
- raspon mjerenja: 2 cm do ~ 4.5 m
- rezolucija mjerenja: 0.3 cm
- kut mjerenja: do 15°
- brzina mjerenja: 40 Hz
- napon napajanja: 4.5 V do 5.5 V
- potrošnja u aktivnom načinu rada: 10 do 40 mA
- potrošnja u mirovanju: manje od 2 mA

Postupak mjerenja je poprilično jednostavan. Prvo je potrebno poslati impuls visoke razine trajanja 10 µs senzoru na *trigger pin*. Senzor tada automatski generira i preko odašiljača šalje ultrazvučni val frekvencije 40 kHz. U trenutku slanja vala *echo pin* se automatski postavlja u stanje

visoke razine koje traje sve dok prijamnik ne detektira reflektirani val, kada se *echo pin* postavlja u stanje niske razine. Unutar programskog koda Arduina, potrebno je izmjeriti vrijeme koliko dugo traje impuls na *echo pin*, te primjeniti (4.1) da bismo izračunali udaljenost. U ovom završnom radu, prethodno opisana funkcionalnost implementirana je korištenjem <*NewPing.h*> biblioteke. Metoda *double ping_cm()* vraća izmjerenu udaljenost u centimetrima, dok metoda *double ping()* vraća trajanje impulsa na *echo pin*-u.

5. *MOTOR SHIELD* S L293D

Motor shield s L293D, prikazan na slici 5.1, je vrsta Arduino *shield*-a koji služi za kontrolu do 4 istosmjerna motora u naponskom opsegu 4.5-25 V, te strujom od 600 mA kontinuirano, odnosno 1.2 A vršno po motoru s kontrolom brzine i smjera. Arduino *shield*-ovi su pločice koje se mogu priključiti na Arduino razvojnu pločicu proširujući njezine mogućnosti. Na priključke za dva istosmjerna motora moguće je spojiti jedan koračni motor, stoga modul također može upravljati do dva koračna motora. Osim toga, modul podržava i upravljanje s dva servo motora za koje postoje dva 3-*pin*-ska konektora. Napajanje tih motora je doduše spojeno na napajanje Arduino razvojne pločice, pa nisu predviđeni za veće snage.

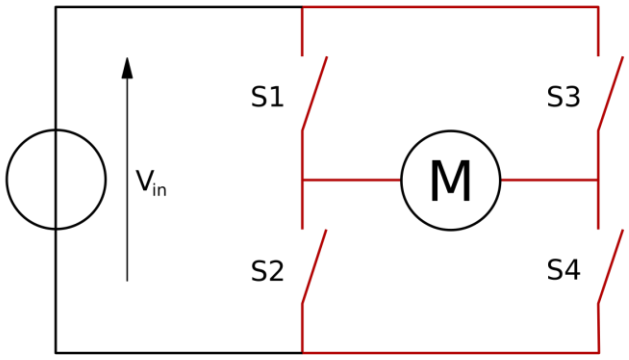


Slika 5.1. Motor shield sa L293D [17].

Na pločici se nalaze dva L293D integrirana kruga i 74HC595 posmačni registar kao što se može vidjeti na slici 5.1. Integrirani krug L293D je dvokanalni H-most *motor driver* koji može kontrolirati dva istosmjerna motora ili jedan koračni motor. Integrirani krug 74HC595 je posmačni registar, koji omogućuje da pomoću četiri Arduino digitalna *pin*-a generiramo osam potrebnih signala za ulaze L293D integriranih krugova.

5.1. Princip rada H-mosta

H-most je sklop koji ima mogućnost mijenjanja polariteta napona na trošilu. Ovi sklopovi se često koriste u robotici kako bi se omogućilo istosmjernim motorima okretanje smjera vrtnje, kao i upravljanje brzinom. Ime je izvedeno iz uobičajenog shematskog prikaza sklopa (vidljivog na slici 5.2), s četiri sklopna elementa konfigurirana kao grane slova „H“.



Slika 5.2. Shematski prikaz H-mosta [18].

Princip rada vrlo je jednostavan. Pravilnim uklapanjem i isklapanjem sklopki može se postići polaritet napajanja uređaja u oba smjera. U slučaju da su zatvorene sklopke S1 i S4 struja će kroz armaturni krug teći „s lijeva na desno“ te će istosmjerni motor početi razvijati elektromehanički moment u jednom smjeru ovisan o iznosu same armaturne struje (izraz 3.3). Ako u nekom drugom trenutku otvorimo sklopke S1 i S4, a u isto vrijeme zatvorimo sklopke S3 i S2 struja koja teče armaturnim krugom promijenit će svoj smjer zbog promijene polariteta napajanja, te će sada teći „od desna na lijevo“. Sada struja ima „negativan“ predznak te će stroj prema izrazu (3.3) početi razvijati elektromehanički moment suprotnog predznaka odnosno suprotnog smjera. Osim promjene smjera rotacije, H-most omogućuje dodatne režime rada, kočni režim i režim slobodnog hoda do zaustavljanja trenjem. Režim kočenja motora je režim gdje se motor iznenada zaustavlja kada su stezaljke motora kratko spojene. Kratkim spajanjem stezaljki, kinetička energija motora se troši na otporu armaturnih namotaja (slika 3.6) te se pretvara u toplinu. Ovaj režim rada poznat je kao elektrodinamičko kočenje. Drugi slučaj omogućuje motoru da se zaustavi uz pomoć trenja. Ovaj slučaj nije ništa drugo nego obično odspajanje motora s napajanja čime nestaje sam napon napajanja, armaturna struja postepeno opada, protu-elektromotorna sila se smanjuje proporcionalno

sa strujom, a kinetička energija rotora predaje se ležajevima koji je pretvaraju u toplinu radi trenja. Vrlo je važno za zamijetiti da dvije sklopke u istoj „grani“ slova H nikada ne smiju biti u isto vrijeme zatvorene zato što to predstavlja kratki spoj naponskog izvora. Slučajevi kratkog spoja opasni su za izvor napajanja i za sklopke. Tablica 5.2 prikazuje stanja elektromotora ovisnih o otvorenosti i zatvorenosti sklopki H-mosta, pri čemu S1-S4 odgovaraju slici 5.5. Stanje 1 se koristi za predstavljanje uključenog stanja prekidača, a 0 za predstavljanje isključenog stanja.

Tablica 5.2. Tablica stanja H-mosta

S1	S2	S3	S4	Rezultat
1	0	0	1	Vrtnja udesno
0	1	1	0	Vrtnja ulijevo
0	0	0	0	Prazan hod
1	0	0	0	
0	1	0	0	
0	0	1	0	
0	0	0	1	
1	1	x	x	Kratki spoj
x	x	1	1	
1	0	1	0	Kočni režim
0	1	0	1	

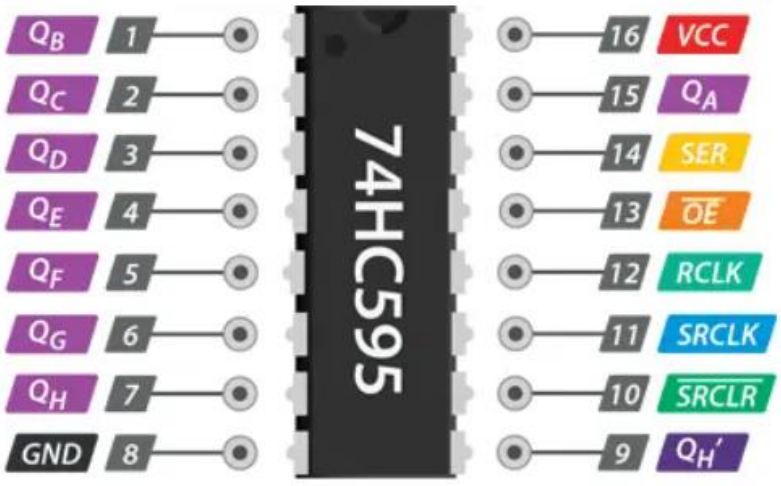
Za sklopke H-mosta mogu se koristiti bipolarni ili unipolarni tranzistori, a u nekim visokonaponskim primjenama i IGBT-ovi.

5.2. 74HC595 Posmačni registar

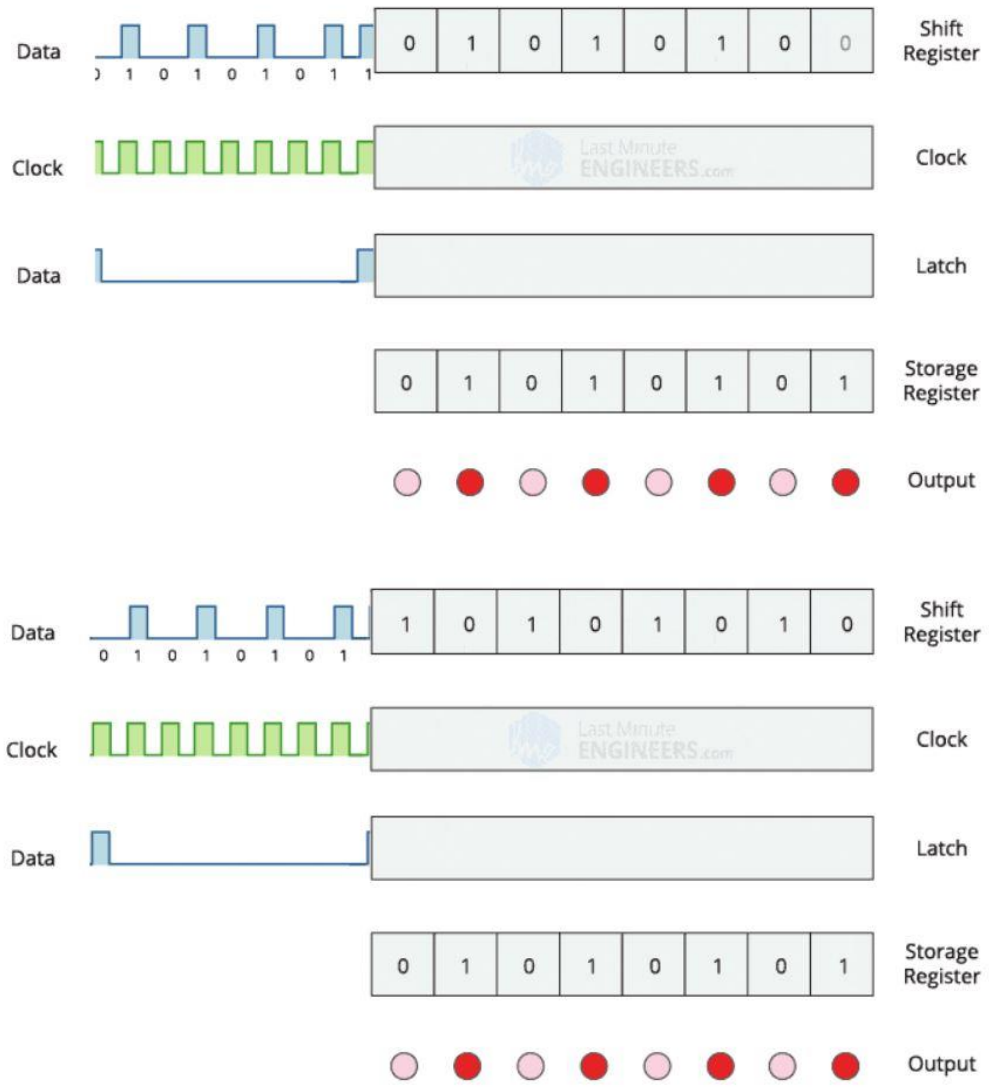
74HC595 je 8-bitni posmični registar sa serijskim ulazom i paralelnim izlazom te 8-bitnim registrom za pohranu. Uređaj ima pin za serijski ulaz podataka, osam paralelnih podatkovnih izlaza i serijski izlaz za kaskadiranje. Obično se koristi za proširenje broja I/O pinova dostupnih na mikrokontroleru, omogućujući kontrolu višestrukih izlaza s nekoliko pinova.

74HC595 ima dva 8-bitna registra. Prvi se naziva *Shift Register*, a drugi *Storage/Latch Register*. Na svaki impuls radnog takta unutar posmačnog registra podatci se miču udesno: bit 0 se pomiče na mjesto bita 1, bit 1 se pomiče na mjesto bita 2, itd., dok se na upražnjeno mjesto bita 0 zapisuje podatak s ulaznog DATA pina.

Kada je *latch enable*-an, sadržaj posmačnog registra se kopira u registar za pohranu. Svaki bit registra za pohranu povezan je s jednim od paralelnih izlaznih *pin*-ova posmačnog registra. Kao rezultat toga, kad god se vrijednost u registru za pohranu promijeni, izlaz se mijenja. Na slici 5.4 prikazan je primjer posmaka bitova unutar registra, dok je na slici 5.3 prikazan popis *pin*-ova.



Slika 5.3. Popis pinova 74HC595 posmačnog registra [19].



Slika 5.4. Princip rada posmačnog registra [20].

Od svih *pin*-ova prikazanih na slici 5.3 izdvojiti ćemo i opisati samo one najvažnije. *Pin* SER (eng. *Serial Input*) je ulazni DATA *pin*, dok je *pin* SRCLK (eng. *Shift Register Clock*) ulaz za radni takt. Kada je *pin* RCLK (eng. *Register Clock*/*Latch*) u visokom stanju sadržaj posmačnog registra se kopira u izlazni registar za pohranu. Pomoću *pin*-a SRCLR (eng. *Shift Register Clear*) možemo pobrisati podatke zapisane u registru, tj. postaviti ih u 0. Kada je *pin* OE (eng. *Output Enable*) aktivan izlazi QA-QH su u stanju visoke impedancije. Zadnji podatkovni izlaz QH omogućuje i kaskadno spajanje više registara, tako da ga spojimo na SER ulaz sljedećeg registra.

6. ALGORITMI ZA RJEŠAVANJE LABIRINATA

Labirinti su stoljećima zaokupljali ljudsku maštu te se pojavljuju u društvu u raznim oblicima, od drevnih labirintskih struktura do modernih izazova u video igrama. Programsko rješavanje labirinta predstavlja zanimljiv problem u računalstvu, s brojnim rješenjima koja se mogu kategorizirati u dvije skupine. Prva skupina su algoritmi kod kojih algoritam nema nikakve informacije o labirintu te slijepo traži izlaz. Druga skupina su algoritmi kod kojih program ima uvid u cijeli labirint. Kako je ideja ovog završnog rada rješavanje labirinta na slijepo, drugu skupinu algoritama nećemo opisivati. Neki od osnovnih algoritama za rješavanje labirinta na slijepo su algoritam slučajnog odabira, algoritam koji slijedi zid, Pledge-ov algoritam te Tréamaux-ov algoritam.

Algoritam slučajnog odabira na križanju nasumično odlučuje o sljedećem smjeru koji će slijediti bez pamćenja kojim je rutama prošao, stoga algoritam može biti vrlo spor i neprimjeren za bilo kakvu praktičnu upotrebu.

Pledge-ov algoritam je tehnika rješavanja labirinta koja pomaže u kretanju labirintom prateći zidove dok ispravlja potencijalne petlje ili zaobilaznice. Princip rada sastoji se od nekoliko koraka. Algoritam započinje odabirom početnog smjera prema kojem se krećemo, s ciljem dostizanja općeg smjera izlaza. Kada naiđe na prepreku, poput zida, počinjemo je slijediti okrećući se desno (ili lijevo, proizvoljno). Pri skretanju vrlo je važno pratiti ukupni kut skretanja. To znači da zbrajamo stupnjeve pri okretu (npr. pozitivno za desna skretanja, negativno za lijeva skretanja). Nastavljamo pratiti zid sve dok ukupni kut ne bude jednak nuli, što znači da smo ponovno okrenuti prema izvornom smjeru. U tom trenutku prestajemo slijediti zid i nastavljamo se kretati ravno u odabranom smjeru. Ako se naiđe na drugu prepreku, postupak ponavljamo. Ovaj algoritam osigurava izlazak iz labirinta, jer izbjegava zarobljavanje u petljama pamćenjem generalnog smjera u kojem se treba gibati.

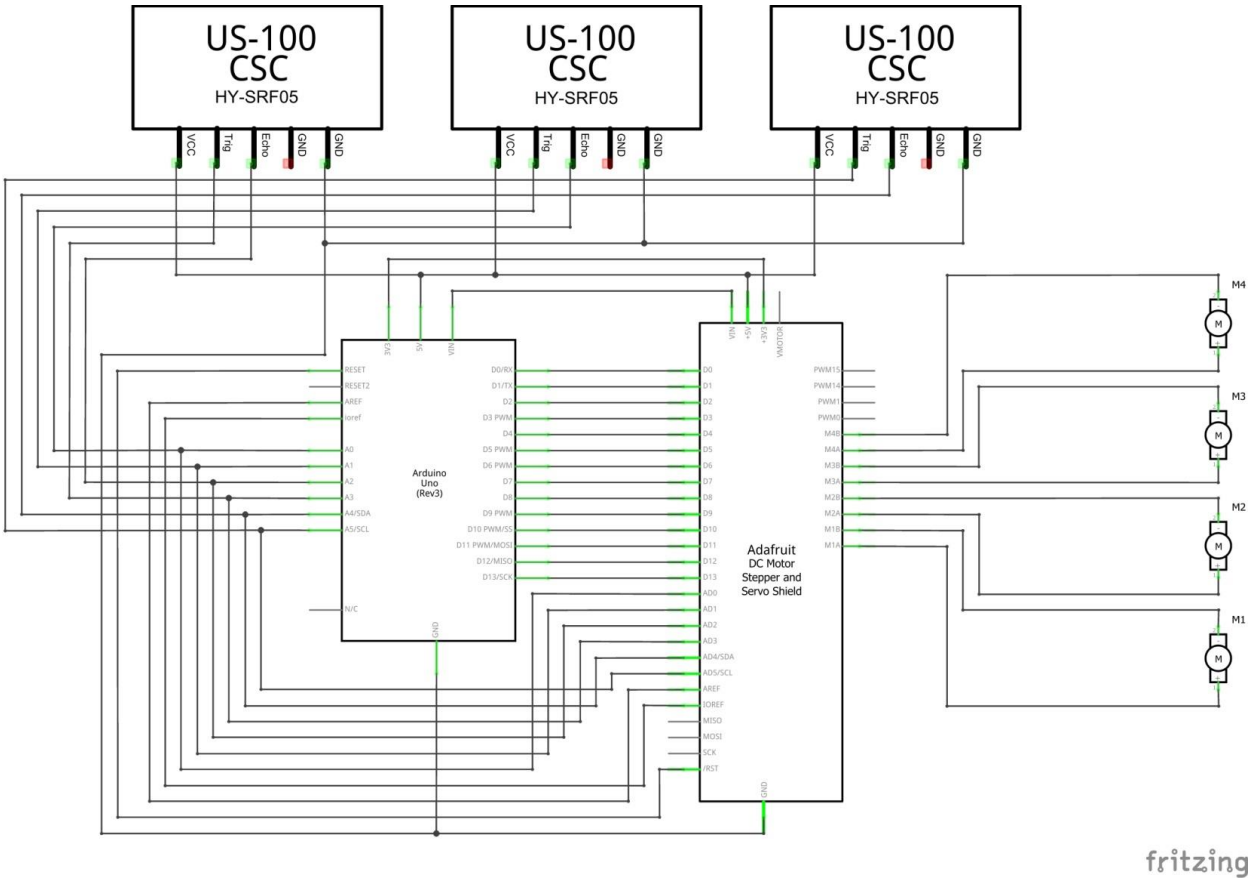
Trémaux-ov algoritam je učinkovita metoda za pronalaženje izlaza iz labirinta koji zahtijeva crtanje linija na podu kako bi se označio put, ali nije zajamčeno da će pronaći najkraći put. Dok prolazimo po labirintu, potrebno je označavati staze po kojima je robot prošao, a na raskrižju uvijek skrećemo prema neoznačenoj stazi, tj. stazi kojom robot još nije prošao. Ako su svi putovi označeni, ili naiđemo na slijepu ulicu, robot se vraća prethodno označenim putem. Ako ponovno posjećujemo

raskrižje, treba odabrati stazu označenu samo jednom. Vrlo je važno izbjegavati dva put označene staze osim ako je potrebno. Ovaj algoritam osigurava da se nijednom stazom ne prijeđe više od dva puta i jamči dolazak do izlaza ili povratak na početak.

Algoritam koji slijedi zid, jedna je od najjednostavnijih i najintuitivnijih metoda za rješavanje labirinta. Ova se tehnika oslanja na načelo praćenja granica labirinta, osiguravajući da na kraju dođemo do izlaza, pod uvjetom da je labirint jednostavno povezan (tj. bez petlji koje bi mogle prekinuti staze). Na početku proizvoljno odabiremo ili lijevi ili desni zid, te robot cijelim svojim putem prati taj zid sve dok ne dođe do izlaza. Učinkovitost ovog algoritma ukorijenjena je u topologiji, posebno u svojstvima jednostavno povezanih prostora. Jednostavno povezani labirint je onaj u kojem su svi zidovi povezani bez izoliranih dijelova. U takvom labirintu, praćenje zida na kraju će nas dovesti do izlaza jer zbog topologije labirinta ne možemo se vrtiti u krug oko izoliranog dijela labirinta. Unatoč svojoj jednostavnosti, algoritam ima široku primjenu u stvarnom svijetu. Može se koristiti u hitnim slučajevima, kao što je navigacija iz zgrade tijekom požara kada je vidljivost slaba ili u istraživanju kada složeniji navigacijski alati nisu dostupni. Jednostavnost algoritma čini ga dostupnim svima, ne zahtijeva posebne vještine ili alate te je upravo iz tog razloga odabran za korištenje u ovom završnom radu.

7. ELEKTRIČNA SHEMA

Na slici 7.1. prikazana je električna shema spoja napravljena u programu Fritzing. Na shemi vidi se korišteni Arduino Uno, *motor shield* sa L293D H-mostovima, tri HY-SRF05 ultrazvučna senzora te četiri istosmjerna motora.

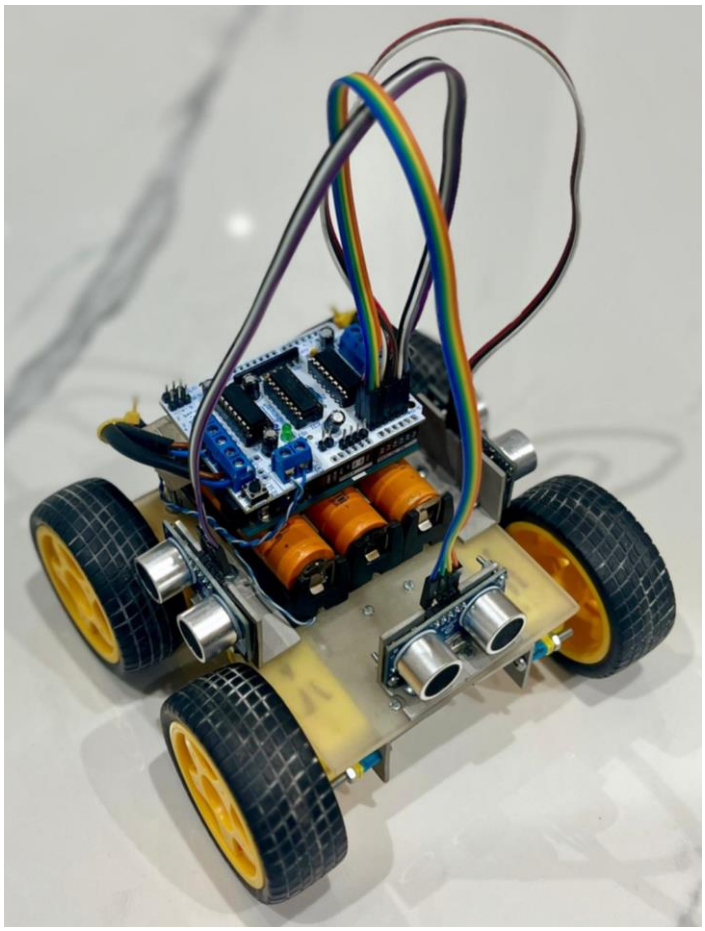


Slika 7.1. Električna shema robotskih kolica.

Sam spoj je poprilično jednostavan zbog činjenice što je korišten *motor shield*. *Motor shield* ne samo što pojednostavljuje upravljanje motorima već smanjuje količinu korištenih žica te znatno olakšava bilo kakav *troubleshoot*-ing u slučaju problema, kvarova ili grešaka. *Motor shield* postavljen je na Arduino Uno razvojnu pločicu. Kako je već opisano u petom poglavlju, analogni *pin*-ove izvedeni od Arduina na *motor shield* popunjeni su zaglavljima te se na njih spajaju tri ultrazvučna senzora udaljenosti tipa HY-SRF05. Za napajanje spomenutih senzora korišten je Arduino 5 V *pin* koji je također izveden na *motor shield* radi lakšeg pristupa. Ultrazvučni senzor koji gleda lijevo od kolica spojen je na analogne *pin*-ove A2 (*echo*) i A3 (*trig*). Senzor koji gleda

ispred kolica spojen je na analogne *pin*-ove A0 (*echo*) i A1 (*trig*), dok je senzor koji gleda desno od kolica spojen na *pin*-ove A4 (*echo*) i A5 (*trig*). Četiri istosmjerna elektromotora korištena za pogonjenje kolica spojena su na redne stezaljke *motor shield*-a pazeći da, kad je u kodu zadana kretnja naprijed, se svi elektromotori vrte u istu stranu. Napajanje Arduina i *motor shield*-a je odvojeno. Razlog tome je što je za kretnje labirintom potrebno vrlo fino upravljanje brzinom i smjerom motora. Kada bismo imali samo jedno napajanje spojeno na Arduino postojao bi problem kod tog finog upravljanja zbog toga što bi *Vin pin* na Arduinu, koji bi u tom slučaju bio korišten za napajanje sva četiri motora, imao poprilično veliku valovitost u naponu. Također, *motor shield* je bolje napajati odvojeno zbog toga što sami elektromotori vuču podosta veliku struju, posebice kod pokretanja. Arduino je napajan preko svog DC priključka na kojeg je spojena baterija od 9 V dok je *motor shield* napajan s tri Li-Ion baterije od 3.7 V spojenih u seriju, stvarajući ukupnih 11.1 V.

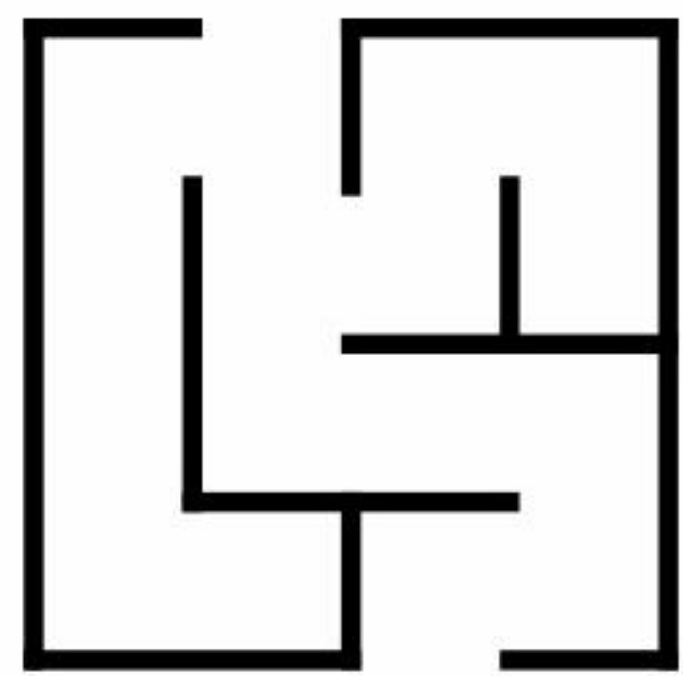
Na slici 7.2 prikazan je izgled robotskih kolica.



Slika 7.2. Izgled robotskih kolica.

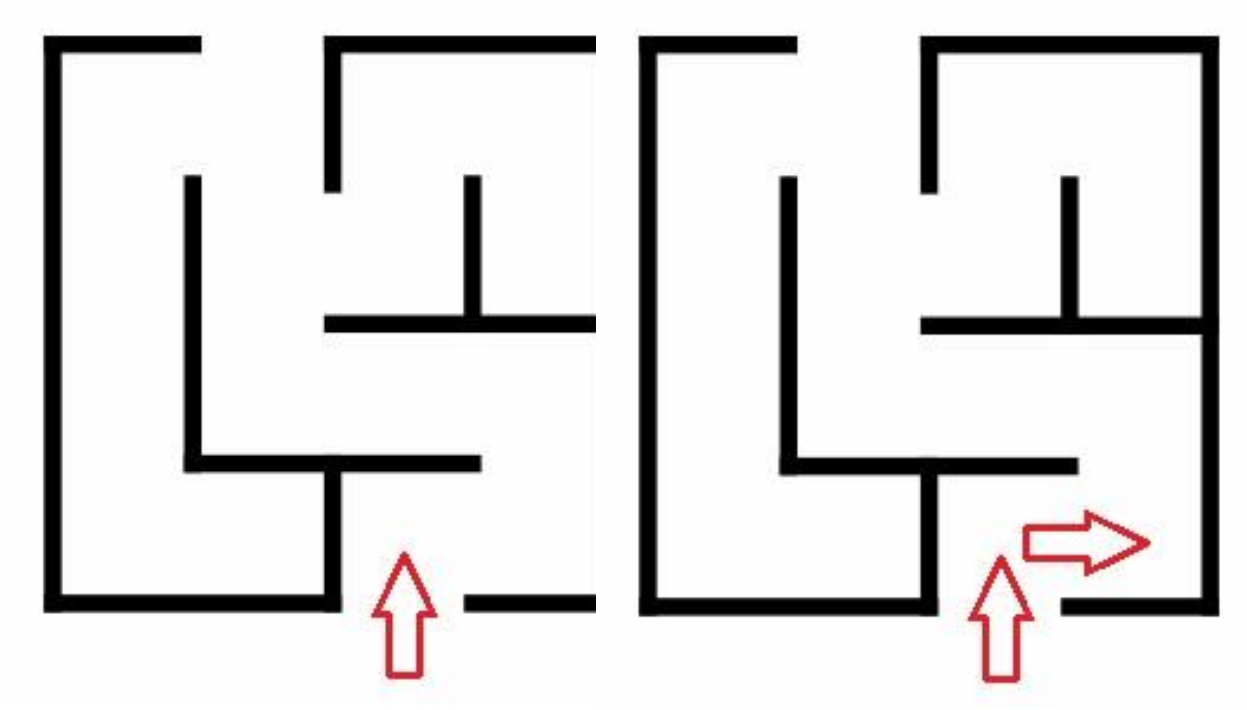
8. PRINCIP RADA

Princip rada objašnjen je na labirintu, prikaznog na slici 8.1, nasumično generiranom koristeći jedan od mnogobrojnih generatora dostupnih na internetu.



Slika 8.1. Izgled labirinta [21].

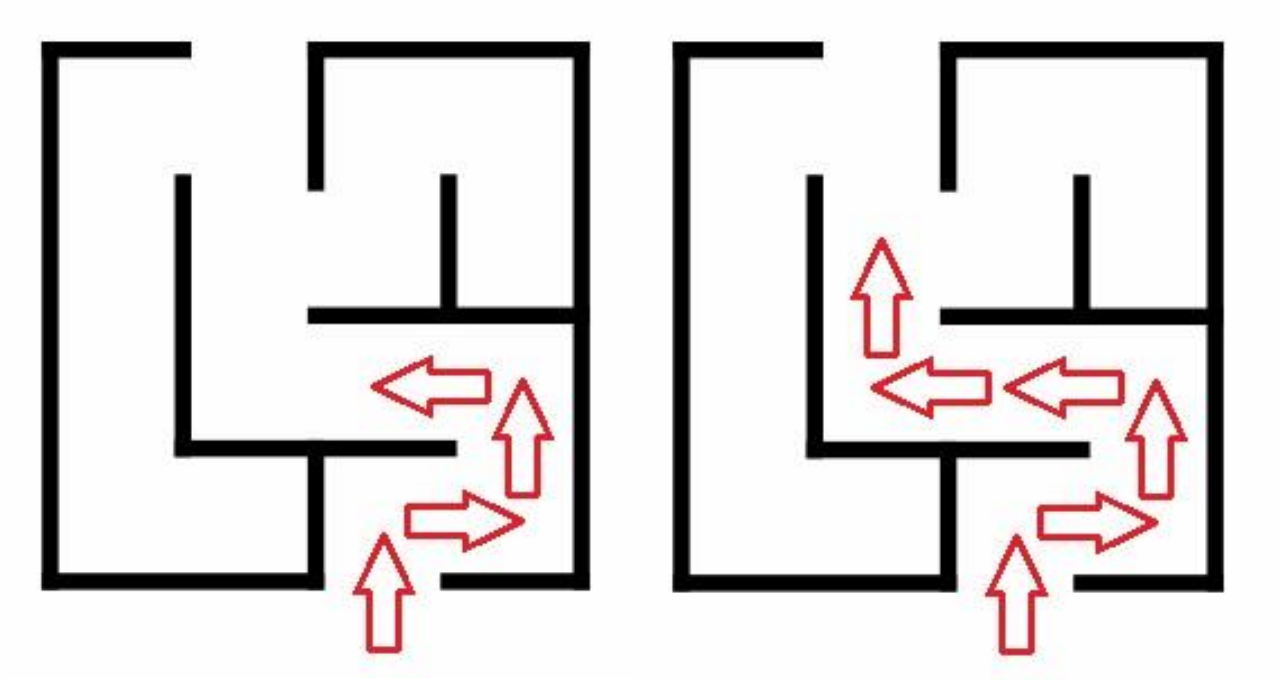
Kao što je opisano u poglavlju šest, algoritam korišten za rješavanje labirinta je algoritam koji slijedi zid. Detekcija zida se vrši pomoću ultrazvučnih senzora, a algoritam je implementiran tako da prati lijevi zid. Praćenje zida se u programskom kodu može rastaviti na nekoliko cjelina. Uz pomoć lijevog senzora detektira se postojanje ili odsutnost lijevog zida. Dok postoji lijevi zid u blizini te dok ispred samih kolica nema nikakve prepreke (u ovom slučaju zida) kolica će se kretati ravno. Kretnja ravno je postignuta tako da se sva četiri istosmjerna motora kreću u istom smjeru istom brzinom. Za dani primjer labirinta sa slike 8.1, na slici 8.2 prikazano je prvo gibanje kolica, ako donji otvor labirinta definiramo kao ulazni.



Slika 8.2. Ulazak u labirint, krenja naprijed te skretanje desno.

Drugi korak je vezan za postojanje lijevog zida te prepreke ispred kolica. U tom slučaju kolica, da bi nastavila pratiti lijevi zid, moraju skrenuti udesno, te je takav slučaj prikazan na slici 8.2. Skretanje udesno implementirano je na način na se lijevi kotači vrte unaprijed, a desni unatrag.

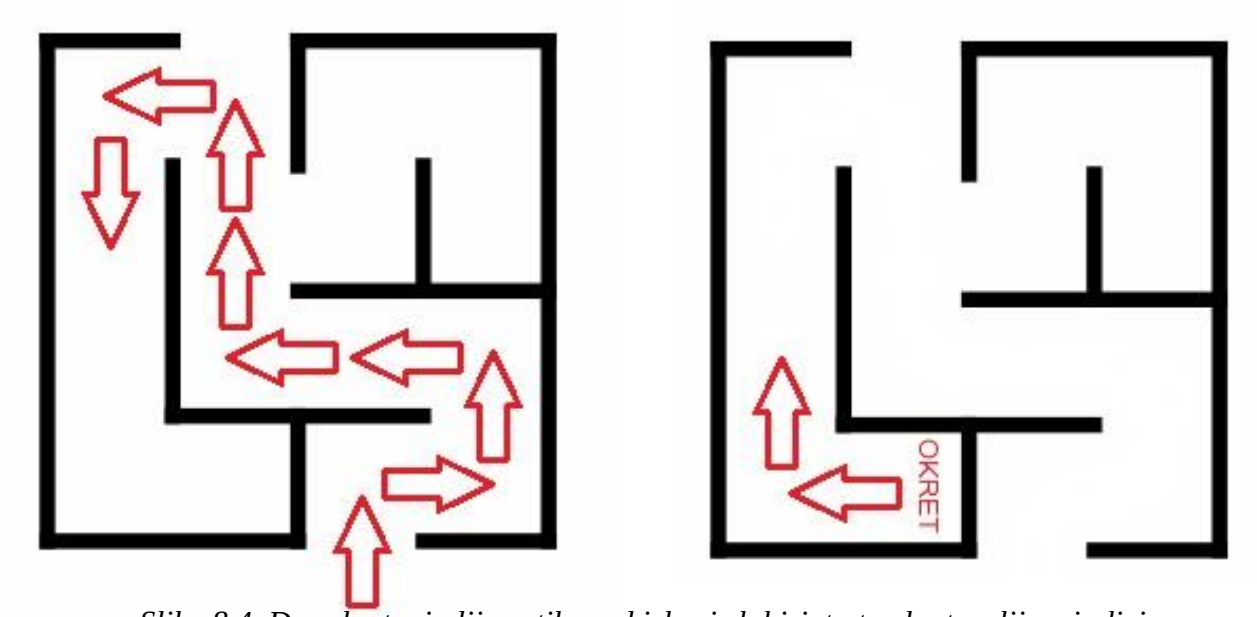
U zadanom primjeru, sada imamo situaciju isto kao i na početku, te se kolica gibaju ravno naprijed. U sljedećem koraku, dolazimo do situacije da lijevi senzor detektira odsutnost zida. Kada pratimo lijevi zid, vrlo je važno da kolica skrenu lijevo neovisno o stanjima preostala dva senzora, ukoliko postoji mogućnost skretanja ulijevo. To je slučaj u ovom koraku, pa će kolica dva puta skrenuti lijevo, a potom nastaviti ravno sve do sljedećeg zida, kao što je prikazano na slici 8.3. Skretanje lijevo, analogno skretanju desno, postignuto je tako da se desni kotači vrte unaprijed, a lijevi unatrag.



Slika 8.3. Dva skretanja lijevo te ponovno skretanje desno.

Na sljedećem raskrižju pojavljuje se prethodno opisana situacija, te će kolica skrenuti udesno, kao što je prikazano na slici 8.3.

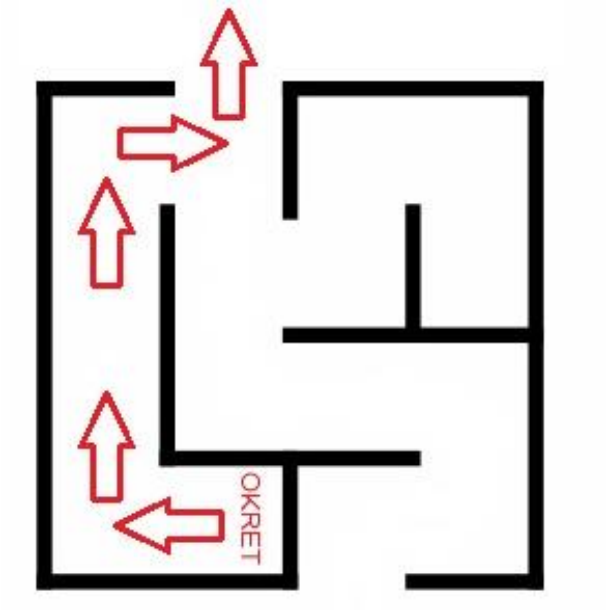
Sada se kolica kreću ravno prema izlazu no kako u jednom trenutku tik ispred izlaza iz labirinta nestaje lijevi zid kolica skreću dva puta ulijevo, kao što je prikazano na slici 8.4.



Slika 8.4. Dva skretanja lijevo tik pred izlaz iz labirinta te okret u slijepoj ulici

Prateći prethodnu putanju, te nakon skretanja ulijevo kolica nailaze na slijepu ulicu. U slijepoj ulici (slika 8.4) sva tri senzora detektiraju postojanje zida, te je jedina mogućnost okretanje kolica. Okret za 180° ostvaren je tako da, ovisno u kojem smjeru se kolica gibaju, kotači jedne strane se okreću u jednom smjeru, a kotači druge strane u suprotnom smjeru. Nakon okreta, kolica skreću desno.

Kolica nastavljaju pratiti sebi lijevi zid koji je sada suprotan od zida kojeg su pratili pri ulazu u istu ulicu. Dolaze opet do situacije postojanja zida lijevo i ispred te skreću desno. Nakon toga kreću se ravno dok postoji lijevi zid. Kolica skreću lijevo i uspješno izlaze iz labirinta (slika 8.5). Kada izađu iz labirinta sva tri senzora ne detektiraju prepreke oko sebe, što je znak da smo izašli iz labirinta, te se svi motori zaustavljaju.



Slika 8.5. Skretanje desno i izlazak iz labirinta.

9. PROGRAMSKA PODRŠKA

Prvi korak prilikom pisanja programske podrške za ovaj završni rad je uključivanje potrebnih biblioteka da bismo mogli koristiti pojednostavljene naredbe za određene senzore. Korištene se dvije biblioteke, prethodno objašnjene u potpoglavlju 2.2.

```
#include <AFMotor.h>
#include <NewPing.h>
```

Nakon toga, definirano je nekoliko konstanti koje će povećati čitljivost programa.

```
#define trigPinFront A1
#define echoPinFront A0
#define trigPinLeft A3
#define echoPinLeft A2
#define trigPinRight A5
#define echoPinRight A4
#define maximumDistance 200
```

Dalje slijedi stvaranje objekata. *NewPing* je konstruktor klase definirane u <*NewPing.h*> biblioteci te kao argumente zahtjeva *trig* i *echo pin*-ove te maksimalnu moguću udaljenost, što su prethodno definirane konstante. *AF_DCMotor* je konstruktor klase definirane u <*AFMotor.h*> biblioteci pomoću koje upravljamo motorima. Imena instanci klase prate lokaciju motora na šasiji kolica (npr. RB - *right back*; desno iza), dok sam konstruktor zahtjeva dva argumenta: prvi argument je oznaka rednih stezaljki na *motor shield*-u na koje je motor spojen, a drugi je konstanta koja definira frekvenciju PWM signala.

```
NewPing sonarFront(trigPinFront, echoPinFront, maximumDistance);
NewPing sonarLeft(trigPinLeft, echoPinLeft, maximumDistance);
NewPing sonarRight(trigPinRight, echoPinRight, maximumDistance);
AF_DCMotor motorRB(1, MOTOR12_8KHZ);
AF_DCMotor motorRF(2, MOTOR12_8KHZ);
AF_DCMotor motorLF(3, MOTOR34_8KHZ);
AF_DCMotor motorLB(4, MOTOR34_8KHZ);
```

Zatim slijedi definicija i inicijalizacija pomoćnih globalnih varijabli.

```
float distanceF = 100;
float distanceR = 100;
float distanceL = 20;
int Close = 6;
int Far = 15 ;
int Front = 20;
int Left = 25;
int Right = 20;
```

```
boolean goesForward = false;
boolean goesStop = true;
```

U *setup()* funkciji nekoliko puta mjerimo udaljenost svih senzora zbog kalibracije senzora. Uz to, definirane su brzine svih motora. Na kraju pokrećemo komunikaciju s računalom za potrebe *troubleshoot*-anja.

```
void setup() {

distanceF = readDistanceFront();
delay(100);
distanceF = readDistanceFront();
delay(100);
distanceF = readDistanceFront();
delay(100);
distanceF = readDistanceFront();
delay(100);
distanceF = readDistanceFront();
delay(100);

distanceL = readDistanceLeft();
delay(100);
distanceL = readDistanceLeft();
delay(100);
distanceL = readDistanceLeft();
delay(100);
distanceL = readDistanceLeft();
delay(100);
distanceL = readDistanceLeft();
delay(100);

distanceR = readDistanceRight();
delay(100);
distanceR = readDistanceRight();
delay(100);
distanceR = readDistanceRight();
delay(100);
distanceR = readDistanceRight();
delay(100);
distanceR = readDistanceRight();
delay(100);

 motorRB.setSpeed(200);
 motorRF.setSpeed(200);
 motorLB.setSpeed(200);
 motorLF.setSpeed(200);
 motorRB.run(RELEASE);
 motorRF.run(RELEASE);
 motorLF.run(RELEASE);
 motorLB.run(RELEASE);

Serial.begin(9600);

}
```

U *loop()* funkciji nalazi se glavni dio koda odnosno kod koji se odnosi na sam algoritam rješavanja labirinta. Funkcije *slightRight() i slightLeft()* koriste se prilikom gibanja unaprijed, a kada kolica nisu postavljena savršeno paralelno zidu. U slučaju da se kolica previše približe lijevom zidu ona se blago udalje od zida, a ako se nađu u poziciji predaleko od lijevog zida onda se blago približe. Zbog toga, u praksi se kolica gibaju cik-cak kroz labirint. Na početku *loop()* funkcije vrši se mjerenje udaljenosti u sva tri smjera, te na temelju dobivenih vrijednosti kroz *if* naredbe se pozivaju funkcije koje vrše specifično gibanje kolica ovisno o izmjerenim udaljenostima.

```
void loop() {


distanceF = readDistanceFront();
distanceL = readDistanceLeft() - 1;
distanceR = readDistanceRight();

if (distanceL > Left) {

  turnLeft();

  }

else if (distanceL < Left && distanceF > Front) {

  if (distanceL > Close && distanceL < Far) {
  moveForward();
  }

  else if (distanceL < Close) {
  moveStop();
  delay(200);
  slightRight();
  }

  else if (distanceL > Far && distanceL <17) {
  moveStop();
  delay(200);
  slightLeft();
  }

 }

else if (distanceF < Front && distanceL < Left && distanceR > Right) {

  turnRight();

  }
else if (distanceF < Front && distanceL < Left && distanceR < Right) {

  moveStop();
  moveRotate();

  }
```

```
else if (distanceF > Front && distanceL > Left && distanceR > Right) {

  moveStop();
  Serial.println("Kraj");
}
}
```

Mjerenje triju udaljenosti implementirano je u tri funkcije *readDistanceFront()*, *readDistanceLeft()* i *readDistanceRight()*. U funkcijama za desno i naprijed, poziva se metoda *ping_cm()* iz pripadne instance klase *NewPing* koja vraća izmjerenu udaljenost u centimetrima kao cijeli broj. Za razliku od te dvije udaljenosti, gdje nam je jedino važno postoji li prepreka ili ne, pomoću lijeve udaljenosti vrši se centriranje kolica unutar staze labirinta, pa nam je potrebna veća točnost. Zato se unutar te funkcije poziva metoda *ping_median(4)* koja četiri puta vrši mjerenje trajanja *echo* impulsa, te vraća njihov medijan. Trajanje impulsa je onda iskorišteno za izračun udaljenosti u centimetrima, a rezultat je spremljen u decimalnu, a ne u cjelobrojnu varijablu. U sve tri funkcije nakon dobivene udaljenosti slijedi *if* naredba. Naime, ako senzori nemaju prepreku unutar mjernog opsega, povratna vrijednost funkcija iz biblioteke je 0, što nam ne odgovara u ostatku koda, pa je u tom slučaju, unutar *if* naredbe udaljenost postavljena na udaljenost veću od maksimalne.

```
float readDistanceFront(){
 delay(70);
 int cm = sonarFront.ping_cm();
 if (cm == 0){
  cm=250;
  }
 return cm;
}

float readDistanceLeft(){
 delay(70);
 float duration = sonarLeft.ping_median(4);
 float cm =  duration*0.034/2;
 if (cm == 0){
  cm=250;
 }
 Serial.println(cm);
 return cm;
}

float readDistanceRight(){
 delay(70);
 int cm = sonarRight.ping_cm();
 if (cm == 0){
  cm=250;
  }
 return cm;
}
```

Funkcija *moveStop()* kao što ime govori je funkcija koja zaustavlja kolica. Metoda *run()* je dio biblioteke za *motor shield* te pomoću nje odabiremo smjer kretanja definiranog motora. U ovoj funkciji, metoda je pozvana četiri puta, za svaki pojedini motor, s argumentom RELEASE koja zaustavlja motor. Također u *moveStop()* funkciji varijablu *goesStop* postavljamo na *true*. Spomenuta varijabla je potrebna u sljedećoj funkciji.

```
void moveStop(){
  motorRB.run(RELEASE);
  motorRF.run(RELEASE);
  motorLF.run(RELEASE);
  motorLB.run(RELEASE);
  goesStop = true;

}
```

Funkcija *moveForward()* je funkcija koja pokreće kolica unaprijed. U njoj imamo jedno grananje s *if* naredbom, ovisno da li su kolica u prethodnoj iteraciji mirovala ili su se već gibala unaprijed. Naime, korišteni elektromotori na sebi imaju premale reduktore, tj. kada se motori vrte nazivnom brzinom kolica su se kretala prebrzo. Kao posljedica toga senzori nisu mogli dovoljno brzo očitavati vrijednosti te Arduino nije dovoljno brzo upravljao s elektromotorima pa bi se kolica, npr. u slučaju dolaska na zid ispred, zabila u zid. Zbog toga je trebalo drastično smanjiti brzinu vrtnje. Korišteni *motor shield* uz upravljanje smjerom motora može upravljati i brzinom vrtnje pojedinih motora. Spomenuto upravljanje brzinom vrši se pomoću metode *setSpeed()* koja kao argument prima cjeli broj od 0 do 255. Taj broj predstavlja *duty cycle* PWM signala korištenog za regulaciju brzine. U (3.4) možemo uočiti načine regulacije brzine istosmjernog elektromotora. Vidimo da regulaciju postižemo na tri načina: promjenom napona napajanja, promjenom magnetskog toka u stroju te promjenom otpora armature. Promjenu magnetskog toka u stroju ne možemo provesti iz razloga što magnetski tok u stroju stvara uzbuda, a u našem slučaju uzbuda je permanentni magnet. Promjena otpora armature provodi se dodavanjem otpora u armaturni krug, no to se u praksi ne izvodi jer povećavamo gubitke. Jedini preostali način je regulacija brzine vrtnje promjenom napona. Iz (3.4) vidimo da smanjenjem napona smanjujemo brzinu vrtnje. To možemo učiniti smanjivanjem efektivne vrijednosti napona korištenjem PWM signala. PWM signal je u biti istosmjerni signal koji se sastoji od perioda kada on ima svoju najveću vrijednost te perioda kada ima vrijednost 0. Povećavanjem trajanja perioda kada je vrijednost signala 0 smanjujemo efektivnu vrijednost napona te tako dobivamo smanjenje brzine. S tako smanjenom brzinom vrtnje motora, nastali su problemi prilikom pokretanja kolica, tj. motori nisu mogli razviti dovoljan moment za pokretanje kolica, što je

rješeno pomoću prethodno spomenute varijable *goesStop*. Kada kolica kreću s mjesta motori se pokreću s većim naponom koji će se onda u sljedećoj iteraciji *void loop()* funkcije smanjiti. Također u slučaju da su se kolica za vrijeme kretanja s mjesta pri većem naponu našla preblizu zidu ispred, kolica se zaustavljaju te se funkcijom *return* izlazi iz ove funkcije.

```
void moveForward(){
if(!goesForward){
  goesForward = true;

  if(goesStop == true) {

  motorRB.run(FORWARD);
  motorRF.run(FORWARD);
  motorLB.run(FORWARD);
  motorLF.run(FORWARD);
  motorRB.setSpeed(60);
  motorRF.setSpeed(60);
  motorLB.setSpeed(80);
  motorLF.setSpeed(80);
  distanceF = readDistanceFront();
    if (distanceF < Front) {
     moveStop();
     return;
    }
  delay(30);

  goesStop = false;

  motorRB.run(FORWARD);
  motorRF.run(FORWARD);
  motorLB.run(FORWARD);
  motorLF.run(FORWARD);
  motorRB.setSpeed(40);
  motorRF.setSpeed(40);
  motorLB.setSpeed(45);
  motorLF.setSpeed(45);
  }
  else {

  motorRB.run(FORWARD);
  motorRF.run(FORWARD);
  motorLB.run(FORWARD);
  motorLF.run(FORWARD);
  motorRB.setSpeed(40);
  motorRF.setSpeed(40);
  motorLB.setSpeed(45);
  motorLF.setSpeed(45);
  goesStop = false;
  }

  }

}
```

Funkcije *slightRight() i slightLeft()* služe za centriranje kolica unutar staze labirinta. Ta funkcionalnost je implementirana tako da kolica najprije stanu, pa se blago okrenu lijevo ili desno, opet staju te nastavljaju kretnju uz kratkotrajni povećani napon radi prije opisanog problema. Također za vrijeme izvršavanja ovih funkcija, ako se kolica nađu preblizu zidu ispred, kretnja se prekida te se vraćamo u *void loop()* funkciju.

```
void slightRight() {
  motorRB.run(RELEASE);
  motorRF.run(RELEASE);
  motorLF.run(RELEASE);
  motorLB.run(RELEASE);
delay(200);

  motorRB.run(FORWARD);
  motorRF.run(FORWARD);
  motorLB.run(FORWARD);
  motorLF.run(FORWARD);
  motorRB.setSpeed(0);
  motorRF.setSpeed(0);
  motorLB.setSpeed(60);
  motorLF.setSpeed(60);
    distanceF = readDistanceFront();
    distanceL = readDistanceLeft() - 1;
    distanceR = readDistanceRight();
        if (distanceF < Front) {
         moveStop();
         return;
        }
        if (distanceL > Left) {
        moveStop();
        return;
        }
  delay(20);

  motorRB.run(RELEASE);
  motorRF.run(RELEASE);
  motorLF.run(RELEASE);
  motorLB.run(RELEASE);
  delay(200);

  motorRB.run(FORWARD);
  motorRF.run(FORWARD);
  motorLB.run(FORWARD);
  motorLF.run(FORWARD);
  motorRB.setSpeed(60);
  motorRF.setSpeed(60);
  motorLB.setSpeed(80);
  motorLF.setSpeed(80);
    distanceF = readDistanceFront();
    distanceL = readDistanceLeft() - 1;
    distanceR = readDistanceRight();
      if (distanceF < Front) {
       moveStop();
       return;
```

```
      }
      if (distanceL > Left) {
      moveStop();
      return;
     }
  delay(30);

  motorRB.run(FORWARD);
  motorRF.run(FORWARD);
  motorLB.run(FORWARD);
  motorLF.run(FORWARD);
  motorRB.setSpeed(40);
  motorRF.setSpeed(40);
  motorLB.setSpeed(45);
  motorLF.setSpeed(45);
    distanceF = readDistanceFront();
    distanceL = readDistanceLeft() - 1;
    distanceR = readDistanceRight();
      if (distanceF < Front) {
      moveStop();
      return;
      }
      if (distanceL > Left) {
      moveStop();
      return;
      }
  delay(300);


}

void slightLeft() {
  motorRB.run(RELEASE);
  motorRF.run(RELEASE);
  motorLF.run(RELEASE);
  motorLB.run(RELEASE);
  delay(200);
  motorRB.run(FORWARD);
  motorRF.run(FORWARD);
  motorLB.run(FORWARD);
  motorLF.run(FORWARD);
  motorRB.setSpeed(60);
  motorRF.setSpeed(60);
  motorLB.setSpeed(0);
  motorLF.setSpeed(0);
    distanceF = readDistanceFront();
    distanceL = readDistanceLeft() - 1;
    distanceR = readDistanceRight();
    if (distanceF < Front) {
     moveStop();
     return;
    }
    if (distanceL > Left) {
     moveStop();
     return;
    }
  delay(20);
  motorRB.run(RELEASE);
  motorRF.run(RELEASE);
```

```
  motorLF.run(RELEASE);
  motorLB.run(RELEASE);
  delay(200);

  motorRB.run(FORWARD);
  motorRF.run(FORWARD);
  motorLB.run(FORWARD);
  motorLF.run(FORWARD);
  motorRB.setSpeed(60);
  motorRF.setSpeed(60);
  motorLB.setSpeed(80);
  motorLF.setSpeed(80);
    distanceF = readDistanceFront();
    distanceL = readDistanceLeft() - 1;
    distanceR = readDistanceRight();
    if (distanceF < Front) {
     moveStop();
     return;
    }
    if (distanceL > Left) {
     moveStop();
     return;
    }
  delay(30);

  motorRB.run(FORWARD);
  motorRF.run(FORWARD);
  motorLB.run(FORWARD);
  motorLF.run(FORWARD);
  motorRB.setSpeed(40);
  motorRF.setSpeed(40);
  motorLB.setSpeed(45);
  motorLF.setSpeed(45);
    distanceF = readDistanceFront();
    distanceL = readDistanceLeft() - 1;
    distanceR = readDistanceRight();
    if (distanceF < Front) {
     moveStop();
     return;
    }
    if (distanceL > Left) {
     moveStop();
     return;
    }
  delay(200);


}
```

Funkcije *turnLeft() i turnRight()* zakreću kolica ulijevo ili udesno, te se sastoje od stajanja, blagog kretanja unazad (da se udalje od zidova naprijed), ponovnog stajanja, zakretanja kolica, ponovnog stajanja, te nastavljanja kretnje unaprijed.

```
void turnLeft(){
  motorRB.run(RELEASE);
```

```
motorRF.run(RELEASE);
motorLF.run(RELEASE);
motorLB.run(RELEASE);
goesStop = false;
delay(200);

motorRB.run(BACKWARD);
motorRF.run(BACKWARD);
motorLB.run(BACKWARD);
motorLF.run(BACKWARD);
motorRB.setSpeed(60);
motorRF.setSpeed(60);
motorLB.setSpeed(60);
motorLF.setSpeed(60);
delay(200);

motorRB.run(RELEASE);
motorRF.run(RELEASE);
motorLF.run(RELEASE);
motorLB.run(RELEASE);
goesStop = false;
delay(300);

motorRB.run(FORWARD);
motorRF.run(FORWARD);
motorLB.run(BACKWARD);
motorLF.run(BACKWARD);
motorRB.setSpeed(90);
motorRF.setSpeed(90);
motorLB.setSpeed(90);
motorLF.setSpeed(90);
delay(450);

motorRB.run(RELEASE);
motorRF.run(RELEASE);
motorLF.run(RELEASE);
motorLB.run(RELEASE);
goesStop = false;
delay(200);

motorRB.run(FORWARD);
motorRF.run(FORWARD);
motorLB.run(FORWARD);
motorLF.run(FORWARD);
motorRB.setSpeed(60);
motorRF.setSpeed(60);
motorLB.setSpeed(80);
motorLF.setSpeed(80);
delay(30);

motorRB.run(FORWARD);
motorRF.run(FORWARD);
motorLB.run(FORWARD);
motorLF.run(FORWARD);
motorRB.setSpeed(40);
motorRF.setSpeed(40);
motorLB.setSpeed(45);
motorLF.setSpeed(45);
delay(400);
```

```
return;
}

void turnRight(){
  motorRB.run(RELEASE);
  motorRF.run(RELEASE);
  motorLF.run(RELEASE);
  motorLB.run(RELEASE);
  goesStop = false;
  delay(200);

  motorRB.run(BACKWARD);
  motorRF.run(BACKWARD);
  motorLB.run(BACKWARD);
  motorLF.run(BACKWARD);
  motorRB.setSpeed(60);
  motorRF.setSpeed(60);
  motorLB.setSpeed(60);
  motorLF.setSpeed(60);
  delay(200);

  motorRB.run(RELEASE);
  motorRF.run(RELEASE);
  motorLF.run(RELEASE);
  motorLB.run(RELEASE);
  goesStop = false;
  delay(200);

  motorRB.run(BACKWARD);
  motorRF.run(BACKWARD);
  motorLB.run(FORWARD);
  motorLF.run(FORWARD);
  motorRB.setSpeed(90);
  motorRF.setSpeed(90);
  motorLB.setSpeed(90);
  motorLF.setSpeed(90);
  delay(350);

  motorRB.run(RELEASE);
  motorRF.run(RELEASE);
  motorLF.run(RELEASE);
  motorLB.run(RELEASE);
  goesStop = false;
  delay(200);

  motorRB.run(FORWARD);
  motorRF.run(FORWARD);
  motorLB.run(FORWARD);
  motorLF.run(FORWARD);
  motorRB.setSpeed(40);
  motorRF.setSpeed(40);
  motorLB.setSpeed(60);
  motorLF.setSpeed(45);
  delay(100);
return;
}
```

Funkcija *moveRotate()* zakreće kolica za 180°. U slučaju da se kolica u trenutku zaustavljanja pred slijepom ulicom nalaze bliže lijevom zidu, okret će se provesti udesno, a ako se nalaze bliže desnom zidu okret će se provesti ulijevo. Takav uvjet smanjuje mogućnost zapinjanja kolica u zidove labirinta. Okret se također sastoji prvo od zaustavljanja pa blagog kretanja unatrag da bi se kolica odmakla od zida ispred sebe. Nakon toga senzori lijevo i desno mjere udaljenosti te na temelju usporedbe se izvršava okret u pripadajuću stranu. Nakon toga, s povećanim naponom na motorima prilikom pokretanja, kolica nastavljaju kretnju naprijed.

```
void moveRotate(){

 motorRB.run(RELEASE);
 motorRF.run(RELEASE);
 motorLF.run(RELEASE);
 motorLB.run(RELEASE);
 goesStop = false;
 delay(200);

 motorRB.run(BACKWARD);
 motorRF.run(BACKWARD);
 motorLB.run(BACKWARD);
 motorLF.run(BACKWARD);
 motorRB.setSpeed(60);
 motorRF.setSpeed(60);
 motorLB.setSpeed(60);
 motorLF.setSpeed(60);
 delay(200);

 motorRB.run(RELEASE);
 motorRF.run(RELEASE);
 motorLF.run(RELEASE);
 motorLB.run(RELEASE);
 goesStop = false;
 delay(1000);

distanceL = readDistanceLeft() - 1;
distanceR = readDistanceRight();
delay(200);

if (distanceL > distanceR) {

 motorRB.run(FORWARD);
 motorRF.run(FORWARD);
 motorLB.run(BACKWARD);
 motorLF.run(BACKWARD);
 motorRB.setSpeed(80);
 motorRF.setSpeed(80);
 motorLB.setSpeed(80);
 motorLF.setSpeed(80);
 delay(1050);
}
else if (distanceR > distanceL) {
 motorRB.run(BACKWARD);
```

```
 motorRF.run(BACKWARD);
 motorLB.run(FORWARD);
 motorLF.run(FORWARD);
 motorRB.setSpeed(80);
 motorRF.setSpeed(80);
 motorLB.setSpeed(80);
 motorLF.setSpeed(80);
 delay(850);
}

 motorRB.run(RELEASE);
 motorRF.run(RELEASE);
 motorLF.run(RELEASE);
 motorLB.run(RELEASE);
 goesStop = false;
 delay(500);

 motorRB.run(FORWARD);
 motorRF.run(FORWARD);
 motorLB.run(FORWARD);
 motorLF.run(FORWARD);
 motorRB.setSpeed(60);
 motorRF.setSpeed(60);
 motorLB.setSpeed(80);
 motorLF.setSpeed(80);
 delay(30);

 motorRB.run(FORWARD);
 motorRF.run(FORWARD);
 motorLB.run(FORWARD);
 motorLF.run(FORWARD);
 motorRB.setSpeed(40);
 motorRF.setSpeed(40);
 motorLB.setSpeed(45);
 motorLF.setSpeed(45);
 delay(50);
return;
}
```

10. ZAKLJUČAK

U ovom završnom radu prikazan je koncept i sam postupak izrade robotskih kolica za rješavanje labirinta temeljena na Arduino platformi. Potreba za robotima za rješavanje labirinata u današnjem društvu pojavljuje se u operacijama traganja i spašavanja. U scenarijima katastrofa, kao što su potresi ili urušavanje zgrada, ovi roboti mogu prolaziti kroz ruševine i ostatke kako bi locirali preživjele. Također, još jedna primjena je u skladištima i tvornicama, gdje se roboti za rješavanje labirinata mogu se koristiti za optimizaciju kretanja robe i materijala što u konačnici predstavlja efikasnije poslovanje. Uz sam postupak izrade kolica, detaljno su analizirane sve komponente te načela vezana za rad istih. Također, objašnjeni su algoritmi potrebni za pravilno funkcioniranje samih kolica u nepoznatom labirintu.

Kolica opisana u ovom završnom radu pojednostavljeni su oblik robota korištenih za prije spomenute zadatke. Kako i kod većine električnih uređaja, tako i kod ovih kolica, uvijek postoji prostora za unaprjeđenje. Pa tako jedan način poboljšanja rada samih kolica bi bilo korištenje koračnih motora umjesto istosmjernih elektromotora. Iako bismo izgubili na brzini samih kolica, postigli bi puno veću kontrolu nad rotacijom te mogućnost povratne informacije o rotaciji pojedinih kotača. Povratne informacije o rotaciji nam omogućuju implementaciju algoritama u slučaju kvarova ili nekih nepredviđenih problema npr. da kolica negdje zapnu ili slično.

Još jedan način poboljšanja kolica bi bilo korištenje akcelerometara i žiroskopa kao što je MPU6050. Prednost korištenja ovakvog modula je ta da bi mogli dobivati povratne informacije o stupnju skretanja robota. Pomoću ovih informacija mogli bi napraviti da robot radi savršene okrete za 90° ili 180° neovisno o vrsti podloge. Također bi mogli mjeriti pređene udaljenosti što bi omogućilo izradu 2D prikaza samog labirinta, te naposljetku korištenjem optimizacijskih algoritama pronaći najkraći put kroz labirint.

11. LITERATURA

[1] Autor nepoznat, „Trademark & Copyright“, s interneta, https://www.arduino.cc/en/trademark, 25.09.2020.

[2] Autor nepoznat, „Arduino Hardware”, s internet, https://www.arduino.cc/en/hardware, 11.04.2022.

[3] Jobit Joseph, „Everything you need to know about the Arduino Hardware“, s interneta, https://circuitdigest.com/article/everything-you-need-to-know-about-arduino-uno-board-hardware, 21.03.2022.

[4] Autor nepoznat, „ Arduino Uno Rev3 - schematic“, s interneta, https://www.arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf, 20.02.2012.

[5] Autor nepoznat, „Arduino IDE“, s interneta, https://www.javatpoint.com/arduino-ide, datum nepoznat

[6] Autor nepoznat, „Motor sa prijenosom i kotačem“, s interneta, https://www.chipoteka.hr/elektronika-i-alati/elektronika/robotika/motor-sa-prijenosom-i-kotacem-joy-it-com-motor01-8203100041, datum nepoznat

[7] Autor nepoznat, „DC Motor Manufacturers | DC Motor Suppliers“, s interneta, https://www.electric-motors.net/dc-motors/, datum nepoznat

[8] Leksikografski zavod Miroslav Krleža, Tehnički leksikon - mrežno izdanje, s interneta, https://tehnicki.lzmk.hr/clanak/pravilo-lijeve-ruke, 2007.

[9] Volarić Ivan, „Istosmjerni motori“, predavanje iz predmeta „Elementi automatizacije postrojenja“, datum nepoznat

[10] Hashmi Fahad „What is the shape of an induced EMF in an armature conductor of a DC machine and why?“, s interneta, https://www.quora.com/What-is-the-shape-of-an-induced-EMF-in-an-armature-conductor-of-a-DC-machine-and-why, 04.06.2018.

[11] Gašparac Ivan, Žarko Damir, „ELEKTROMOTORNI POGONI S ISTOSMJERNIM MOTORIMA“, predavanje iz predmeta „Elektromotorni pogoni“, FER, 18.03.2015.

[12, 13, 14] Autor nepoznat, „Prva auditorna vježba - radni mehanizmi“, auditorne vježbe iz predmeta „Elektromotorni pogoni“, datum nepoznat

[15] Autor nepoznat, „HC-SR04 Ultrasonic Sensor Guide with Arduino Interfacing“, s interneta, https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.electronicwings.com%2Fsensors-modules%2Fultrasonic-module-hc-sr04&psig=AOvVaw36CHSs81NveBdy7EoLnOY-&ust=1724439138745000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCND41cSiiYgDFQAAAAAdAAAAABAQ, datum nepoznat

[16] Autor nepoznat, „hy-srf05-main“, s interneta, https://www.artekit.eu/wp-content/uploads/hy-srf05-main.jpg, datum nepoznat

[17] Autor nepoznat, „Motor shield sa L293D, za Arduino, VMA207“, s interneta, https://www.chipoteka.hr/elektronika-i-alati/elektronika/arduino-elementi/motor-shield-sa-l293d-za-arduino-vma207-9150036250, datum nepoznat

[18] Autor nepoznat, „H-bridge“, s interneta, https://en.wikipedia.org/wiki/H-bridge#/media/File:H_bridge.svg, 02.07.2024.

[19, 20] Autor nepoznat, „How 74HC595 Shift Register Works & Interface it with Arduino“, s interneta, https://lastminuteengineers.com/74hc595-shift-register-arduino-tutorial/, datum nepoznat

[21] Autor nepoznat, „Maze Generator“, s interneta, https://www.mazegenerator.net/, datum nepoznat

12. SAŽETAK I KLJUČNE RIJEČI

U sklopu ovog završnog rada izrađena su robotska kolica s pogonom na sva četiri kotača, tj. s četiri DC motora. Robotska kolica su opremljena s tri ultrazvučna senzora udaljenosti, tipa HY-SRF05, orijentirana lijevo, desno, te ispred kolica. Tri senzora, te četiri H-mosta za upravljanje motorima, implementiranih u *motor driver*-u s dva L293D integrirana kruga, spojena su na Arduino Uno razvojnu pločicu. Također, detaljno su opisana načela rada svih komponenti te algoritmi i programska podrška potrebni za rješavanje samog labirinta.

Ključne riječi: labirint, robotska kolica, Arduino, ultrazvučni senzori, istosmjerni motori

13. SUMMARY AND KEYWORDS

In this bachelors thesis, a robotic car with four-wheel drive, i.e. with four DC motors, has been constructed. The robotic car is equipped with three ultrasonic distance sensors, type HY-SRF05, oriented to the left, right, and in front of the cart. Three sensors, and four H-bridges for motor control, which are implemented in a motor driver with two L293D integrated circuits, are connected to the Arduino Uno development board. Also, the operation principles of all components have been described in details, as well as the algorithms and software support needed in order to solve the maze itself.

Keywords: maze, robotic car, Arduino, ultrasonic sensors, DC motors