Projektiranje i izrada robotskog sustava koji igra šah

Herceg-Rušec, Matija

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:235:320747

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Matija Herceg-Rušec

Zagreb, 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor:

Prof. dr. sc. Dubravko Majetić

Student:

Matija Herceg-Rušec

Zagreb, 2021.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se profesoru Dubravku Majetiću na mentorstvu te pruženoj pomoći i mnogobrojnim savjetima koji su mi uvelike olakšali izradu ovog diplomskog rada.

Posebno se zahvaljujem svima koji su na bilo koji način pripomogli u fizičkoj realizaciji dijelova robota: tokarenje, glodanje, bušenje, rezanje i savijanje lima.

Također, zahvaljujem se obitelji i djevojci Ivani na potpori i strpljenju tijekom čitavog studija.

Matija Herceg-Rušec

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske radove studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment,
inženjerstvo materijala te mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum:	Prilog:
Klasa: 602-04/21-6/1	
Ur. broj: 15-1703-21	

DIPLOMSKI ZADATAK

Student: **MATIJA HERCEG-RUŠEC** Mat. br.: 0035208477

Naslov rada na hrvatskom jeziku: **Projektiranje i izrada robotskog sustava koji igra šah**

Naslov rada na engleskom jeziku: **Design and implemenation of chess-playing robotic system**

Opis zadatka:

Robotski sustav koji igra šah predstavlja izazovni mehatronički zadatak. Za njegovu izvedbu potrebno je kombinirati znanja iz područja strojarstva, elektrotehnike, automatizacije, umjetne inteligencije i programiranja. Sam sustav se sastoji od nekoliko složenih podsustava. To su robotska ruka za pomicanje šahovskih figura, elektronička šahovska ploča i algoritam umjetne inteligencije koji generira poteze robota kao odgovor na poteze protivnika.

U radu treba načiniti sljedeće:
1. Projektirati i izraditi robotsku ruku koja je pokretana koračnim i servo motorima te je upravljana pomoću mikrokontrolera.
2. Predložiti i implementirati odgovarajući mikrokontroler.
3. Projektirati i izraditi elektroničku šahovsku ploču s pripadajućim figurama koja će robotu omogućili lakše pozicioniranje i prihvat šahovskih figura.
4. Predložiti i primijeniti algoritam umjetne inteligencije, a u ovom slučaju umjetne neuronske mreže, koja će učiti šahovske poteze iz dostupnih baza šahovskih igara.
5. Izvesti zaključke rada.
6. Navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan: 6. svibnja 2021.

Rok predaje rada: 8. srpnja 2021.

Predviđeni datum obrane: 12. srpnja do 16. srpnja 2021.

Zadatak zadao: prof. dr. sc. Dubravko Majetić

Predsjednica Povjerenstva: prof. dr. sc. Biserka Runje

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

Oznaka	Jedinica	Opis
q_1, q_2, q_3	rad	Kutovi zakreta baze i krakova robotske ruke
p_x, p_y, p_z	mm	Pomaci prihvatnice robota u odnosu na osi k.s.
p_{y2}	mm	Pomak prihvatnice po osi Y lokalnog k.s.
L_1, L_2, L_3, L_4, L_5	mm	Dimenzije krakova robotske ruke
s	rad	Put (kut) kretanja motora
v	rad/s	Brzina kretanja motora
a	rad/s^2	Ubrzanje motora
t	s	Vrijeme

SAŽETAK

Ovaj diplomski rad prikazuje način rada i postupak izrade robotske ruke koja igra šah na bazi umjetne inteligencije. Prikazan je postupak odabira aktuatora i upravljačkih komponenti te sama konstrukcija robotske ruke i elektroničke šahovske ploče. Nakon konstrukcije i izrade, na temelju stvarnih dimenzija prikazana je inverzna kinematika, odnosno pretvorba željenih Kartezijevih koordinata u rotacijske pozicije zglobova robota. Napravljen je program za mikrokontroler koji će upravljati gibanjem robotske ruke te program koji će upravljati elektroničkom šahovskom pločom. Nakon što je fizički dio gotov, objašnjena je inteligencija koja će odlučivati o sljedećem šahovskom potezu robota. Prikazani su i objašnjeni načini na koje se može ostvariti odabir poteza te dobre i loše strane istih. Nakon toga je formiran algoritam za učenje neuronske mreže te prikazana primjena konačnog modela neuronske mreže. Na kraju je ukratko objašnjeno kreiranje mobilne aplikacije kojom će se upravljati postavkama robota. Tijekom izrade rada korišteni su programski paketi SolidWorks, MATLAB, Arduino, Android Studio, Python IDLE, Tensorflow.

Ključne riječi: šahovski robot, robotska ruka, šah, umjetna inteligencija

SUMMARY

This thesis presents the procedure of making a robotic arm that plays chess by artificial intelligence. The methods of selecting actuators and control components, construction of robotic arm and development of electronic chessboard are presented. After construction and fabrication, based on the actual dimensions, the inverse kinematics is shown, i.e. the conversion of Cartesian coordinates in the rotational positions of the robot joints. A program for a microcontroller that will control the movement of the robotic arm and a program that will control the electronic chessboard were created. Once the mechanical and electronical part is done, the intelligence that will decide the robot's next chess move is explained. The ways in which the selection of moves is made and the pros and cons of every method are presented and explained. Then a neural network learning algorithm is formed and application of a final neural network model is presented. Finally, the process of creating mobile application that will manage robot settings is briefly explained. SolidWorks, MATLAB, Arduino, Android Studio, Python IDLE, Tensorflow software were used while working on the project.

Key words: chess playing robot, robotic arm, chess, artificial intelligence

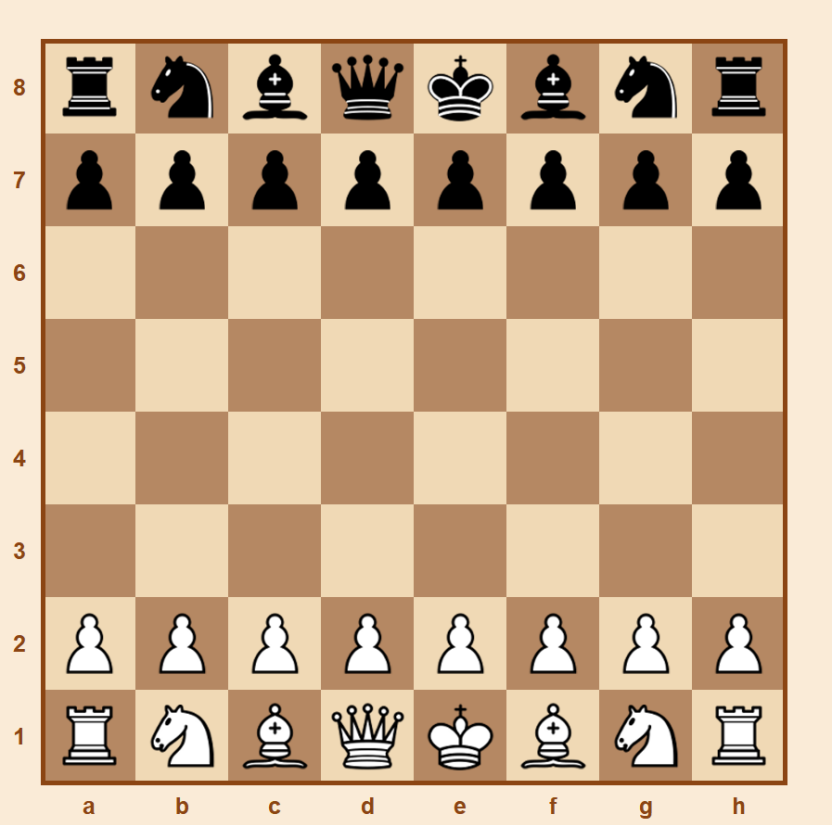
1. UVOD

Ideja ovog diplomskog rada je prikazati mogućnost povezivanja mnoštva različitih elektromehaničkih komponenata u smislenu cjelinu na primjeru robotskog sustava koji igra šah. Šah je jedna od zanimljivijih igara na kojoj se vrlo lako mogu isprobavati nove tehnologije. Upravo to se dogodilo 1997. godine [1], kada je prvi puta računalo pobijedilo svjetskog prvaka Garrya Kasparova u šahu. Kako se kroz godine tehnologija razvijala, u današnje vrijeme možemo vidjeti primjere (slika 1.) gdje robotska ruka igra šah protiv Vladimira Kramnika, te time približava računalo čovjeku. Kako bi se predstavio jedan ovakav sustav, u sklopu ovog rada biti će prezentirani svi dijelovi tog sustava. Prezentirat će se sve od konstrukcije robota, programiranja, izrade elektroničke ploče i na kraju do umjetne inteligencije koja određivati koji potezi trebaju biti odigrani od strane robota. Cilj ovog rada nije napraviti savršeno precizni robot visokih performansi i umjetnu inteligenciju koju je nemoguće pobijediti, već je cilj prikazati način na koji se može pomoću hobističkih komponenti napraviti dovoljno dobro konstrukcijsko rješenje koje je pokretano dovoljno dobrom umjetnom inteligencijom da jednu partiju šaha učini zanimljivom. Ovakav jedan robotski sustav se sastoji od mnogo dijelova te će i nakon završetka izrade uvijek biti mjesta za poboljšanje.

Slika 1. Robot koji igra šah protiv Vladimira Kramnika [1]

2. ŠAHOVSKE OSNOVE

Da bismo uopće mogli koristiti šahovsku terminologiju, potrebno je prvo objasniti osnove šaha. Šah je igra za dva igrača (crni i bijeli), gdje se naizmjenično povlače potezi počevši od bijelog igrača. Potezi se ne mogu preskakati. Šahovska ploča se sastoji od 64 polja, gdje svako polje ima svoje ime prema redu i stupcu (npr. a1, e7,…). Na slici 2. vidljiva su šahovska polja i početna pozicija svih figura. Figura na polju a2 zove se *pješak* (ili pijun), na polju a1 zove se *top* (ili kula), na polju b1 zove se *skakač* (ili konj), na polju c1 zove se *lovac*, na d1 zove se *dama* (ili kraljica), a na polju e1 nalazi se najvažnija figura *kralj*. Niti jedna figura osim skakača ne može preskakati druge figure prilikom svog kretanja. Kada jedna figura uzima (ili pojede) drugu figuru, uzeta figura se miče s ploče dok prva figura ostaje. *Pješak* se kreće za jedno polje unaprijed, osim kod prvog poteza tim pješakom, kada može ići za dva polja unaprijed (ali ne mora). Pješak ne može ići natrag, a može uzimati samo kada se protivnička figura nalazi na prvom dijagonalnom polju u smjeru kretanja pješaka. *Top* se giba u bilo kojem smjeru po redovima i stupcima, isto tako može i uzimati figure koje se nađu na njegovom redu ili stupcu. *Skakač* se giba u obliku slova L, odnosno dva polja ravno i jedno polje bočno. *Lovac* je sličan topu, on se giba dijagonalno u bilo kojem smjeru za razliku od topa koji se giba ravno u bilo kojem smjeru. *Kraljica* je najjača figura u šahu i ona je kombinacija gibanja topa i lovca, tj. Može se gibati u bilo kojem smjeru koliko daleko želi. *Kralj* je najvažnija figura jer o njemu ovisi ishod šahovske partije. Kralj se giba za jedno polje u bilo kojem smjeru. Pravilo je da se kralj ne može pomaknuti na polje na kojem je šah (na kojem je napadnut od strane druge figure), a isto tako se niti jedna druga figura ne može pomaknuti ako bi svojim pomicanjem prouzročila šah na svojem vlastitom kralju. *Šah* je, dakle, termin koji označava napadanje na kralja od protivničke figure. *Šah-mat* se dobiva kada je kralj napadnut (šah) a ne može napraviti niti jedan potez koji bi spriječio šah, te to označava kraj igre. Ako se niti jedna druga figura ne može pomaknuti, trenutno nije šah, a svaki potez s kraljem dovodi do šaha, takva pozicija se zove *Pat* te ona dovodi do remija. Kako bi se izbjegla pre duga ili u nekim slučajevima beskonačna igra, uvedena su dodatna pravila. Nakon što se ista pozicija na ploči ponovi 3 puta, igrači mogu zatražiti remi. Ako se u 50 poteza ne dogodi niti jedan pokret pješaka ili uzimanje, tada ponovo igrači mogu tražiti remi. Isto tako remi (izjednačeno) je moguće postići dogovorom.



Slika 2. Šahovska ploča

Postoje i neki posebni potezi koji su malo drugačiji od klasičnih. Poseban pješakov potez je *en passant* kada pješak uzima drugog pješaka koji se giba za 2 polja i dođe do bočne strane prvog pješaka. Pješak se također može takozvanom *promocijom* pretvoriti u bilo koju drugu figuru osim kralja, kada taj pješak dođe unaprijed do zadnjeg polja. Najčešće se pješak promovira u damu jer je ona najjača figura. Rokada (ili rošada) je potez kada se kralj giba za dva polja u lijevo ili u desno a top ga preskače, a moguće ga je odigrati jedino u slučaju da su: i kralj i top netaknuti, da nema drugih figura na poljima između kralja i topa prije nego se napravi rokada, ne smije u trenutku rokade biti šah, a također ni jedno polje između kralja i topa ne smije biti napadnuto. Tijekom šahovske igre, često se čuje riječ *otvaranje* koja označava skup poteza koji se igraju od početne pozicije figura koje igrač zna napamet i koristi ih kako bi odigrao solidnu partiju šaha ili navukao protivnika na tok partije na koji je samo on spreman.

3. PRINCIP RADA ŠAHOVSKOG ROBOTA

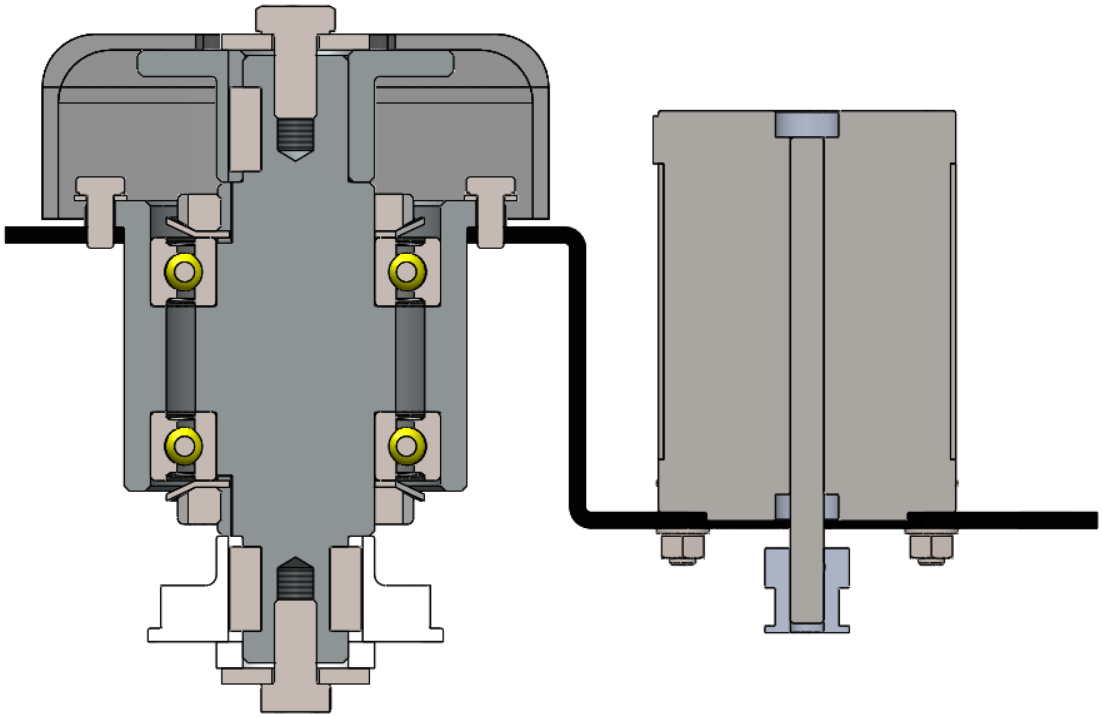
Kako bi se pravilno objasnio princip rada, robotski sustav treba podijeliti u funkcionalne cjeline i promatrati funkciju svakog pojedinog dijela. Robotska ruka je glavni pokretni dio koji će obavljati sve pokrete od strane robota. Ona se sastoji od nekoliko krakova povezanih motorima koji su upravljani pomoću mikrokontrolera. Mikrokontroler izračunava inverznu kinematiku i usklađuje pokrete s obzirom na traženu poziciju u koju robot treba doći. Hvataljka se sastoji od servo motora koji omogućuje otvaranje i zatvaranje iste. Sve radnje pomicanja i hvatanja sinkroniziraju se od strane mikrokontrolera. Drugi glavni dio je elektronička šahovska ploča koja se sastoji od mreže hallovih senzora koji detektiraju prisutnost magneta iznad svakog polja. Ispod svake figure je nalijepljen magnet koji aktivira hallov senzor kada se nalazi na polju. Mreža hallovih senzora se učitava pomoću mikrokontrolera te se u memoriji stvara matrica polja označenih s 0 ili 1 ovisno o tome nalazi li se figura na polju ili ne. Svaka šahovska partija počinje s istom početnom pozicijom te se tako praćenjem pomaka figura može jednoznačno odrediti o kojem se odigranom potezu radi. Glavni program koji je pokretan na mini računalu Raspberry Pi povezuje sve dijelove sustava u cjelinu. Taj program preuzima podatke o poziciji iz mikrokontrolera spojenog na elektroničku šahovsku ploču te putem logičkih operacija određuje o kojem se odigranom potezu radi. Nakon što je igračev potez prepoznat, program traži od umjetne inteligencije povratni šahovski potez koji robotska ruka treba odigrati. Kada je taj potez određen, kreira se lista pokreta koje robotska ruka mora napraviti kako bi taj potez bio odigran. Koordinate za te pokrete se slijedno šalju na mikrokontroler povezan na robotsku ruku te se iščekuje povratna informacija da je pokret obavljen. Taj ciklus se obavlja za svaki potez odigran od strane računala. Kako bi računalo znalo koji potez treba odigrati, napravljen je algoritam koji pretražuje moguće poteze do određene dubine (koliko računalo vidi u budućnost šahovske partije). Koristeći taj algoritam, računalo za svaku poziciju poziva istrenirani model neuronske mreže koji vraća svoju procjenu pozicije na ploči. Ovisno o tome koliku ocjenu pozicija dobije, računalo će dalje težiti tom potezu ili ga izbjegavati ukoliko se taj potez pokaže kao loš u budućnosti. Na kraju je napravljena mobilna aplikacija kako bi se postigla dodatna mobilnost i olakšala upotreba šahovskog robota. Upotrebom aplikacije se također može vidjeti je li došlo do razlike između pozicije na ploči i pozicije u memoriji, a samim time i greške u radu.

4. KONSTRUKCIJA ROBOTA I ODABIR ELEKTRONIČKIH KOMPONENTI

Robotski sustav se sastoji od nekoliko osnovnih komponenti, a to su: robotska ruka, prihvatnica, elektronička šahovska ploča, figure i kućište u koje su smještene sve elektroničke komponente. U daljnjem tekstu će svaka komponenta biti posebno razmatrana.

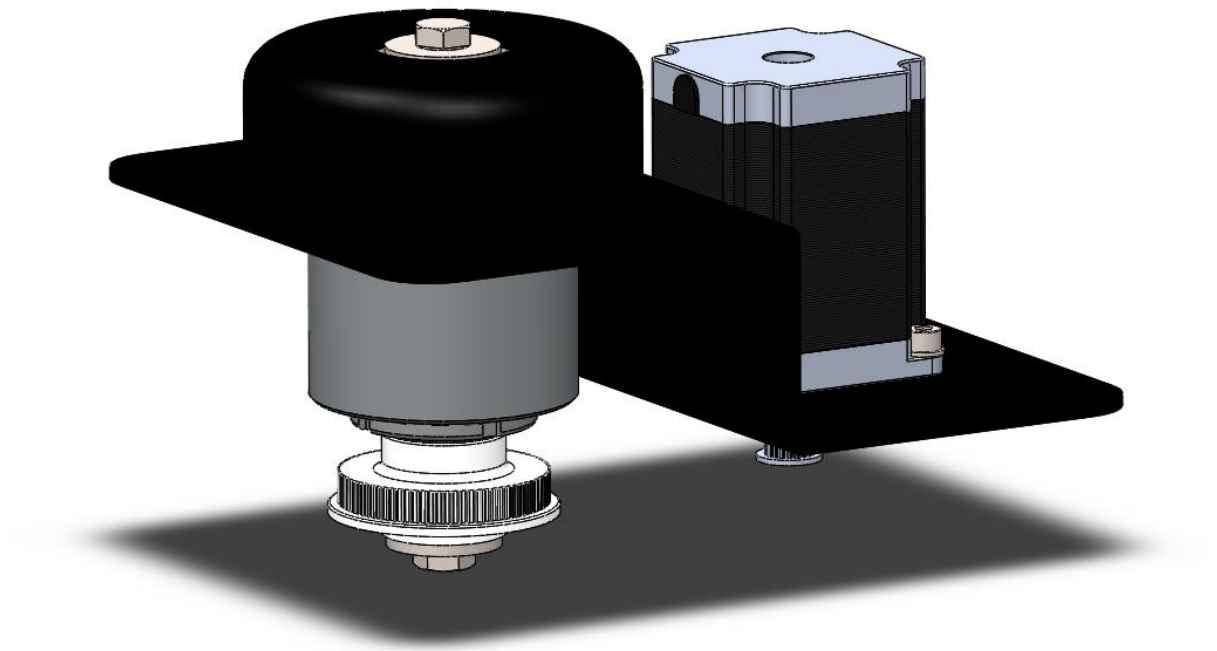
4.1. Robotska ruka

Robotsku ruku možemo ponovo podijeliti u dva dijela: bazu robotske ruke koja mora biti robusnija te koja će se pobrinuti oko okretanja robotske ruke oko Z-osi i gornji dio robotske ruke koji se sastoji od dva kraka kako bi se moglo ostvariti pozicioniranje u prostoru. Na slici 3. mogu se vidjeti svi dijelovi baze u presjeku.



Slika 3. Prikaz presjeka baze robotske ruke

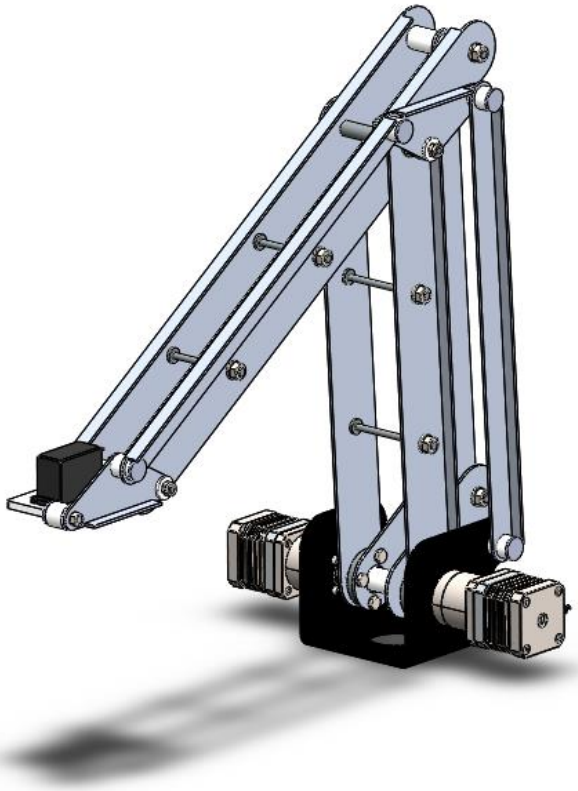
Baza robotske ruke se sastoji od glavnog vratila koje će rotirati robotsku ruku oko Z-osi. Vratilo je uležišteno pomoću 2 ležaja 6006 u kućište vratila, dok je kućište vratila učvršćeno za temeljnu ploču koja će se kasnije ugrađivati kao cjelina na glavno drveno kućište. Na vrhu uležištenja postavljeno je zaštitno zvono koje sprječava ulazak stranih tijela u prostor gdje se okreće vratilo. Vratilo je pokretano pomoću koračnog motora Nema23 preko GT2 remena. Prijenosni omjer između remenice motora i remenice na vratilu je 1:4.



Slika 4. Prikaz sklopa baze robotske ruke

Na bazu robotske ruke (slika 4.), iznad zaštitnog zvona, smješta se gornji dio robotske ruke (slika 5.). Gornji dio se sastoji od nosača na koji se pričvršćuju svi elementi, motora, krakova robota i sustava poluga. Krakovi robotske ruke napravljeni su od para aluminijskih L profila razmaknutih odstojnicima. Aluminij je odličan materijal za izradu robotske ruke zbog svoje relativno male mase. Međutim, aluminij nije posebno krut konstrukcijski materijal, pa su upravo zbog toga korišteni L profili koji pospješuju krutost krakova, a time i preciznost robota. Kako bi se omogućilo kretanje (rotacija) krakova robotske ruke, korišteni su koračni motori Nema17 s reduktorom prijenosnog omjera 1:27. Jedan motor je direktno spojen s donjim krakom robotske ruke, dok je dugi krak spojen s motorom preko poluge.

Za premještanje figura na šahovskoj ploči prihvatnica mora uvijek biti paralelna sa šahovskom pločom. Kako bi se izbjegla upotreba skupih aktuatora, paralelnost prihvatnice je osigurana sustavom poluga koje geometrijski drže prihvatnicu paralelnom. Na glavi robotske ruke pričvršćen je servo motor koji će okretati prihvatnicu oko njene Z-osi. Ta orijentacija je bitna kako bi se prsti prihvatnice uvijek prilikom otvaranja smjestili između figura umjesto da se otvaranjem pomaknu sve figure koje se nalaze u okolini oko figure koja se pokušava premjestiti. Direktno na vratilo servo motora, pričvrstit će se prihvatnica. Robotska ruka je upravljana pomoću mikrokontrolera ESP32 koji preko drivera za koračne motore TB6600 obavlja funkcije rada robotske ruke.

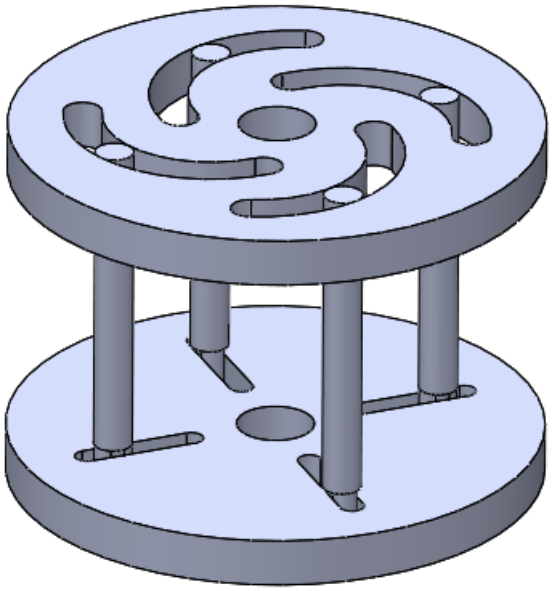


Slika 5. Gornji dio robotske ruke

4.2. Prihvatnica

Važan dio robotskog sustava za šah je sama prihvatnica. Konstrukcija prihvatnice mora biti dobro obavljena kako bi robot figure pomicao ispravno i precizno. Isto tako potrebna je čim veća ušteda prostora, jer je razmak između figura malen. Zbog navedenih uvjeta potrebno je konstruirati novi tip hvataljke koja će zadovoljavati kriterije.

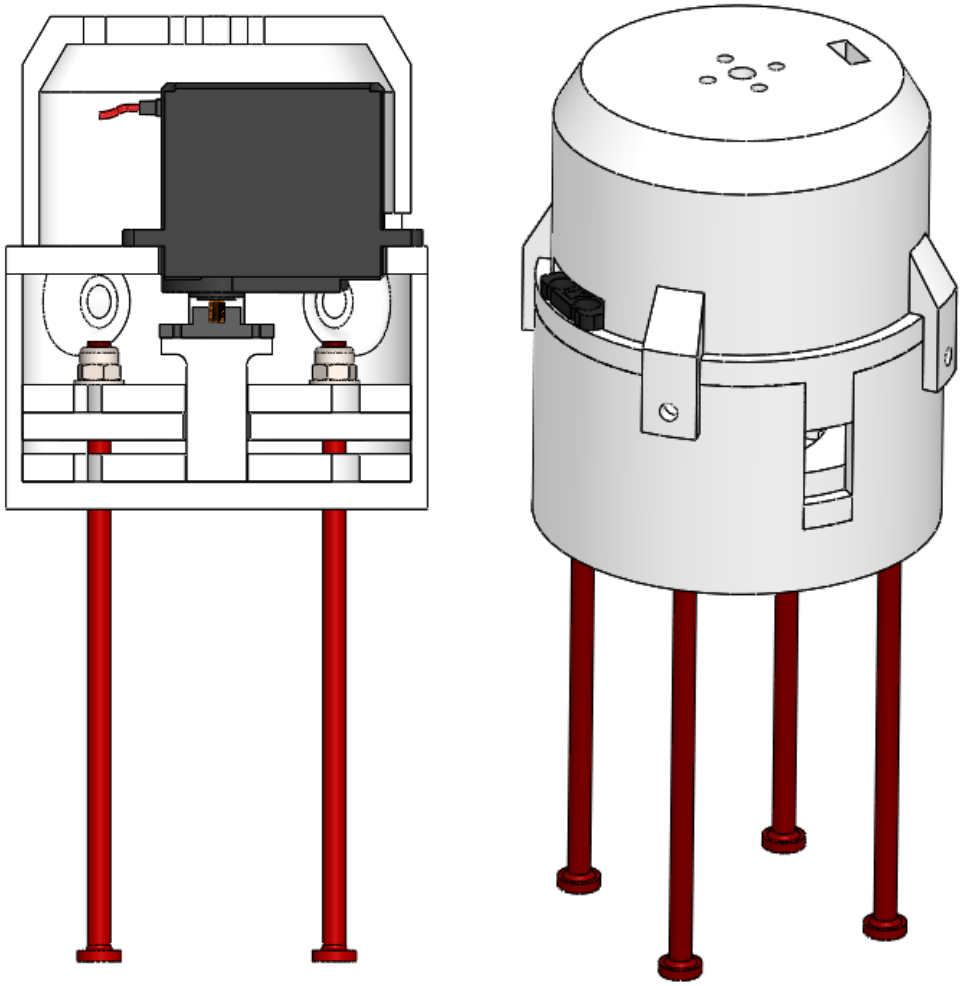
Klasična dvoprstna hvataljka bi mogla zadovoljiti uvijete, no ona prilikom zatvaranja ne centrira figure, a u isto vrijeme zauzima puno mjesta. Kako bi se anulirali ti problemi, prihvatnica je konstruirana na način da se četiri okrugla prsta pomiču sinkrono prema centru uz pomoć dvije različite ploče s utorima. Jedna ploča ima ravne utore, dok druga ploča ima spiralne utore. Kada se te dvije ploče spoje, gledano s tlocrta imamo četiri prolaza u koji se smještaju okrugli prsti prihvatnice. Pomicanjem gornje ploče, simetrično se pomiču svi okrugli prsti, na način da se okretanjem u smjeru kazaljke na satu događa zatvaranje hvataljke, dok se u obrnutom slučaju događa otvaranje. Na slici 6. vizualizirana je ideja osnovnog mehanizma prihvatnice, a ploče su razmaknute kako bi način funkcioniranja bio uočljiviji.



Slika 6. Princip otvaranja i zatvaranja prihvatnice

Kada je definirana osnovna ideja kako će funkcionirati hvataljka, dodatnom razradom rješenja i dodavanjem aktuatora dobiva se konačno konstrukcijsko rješenje. Konačno rješenje se sastoji od dva para simetrično postavljenih ploča s različitim utorima kako bi se smanjio nepoželjan hod prstiju hvataljke prilikom djelovanja sile stezanja. Prsti hvataljke izvedeni su kao čavli na koje je narezan navoj, te su s gornje strane maticom pričvršćeni kako bi se spriječilo ispadanje van hvataljke zbog utjecaja gravitacije. Putem četvrtastog vratila pokretane su spiralne ploče te je time postignuta funkcija otvaranja i zatvaranja.

Servo motor smješten u gornjem dijelu prihvatnice pokreće vratilo kada se želi promijeniti stanje prihvatnice. Cijela prihvatnica (slika 7.) je na kraju oklopljena kako bi se postigao neometan rad.

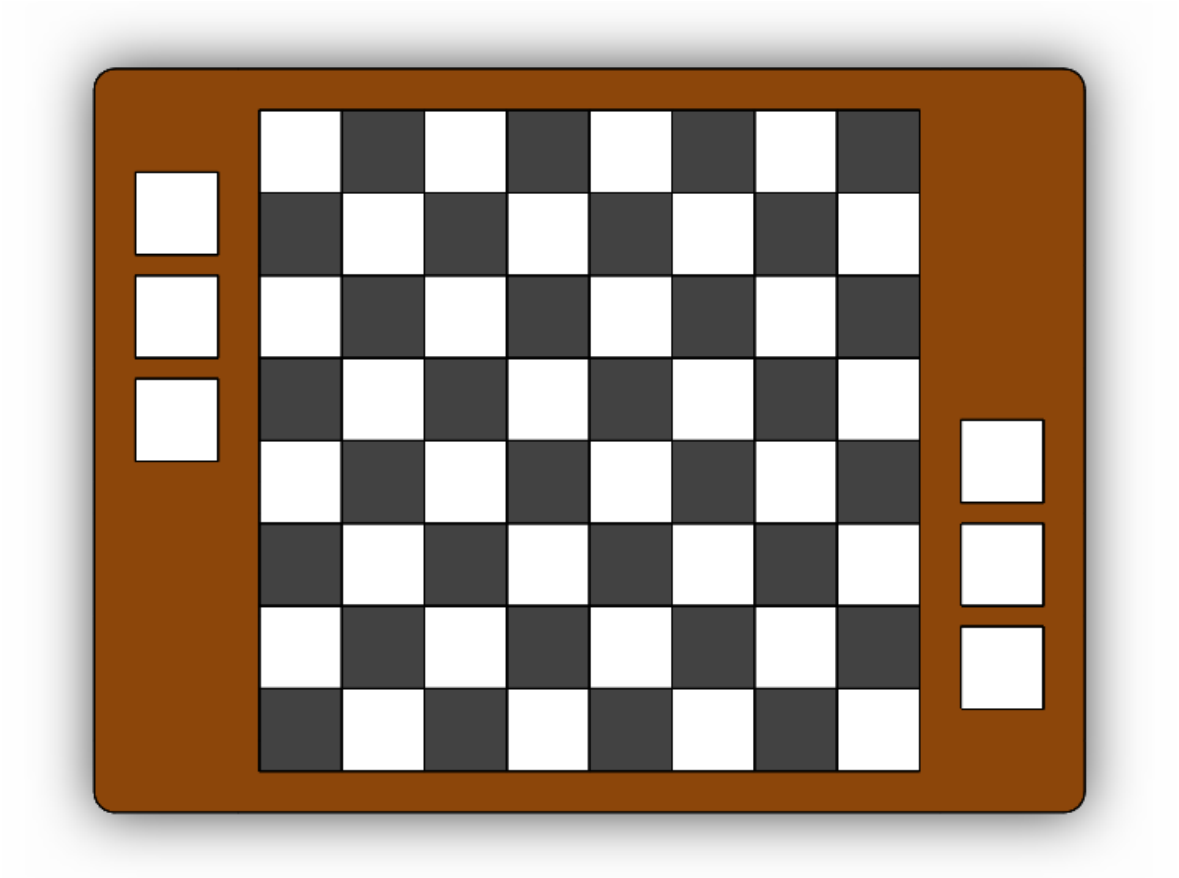


Slika 7. Prikaz sklopa prihvatnice u presjeku (lijevo) i u 3D prikazu (desno)

4.3. Elektronička šahovska ploča

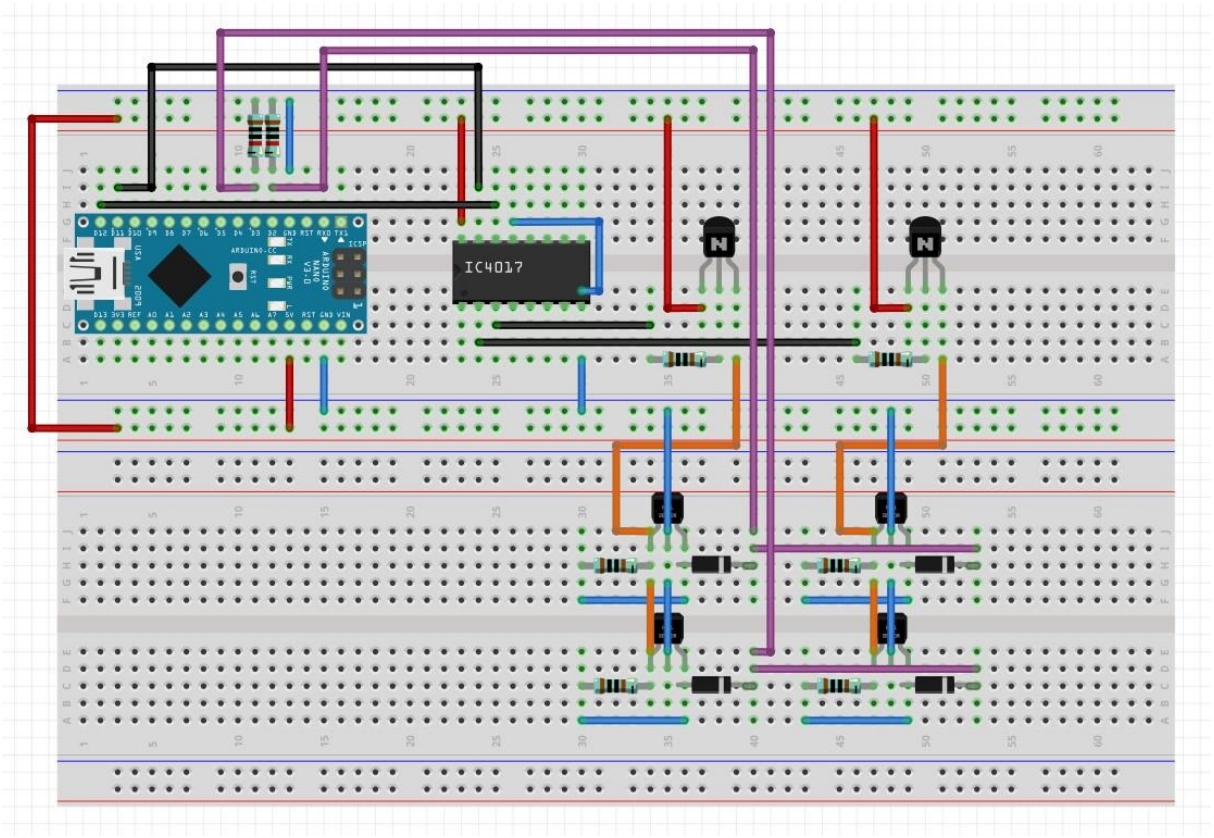
Za funkciju igranja šaha, naravno, potrebna je šahovska ploča. Da bi se maksimalno pojednostavila izrada ploče, korištena je polimerna ploča na koji je otisnuta naljepnica s izgledom šahovske ploče. U odnosu na običnu šahovsku ploču, ova ploča mora imati dodatna polja na koja se stavljaju figure za promociju pješaka, kako bi robot znao s koje pozicije uzeti figuru prilikom obavljanja pokreta promocije pješaka.

Isto tako, promocija je ograničena na damu i skakača jer su to dvije figure koje su korištene kod promocije kod gotovo svih slučajeva. Nacrtana naljepnica koja se nalijepila na šahovsku ploču prikazana je na slici 8.



Slika 8. Izgled šahovske ploče

Kako bi se odigrani potez mogao pratiti, na napravljenu šahovsku ploču potrebno je dodati senzore. Najprije su korišteni reed releji, no testiranjem je ustanovljeno da se prilikom savijanja ploče kod premještanja lomi staklo reed releja. Oni su kasnije zamijenjeni hall-ovim senzorima koji su tvornički napravljeni u kućištu koje je puno otpornije. Pošto se koristi velik broj senzora, koji su ispod svakog polja, potrebno je puno vodova koji bi spajali te sve senzore. Takav sustav moguće je smanjiti spajanjem senzora u mrežu i korištenjem integriranog kruga 4017 koji će slijedno uključivati napon za svaki stupac mreže senzora. Time je potrebno na mikrokontroler spojiti samo 8 vodova koji će čitati pozicije, a matrica cijele šahovske ploče će se stvarati postepeno prolazeći kroz ciklus sklopa 4017. Spoj za matricu senzora 2x2 vidljiv je na slici 9.

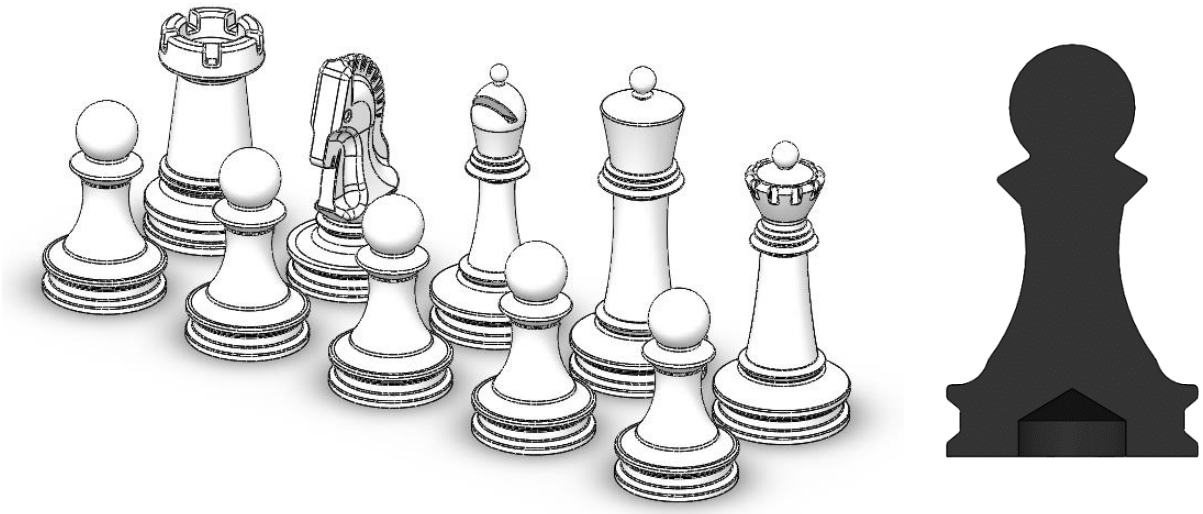


Slika 9. Prikaz spajanja hall senzora na breadboardu

Kako bi se postigla dovoljna snaga za napajanje svih senzora koji se uključe odjednom pomoću sklopa 4017, mora se za svaki stupac napraviti tranzistorska sklopka koja će uzimati napon direktno s izvora umjesto da se napon uzima s digitalnih pinova mikrokontrolera koji ne može dati dovoljnu struju za napajanje. Isto tako, potrebno je iza svakog hallovog senzora dodati diodu kako bi se spriječio povratak struje kroz vodiče koji nisu trenutno aktivirani preko integriranog kruga 4017, te se time sprječavaju pogrešne informacije. Cijela logika se obavlja pomoću mikrokontrolera Arduino Nano.

4.4. Figure

Figure su modelirane na način da izgledaju oku ugodno, a u isto vrijeme olakšavaju primjenu. Na slici 10. se tako može vidjeti dizajn različitih figura koji je napravljen u sklopu ovog diplomskog rada. Proporcije figure (odnos širine i visine) su određene prema standardu za turnirske setove šahovskih figura ovisno o tipu figure.

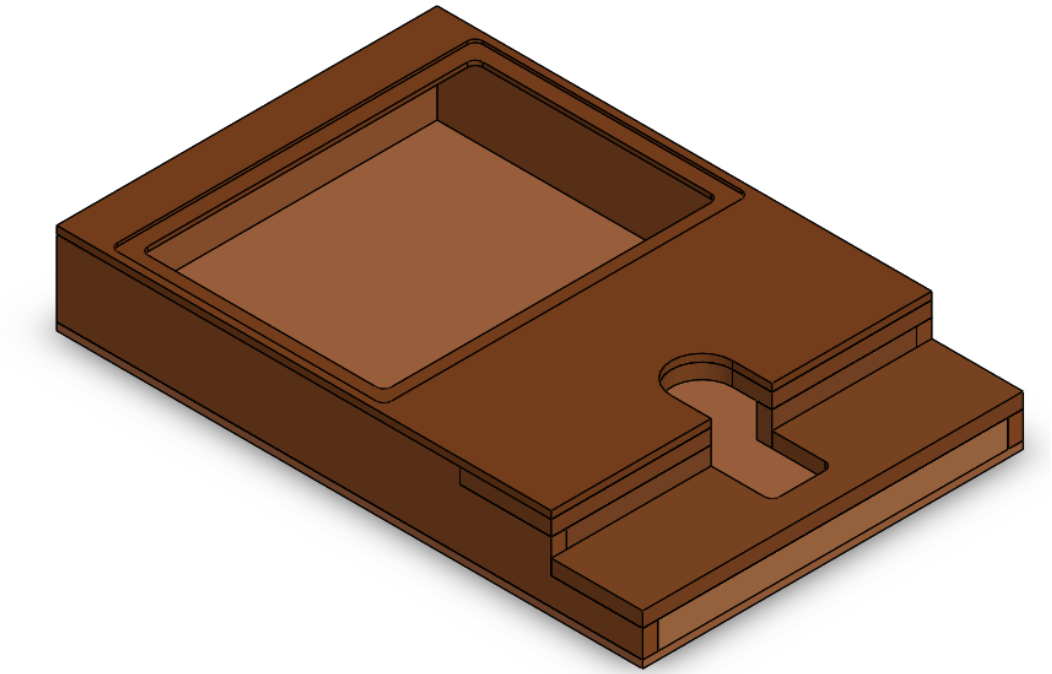


Slika 10. Dizajn figura (lijevo), presjek figure (desno)

Na slici 10. s desne strane možemo vidjeti presjek figure. Donji provrt je rađen kako bi se u njega smjestio magnet koji će onda moći aktivirati hall-ove senzore kada figura bude stavljena na polje šahovske ploče. Također, sa slike je vidljiv utor na bazi figure. Taj utor služi kao dodatno osiguranje prilikom prihvaćanja figure pomoću hvataljke. Glava prsta hvataljke (čavao) ulazi u taj utor prilikom zatvaranja hvataljke te dodatno osigurava od ispadanja figure.

4.5. Kućište

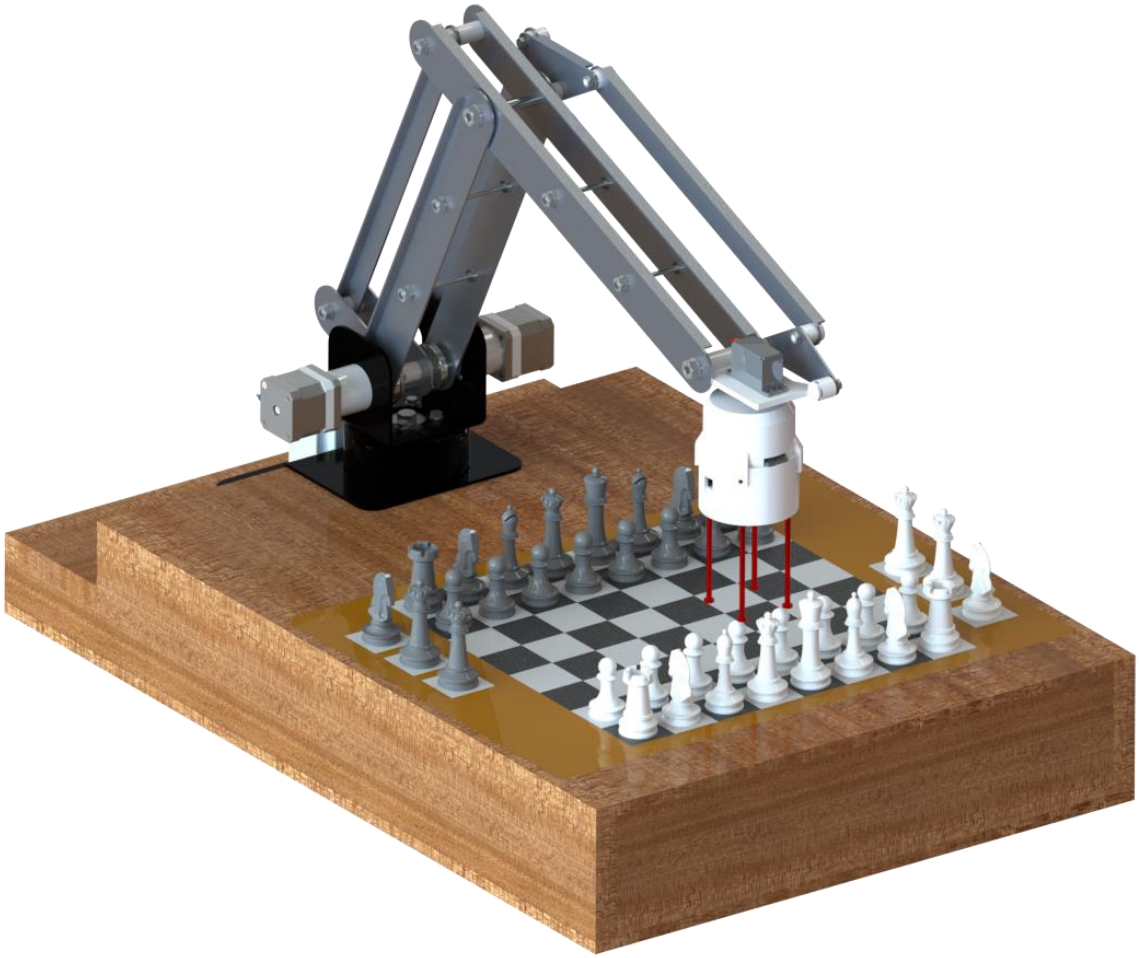
Sve dijelove robotskog sustava je potrebno spojiti u mobilnu cjelinu kako bi se robotska ruka znala pozicionirati, jer fiksacijom šahovske ploče u odnosu na robotsku ruku postižemo stalni koordinatni sustav u kojem onda robot može premještati figure. Kako bi se osigurala krutost kućišta, a u isto vrijeme što manja masa, kućište je rađeno od drva (šperploča). Na slici 11. vidi se da je kućište zamišljeno kao spoj od više drvenih dasaka koje su zajedno učvršćene ljepilom za drvo. Na mjestu gdje se učvršćuje robotska ruka napravljena su ojačanja od deblje daske. Na gornjoj ploči napravljen je utor koji odgovara obliku elektroničke šahovske ploče. Pozicija tog utora je pomno odabrana s obzirom na to koliko blizu sebe robotska ruka može doći, a s druge strane koja je najveća udaljenost na koju robotska ruka bez problema može doći. Također, napravljen je i manji utor kako bi baza robotske ruke bez problema mogla ući u kućište. Visina kućišta je određena prema visini vratila, tako da se remenski prijenos između motora i vratila Z-osi robota može odvijati bez problema.



Slika 11. Izgled drvenog kućišta

4.6. Sklop

Nakon što su sve komponente modelirane, potrebno ih je sklopiti u cjelinu. Na donju podlogu drvene kutije, koja je zamišljena kao rastavljiva ploča koja se vijcima spaja s ostalim zalijepljenim dijelom drvenog kućišta, najprije su montirane elektroničke komponente koje će biti unutar kućišta. To su napajanje, driveri koračnih motora, mikrokontroler robotske ruke, Raspberry Pi i ožičenje. Nakon toga je baza robotske ruke pozicionirana u utor namijenjen za nju, te je temeljna ploča baze robotske ruke vijcima pričvršćena za drveno kućište. Gornji dio robotske ruke je potom vijcima pričvršćen na bazu robotske ruke, te je time ona kompletna. Na glavu ruke, direktno na vratilo servo motora, se spaja hvataljka. U veliki utor drvenog kućišta se ugrađuje elektronička šahovska ploča te se na nju smještaju figure. Zbog simetričnosti šahovske ploče potrebno je paziti na orijentaciju ploče prilikom umetanja u utor, jer programska podrška mikrokontrolera omogućuje samo jednu moguću orijentaciju kako bi prepoznati odigrani potezi bili ispravni. Tako sklopljen robotski sustav moguće je jednostavno inicijalizirati i obaviti homing osi robota, jer su sve pozicije polja uvijek konstantne. Kao dodatak modelu sklopa, potrebno je još dodati krajnje prekidače na pozicije u kojima će ih tijelo robota moći dodirnuti, ali van aktivnog radnog prostora robota.



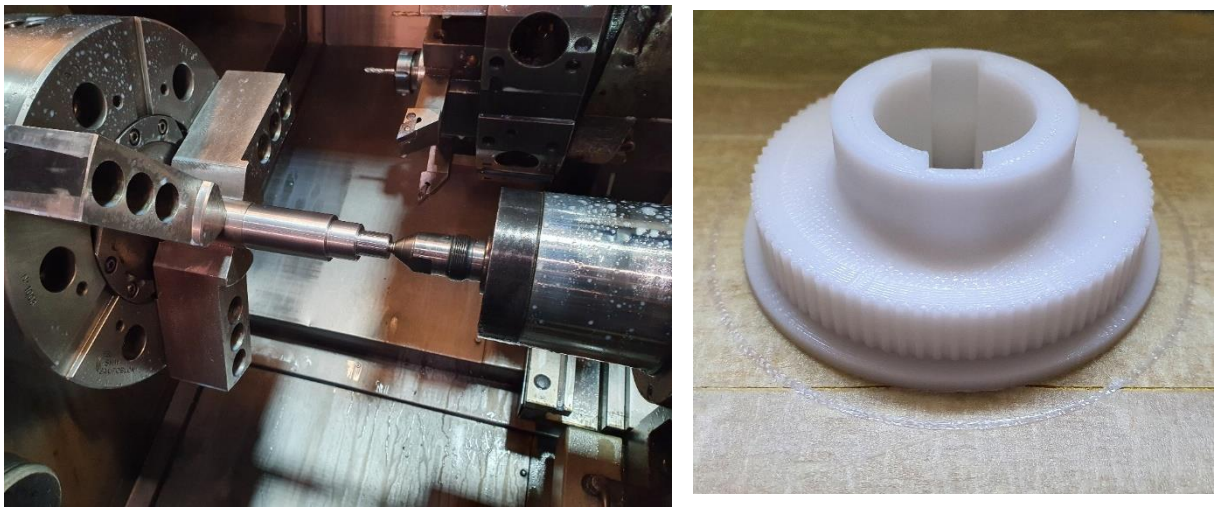
Slika 12. Render konačnog sklopa robotske ruke

Također, potrebno je dodatno napraviti posudu u koju će robot bacati uzete figure. Posuda nije modelirana, već će se napraviti na najboljoj mogućoj poziciji nakon gotovog sklopa svih drugih dijelova robotskog sustava. Pozicija posude može biti proizvoljna unutar granica kretanja robota, jer će se nakon njenog postavljanja u program robotske ruke unijeti koordinate. Na slici 12. može se vidjeti kako u programskom paketu SolidWorks izgleda cijeli sklopljeni robotski sustav (bez krajnjih prekidača i posuda za odlaganje figura) nakon renderiranja slike. Pomoću modela u 3D CAD programu moguće je pomicati robotsku ruku i vidjeti koje su joj krajnje pozicije. Na taj način može se prije fizičke izrade provjeriti i ukloniti potencijalne probleme.

5. IZRADA ROBOTA

Za izradu robotskog sustava potrebni su razni tehnološki postupci izrade dijelova kao što su: rezanje, savijanje, tokarenje, glodanje, brušenje, lakiranje, 3D ispis, lijepljenje, montaža, itd. U ovom poglavlju biti će opisano kako su neki od dijelova izrađeni i montirani. Također će biti prikazane slike pojedinih dijelova tijekom i nakon izrade, kako bi se što bolje dočarao postupak.

Kod izrade dijelova baze robotske ruke, velik dio je rađen tokarenjem zbog toga što su svi dijelovi koji će se okretati oko Z-osi cilindričnog oblika. Na slici 13. lijevo je prikazana CNC obrada vratila baze robotske ruke, odnosno tokarenje jedne strane vratila. Nakon tokarenja narezan je navoj gdje će prilikom montaže doći matice ležaja te je glodan utor za pero koji će prenositi moment na gornji dio robotske ruke. Temeljna ploča je izrađena rezanjem i savijanjem lima od nehrđajućeg čelika prema zadanim dimenzijama. Osim izrađenih dijelova, kupljeni su ležajevi, matice ležaja i sigurnosne pločice za osiguravanje matice ležaja od odvrtanja te motor, remenica i remen GT2. Kako bi se dobio povoljan prijenosni omjer modelirana je vlastita remenica te je nakon toga isprintana na 3D printeru. Gotova remenica može se vidjeti na slici 13. desno. Nakon prikupljanja i izrade svih dijelova baza robotske ruke je sklopljena prema napravljenom modelu iz SolidWorksa koji je prezentiran u prethodnoj temi.



Slika 13. Tokarenje vratila baze robotske ruke (lijevo) i 3D ispis remenice (desno)

Gornji dio robotske ruke sastoji se od nosača, krakova ruke, sustava poluga i motora. Nosač je napravljen od nehrđajućeg čeličnog lima debljine 2mm kako bi se postigla zadovoljavajuća krutost koja je potrebna za preciznost (robot se prilikom rada ne smije savijati). Nosač je savijen prema modelu te su odgovarajući provrti napravljeni glodanjem kako bi se ostvarila čim bolja suosnost provrta za motore s lijeve i desne strane nosača. Krakovi su napravljeni od aluminijskih L profila, koji su rezani na odgovarajuću debljinu. Nakon rezanja, provrti su rađeni u paru (lijeva i desna strana svakog kraka su prije bušenja poravnate) kako bi se provrti napravili na istoj osi. To je bitno zbog toga jer se nakon bušenja između profila stavljaju tokareni odstojnici koji bi prilikom pritezanja matice savijali konstrukciju ako provrti profila ne bi bili poravnati. Bočno se na sastavljene krakove robotske ruke smješta sustav poluga koji će osiguravati paralelnost hvataljke s podlogom. Sustav poluga je napravljen također od aluminijskih L profila te je postupak izrade sličan. Pažljivom montažom svih komponenata dobiva se gornji dio robotske ruke koji se potom spaja s bazom robotske ruke pomoću četiri vijka. Nakon sklapanja, sustav se giba kao što je to predviđeno modelom.

Hvataljka se sastoji od 3D printanih dijelova, prstiju i servo motora. Svi 3D printani dijelovi su nakon printa obrađeni brusnim papirom kako bi se ispravile eventualne netočnosti printera. Prsti hvataljke su napravljeni tako da su čavli odrezani na određenu duljinu te su na krajevima narezani navoji kako bi se kasnije maticom spriječilo ispadanje iz hvataljke. Pažljivim sklapanjem prihvatnice (hvataljke) prema prije napravljenom modelu dobiva se lagana i funkcionalna hvataljka koja minimalno interferira s figurama na šahovskoj ploči.

Elektronička šahovska ploča se sastoji od polimerne ploče na koju su s donje strane zalijepljeni senzori. Sama šahovska ploča je dobivena na način da se izgled šahovske ploče modeliran u SolidWorksu ispiše na foliju koja se pomoću topline lijepi za polimernu ploču. Ispod svakog polja su potom nalijepljeni hallovi senzori koji detektiraju magnetsko polje. Kod lijepljenja senzora potrebno je sve senzore okrenuti na istu stranu jer hallovi senzori su polarizirani, te mogu s jedne strane detektirati samo jedan pol magneta. Kada su senzori pričvršćeni, između njih su zalemljene ostale elektroničke komponente: diode, tranzistori, otpornici i vodovi. Također je napravljena mala elektronička pločica na koju je smješten Arduino Nano mikrokontroler na kojega su spojeni svi vodovi koji su potrebni za rad ploče.

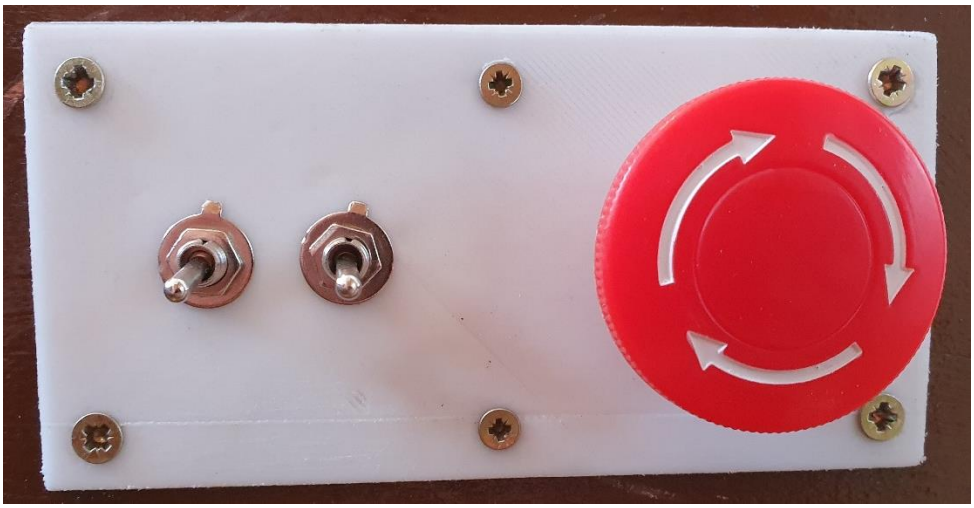
Figure su izrađene 3D printom korištenjem filamenta različitih boja (crna i bijela), a mogu se vidjeti na slici 14. Ispod svake figure nalazi se utor za magnet kao što je modelirano u CAD programu. Provjerom jačine i pozicije detekcije hallovih senzora, metodom pokušaja i pogreške određeni su magneti koji su korišteni u figurama. Ti magneti su neodimijski magneti promjera 10 mm i debljine 2,5 mm. Prilikom stavljanja figure na šahovsko polje, ona je detektirana točno unutar granica polja, a detekcija se ne događa kada je figura izvan polja, što je traženi efekt. Kada su odabrani magneti, oni su lijepljeni na figuru točno tako da strana magneta koju detektira hallov senzor bude s donje strane. Isto tako je potrebno paziti da su svi magneti dobro okrenuti jer inače neće doći do detekcije prisutnosti figure. Kada su svi magneti zalijepljeni, preko dna figure su nalijepljeni podlošci za namještaj, kako bi se postigla ravna površina koja neće grepsti naljepnicu na vrhu šahovske ploče i time produljiti vijek trajanja.

Slika 14. Šahovske figure napravljene 3D printom

Drveno kućište je napravljeno rezanjem dasaka dimenzija određenih 3D modelom te njihovim lijepljenjem. Nakon što se ljepilo osušilo s gornje strane kućišta su glodanjem napravljeni utori za šahovsku ploču i utor za bazu robotske ruke. Nakon toga s donje strane je vijcima pričvršćena pomična ploča koja mora biti rastavljiva. Kada je sve provjereno i zaključeno da sve odgovara, kućište je obojano bojom za drvo u 2 premaza kako bi se drvo zaštitilo od vanjskih utjecaja. Prilikom izrade dodan je malen stalak za robotsku ruku gdje će se hvataljka robotske ruke smjestiti prilikom isključivanja i pokretanja sustava kako bi se osigurao ispravan rad i minimalno savijanje dijelova robotske ruke tijekom vremena ne korištenja.

Kada su svi podsklopovi gotovi, prema 3D modelu sklopa se sve sklapa u cjelinu. Tako je prema planu temeljna ploča vijcima pričvršćena za glavno drveno kućište, a hvataljka spojena na vratilo servo motora na glavi robotske ruke. Sve elektroničke komponente: napajanje, servo driveri, ispravljači napona, Raspberry PI i mikrokontroler ruke smještene su na donju rastavljivu ploču te je ploča ponovno vijcima pričvršćena za ostatak kućišta. Nakon toga je konačno ugrađena elektronička šahovska ploča u utor predviđen za nju.

Kako bi cijeli sustav radio potrebno je strateški smjestiti krajnje prekidače koji pomažu robotu da odredi početnu poziciju na temelju koje će dalje znati gdje se koji dio sustava nalazi i na temelju toga će se moći gibati. Krajnji prekidači moraju biti smješteni van radnog područja robota kako se robot dokom rada ne bi slučajno zaletio u njih, tj. da ne bi došlo do kolizije. Nakon toga moguće je pokrenuti homing osi, te je robot uspješno pokrenut. Prilikom rada primijećena je dosta velika zračnost između zuba zupčanika reduktora 1:27 koji su došli u paketu s koračnim motorima. Kako bi se taj problem barem djelomično anulirao, dodane su opruge koje će vući robotsku ruku u krajnji položaj te time povećati točnost pokreta.



Slika 15. Sklopke i sigurnosna gljiva

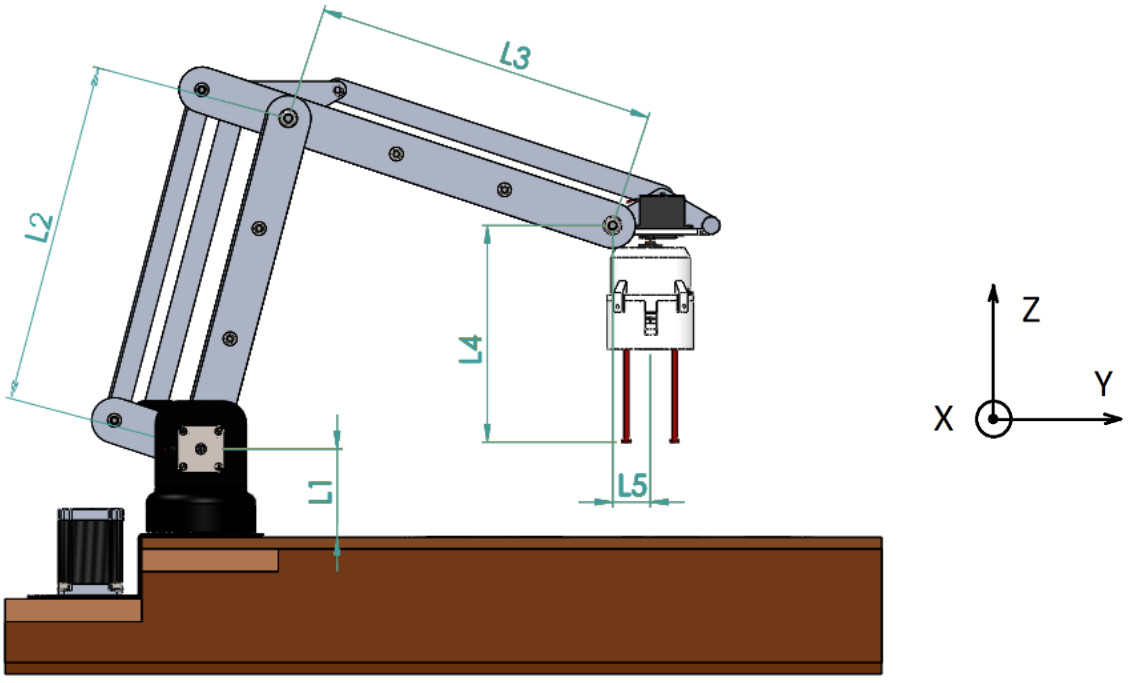
Dodane su posebne sklopke i sigurnosna gljiva koje se mogu vidjeti na slici 15. Lijeva sklopka uključuje motore, dok desna sklopka uključuje Raspberry Pi i mikrokontrolere. Sigurnosna gljiva je ugrađena za slučaj nenadane greške. Također, dodani su spremnici za figure kako bi robotska ruka mogla izbaciti uzete (pojedene) figure uvijek na isto mjesto.

6. KINEMATIKA ROBOTSKE RUKE

U ovom poglavlju biti će opisana inverzna kinematika robotske ruke, tj. izračun potrebnih pozicija rotacije motora u odnosu na željenu poziciju u Kartezijevom koordinatnom sustavu. Isto tako, da bi se spriječilo trzanje i neusklađen rad, potrebno je uskladiti brzine svih motora kako bi prilikom pokreta robot gibanje svih motora počeo i završio u isto vrijeme.

6.1. Inverzna kinematika

Kako bi se odredila inverzna kinematika robotske ruke, potrebno je prvo definirati koordinatni sustav. Ishodište koordinatnog sustava je postavljeno u centar rotacije robotske ruke na visini gornje strane kućišta, a osi su definirane prema slici 16. desno. Na istoj slici su prikazane definirane veličine robotske ruke koje će biti korištene tijekom procesa izračuna.



Slika 16. Dimenzije korištene za proračun

Definirani su kutovi q_1, q_2, q_3 koji redom označavaju zakret robota oko Z-osi, nagib donjeg kraka u odnosu na Z-os te nagib gornjeg kraka u odnosu na horizontalnu ravninu. Pomaci p_x, p_y, p_z definiraju pomake prihvatnice robota u odnosu na sve tri osi (X, Y i Z) zadanog koordinatnog sustava sa slike 16.

Kao što je već rečeno, ideja inverzne kinematike je dobiti kutove q_1, q_2, q_3 za koje se krakovi robota okreću iz pomaka hvataljke p_x, p_y, p_z. Najlakše je dobiti prvi kut q_1, uz pomoć pravila trigonometrije $arctg\,(-X) = -arctg\,(X)$:

$$tg(q_1) = \frac{-p_x}{p_y}$$

$$q_1 = -arctg\left(\frac{p_x}{p_y}\right) \quad \textbf{(6.1)}$$

Kada se robotska ruka okreće oko svoje baze za pomak q_1 potrebno je definirati novi lokalni koordinatni sustav koji je analogan početnom, samo što prati novu orijentaciju robota. Z-os se prilikom rotacije ne mijenja, a u lokalnom koordinatnom sustavu će se novonastale osi označavati s x2 i y2. Prema tome se dobiva pomak po osi y2 kao:

$$p_{y2} = \sqrt{{p_x}^2 + {p_y}^2} \quad \textbf{(6.2)}$$

Kako bi se dobile sljedeće vrijednosti kutova q_2, q_3 potrebno je uvesti pomoćne varijable. Krakovi duljina L_2 i L_3 zajedno s imaginarnom dužinom L_x, koja spaja ishodište donjeg kraka i vrh gornjeg kraka, tvori trokut. Kut između L_2 i L_x je određen kao α, kut između L_3 i L_x kao β, dok je kut između osi y2 i L_x označen kao δ. Duljina imaginarne dužine određena je s:

$$L_x = \sqrt{\left(p_{y2} - L_5\right)^2 + (p_z + L_4 - L_1)^2} \quad \textbf{(6.3)}$$

Sada kada su poznate sve tri stranice trokuta $L_2, L_3\ i\ L_x$, može se pomoću kosinusovog poučka $a^2 = b^2 + c^2 - 2bc \cdot \cos(\alpha)$ dobiti kut α :

$${L_3}^2 = {L_2}^2 + {L_x}^2 - 2L_2L_x \cdot \cos(\alpha)$$

$$\cos(\alpha) = \frac{{L_2}^2 + {L_x}^2 - {L_3}^2}{2L_2L_x}$$

$$\alpha = arccos\left(\frac{{L_2}^2 + {L_x}^2 - {L_3}^2}{2L_2L_x}\right) \quad \textbf{(6.4)}$$

Na isti način dobiva se kut β :

$${L_2}^2 = {L_x}^2 + {L_3}^2 - 2L_xL_3 \cdot \cos(\beta)$$

$$\cos(\beta) = \frac{{L_x}^2 + {L_3}^2 - {L_2}^2}{2L_xL_3}$$

$$\beta = arccos\left(\frac{{L_x}^2 + {L_3}^2 - {L_2}^2}{2L_xL_3}\right) \tag{6.5}$$

I na kraju, pomoću trigonometrije dobiva se kut δ :

$$\mathrm{tg}(\delta) = \frac{p_z + L_4 - L_1}{p_{y2} - L_5}$$

$$\delta = arctg\left(\frac{p_z + L_4 - L_1}{p_{y2} - L_5}\right) \tag{6.6}$$

Jednostavnim pravilom trokuta iz pomoćnih kutova se mogu preračunati potrebni kutovi, gdje postupak izgleda ovako:

$$q_3 = \pi - \beta - (\pi - \delta) = \delta - \beta \tag{6.7}$$

$$q_2 = \frac{\pi}{2} - (\pi - \alpha - \delta) = \alpha + \delta - \frac{\pi}{2} \tag{6.8}$$

Na kraju se uvrštavanjem jednadžbi **6.4**, **6.5** i **6.6** u **6.7** i **6.8** dobiva elegantan zapis pretvorbe pomaka prihvatnice u potrebne pomake kutova robotske ruke:

$$q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} -arctg\left(\frac{p_x}{p_y}\right) \\ \alpha + \delta - \frac{\pi}{2} \\ \delta - \beta \end{bmatrix} = \begin{bmatrix} -arctg\left(\frac{p_x}{p_y}\right) \\ arccos\left(\frac{{L_2}^2 + {L_x}^2 - {L_3}^2}{2L_2L_x}\right) + arctg\left(\frac{p_z + L_4 - L_1}{p_{y2} - L_5}\right) - \frac{\pi}{2} \\ arctg\left(\frac{p_z + L_4 - L_1}{p_{y2} - L_5}\right) - arccos\left(\frac{{L_x}^2 + {L_3}^2 - {L_2}^2}{2L_xL_3}\right) \end{bmatrix} \tag{6.9}$$

Prilikom unošenja jednadžbi u program treba pripaziti na korištenje *arctan()* funkcije. Navedena funkcija mora biti zamijenjena s *arctan2()* funkcijom kako bi se dobilo ispravno rješenje za sva četiri kvadranta Kartezijevog koordinatnog sustava.

6.2. Usklađivanje pokreta

Jednom kada je inverzna kinematika izračunata, robot se može gibati u odnosu na zadane koordinate na ploči. Međutim, kada je kut jednog kraka veći od kuta drugog kraka robota, on pokrete obavlja nesinkronizirano, te gibanja sva tri motora završavaju u različitim vremenima što je nepoželjan efekt kod robotske ruke. Kako bi se tome doskočilo, potrebno je izračunati potreban hod za svaki kut te nakon toga za svaki motor posebno uskladiti brzinu gibanja.

Svako gibanje se sastoji od dijela u kojem motor ubrzava, okreće se konstantnom brzinom i usporava. Vrijeme ubrzavanja i usporavanja je prilikom gibanja koračnog motora jednako te će biti označavano s t_a. Vrijeme u kojem se motor okreće konstantnom brzinom označeno je s t_c. Ukupno vrijeme svakog gibanja može biti određeno s:

$$\Delta t = 2t_a + t_c \tag{6.10}$$

Vrijeme ubrzavanja može se odrediti pomoću maksimalne brzine kojom se pokreće motor i njegovog maksimalnog ubrzanja:

$$t_a = \frac{v_{max}}{a_{max}} \tag{6.11}$$

Ukupni prijeđeni put (kut zakreta) može se dobiti kao površina ispod grafa brzine (integral brzine). Pa se tako dobiva:

$$s = v_{max}t_a + v_{max}t_c \tag{6.12}$$

U sljedećem koraku će se uvesti nove oznake. Pošto se želi odrediti konstantnu brzinu kojom motor mora raditi kako bi posao obavio u određenom vremenu, ta brzina biti će nazvana v_x. Vrijeme ukupnog pokreta motora uključujući ubrzavanje i usporavanje biti će u ovom slučaju označeno s t_1. Ubrzanja će biti konstantno podešena za svaki motor i neće se mijenjati. Uvrštavanjem izraza **6.10** i **6.11** u izraz **6.12** i naknadnim sređivanjem dobiva se:

$$s = v_x t_c + v_x t_a$$

$$s = v_x \left(t_1 - 2\frac{v_x}{a}\right) + \frac{v_x^2}{a}$$

$$s = v_x t_1 - 2\frac{v_x^2}{a} + \frac{v_x^2}{a}$$

$$s = v_x t_1 - \frac{v_x^2}{a}$$

$$v_x^2 - a t_1 v_x + a s = 0 \tag{6.13}$$

Rješavanjem kvadratne jednadžbe se za zadano vrijeme t_1 dobiva potrebna brzina okretanja motora v_x. Postupak usklađivanja brzina motora provodi se kroz dvije faze. Prva faza je izračunavanje vremena potrebnog za obavljanje pokreta punom brzinom za sva 3 motora. U drugoj fazi će najveće vrijeme postati referentno vrijeme t_1. Na temelju tog vremena će se izračunati potrebne brzine okretanja preostala dva motora (te brzine su manje od motora koji je u prvom ciklusu imao najdulje vrijeme kretanja) i time će pokreti biti savršeno usklađeni.

7. PROGRAMIRANJE MIKROKONTROLERA

U ovom poglavlju biti će prokomentirani dijelovi koda za mikrokontrolere. Zbog velike količine koda, razmotrit će se samo glavne funkcije za svaki mikrokontroler. Kao što je već prije navedeno za robotsku ruku se koristi dvojezgreni ESP32, dok je za šahovsku ploču dovoljan Arduino Nano. Oba programa su pisana u programu Arduino IDE.

7.1. Upravljački program robotske ruke

Prva funkcija koja će biti razmatrana je funkcija inverzne kinematike koja, kako joj samo ime govori, pretvara željenu poziciju u koordinatnom sustavu u korake koje koračni motor mora obaviti. Funkcija izgleda ovako:

```
void pozicija_u_korake(float px, float py, float pz){
  float py2 = sqrt(sq(px)+sq(py));
  float Lx = sqrt(sq(py2-L5)+sq(pz+L4-L1));
  float alfa = acos((sq(L2)+sq(Lx)-sq(L3))/(2*L2*Lx));
  float beta = acos((sq(Lx)+sq(L3)-sq(L2))/(2*Lx*L3));
  float delta = atan2((pz+L4-L1),(py2-L5));

  float q1 = -atan2(px,py);
  float q2 = alfa+delta-PI/2;
  float q3 = delta-beta;

  if(homirano == true && traje_sklapanje == false){
    Servo1.write(90+round((180/PI)*q1));
  }

  koraci1 = round((180/PI)*q1*prijenosni_omjer_veliki_motor*
broj_koraka_po_okretu*microstepping/360);
  koraci2 = round((180/PI)*q2*prijenosni_omjer_mali_motor*
broj_koraka_po_okretu*microstepping/360);
  koraci3 = round((180/PI)*q3*prijenosni_omjer_mali_motor*
broj_koraka_po_okretu*microstepping/360);
}
```

U prvom dijelu funkcije implementirane su jednadžbe izvedene u prethodnom poglavlju. Inicijalizacijom varijabli i izračunom dobivamo kutove q_1, q_2, q_3 u radijanima. Ti kutovi u radijanima direktno ne znače ništa, već ih treba pretvoriti u korake koračnih motora, kako bi mikrokontroler znao koliko koraka mora motor napraviti. Pretvorba je vidljiva u donjem dijelu funkcije. Isto tako, odmah prilikom izračuna kuta servo motor, koji vrši rotaciju hvataljke u svrhu poravnanja sa šahovskim poljima, se okreće za isti kut koji se i robot okreće oko Z-osi. Pozivom funkcije u memoriji se spremaju varijable koraka te su spremne za izvršavanje.

Sljedeća stvar koju je potrebno napraviti je usklađivanje pokreta motora.

```
if(kor1f < 1){
    vrijeme1 = 0;
}else if(kor1f >= 1 && kor1f <= vmax_v*ta_v){
    vrijeme1 = 2*sqrt(kor1f/a_v);
}else{
    vrijeme1 = 2*ta_v + (kor1f-vmax_v*ta_v)/vmax_v;
}
```

Ovim kodom se prvo provjerava je li broj koraka koje motor mora obaviti veći od 1. Ako je veći od 1 onda se provjerava je li motor uspio uslijed ograničenog ubrzanja dostići maksimalnu brzinu vrtnje ili se uslijed malog broja koraka mora zaustaviti prije toga. Ako je to slučaj onda se vrijeme potrebno za pokret mora računati po specifičnoj formuli. Ako se ne dogodi ni jedan od prošla dva slučaja, to znači da se radi o normalnom gibanju gdje je motor ubrzavao, postigao maksimalnu brzinu, vrtio se neko vrijeme tom brzinom i onda usporio i stao. Isti kod se pokreće za svaki motor za broj koraka koji on mora napraviti.

U sljedećem koraku se provjerava od kojeg motora je najveće vrijeme potrebno za izvršenje.

```
if(vrijeme1 > vrijeme2 && vrijeme1 > vrijeme3){
  vmax_1 = vmax_v;
  vmax_2 = (a_m*vrijeme1-sqrt(sq(a_m*vrijeme1)-4*a_m*kor2f))/2;
  vmax_3 = (a_m*vrijeme1-sqrt(sq(a_m*vrijeme1)-4*a_m*kor3f))/2;
}else if(vrijeme2 > vrijeme1 && vrijeme2 > vrijeme3){
  vmax_1 = (a_v*vrijeme2-sqrt(sq(a_v*vrijeme2)-4*a_v*kor1f))/2;
  vmax_2 = vmax_m;
  vmax_3 = (a_m*vrijeme2-sqrt(sq(a_m*vrijeme2)-4*a_m*kor3f))/2;
}else if(vrijeme3 > vrijeme1 && vrijeme3 > vrijeme2){
  vmax_1 = (a_v*vrijeme3-sqrt(sq(a_v*vrijeme3)-4*a_v*kor1f))/2;
  vmax_2 = (a_m*vrijeme3-sqrt(sq(a_m*vrijeme3)-4*a_m*kor2f))/2;
  vmax_3 = vmax_m;
}
```

Ono vrijeme koje je najveće postaje referentno, a prema formuli **6.13** se izračunavaju brzine za preostala 2 motora. Rezultat toga je da će se motor koji se mora pomaknuti za najveći put gibati maksimalnom brzinom, dok će preostala dva motora svoje gibanje napraviti usporeno kako bi sva tri motora stigla u odredište u istom vremenu. Nakon primjene ovog koda motori se mogu gibati bez trzanja, a samim time i s većom točnošću. Isto tako, opterećenje na motorima je manje jer se kod svakog pokreta mijenja motor koji radi punom snagom, pa se može očekivati dulji vijek trajanja motora.

Kada su sve brzine i potrebni koraci izračunati potrebno je pokrenuti motore pod tim proračunatim režimima rada.

```
    stepper1.setMaxSpeed(vmax_1_final);
    stepper2.setMaxSpeed(vmax_2_final);
    stepper3.setMaxSpeed(vmax_3_final);

    stepper1.moveTo(kor1);
    stepper2.moveTo(-kor2);
    stepper3.moveTo(kor3);

    while(abs(stepper1.distanceToGo()) > 0 or abs(stepper2.distanceToGo()) > 0
  or abs(stepper3.distanceToGo()) > 0){
      stepper1.run();
      stepper2.run();
      stepper3.run();
    }

    if (traje_sklapanje == false){
      Serial.print(slanje_zavrseno);
    }
```

Prvo se postavlja brzina do koje svaki od motora ubrzava na temelju izračunatih vrijednosti brzina prilikom usklađivanja pokreta motora. Nakon toga postavlja se cilj u koji svaki od motora mora stići. Kada je sve spremno, motori se u *while* petlji vrte tako dugo dok sva tri motora ne obave predviđeni broj koraka. Kada su motori gotovi s radom, prema glavnom računalu se šalje informacija da je pokret obavljen.

Za otvaranje i zatvaranje hvataljke definirane su posebne funkcije.

```
void otvori_hvataljku(){
  Servo2.write(20);
  delay(250);
  Serial.print(slanje_zavrseno);
}
void zatvori_hvataljku(){
  Servo2.write(130);
  delay(250);
  Serial.print(slanje_zavrseno);
}
```

Servo motor je prilikom otvaranja hvataljke okretan suprotno od kazaljke na satu, a prilikom zatvaranja u smjeru kazaljke na satu kao što je bilo i konstruirano. Također kao i nakon obavljanja pokreta, nakon zatvaranja ili otvaranja hvataljke na glavno računalo se šalje kratka informacija kako bi računalo znalo da je pokret hvataljke obavljen. Nakon naredbe za otvaranje ili zatvaranje dodan je malen delay kako bi hvataljka imala vremena obaviti funkciju.

Sljedeći važan dio programa na mikrokontroleru je komunikacija. Zbog visoke pouzdanosti mikrokontroler ESP32 je preko bluetootha spojen na mobitel, jer bluetooth modul na Raspberry Pi-u često proizvodi greške, tj. nije pouzdan.

```
if ((millis() % (vrijeme_primanja_bt)) == 0) {
   while (SerialBT.available()) {
     delay(1);
     char c = SerialBT.read();
     if (c == '#') {
       break;
     }
     procitani_string += c;
   }

   if (procitani_string.length() > 0) {

     if(procitani_string == "pomoc"){
       Serial.print("pomoc#");
     }
     else if(procitani_string == "vrati"){
       Serial.print("vrati#");
     }
     else if(procitani_string == "homing"){
       Serial.print("homingMOB#");
     }
     else if(procitani_string == "sklopi"){
       Serial.print("sklopiMOB#");
     }
     else if(procitani_string == "iskljuci"){
       Serial.print("iskljuci#");
     }
     else if(procitani_string.substring(0,4) == "nova"){
       Serial.print(procitani_string+"#");
     }
     procitani_string="";
   }
 }
```

Kako bi se ograničilo prekomjerno procesiranje, pomoću vanjske *if* petlje ono se obavlja samo u određenom vremenskom trenutku. Ako postoji dolazna informacija u serijskoj bluetooth sabirnici, *while* petlja se počinje obavljati. Svaki dolazni znak se dodaje u *string* sve dok više nema ničega na dolaznom bufferu ili dođe znak # . Znak # je isprogramiran da se koristi na kraju svake poruke kako bi se moglo odvojiti informacije prilikom brzog dolaska više poruka odjednom. Nakon što je *string* kompletan on se uspoređuje s ključnim riječima za koje je predefiniran postupak. Ukoliko se dolazna riječ poklapa s nekom od gore navedenih riječi, putem serijske komunikacije se na glavno računalo šalje informacija koja je poslana iz mobilne aplikacije. Nakon što je dolazna riječ uspoređena sa svim memoriranim riječima, ona se briše te je varijabla spremna za sljedeći ciklus čitanja.

Uz bluetooth komunikaciju s mobitelom, ESP32 u isto vrijeme obavlja i komunikaciju s glavnim računalom preko serijske veze brzine 115200 bitova po sekundi.

```
if (Serial.available() > 0) {
  while (Serial.available() > 0) {
    delayMicroseconds(100);
     char c = Serial.read();
     if (c == '#') {
       break;
     }
     ulaz += c;
   }
   if (ulaz.length() > 0) {
     if (ulaz == "homing"){
       homing();
     }else if (ulaz == "on"){
       zatvori_hvataljku();
     }else if (ulaz == "off"){
       otvori_hvataljku();
     }else if (ulaz == "sklopi"){
       sklapanje_robota();
     }else if (ulaz.substring(0,3) == "poz"){
       SerialBT.print(ulaz+"#");
     }else if (ulaz.substring(0,3) == "kor"){
       ulaz.remove(0, 4);
       float px = getValue(ulaz, ',', 0).toFloat();
       float py = getValue(ulaz, ',', 1).toFloat();
       float pz = getValue(ulaz, ',', 2).toFloat();
       pozicija_u_korake(px, py, pz);
       kreni_u_poziciju(koraci1, koraci2, koraci3);
     }
     ulaz = "";
   }
 }
```

Serijska komunikacija radi na identičan način kao i serijska bluetooth komunikacija. Riječ se naslaguje znak po znak sve dok ne dođe do kraja poruke ili se pojavi znak #. Tada se riječ uspoređuje s memoriranim riječima te se obavljaju zadane funkcije. Ako dođe riječ *homing*, obavlja se homiranje osi robota, na *on* i *off* se obavlja otvaranje ili zatvaranje hvataljke, dok se na naredbu *sklopi*, sklapa robotska ruka u početnu poziciju. Kada su prva 3 slova riječi *'poz'* onda se pozicija iz memorije glavnog programa šalje na mobilnu aplikaciju kako bi se obavio prikaz pozicije. I na kraju, kada su prva 3 slova riječi *'kor'*, tada je robot primio koordinate u koje mora doći. Najprije se pozivom funkcije obavlja proračun inverzne kinematike, a potom proračun brzina, te se nakon toga gibanje motora pokreće. Isto kao i kod bluetootha, na kraju se ulazna riječ briše kako bi se mogla ponovo puniti ulaznim znakovima.

7.2. Program šahovske ploče

Kao što je i zamišljeno, mikrokontroler šahovske ploče obavlja puno procesorski lakše operacije nego što je to slučaj kod mikrokontrolera robotske ruke.

```
void loop(){
  for(int i = 0; i<10; i++){
    bool stanje1 = digitalRead(2);
    bool stanje2 = digitalRead(3);
    bool stanje3 = digitalRead(4);
    bool stanje4 = digitalRead(5);
    bool stanje5 = digitalRead(6);
    bool stanje6 = digitalRead(7);
    bool stanje7 = digitalRead(8);
    bool stanje8 = digitalRead(9);
    ploca[0][i] = stanje1;
    ploca[1][i] = stanje2;
    ploca[2][i] = stanje3;
    ploca[3][i] = stanje4;
    ploca[4][i] = stanje5;
    ploca[5][i] = stanje6;
    ploca[6][i] = stanje7;
    ploca[7][i] = stanje8;
    digitalWrite(clck,HIGH);
    delayMicroseconds(5);
    digitalWrite(clck,LOW);
    delay(10);
  }

  String poruka = "";
  for(int j = 0; j < 8; j++){
    for(int k = 0; k < 10; k++){
      poruka += String(ploca[j][k])+",";
    }
  }
  poruka.remove(poruka.length()-1);
  Serial.println(poruka);
}
```

Vodovi s redova šahovske ploče spojeni su direktno na digitalne ulaze mikrokontrolera dok je 10 stupaca ploče spojeno na sklop 4017. Program radi tako da očitava stanja svih 8 ulaza te ih sprema u prvi stupac matrice. Nakon toga daje se impuls na sklop 4017 koji prebacuje napon s prvog stupca na drugi stupac te se sada mogu čitati svi senzori na drugom stupcu ploče. Nakon što se taj proces obavi kroz 10 koraka, dobiva se konačna matrica prisutnosti figura. Ako je figura na polju, na tom mjestu u matrici nalazi se 0, a u suprotnom se nalazi 1. Kada je matrica kreirana ona se odvaja zarezima te se pretvara u varijablu tipa *string* te se šalje serijskom komunikacijom na glavno računalo koja dalje procesira informaciju. Možda se ovaj proces čini sporim, no informacija o poziciji se šalje otprilike svakih 100ms, što je 10 puta u sekundi. To je više nego dovoljno za pouzdanu informaciju o poziciji.

8. PYTHON PROGRAM ZA RASPBERRY PI

U ovom poglavlju prezentirat će se glavni dijelovi koda koji se pokreću na Raspberry Pi-u. Taj program razmjenjuje informacije s oba mikrokontrolera te vrši prikaz šahovske igre pomoću korisničkog sučelja. Program se može podijeliti u četiri osnovne cjeline: dio za elektroničku šahovsku ploču, dio za generiranje pokreta robotske ruke, korisničko sučelje i šahovski engine. Prva tri dijela biti će predstavljena u ovom poglavlju dok će se šahovski engine razmotriti kroz sljedeća poglavlja. Program je pisan u Python-u.

8.1. Detekcija poteza na elektroničkoj šahovskoj ploči

Arduino Nano koji upravlja elektroničkom šahovskom pločom preko serijske komunikacije šalje na Raspberry Pi informaciju o poziciji. Struktura pozicije, kao što je to već navedeno u prethodnom poglavlju, se sastoji od 0 i 1 odvojenih zarezom. Za svako polje posebno određuje se 0 ako je na polju figura i 1 ako je polje prazno. Niz odvojen zarezom se prima kao *string* kodiran UTF-8 kodiranjem preko serijske komunikacije. Prva funkcija koja je definirana pretvara virtualnu šahovsku ploču iz memorije u matricu nula i jedinica kako bi se kasnije moglo usporediti s dolaznom matricom.

```
def stanje_figura():
    ploca_redovi = str(ploca_trenutno).splitlines()
    ploca = [x.split() for x in ploca_redovi]
    if odabir_boje.get() == 2:
        ploca = [i[::-1] for i in ploca[::-1]]
    for red in range(8):
        for stupac in range(8):
            if ploca[red][stupac] == '.':
                ploca[red][stupac] = 1
            else:
                ploca[red][stupac] = 0
    ploca_matrica = np.array(ploca)
    return ploca_matrica
```

Virtualna šahovska ploča u memoriji za svaku figuru ima slovo te figure a za prazna polja ima točke. Kako bi se dobila željena matrica prvo se ploča razdvaja po redovima i onda po stupcima tako da se dobije 2D polje slova. Ako igrač igra s crnim figurama onda je ta matrica reverzirana kako bi se dobila ispravna informacija. Nakon toga se pomoću duple for petlje prolazi kroz 2D polje i ako je znak točka onda se stavlja 1 što znači da je polje prazno, ako je bilo koje drugo slovo onda se u matricu stavlja 0 što znali da je figura na tom polju prisutna.

Da bismo mogli, kod usporedbe vrijednosti s ploče i vrijednosti s memorije, prepoznati koji je potez valjan, potrebno je generirati listu svih mogućih poteza. Iteracijom kroz tu listu može se nadalje provjeriti ispravnost poteza koji se u tom trenutku nađe na ploči.

```
def generiraj_moguce_poteze():
    global ploca_trenutno
    ploca_trenutno = board.copy()
    moguci_potezi = list(ploca_trenutno.legal_moves)
    lista_mogucih_poteza_ploca = []
    for potez in moguci_potezi:
        ploca_trenutno.push(potez)
        lista_mogucih_poteza_ploca.append(stanje_figura())
        ploca_trenutno.pop()
    return moguci_potezi, lista_mogucih_poteza_ploca
```

U varijabli *board* spremljena je virtualna šahovska ploča pomoću *python-chess* [2] modula koji je besplatan i dostupan svima. Taj modul cijelo vrijeme u memoriji sadrži virtualnu šahovsku ploču, prati valjanost poteza, a ima i mnogo dodatnih funkcija koje se mogu iskoristiti. U prikazanom kodu prvo se ploča sprema u pomoćnu varijablu kako ne bi došlo do interferencije s ostalim modulima koji koriste varijablu *board*. Nakon toga se svi mogući potezi koje igrač može odigrati spremaju u listu. Nakon što su mogući potezi u listi se stvara nova lista koja za svaki taj potez tvori matricu nula i jedinica pomoću funkcije *stanje_figura()* koja je objašnjena na početku ovog poglavlja. Time je objašnjena pretvorba iz virtualne ploče u memoriji u matricu koja može direktno biti uspoređivana s matricom koja dolazi s ploče. U sljedećem dijelu objasnit će se kod za povezivanje i pretvaranje dolazne poruke u matricu nula i jedinica.

```
poruka = arduino.readline().decode("utf-8").rstrip()
vektor_str = poruka.split(',')
vektor = np.array([int(element) for element in vektor_str])
matrica = vektor.reshape(8, 10)
matrica_ploca = np.delete(matrica, 0, axis=1)
matrica_ploca = np.delete(matrica_ploca, 8, axis=1)
matrica_memorija = stanje_figura()
matrica_promjene = matrica_memorija ^ matrica_ploca
if np.all((matrica_promjene == 0)):
    polja_s_promjenama = []
    promocija_odabrano = []
```

Poruka s mikrokontrolera se čita preko serijske komunikacije te se dekodira s formata UTF-8. Nakon toga se poruka formata *string* razdvaja u listu na mjestima svakog zareza. Kada je jednodimenzijska lista u memoriji, jednostavno se ta lista pretvori u 2D polje 8x10. S ploče dolaze informacije i o poljima s figurama za promociju. Ta polja moramo izdvojiti s matrice kako bi se matrice mogle direktno uspoređivati. Tako se brišu 1. i 10. stupac matrice da bi se dobila čista matrica šahovske ploče veličine 8x8.

Kako bi se detektirala promjena na šahovskoj ploči u odnosu na ploču u memoriji se jednostavno provodi logička operacija XOR. Operacija *isključivo ili* vraća vrijednost 1 samo u slučaju kada su različite vrijednosti (0 ili 1) u matricama u memoriji i matrici koja dolazi s ploče. Ako postoji barem jedno polje koje se razlikuje, program zabilježi promjenu, a ako ne postoji razlika u nijednom polju, tada znači da do promjene na ploči nije došlo.

Kod brzog rada mikrokontrolera moguće su greške u slanju pa se informacija o poziciji mora provjeriti više puta. Isto tako, ako bi se prilikom prve promjene na polju definirao odigrani potez često bi došlo do greške. Zbog toga treba napraviti više provjera prije nego što se neki potez proglasi odigranim potezom. Kod te provjere vidljiv je niže u dva dijela:

```
moguci_potezi, lista_mogucih_poteza_ploca = generiraj_moguce_poteze()

if matrica_ploca_prosla.size != 0:
    matrica_promjene = matrica_ploca_prosla ^ matrica_ploca
    if odabir_boje.get() == 1:
        matrica_promjene = np.flip(matrica_promjene, axis=0)
    elif odabir_boje.get() == 2:
        matrica_promjene = np.flip(matrica_promjene, axis=1)
    matrica_promjene = matrica_promjene.flatten()
    for i in range(len(matrica_promjene)):
        if matrica_promjene[i] == 1:
            polja_s_promjenama.append(chess.SQUARE_NAMES[i])

matrica_ploca_prosla = matrica_ploca
promocija_figure_prosla = promocija_figure

provjera_jednakosti = False
```

U ovom dijelu koda, kada je primijećeno da se matrica s šahovske ploče i matrica s memorije ne poklapaju, stvara se nova varijabla koja sprema taj potencijali odigrani potez. Isto tako, u novu listu se spremaju i polja na kojima se dogodila promjena kako bi se u slučaju dvosmislenih poteza odabrao pravilni. Kao primjer dvosmislenog poteza može se navesti top je između 2 pješaka na istom retku šahovske ploče. Ako se top digne i pojede jednog od ta dva pješaka, to se može sa stanovišta ploče protumačiti kao da je top uzeo ili jednog ili drugog pješaka jer je krajnja binarna pozicija figura na ploči ista. Upravo zbog toga se dodatno koristi i lista polja s promjenama kako bi se odabrao pravi potez koji je zapravo odigran. Nakon što je potez potvrđen, sve liste vezane za taj potez se brišu kako se ne bi dogodila greška prilikom sljedećeg ciklusa prepoznavanja nove odigrane pozicije. Takav način je robustan i bez problema može prepoznati sve odigrane poteze.

Nakon razmatranja koda koji upravlja matricama, još je potrebno prokomentirati kod koji se zapravo bavi odabirom poteza.

```
index_jednakosti = -1
lista_indexa = []
for ploca in lista_mogucih_poteza_ploca:
    index_jednakosti += 1
    if np.array_equal(matrica_ploca, ploca):
        provjera_jednakosti = True
        lista_indexa.append(index_jednakosti)
if provjera_jednakosti:
    brojac_provjere += 1
else:
    brojac_provjere = 0
if brojac_provjere >= 10:
    potez_ploca = None
    if len(lista_indexa) == 1:
        potez_ploca = moguci_potezi[lista_indexa[0]]
```

Gore navedeni kod prolazi kroz listu svih matrica nula i jedinica za svaki mogući potez. Ako je među mogućim potezima nađen odigran potez, u listu valjanih poteza se dodaje indeks tog poteza. Ako se 10 puta za redom pojavi isti potez i samo je jedan potez u listi indeksa (to znači da potez nije dvosmislen) onda se taj potez odigra. Ako je potez dvosmislen (nije prikazano u kodu) onda se koristi malo kompleksnija metoda gdje se gleda koji od tih mogućih poteza odgovara na temelju polja na kojima se dogodila promjena. Kada se nađe taj potez on je proglašen odigranim potezom. Kada je odigrani potez određen, povlači se potez u virtualnoj šahovskoj ploči u memoriji, a glavni program pokreće razmišljanje algoritma kako bi se pronašao potez kao odgovor računala. Taj potez će robotska ruka tada odigrati na ploči. Time je objašnjen dio logike iza prepoznavanja poteza na elektroničkoj šahovskoj ploči.

8.2. Generiranje pokreta robotske ruke za dobiveni potez računala

Kada računalo generira potez kao odgovor na korisnikov potez, robotska ruka mora taj potez i odigrati. Kada se robotska ruka giba, to gibanje se sastoji od nekoliko pokreta. Kako bi se moglo pratiti obavlja li robotska ruka posao kako treba, potrebno je sve te pokrete slati jedan po jedan na mikrokontroler robotske ruke. Kada se pokret obavi, iz mikrokontrolera dolazi povratna informacija da je sve u redu i da se može poslati sljedeći pokret. Ako informacija uslijed greške ne dođe unutar 5 sekundi, ponovo se šalje naredba da se obavi isti pokret. Time je dodatno osigurano da greška neće prouzročiti prestanak rada robota uslijed šahovske partije. U nastavku će biti prikazani najvažniji dijelovi koda zajedno s objašnjenjem.

Najprije je napravljeno nekoliko funkcija koje pokrivaju pomicanje figura, uzimanje figura, rokadu, en passant i promociju. U nastavku jedan primjer funkcije za pomicanje figura:

```
def napravi_listu_instrukcija_pomicanje(uci):
    if odabir_boje.get() == 1:
        p1 = koordinate_polja[brojevi_bijeli.index(uci[1])][slova_bijeli.in-
dex(uci[0])]
        p2 = koordinate_polja[brojevi_bijeli.index(uci[3])][slova_bijeli.in-
dex(uci[2])]
        z1 = visine_polja[brojevi_bijeli.index(uci[1])][slova_bijeli.in-
dex(uci[0])]
        z2 = visine_polja[brojevi_bijeli.index(uci[3])][slova_bijeli.in-
dex(uci[2])]
    elif odabir_boje.get() == 2:
        p1 = koordinate_polja[brojevi_crni.index(uci[1])][slova_crni.in-
dex(uci[0])]
        p2 = koordinate_polja[brojevi_crni.index(uci[3])][slova_crni.in-
dex(uci[2])]
        z1 = visine_polja[brojevi_crni.index(uci[1])][slova_crni.in-
dex(uci[0])]
        z2 = visine_polja[brojevi_crni.index(uci[3])][slova_crni.in-
dex(uci[2])]
    x1 = p1[0]
    y1 = p1[1]
    x2 = p2[0]
    y2 = p2[1]
    izlaz = ["kor/"+str(x1)+','+str(y1)+','+str(visina_iznad),
             "kor/"+str(x1)+','+str(y1)+','+str(z1),
             "on",
             "kor/"+str(x1)+','+str(y1)+','+str(visina_iznad),
             "kor/"+str(x2)+','+str(y2)+','+str(visina_iznad),
             "kor/"+str(x2)+','+str(y2)+','+str(z2),
             "off",
             "kor/"+str(x2)+','+str(y2)+','+str(visina_iznad)]
    return izlaz
```

U 2D poljima *koordinate_polja* i *visine_polja* nalaze se koordinate na koje robotska ruka mora doći prilikom operacije na tom polju šahovske ploče. Ovisno o boji figura s kojima igra robot, iz navedenih polja se izvlače koordinate u koje mora doći. Kod ovog primjera za pomicanje figura najvažnije su koordinate polja s kojeg se pomiče figura i koordinate polja na koja se stavlja figura. Kada su koordinate ta dva polja spremljene u varijable *x1*, *y1*, *x2*, *y2*, može se generirati lista pokreta koje robot mora obaviti. Kao što je to vidljivo u kodu, robotska ruka mora prvo doći iznad polja gdje se prvotno nalazi figura, zatim se u sljedećem pokretu mora spustiti na visinu za hvatanje figure. Nakon toga se uključuje hvataljka te se figura prihvaća i u sljedećem pokretu ponovo diže na određenu visinu iznad polja kako prilikom premještanja ne bi udarila u druge figure. Zatim se figura prenosi do odredišnog polja, spušta se, potom se otvara hvataljka, hvataljka se diže iznad figure i robotska ruka se vraća u početni položaj. Svaki taj pokret će se pojedinačno slati na robotsku ruku.

Kada smo napravili sve funkcije za različite poteze, potrebno je napraviti novu funkciju koja će određivati kada će se koja od tih funkcija pozivati.

```
def napravi_listu_instrukcija():
    potez_uci = board.pop()
    potez_uci_string = str(potez_uci)
    potez_san = board.san(potez_uci)

    if board.is_castling(potez_uci):
        if board.is_kingside_castling(potez_uci):
            lista_funkcija = napravi_listu_instrukcija_rokada(True)
        elif board.is_queenside_castling(potez_uci):
            lista_funkcija = napravi_listu_instrukcija_rokada(False)
    elif board.is_en_passant(potez_uci):
        lista_funkcija = napravi_listu_instrukcija_enpassant(potez_uci_string)
    elif potez_uci_string[-1]=='q' or potez_uci_string[-1]=='n':
        uzimanje = False
        if board.is_capture(potez_uci):
            uzimanje = True
        if potez_uci_string[-1]=='q':
            lista_funkcija = napravi_listu_instrukcija_promocija(po-
tez_uci_string, 'q', uzimanje)
        elif potez_uci_string[-1]=='n':
            lista_funkcija = napravi_listu_instrukcija_promocija(po-
tez_uci_string, 'n', uzimanje)
    elif board.is_capture(potez_uci):
        lista_funkcija = napravi_listu_instrukcija_uzimanje(potez_uci_string)
    else:
        lista_funkcija = napravi_listu_instrukcija_pomicanje(potez_uci_string)

    board.push(potez_uci)
    lista_pokreta = ["kor/"+"0,180,150"]
    lista_pokreta.extend(lista_funkcija)
    lista_pokreta.extend(["kor/"+"0,180,150"])
    global buffer_instrukcija
    buffer_instrukcija.extend(lista_pokreta)
```

Pomoću gore prikazane funkcije prvo se provjerava je li odigrani potez od računala rokada. Ako je rokada poziva se funkcija za određivanje pokreta za rokadu. Nakon toga se provjerava je li potez *en passant* te se isto tako poziva pripadna funkcija. Potom se provjerava je li potez promocija, a nakon toga je li potez uzimanje. Na kraju, ako ništa od toga nije slučaj, zaključuje se da je potez obično pomicanje figure, te se poziva ta funkcija. Kao dodatak na pokrete određene funkcijom kasnije se dodaje ispred i iza te liste još jedan pokret koji vraća robotsku ruku u početni položaj. Kada je lista pokreta napravljena, potez računala se odigra u glavnoj virtualnoj ploči. Nakon toga se u *buffer* dodaje lista svih pokreta koje treba obaviti. Iz tog *buffera* se pojedinačno šalju svi pokreti i očekuje se odgovor mikrokontrolera. U sljedećem dijelu koda će se objasniti način na koji se ti pokreti šalju i kako se za njih očekuje povratna informacija nakon što se oni obave.

```
def funkcija_thread_ruka():
    global buffer_instrukcija
    global buffer_slanje
    global pokret_checkpoint
    while True:
        time.sleep(0.1)
        try:
            if homing_checkpoint:
                if buffer_instrukcija:
                    trenutna_instrukcija = buffer_instrukcija.pop(0)
                    buffer_slanje.append(trenutna_instrukcija)
                    start_vrijeme = time.time()
                    while not pokret_checkpoint:
                        trenutno_vrijeme = time.time()
                        proteklo_vrijeme = trenutno_vrijeme - start_vrijeme
                        if proteklo_vrijeme > 5:
                            buffer_slanje.append(trenutna_instrukcija)
                            start_vrijeme = time.time()
                        time.sleep(0.1)
                    pokret_checkpoint = False
        except Exception as e:
            print(e)
```

Gore navedena funkcija je zamišljena da se pokreće na posebnom *threadu* kako bi se ostvarilo multiprocesiranje, odnosno da glavni program ne bi čekao izvršenje ove funkcije nego se ta funkcija cijelo vrijeme vrti u pozadini. U *while* petlju je dodana pauza od 100 milisekundi kako se procesor ne bi bespotrebno zagušio. Prvo se provjerava je li obavljen *homing* robota, jer ako nije onda robotska ruka ne zna gdje se nalazi u prostoru i bilo kakav pomak bi mogao oštetiti robotski sustav. Ako je *homing* obavljen, iz *buffera* se uzima prvi pokret te se šalje na mikrokontroler robotske ruke. Pokreće se nova beskonačna petlja i isto vrijeme se krene bilježiti početno vrijeme. Ako s mikrokontrolera uslijed greške ne dođe povratna informacija da je pokret obavljen, ta petlja bi se vrtjela beskonačno, a program se nikada ne bi pomakao dalje iz te petlje. Zbog toga se mjeri vrijeme proteklo u toj petlji. Ako prođe više od 5 sekundi, tada se ponovo na mikrokontroler šalje da robot treba doći u istu poziciju, te se time sprječava bilo kakva pojava greške koja bi prekinula rad cijelog programa. Kada dođe povratna informacija da je pokret obavljen, petlja se gasi, a funkcija poziva sljedeći pokret iz *buffera*.

Na mikrokontroler robotske ruke povezuje se i mobitel preko bluetootha te se s tog mikrokontrolera očekuju naredbe za upravljanje funkcijama robota. Sljedećim kodom biti će definirane ključne riječi, koje kada dođu na računalo pokreću razne funkcije. Isto tako, vidljiv je cijeli postupak kako se procesira znak po znak prilikom dolaska putem serijske komunikacije te kako se prepoznaje kada je obavljen neki pokret robotske ruke.

```
if esp32.in_waiting:
    dolazno = ""
    while esp32.in_waiting:
        char = esp32.read().decode("utf-8")
        if char == '%':
            homing_checkpoint = True
        elif char == '$':
            pokret_checkpoint = True
        elif char == '#':
            break
        else:
            dolazno += char
    print(dolazno)
    if len(dolazno) > 0:
        if dolazno == "homingMOB":
            thread_homing.start()
        elif dolazno == "sklopiMOB":
            if homing_checkpoint:
                buffer_slanje.append("sklopi")
        elif dolazno == "iskljuci":
            iskljuciRPI()
        elif dolazno == "vrati":
            vrati_potez()
        elif dolazno == "pomoc":
            if homing_checkpoint:
                pomoc()
        elif dolazno[0:4] == "nova":
            dijelovi = dolazno.split('/')
            odabir_boje.set(dijelovi[1]) # bijeli - 1, crni - 2
            odabir_boje_radio_button.set(dijelovi[1])
            odabir_engine.set(dijelovi[2]) # stockfish - 1, AI - 2
            postavi_tezinu(dijelovi[3])
            provjeri_slider(dijelovi[3])
            root.update_idletasks()
            nova_igra()
```

Analogno obradi dolaznih podataka putem serijske komunikacije koja je objašnjena u prošlom poglavlju na primjeru mikrokontrolera, ovdje se vrši identična petlja za obradu podataka. Svaki dolazni znak se pojedinačno čita u varijablu. Ako je taj znak „%“, to znači da je mikrokontroler poslao informaciju da je *homing* obavljen. Ako je znak „$“, to znači da je obavljen pokret robotske ruke. Znakom „#“ označen je kraj svake dolazne naredbe. Kada je cijela naredba očitana, ona se u donjem dijelu koda uspoređuje s prije definiranim naredbama. Ako dolazna naredba odgovara nekoj od zadanih naredbi, tada se pokreće pripadna funkcija. Sve moguće naredbe su lako vidljive iz koda iznad. Kada se želi pokrenuti nova igra, tada se još dodatno isčitavaju zadane postavke nove igre. To su šahovski *engine*, težina računalnog protivnika i boja figura s kojom igrač igra. Na temelju tih informacija se dinamički podese varijable nove igre, te se virtualna šahovska ploča resetira u početnu poziciju. Ovim kodom pokrivena je cijela dolazna komunikacija s mikrokontrolera.

8.3. Korisničko sučelje (GUI)

Kako bi se virtualna šahovska ploča koja je u memoriji mogla lijepo prikazati, pomoću Pythonovog modula Tkinter napravljeno je grafičko korisničko sučelje. U sklopu ovog poglavlja biti će prikazano nekoliko glavnih dijelova koda koji omogućuju prikaz igre.

```
canvas=Canvas(root, width=900, height=900)
canvas.place(relx=0.375, rely=0.5, anchor=CENTER)
canvas.config(bg='antique white')
canvas.create_rectangle(50, 50, 850, 850, fill='#F0D9B5')
for i in range(8):
    for j in range(8):
        if (i+j)%2!=0:
            canvas.create_rectangle(50+100*i, 50+100*j, 50+100*(1+i),
50+100*(1+j), fill='#B58863', outline='')
canvas.create_rectangle(50, 50, 850, 850,width=5, fill='', outline='saddle
brown')
```

Kao što je to vidljivo u kodu iznad, prvo se kreira *canvas* dimenzija 900x900 piksela na kojem će se dalje crtati šahovnica, imena polja i figure. Unutar tog *canvasa* crta se obrub i pomoću *for* petlje se radi šahovnica. Jednostavnim zbrajanjem piksela unutar *for* petlje crtaju se tamna šahovska polja na svakom drugom mjestu, dok ostala polja ostaju boje pozadine. U globalu nakon što je petlja završila, dobiva se izgled šahovnice šahovske ploče. Preostalo je još napraviti oznake za slova i brojeve polja šahovske ploče:

```
def napravi_oznake(boja):
    izbrisi_oznake()
    oznake.clear()
    brojevi = list('12345678')
    slova = list('abcdefgh')
    if boja == "bijeli":
        for i in range(8):
            oznake.append(canvas.create_text(100*(i+1),875,fill="saddle
brown",font="Arial 15 bold",text=slova[i]))
        for j in range(8):
            oznake.append(canvas.create_text(25,100*(j+1),fill="saddle
brown",font="Arial 15 bold",text=brojevi[7-j]))
    if boja == "crni":
        for i in range(8):
            oznake.append(canvas.create_text(100*(i+1),875,fill="saddle
brown",font="Arial 15 bold",text=slova[7-i]))
        for j in range(8):
            oznake.append(canvas.create_text(25,100*(j+1),fill="saddle
brown",font="Arial 15 bold",text=brojevi[j]))
```

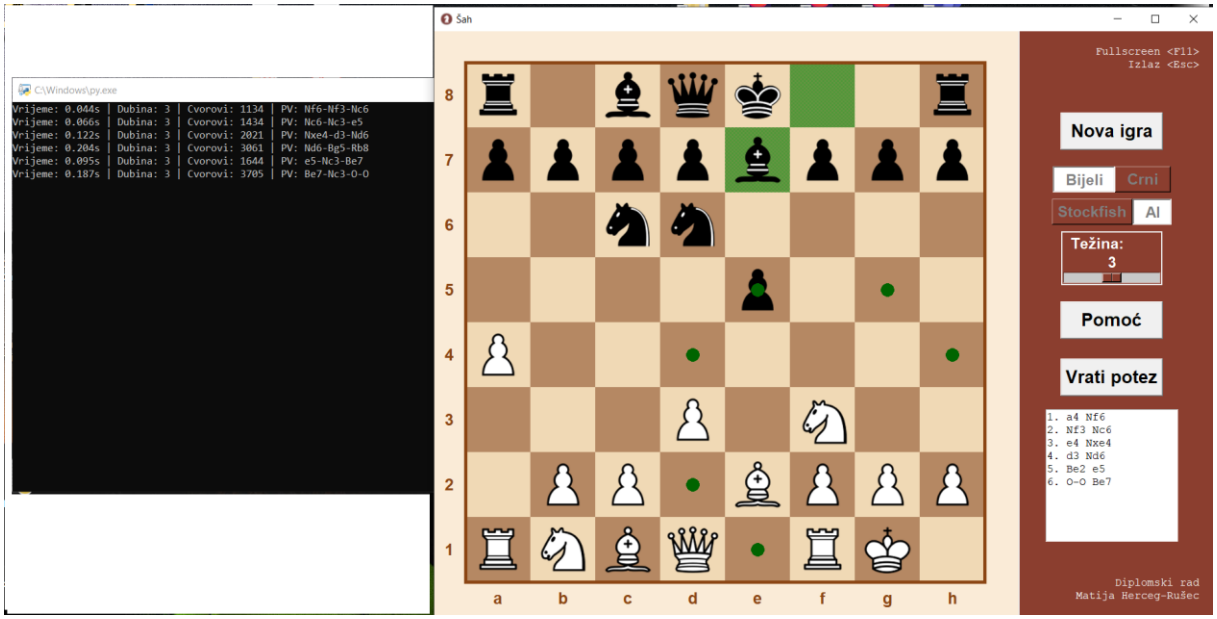
Ovisno o tome je li igrač crne ili bijele boje pokreću se dvije *for* petlje koje stvaraju različite oznake. Prva for petlje horizontalno na dnu ploče ispisuje slova koja predstavljaju imena stupaca. Druga for petlja ispisuje vertikalno brojeve redova na šahovskoj ploči. Na taj jednostavan način izvedene su oznake šahovskih polja.

Na kraju se na gotovu šahovsku ploču dodaju slike figura [3] koje su preuzete s open-source šahovske web stranice *Lichess* pomoću for petlje.

```
    for red in range(8):
        for stupac in range(8):
            if ploca[red][stupac] == 'r':
                figure.append(canvas.create_image(50+stupac*100,50+red*100,
anchor=NW, image=bR))
            if ploca[red][stupac] == 'n':
                figure.append(canvas.create_image(50+stupac*100,50+red*100,
anchor=NW, image=bN))
            if ploca[red][stupac] == 'b':
                figure.append(canvas.create_image(50+stupac*100,50+red*100,
anchor=NW, image=bB))
            if ploca[red][stupac] == 'q':
                figure.append(canvas.create_image(50+stupac*100,50+red*100,
anchor=NW, image=bQ))
            if ploca[red][stupac] == 'k':
                figure.append(canvas.create_image(50+stupac*100,50+red*100,
anchor=NW, image=bK))
            if ploca[red][stupac] == 'p':
                figure.append(canvas.create_image(50+stupac*100,50+red*100,
anchor=NW, image=bP))
            if ploca[red][stupac] == 'R':
                figure.append(canvas.create_image(50+stupac*100,50+red*100,
anchor=NW, image=wR))
            if ploca[red][stupac] == 'N':
                figure.append(canvas.create_image(50+stupac*100,50+red*100,
anchor=NW, image=wN))
            if ploca[red][stupac] == 'B':
                figure.append(canvas.create_image(50+stupac*100,50+red*100,
anchor=NW, image=wB))
            if ploca[red][stupac] == 'Q':
                figure.append(canvas.create_image(50+stupac*100,50+red*100,
anchor=NW, image=wQ))
            if ploca[red][stupac] == 'K':
                figure.append(canvas.create_image(50+stupac*100,50+red*100,
anchor=NW, image=wK))
            if ploca[red][stupac] == 'P':
                figure.append(canvas.create_image(50+stupac*100,50+red*100,
anchor=NW, image=wP))
```

Kao što je to već objašnjeno u prethodnim cjelinama, virtualna šahovska ploča može biti prikazana kao 2D matrica koja na svakom popunjenom polju ima slovnu oznaku figure, dok su prazna polja popunjena točkom. Pomoću for petlje se jednostavno prolazi kroz tu 2D matricu. Na svim mjestima na kojima je neko od slova se pomoću gornjeg koda prepoznaje o kojoj se figuri radi te se na pripadno polje postavlja slika figure. Ponovo se slika jednostavno može dodati pomoću koordinata u *canvasu* koji je kreiran na početku. Tako dodane slike je uvijek moguće pomaknuti uređivanjem koordinata na kojima se one nalaze. Kako bi se taj pomak dogodio koristi se posebna funkcija koja prati pokret pokazivača miša te bilježi koordinate.

Ako je kliknuto na figuru, pomicanjem miša se za tu sliku uređuju koordinate pozicije. Nakon otpuštanja klika miša, provjerava se je li moguće po pravilima šaha tu figuru pomaknuti na to polje. Također, dodane su zelene točkice koje označavaju moguće poteze prilikom klika na figuru. Kod za tu funkciju je kompleksan te zbog toga neće biti razmatran u sklopu ovog poglavlja. Početno i krajnje polje posljednjeg poteza označeno je zelenom bojom kako bi se moglo lakše pratiti tijek igre.



Slika 17. Izgled grafičkog korisničkog sučelja

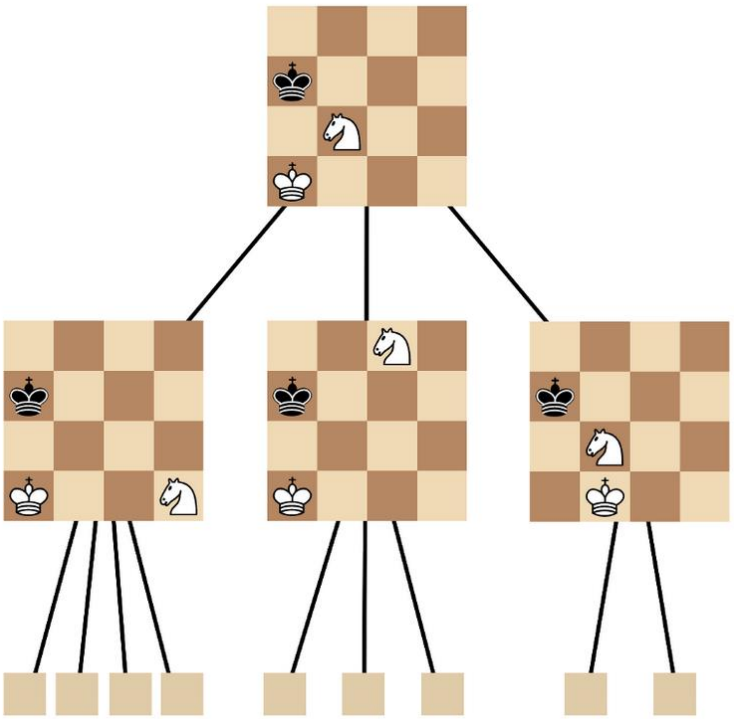
Na slici 17. prikazan je konačan izgled grafičkog korisničkog sučelja. Opisano kreiranje izgleda šahovske ploče vidljivo je s lijeve strane glavnog prozora, dok je desna strana rezervirana za tipke i ispis odigranih poteza u šahovskoj partiji. S lijeve strane otvara se konzolni prikaz koji daje rezultate algoritma za odabir poteza računalnog protivnika. Više o algoritmu i umjetnoj inteligenciji opisano je u sljedećim poglavljima.

9. ALGORITAM TRAŽENJA ZA IGRANJE ŠAHA

Program koji određuje potez koji će odigrati računalo se sastoji od dva dijela: algoritma za traženje i funkcije za evaluaciju. U ovom poglavlju će biti pokriven algoritam za traženje, dok će se u sljedećem poglavlju pokriti funkcija evaluacije pomoću neuronske mreže. Algoritam za traženje funkcionira na način da se prolazi kroz sve moguće poteze, za svaki od tih mogućih poteza se traže mogući potezi protivnika, zatim se ponovo traže svi mogući potezi na te poteze i tako do dubine do koje je traženje zadano. Ti potezi se eksponencijalno granaju te se takva struktura naziva *stablo poteza*. Na svakoj poziciji u tom stablu poteza poziva se evaluacijska funkcija koja boduje poziciju, te se na temelju tih bodova može pratiti koji je potez najbolji. Kako se stablo poteza eksponencijalno grana s povećavanjem dubine traženja, koriste se razne metode pomoću kojih se to stablo smanjuje eliminacijom očigledno loših poteza. Upravo su te metode dio koji će se razmatrati u ovoj cjelini, zajedno s osnovnim načinom rada traženja po stablu poteza. Zbog izrazite složenosti i isprepletenosti različitih dijelova koda, neće se posebno prikazivati i objašnjavati kod koji izvršava funkcije pokrivene u ovom poglavlju, već će se detaljno razmotriti ideja iza svake metode algoritma traženja za igranje šaha. Kod za algoritam traženja i za funkciju evaluacije pomoću umjetne inteligencije je pisan u Python-u.

9.1. Stablo poteza

Kao što je objašnjeno u uvodu ovog poglavlja, stablo poteza nastaje tako da se prolazi kroz sve moguće pozicije do određene dubine počevši od trenutne pozicije na šahovskoj ploči. Na slici 18. grafički je prikazan način rada stabla poteza. Sve počinje od početne pozicije prikazane u gornjem dijelu slike. Nakon toga, na dubini 1, gledaju se svi mogući potezi koji se mogu napraviti bijelim skakačem. Nakon toga na sljedećoj dubini 2, gledaju se svi mogući potezi crnog igrača kao odgovor na bilo koji od poteza bijelog igrača iz prošle dubine. Proces se tako nastavlja do određene zadane dubine prilikom pokretanja algoritma. Prosječno se na dubini 1 može očekivati oko 40 različitih pozicija, dok se eksponencijalno na dubini 2 već može očekivati 1600 poteza [5]. Gledajući te podatke, čini se da to nije složen problem, te računala mogu bez problema brzo prolaziti kroz pozicije. Šah je igra kojoj bi se mogli izračunati svi mogući potezi, no za to bi bili potrebni enormni resursi koji su daleko ispred trenutne tehnologije. U nastavku će se razmotriti konkretni podaci.



Slika 18. Grafički prikaz stabla šahovskih poteza [4]

Iako je 1600 pozicija na dubini 2 relativno jednostavno obraditi današnjom računalnom snagom, na primjer na dubini 8 potrebno je prosječno obraditi 6 553 600 000 000 [5]. Najbolji šahovski programi mogu obraditi oko 50 milijuna pozicija u sekundi [6]. Ako se preračuna, za dubinu 8 najboljem šahovskom programu treba 91 dan punog rada. Još zanimljivija činjenica je da današnji najbolji šahovski programi igraju prosječno na dubini između 20 i 25. Broj mogućih pozicija na dubini 20 je enormno velik (40^{20}), pa na prvu nikako nije jasno kako šahovski program to uspijeva. Naravno, kako bi se postigla ta dubina, programi se koriste raznim trikovima koji eliminiraju veliku količinu poteza koji su loši. Većina tih metoda će biti objašnjena u sljedećim dijelovima ovog poglavlja. Također, treba imati na umu da eliminacijom pozicija dobivamo na dubini koja se može postići u realnom vremenu, ali gubimo na točnosti određivanja najboljeg poteza. Tako je moguće da jedan šahovski program bude bolji od drugog jer nađe neko kreativno rješenje koje drugi program na početku izbaci jer se čini loše, no dugotrajno taj potez ima velike benefite. Šahovski programi će se uvijek moći poboljšavati, jer se nikada tijekom odabira poteza ne provjere svi mogući potezi pa prema tome uvijek je moguće da se najbolji mogući potez izostavi. Korištenjem umjetne inteligencije, na primjer, moguće je ostvariti drugačiji pristup odabiru poteza, te time sagledati i druge mogućnosti.

9.2. Alfa-beta pruning metoda

Prva i najvažnija metoda redukcije broja pozicija koje se pregledavaju je metoda zvana *alfa-beta pruning*. Pomoću te metode se uklanjaju oni potezi (pozicije) koji su stvarno loši, dok svi preostali potezi ostaju te se za njih pokreće evaluacija. Zbog toga se ovom metodom ne gubi točnost algoritma, već se samo ubrzava izvođenje. Upravo zbog te metode je računalima omogućeno *brute force* procesiranje šahovskih poteza, jer bi bez toga mogli maksimalno doseći dubinu 5-6 što nikako nije dovoljno da se pobijedi u igri protiv profesionalnih šahista.

Ova metoda funkcionira na način da se tijekom traženja poteza prate dvije nove varijable alfa i beta. Te varijable spremaju najgori mogući ishod u trenutnoj grani stabla poteza. Prilikom prolaska kroz poteze uvijek se pokušava odrediti koji najbolji mogući potez protivnik može odigrati, tj. koji je najnepovoljniji potez sa stanovišta onoga tko računa potez pomoću algoritma (računalo). Ako je u jednoj grani pronađen izrazito nepovoljan potez koji je spremljen u varijablu alfa ili beta, on služi kao referenca. Ako se u sljedećoj grani pronađu nepovoljni potezi koji nisu toliko nepovoljni kao referenca, ti svi potezi se mogu zanemariti jer će protivnik uvijek birati najnepovoljniji. Time se stablo poteza drastično sužava.

Tablica 1. Poboljšanje u broju obrađenih pozicija nakon primjene alfa-beta pruninga [5]

Dubina	Čisto stablo poteza	Alfa-beta pruning
0	1	1
1	40	40
2	1600	79
3	64 000	1 639
4	2 560 000	3 199
5	102 400 000	65 569
6	4 096 000 000	127 999
7	163 840 000 000	2 623 999
8	6 553 600 000 000	5 119 999

U tablici 1. vidljivo je ekstremno poboljšanje performansi algoritma. Za dubinu 8 više nije potreban 91 dan, već samo 100 milisekundi (pri brzini od 50 milijuna poteza u sekundi [6]). Međutim, pogledom na tablicu je vidljivo da je za očekivati da će se s povećanjem dubine broj poteza i dalje drastično povećavati. Kako bi se to širenje stabla pozicija još dodatno smanjilo, razmotrit će se još nekoliko metoda u nastavku.

9.3. Quiescence pretraga

Ova metoda nije zamišljena da smanji broj pozicija koje se obrađuju, nego da ukloni takozvani *horizon* efekt. Ako na primjer algoritam računa pozicije do dubine 8 i na toj dubini je moguć potez uzimanje kraljice, taj potez će dobiti najveći broj bodova i računalo će ga gotovo vjerojatno odigrati. Međutim, računalo u tom slučaju ne vidi što se događa iza toga na dubini 9+. Možda će nakon toga računalo izgubiti i svoju kraljicu i još nekoliko drugih figura te time izgubiti partiju. Pomoću *quiescence* pretrage taj efekt se eliminira. Pretraga se pokreće za svaki potez koji je uzimanje ili šah na zadnjoj dubini stabla pozicija. Ona prolazi kroz sva uzimanja ili šahove koji su mogući do beskonačne dubine. Suprotno logici stabla pozicija, potezi uzimanja ili šahova ne rastu eksponencijalno, nego nakon nekoliko dodatnih dubina trnu. Kod središnjice šahovske partije, *quiescence* pretraga puno produbljuje osnovno stablo pozicija jer postoji mnogo poteza koji su uzimanje ili šah. Zbog toga je algoritam najsporiji tijekom središnjice, a najbrži tijekom otvaranja ili završnice šahovske partije. Pomoću opisane metode uspješno se uklanja *horizon* efekt, a algoritam više ne odabire poteze koji bi u nastavku igre mogli biti pogubni za ishod šahovske partije.

9.4. Glavna varijacija

Kada algoritam traženja odabere potez, taj potez je odabran na temelju ishoda na posljednjoj zadanoj dubini pretrage. Na primjer ako je zadana dubina pretrage 4, na temelju 4 poteza unaprijed biti će odabran potez. Ako se zapišu svi ti potezi do zadane dubine 4 dobije se glavna varijacija. Varijacija za dubinu 4 prema tome može izgledati ovako: e4-e5-Nf3-Nc6. U toj glavnoj varijaciji su vidljivi svi potezi po redu za koje računalo misli da će biti za njega najbolji ishod, a u isto vrijeme za sve igračeve poteze iz te liste 4 poteza računalo smatra da su to najbolji potezi koje igrač može odigrati. Glavna varijacija sama po sebi nema nikakvih benefita, no u kombinaciji s *iterativnim produbljivanjem* (sljedeća tema) postiže dobre rezultate.

9.5. Iterativno produbljivanje

Iterativno produbljivanje je metoda kod koje se pomoću stabla pozicija ne pretražuje odmah na najvećoj dubini nego se postepeno pretražuje od dubine 1 do željene dubine. Na prvu se to čini kao dodatno bespotrebno povećavanje broja poteza koji se traže, no gledajući tablicu 1. vidljivo je da zbroj svih traženih pozicija na dubini *1* do *n-1* uvijek značajno manji nego broj poteza na dubini *n*. Ta metoda se uvijek koristi u kombinaciji s glavnom varijacijom, jer se na taj načina dodatno smanjuje broj traženih pozicija. Počevši od dubine 1 stvara se glavna varijacija. Na sljedećoj dubini 2 se kao prvi potez koji se provjerava uzima potez iz glavne varijacije s dubine 1. Time se postižu dovoljno dobre početne vrijednosti alfa i beta (alfa-beta pruning) pomoću kojih se ocjenjuju pozicije. Ako su te vrijednosti alfa i beta na početku dobro određene, puno poteza će se ignorirati te će se pretraga uvelike poboljšati. Na svakoj sljedećoj dubini se uzima glavna varijacija s prethodne dubine. Kao rezultat toga dobiva se oko 20% manje obrađenih pozicija, a time i toliko brže vrijeme izvođenja.

9.6. Null-move pruning metoda

Null-move pruning se zasniva na ideji da ako se potez ne pokaže dobrim u narednih nekoliko poteza da se onda vjerojatno niti neće pokazati dobrim u budućnosti. Tako se pozivom te metode prvo provjerava prvih nekoliko poteza dubine i ako se ustanovi velika nepovoljnost pozicije, takvi potezi se eliminiraju i više se ne pretražuju kada se pretražuje do pune dubine. Isto tako, pošto se prvo poziva ta metoda, postavljaju se inicijalne vrijednosti alfa i beta koje kasnije služe za bržu eliminaciju loših poteza kada se uđe u alfa-beta pruning petlju. Međutim, potrebna je velika opreznost prilikom programiranja dotične metode. Metoda se nikako ne smije pozivati ako je trenutno na ploči šah, a računalo treba napraviti potez. Isto tako, treba izbjegavati tu metodu ako je na ploči vrlo malo figura jer se u tom slučaju može javiti *zugzwang*, što drugim riječima znači neizbježno loš potez. To se može reprezentirati, kada se npr. kralj mora maknuti od figure koju brani jer nema drugih poteza i onda ta figura biva uzeta, gdje je rezultat toga vrlo loš za računalo. Korištenje *null-move pruninga* u tom slučaju bi eliminiralo takav loš potez koji je ujedno jedini potez te bi nastala greška u algoritmu, a ako ima više mogućih poteza možda bi se eliminirali potezi koji su u budućnosti zapravo dobri kada se pretraže punim opsegom dubine pretrage. Korištenjem te metode ponovo se može očekivati oko 20% manje pozicija koje se pretražuju a time i dodatno ubrzanje algoritma.

9.7. „Ubojiti“ potezi (engl. Killer moves)

Prilikom prolaženja kroz poteze do sada se prvo provjeravala glavna varijacija a nakon toga svi ostali potezi generirani po nasumičnom redoslijedu. Te sve poteze moguće je na neki način poredati kako bi se čim prije podesile alfa i beta vrijednosti a time i čim prije dobilo smanjivanje broja poteza. „Ubojiti“ potezi su svi potezi koji su prouzročili eliminaciju grane tijekom izvođenja *alfa-beta pruninga*. Ti potezi se spremaju u posebnu listu. Prilikom generiranja mogućih poteza u svakoj grani, nakon poteza iz glavne varijacije provjerava se jeli koji od preostalih mogućih poteza iz liste „ubojitih“ poteza. Ako je ti potezi se provjeravaju odmah nakon poteza iz glavne varijacije kako bi se čim brže postigle referentne vrijednosti alfa i beta. Ovom metodom postignuto je ubrzanje od oko 10%.

9.8. MVV-LVA sortiranje

Sada se na temelju prethodnih metoda uz potez iz glavne varijacije na početku svake grane provjeravaju i „ubojiti“ potezi. Kako bi se postiglo još brže određivanje dobrih referentnih vrijednosti alfa i beta, preostali mogući potezi se moraju poredati. Kratica MVV-LVA dolazi iz engleskog jezika i znači Most Valuable Victim – Least Valuable Aggressor, što u prijevodu znači najvrijednija meta – najmanje vrijedan napadač. Ta metoda označava preostale poteze koji su uzimanje pomoću određenog broja bodova te ih potom poreda od najvećeg broja bodova do najmanjeg. Najveći broj bodova dobije onaj potez gdje je figura koja se uzima najvrjednija, a u isto vrijeme figura koja uzima tu figuru najmanje vrijedna. Npr. kada pješak uzima damu, taj potez će biti bodovan s najviše bodova, te će se prvo provjeravati. Uz korištenje ove metode, sada se prvo provjeravaju potezi iz glavne varijacije, nakon toga „ubojiti“ potezi, te svi potezi koji su uzimanje poredani pomoću bodova. Na kraju se ostavljaju svi potezi koji nisu uzimanje. Na taj način je postignut dobar poredak poteza koji će jako brzo postići dobre referentne vrijednosti alfa i beta, a time i smanjiti broj poteza koji se pretražuju. Korištenjem ove metode postiže se poboljšanje od 15%.

10. EVALUACIJA UMJETNOM INTELIGENCIJOM

Nakon što je opisan dio šahovskog programa za pretragu kroz poteze, preostaje dio koji će bodovati (evaluirati) svaku poziciju koja se pretražuje. Obično najbrži šahovski programi pozicije evaluiraju na način da se boduje zbroj vrijednosti svih figura na ploči za svaku boju, ključne pozicije figura, mobilnost i drugi aspekti pozicije. Na taj način dobivaju se gotovo savršene evaluacije pozicija, no potezi koje odigra računalo gotovo uvijek izgledaju baš kao da ih je odigralo računalo – savršeno i nemaštovito. Zbog toga je u ovom radu za evaluaciju korištena umjetna inteligencija u obliku neuronske mreže. Najprije se na temelju podataka za učenje istrenira model neuronske mreže pomoću kojeg će se kasnije evaluirati pozicija. Takve evaluacije su nesavršene, a odabrani potezi su gotovo uvijek kreativniji i bliži ljudskom načinu razmišljanja. Također je bitno da napomenuti da je primjenom neuronske mreže kao funkcije evaluacije značajno usporeno traženje poteza, te će ukupna dubina pretrage biti dosta manja.

10.1. Ideja

Ideja korištenja neuronske mreže je napraviti evaluaciju koja će omogućiti odabir kreativnih poteza, koji će više izgledati kao da ih igra čovjek. Nije ideja napraviti savršenu evaluaciju, već dovoljno dobru da učini šahovsku partiju zanimljivom. Neuronska mreža također mora davati dovoljno dobru evaluaciju, kako bi program mogao pobijediti srednje vještog protivnika. Kako bi se to ostvarilo koriste se partije šahovskih velemajstora da bi se izradili podaci za učenje. Na temelju tih podataka trenira se duboka neuronska mreža s 3 skrivena sloja koja može naučiti komplicirane sustave pravila. Jednom kada se mreža istrenira, gotov model će se uvesti prilikom pokretanja glavnog šahovskog programa. Evaluacija će se pokretati u svakoj poziciji koju će obraditi algoritam traženja i davati evaluacije prema istreniranom modelu.

10.2. Podaci za učenje

Kao što je prije spomenuto, podaci za učenje se kreiraju na temelju partija šahovskih velemajstora iz lako dostupnih besplatnih baza na internetu [7]. Sve partije su zapisane u PGN formatu te ih je potrebno obraditi kako bi se dobili valjani podaci kao ulaz u neuronsku mrežu. Kada se partije preuzmu iz baze, one su spremljene u tekstualne datoteke koje izgledaju ovako:

```
[Event "F/S Return Match"]
[Site "Belgrade, Serbia JUG"]
[Date "1992.11.04"]
[Round "29"]
[White "Fischer, Robert J."]
[Black "Spassky, Boris V."]
[Result "1/2-1/2"]

1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 4. Ba4 Nf6
5. O-O Be7 6. Re1 b5 7. Bb3 d6 8. c3 O-O
9. h3 Nb8 10. d4 Nbd7 11. c4 c6 12. cxb5 axb5
13. Nc3 Bb7 14. Bg5 b4 15. Nb1 h6 16. Bh4 c5
17. dxe5 Nxe4 18. Bxe7 Qxe7 19. exd6 Qf6
20. Nbd2 Nxd6 21. Nc4 Nxc4 22. Bxc4 Nb6
23. Ne5 Rae8 24. Bxf7+ Rxf7 25. Nxf7 Rxe1+
26. Qxe1 Kxf7 27. Qe3 Qg5 28. Qxg5 hxg5
29. b3 Ke6 30. a3 Kd6 31. axb4 cxb4
32. Ra5 Nd5 33. f3 Bc8 34. Kf2 Bf5 35. Ra7 g6
36. Ra6+ Kc5 37. Ke1 Nf4 38. g3 Nxh3
39. Kd2 Kb5 40. Rd6 Kc5 41. Ra6 Nf2
42. g4 Bd3 43. Re6 1/2-1/2
```

Kako bi se dobio valjan ulaz za neuronsku mrežu, prvo je potrebno obrisati informacije o partiji s početka zapisa, rezultat i brojeve polja, a poteze u PGN formatu spremiti u slijednu listu. Iz te liste će se potezi redom pozivati u virtualnoj šahovskoj ploči te se na temelju te ploče stvarati binarni vektor koji će direktno služiti kao ulaz u neuronsku mrežu. Vektor se sastoji od 768 bitova, a napravljen je iz 3D matrice šahovske ploče. 3D matrica šahovske ploče se dobiva tako da se uzmu prve dvije očigledne dimenzije – horizontalna i vertikalna pozicija polja, koje tvore 2D matricu dimenzija 8x8 te se na njih doda treća dimenzija veličine 12. veličina treće dimenzije je 12 jer postoji 6 različitih tipova figura u dvije različite boje. Konačna 3D matrica je dimenzija 8x8x12, a nakon što se kreira se pretvara u već spomenuti binarni vektor duljine 768. Nakon toga, pomoću profesionalnog šahovskog programa Stockfish [8] se kreiraju evaluacije koje će učiti neuronska mreža. Ključno je postaviti kapacitet neuronske mreže manji nego je količina podataka za učenje kako bi se spriječio *overfitting*, što znači da bi se podaci samo naučili bez generalizacije na druge nepoznate ulazne podatke. Za učenje neuronske mreže korišteno je 5 010 315 šahovskih pozicija pretvorenih u vektor i njima pripadnih evaluacija. Korišteni model neuronske mreže ima 100 569 parametara koje je moguće trenirati, tako da će teško doći do *overfittinga* zbog velike količine ulaznih podataka. Kada neuronska mreža bude istrenirana, treba očekivati logičan izlaz za dani ulaz. Ako su sve evaluacije normalizirane između -1 i 1, tada mreža mora također davati takav izlaz na bilo koji ulazni podatak. Nakon što su svi podaci spremni, potrebno je napraviti algoritam za učenje i model neuronske mreže.

10.3. Algoritam za učenje

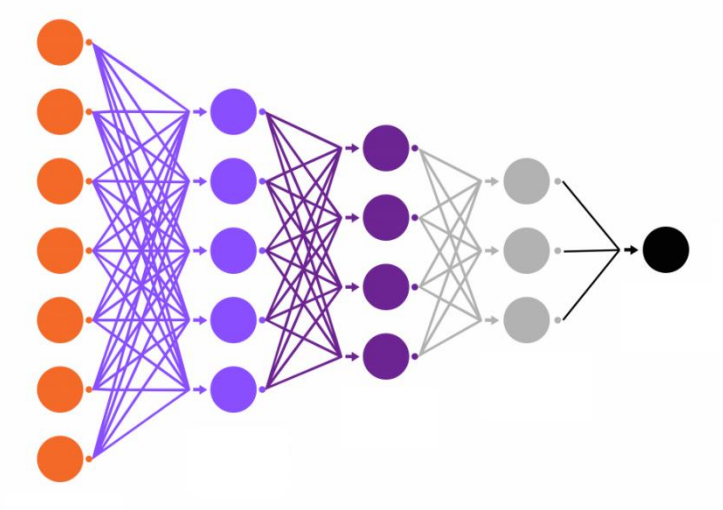
Za algoritam učenja neuronske mreže, a isto tako za kasnije pokretanje gotovog modela, koristit će se Tensorflow [9]. To je biblioteka koja omogućuje jednostavno i izuzetno efikasno i brzo treniranje neuronske mreže pomoću grafičke kartice računala, a vrlo je jednostavan za korištenje iz Python-a gdje je do sada izrađen ostatak programa za šah.

Prvi korak kod izrade algoritma za učenje je izrada ulaznog *buffera* za podatke. Taj korak je potreban jer se tolika količina podataka (5 milijuna) ne može direktno učitati u RAM memoriju. Ideja *buffera* je da se podaci redom učitavaju u grupe određene veličine, koje najčešće iznose 16,32,64,128 podataka po grupi. Uvijek se stvara desetak grupa unaprijed kako bi prilikom završetka obrade prve grupe podataka mogla odmah pokrenuti obrada druge grupe, kako bi se maksimalno iskoristilo vrijeme. Takav način obrade podataka je brz i efikasan.

Srce svakog algoritma za učenje je optimizacijska funkcija. Tijekom izrade ovog diplomskog rada, jedna od najnaprednijih, najbržih i najčešće korištenih metoda optimizacije je metoda *Adam* [10]. Ta metoda je dostupna unutar Tensorflow biblioteke te se jednostavno poziva pomoću jedne linije koda. *Adam* se temelji na gradijentu prvog reda za optimizaciju stohastičkih funkcija. Bazira se na adaptivnim estimacijama momenata nižeg reda. Ta metoda je jednostavna za implementaciju te izrazito računalno efikasna, a pritom zahtijeva vrlo malo memorije. Kao što je to navedeno u znanstvenom radu [10], korištenjem te metode se dobivaju jedni od najboljih rezultata u vidu brzine konvergencije točnosti modela. To svojstvo je iznimno važno prilikom treniranja mreže s velikim brojem podataka, jer se inače treniranje ne bi moglo izvesti u realnom vremenu bez korištenja posebnih specijalnih računala.

10.4. Model neuronske mreže

Kako bi se kod kompliciranе evaluacije u igri poput šaha uspjele unutar modela prepoznati neke karakteristike i uzorci koji se boduju, potrebno je koristiti duboku neuronsku mrežu. Svaka neuronska mreža se sastoji od ulaznog sloja, skrivenih slojeva i izlaznog sloja [11]. U ovoj strukturi neuronske mreže, svi neuroni su između slojeva međusobno povezani, kao što je to prikazano na slici 19, a svaka veza ima svoju određenu težinu koja se podešava tijekom učenja. Učenje završava jednom kada trenutne težine daju zadovoljavajuće rezultate na izlazu.



Slika 19. Grafički simbolički prikaz korištene neuronske mreže [12]

Korištena mreža na ulaznom sloju ima 768 neurona, gdje se svaki bit iz ulaznog vektora pozicije šalje na jedan neuron. Nakon toga su postavljena tri skrivena sloja, redom veličina 128, 16 i 4 neurona. A na izlazu se nalazi jedan neuron koji daje konačnu evaluaciju pozicije. Ovakvim postepenim smanjivanjem boja neurona u svakom sljedećem sloju, dobiva se efekt prepoznavanja uzoraka, što je korisno u slučaju učenja evaluacije šahovskih poteza. Navedeni brojevi neurona u skrivenim slojevima dobiveni su testiranjem. Ako se koristi pre-mali broj neurona u skrivenim slojevima mreža neće moći naučiti kako evaluirati ulazne podatke, dok preveliki broj neurona može prouzročiti *overfitting*, tj. „učenje na pamet“ bez generalizacije, dok se ujedno bespotrebno povećava model. Za veći model, s većim brojem težina, potrebno je više vremena tijekom izračuna evaluacije, što je nepogodna pojava kada se što brže želi dobiti evaluacija pozicije unutar stabla pozicija prilikom traženja najboljeg poteza.

Model je postigao zadovoljavajuću točnost nakon 60 epoha učenja, što znači da se kroz 5 milijuna ulaznih podataka za učenje prošlo 60 puta. Na računalu na kojem je treniran model neuronske mreže, za jednu epohu je bilo potrebno oko 14 minuta što je poprilično mala brojka za toliku količinu podataka. Na kraju su se izlazi neuronske mreže malo razlikovali od onih zadanih u bazi za učenje, no to nije problem jer se od neuronske mreže traži subjektivna evaluacija za koju se ne zna koja bi točna vrijednost zapravo morala biti, jer se u šahu zapravo nikada ne zna za koliko je točno bodova određena pozicija bolja za nekog od igrača.

10.5. Primjena gotovog modela

Kada je model provjeren i daje logičan izlaz u vidu danih bodova na poziciju, može se primijeniti unutar algoritma za traženje najboljeg poteza. Treba imati u vidu da izlaz neuronske mreže sada direktno ovisi o tome koji će potez biti odabran kao najbolji. Ako model ne daje dobar izlaz, loši potezi će biti odabrani, a time će snaga šahovskog algoritma biti nedovoljna da pobijedi čovjeka koji odabire razumne poteze protiv računala. Kod za pokretanje glasi:

```
evaluacija_a = load_model('./model/evaluacija.h5')
evaluacija = evaluacija_a(vektor)
```

U prvoj liniji koda vidljiv je način na koji se na početku koda glavnog programa inicijalizira model neuronske mreže. Druga linija koda prikazuje kako se na jednostavan način može iz ulaznog binarnog vektora duljine 768 koji predstavlja trenutnu poziciju ploče dobiti evaluacija te pozicije pomoću Tensorflow-a.

Nakon što je model uspješno učitan u kodu i evaluacija se uspješno izvršava unutar algoritma za traženje poteza, vrijedi još ispitati logičnost dobivenih poteza. Postavljanjem igre između računalnog protivnika težine 5 od 8 (rejting oko 2000) na internetskoj stranici Lichess (bijeli igrač) i napravljenog šahovskog algoritma (crni igrač), dobiva se sljedeća šahovska partija:

```
1. Nf3 d5 2. Nc3 Nf6 3. d4 g6 4. e3 Bh6 5. Ne5 O-O 6. Bd3 c5 7. h3 Be6 8. O-O
Nbd7 9. Bd2 Nxe5 10. dxe5 Nd7 11. f4 f6 12. f5 Bxf5 13. exf6 Be6 14. e4 Bxd2
15. Nxd5 Bxd5 16. Qxd2 Be6 17. Bb5 exf6 18. Qc3 Ne5 19. Rad1 Qb6 20. Rxf6 Rxf6
21. a4 c4+ 22. Kh2 Qc7 23. Rd5 Ng4+ 24. Kh1 Qh2# 0-1
```

Vidljivo je da je pobjednik crni igrač, tj. algoritam koji koristi neuronsku mrežu napravljen u sklopu ovog diplomskog rada. Time je dokazano da je istrenirani model dovoljno dobar da odigra ozbiljnu partiju šaha te je time prihvaćen i korišten za dobivanje evaluacije tijekom rada algoritma koji traži najbolji potez. Nakon testiranja, procijenjeni rejting algoritma na dubini 4 je oko 2100, dok bi se za veći rejting morala povećati dubina pretrage. Povećanjem dubine pretrage eksponencijalno se povećava i vrijeme pretrage. Prema samom načinu rada, dobivanje izlaza neuronske mreže je puno sporiji postupak od čiste procjene i bodovanja određenih značajki pozicije kao što to radi *Stockfish*. Da bi se postigle visoke performanse, kao što je to slučaj kod najboljih šahovskih programa, trebalo bi dodati još neke dodatne metode redukcije broja poteza u algoritmu za traženje poteza, koristiti brži programski jezik od Pythona i napraviti da se program pokreće na više jezgri odjednom.

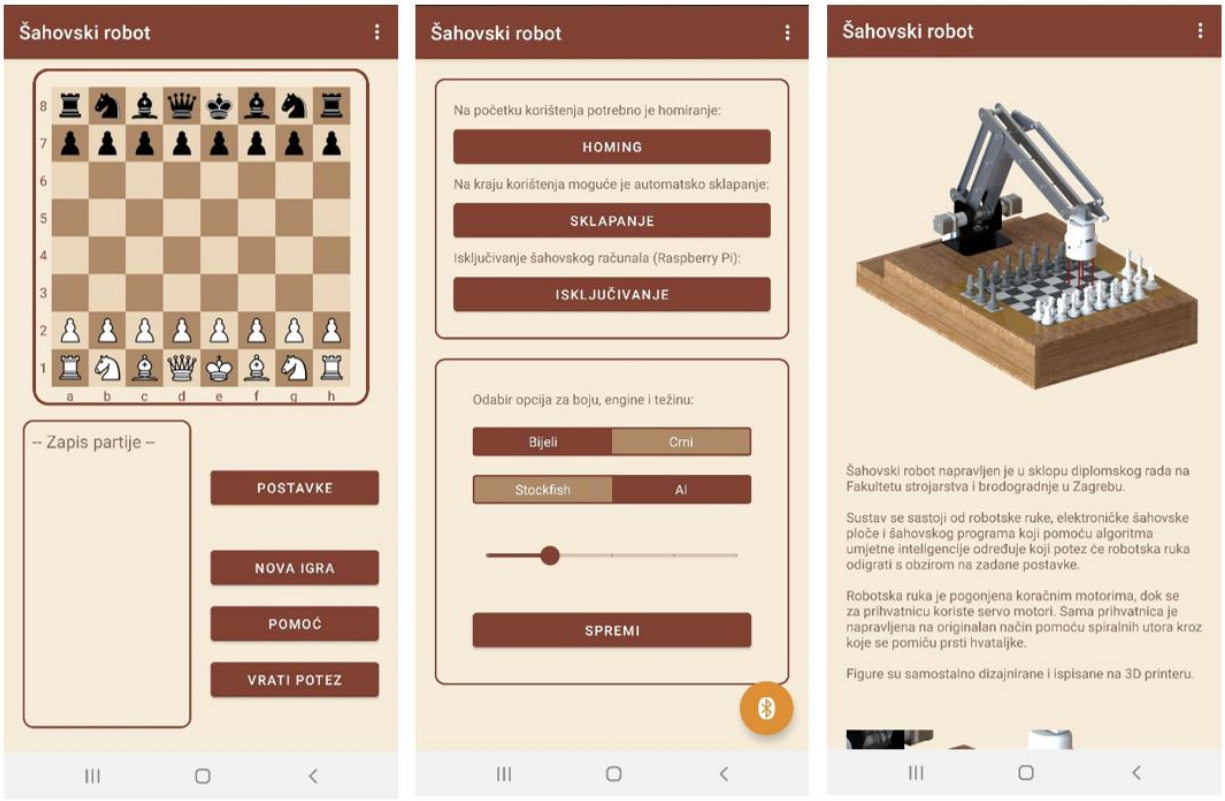
11. MOBILNA APLIKACIJA ZA KONTROLU

Do sada se na glavno računalo unutar robotskog sustava trebalo svaki puta povezati preko mreže, kako bi se pokrenula nova igra ili promijenile postavke. Da bi se olakšalo korištenje i povećala mobilnost cijelog robotskog sustava za šah, napravljena je mobilna Android aplikacija koja se može preuzeti pomoću sljedećeg QR koda:

Android aplikacija je rađena u programskom okruženju Android Studio, te je nakon konačne verzije učitana u najpoznatiju trgovinu Google Play.

Kada se robot pokrene, nakon nekog vremena se čuje ton koji označava da je pokretanje izvršeno do kraja. Nakon tog tona potrebno je pokrenuti mobilnu aplikaciju i pritisnuti na tipku *Postavke*. S donje lijeve strane pojavljuje se tipka za povezivanje putem bluetootha. Ako je robot uparen, pritiskom na tipku se mobitel spaja na računalo šahovskog robota, a ako robot nije uparen, potrebno ga je prvo upariti pa onda spojiti pomoću tipke. Kada tipka dobije zelenu boju, to znači da je bluetooth veza uspostavljena. Prvo se pokreće *homing* pritiskom na tipku, te se čeka da robotska ruka izvrši poletnu inicijalizaciju. Nakon toga odabiru se na dnu ekrana željene postavke težine, boje i algoritma te se pritisne tipka spremi. Povratkom na prethodnu stranicu i pritiskom na tipku *Nova igra*, pokreće se nova igra te ako je robot bijele boje, on igra prvi potez. Daljnja igra se odvija preko elektroničke šahovske ploče u sklopu robota. Ako je igraču tijekom igre potrebna pomoć, može ju dobiti pritiskom na tipku *Pomoć,* gdje se nakon toga potez odigra na virtualnoj ploči u aplikaciji, a igrač taj potez ako želi ponovi na ploči.

Ukoliko igrač napravi neku grešku, kada robot miruje može vratiti svoj potez i potez robota te nakon toga pritisnuti tipku *Vrati potez*. Potez će biti vraćen te će stara pozicija biti prikazana na virtualnoj ploči unutar aplikacije. Nakon igre, moguće je pritiskom na tipku *sklapanje*, sklopiti robotsku ruku na stalak predviđen za nju. Kada je robotska ruka na stalku, potrebno je pritisnuti tipku *isključivanje*, kako bi se glavno računalo robota ugasilo. Taj korak je potreban jer se u suprotnom naglim prekidom napajanja putem sklopke, naglo prekida rad računala koje možda koristi memorijsku karticu u tom trenutku. To može dovesti do greške na memorijskoj kartici i gubitka podataka, što je nepoželjna pojava.



Slika 20. Izgled mobilne aplikacije

Na slici 20. prikazani su glavni dijelovi aplikacije. Lijevi dio slike predstavlja glavni prikaz gdje se tijekom igre prikazuje trenutna pozicija na ploči unutar memorije računala robota, vidi se zapis partija i moguće je kontrolirati aspekte igre. Na se srednjem dijelu slike vidi se prikaz postavki gdje se odabire način i težina igre te se mogu pokretati funkcije robotske ruke. Desni dio slike prikazuje ekran s nekoliko kratkih informacija o projektu i slikama robotskog sustava.

12. TESTIRANJE I REZULTATI

Jednom kada su svi dijelovi robotskog sustava napravljeni i sastavljeni, programi napisani i neuronska mreža istrenirana, na redu je testiranje cijelog sustava i pronalaženje mogućih poboljšanja koja se mogu primijeniti odmah ili prilikom izrade druge verzije robotskog sustava.

Robotska ruka se nakon usklađivanja brzina giba jednolično, te svaki puta uspijeva pravilno prenijeti figuru s jednog polja na drugo. Zbog toga što su krakovi robotske ruke rađeni ručno, postoji mala zračnost koja za posljedicu ima to da hvataljka preko sustava poluga nije 100% poravnata s podlogom. Kupljeni motori s reduktorom imaju veliku zračnost između zuba zupčanika u reduktoru, te time katastrofalno utječu na preciznost kretanja. Kako bi se taj problem i problem zračnosti barem djelomično otklonio postavljene su opruge koje vuku krakove i prihvatnicu u krajnji položaj te je time preciznost poboljšana. Međutim, kada se robotska ruka raširi u krajnje ispruženi položaj kako bi pomakla najudaljenije figure, onda se zbog velikog kraka sile opruga rastegne, te robot propadne u drugi krajnji položaj u koji mu dopušta zračnost zupčanika. Ta zračnost nije velika, ali je dovoljna da nastane taj negativan efekt propadanja. Taj problem nije moguće riješiti bez ponovne drugačije konstrukcije robotske ruke. Potrebno je ili koristiti jači motor ili napraviti redukciju pomoću remenskog prijenosa koji ima minimalnu zračnost. Prihvatnica obavlja svoju funkciju bez problema, a jedino unaprjeđenje bi moglo biti korištenje profesionalnijih servo motora. Hobistički servo motori, kao što su ovi korišteni u prihvatnici robota, nemaju 100% pouzdan rad, te zbog toga često imaju razne vibracije i sitne greške u položaju. Isto tako, korištenjem profesionalnijih servo motora, mogla bi se postići puno veća preciznost pozicioniranja jer bi se tada sva gibanja vršila s povratnom vezom, što nije slučaj kod upravljanja koračnim motorima. Cijela robotska ruka se također mogla napraviti od tvrđeg polimernog materijala što bi dodatno smanjilo masu.

Elektronička šahovska ploča prepoznaje poteze bez problema, a očitanja su precizna i vrlo brza. Jedino poboljšanje bi bilo da se cijela ploča napravi iz jedne elektroničke pločice gdje bi se na jednu stranu postavile SMD elektroničke komponente a na drugu stranu bi se otisnula šahovska ploča. Time bi se anulirala velika količina žica ispod ploče koje je nemoguće izbjeći kada je ploča rađena ručno lijepljenjem senzora ispod polja i lemljenjem drugih komponenata na njih.

Ožičenje unutar drvenog kućišta robotske ruke rađeno je s izrazitim oprezom, jer se na glavni sklop za napajanje dovodi napon kućne električne mreže 230 V. Svi metalni dijelovi robotske ruke su uzemljeni. Kada bi se robotski sustav izrađivao za serijsku proizvodnju, svi priključci bi se u sklopu poboljšanja mogli oklopiti u konektore koji bi tada bili direktno i uredno priključeni na pripadne dijelove robotskog sustava koje povezuju.

Cijeli robotski sustav na kraju vrši svoju funkciju i bez problema igra šah protiv ljudskog protivnika. Nakon testiranja i s bijelim i s crnim figurama, sve partije su na kraju završile, a robotski sustav nigdje nije upao u neizlaznu grešku. Ponekad su vidljive greške u serijskoj komunikaciji gdje robotska ruka stane jer ne zna što treba napraviti. Te greške su neizbježne, a kada dođe do njih program to prepoznaje te se izvršava određena funkcija koja ponovo šalje istu informaciju, te robot nastavlja svoje gibanje.

Algoritam za igranje šaha obavlja odlično svoju zadaću, te su dobiveni potezi kreativni i dovoljno točni da može pobijediti poprilično jakog protivnika. Već na razini 1, algoritam predstavlja ozbiljan izazov za šahovskog početnika. Najveća razina 5 daje najbolje rezultate, no za to je potrebno dosta vremena, pa se ta razina ne može primijeniti za brze igre. Kao što je već i navedeno prilikom opisa algoritma, da bi se postigla veća brzina, potrebno je algoritam napisati u bržem programskom jeziku (na primjer C ili C++), a traženje poteza pokretati na više jezgri procesora. Time bi se performanse uvelike približile vodećim šahovskim programima. Isto tako Raspberry Pi na kojem se pokreće algoritam, nema posebno moćan procesor, pa je i zbog toga vrijeme traženja najboljeg poteza malo produženo.

Kako bi se smanjila cijena i kompleksnost cijelog robotskog sustava, mogle bi se sve elektroničke i elektromehaničke komponente povezati na jedan jaki mikrokontroler s puno ulaza (na primjer Arduino Due). Taj mikrokontroler bi tada bio povezan s korisnikovim računalom pomoću USB kabela, a korisnik bi glavni program za šah pokretao na svojem računalu. Time bi se smanjila i kompleksnost i cijena, jer više nije potreban Raspberry Pi i poseban mikrokontroler za elektroničku šahovsku ploču. Na korisnikovom osobnom računalu bi se program pokretao puno brže, a tijekom vremena s poboljšanjem performansi osobnih računala bi se poboljšavalo i vrijeme koje je potrebno za dobivanje najboljeg poteza.

13. PROCJENA VRIJEDNOSTI

Budući da je šahovski robot fizički izveden, popis troškova je prikazan u tablici 2. ispod. Navedene cijene su cijene na dan kupnje dijelova te mogu varirati s vremenom.

Tablica 2. Okvirne cijene komponenata robota

Komponenta	Cijena (kn)
Aluminijski L profili	105
Spojni elementi (vijci, matice, ležajevi, …)	81
Nema17 koračni motor s reduktorom – 2 kom	537
Nema23 koračni motor	156
Servo motor MG995R – 2 kom	194
Driver TB6600 – 3 kom	195
Mikrokontroler ESP32	63
Mikrokontroler Arduino Nano	20
Raspberry Pi 3B+	398
Modul za napajanje 230 VAC na 24 VDC	117
Stabilizator napon 24V na 5V – 3 kom	139
Ostale elektroničke i elektromehaničke komponente	124
Sigurnosna gljiva	21
Mini USB kabel – 2 kom	31
Žice	35
Tisak šahovske ploče na polimernu ploču	250
Neodimijski magneti	106
Podloške za noge namještaja	40
Kutije za figure – 2 kom	30
Filament za 3D printer bijeli+crni	348
Ukupno	**2990**

U ovu cijenu nije uključena strojna obrada, izrada drvene kutije, ručna obrada pojedinih dijelova, montaža i cijena materijala koji nije naveden u tablici. Prilikom planiranja serijske proizvodnje potrebno je uključiti i navedene troškove u konačnu cijenu.

14. ZAKLJUČAK

Ovim diplomskim radom prikazan je cijeli postupak pristupa problemu izrade robotskog sustava koji igra šah. Kroz poglavlja su detaljno objašnjeni mehanički, elektronički i programski dijelovi šahovskog robota kao i ideje na temelju kojih ti dijelovi rade.

Testiranjem je pokazano da robotski sustav uspješno obavlja svoju zadaću te može odigrati šahovsku partiju od početka do kraja, a uz to pruža i puno dodanih mogućnosti kao što su odabir težine, pomoć, vraćanje poteza i slično. Uočeno je da također ima i puno mjesta za napredak. Prvenstveno bi prilikom konstruiranja bilo bolje koristiti jače motore bez reduktora ili ako se redukcija koristi, izvesti ju preko zupčastog remena gdje je zračnost zanemarivo mala. Također, cijena bi se mogla smanjiti za oko 20% ako se umjesto mini računala unutar robotske ruke, putem USB kabela spoji mikrokontroler robota na korisnikovo računalo gdje se pokreće šahovski program. Tako bi se dobila manja cijena i bolje performanse, ali bi bilo potrebno dodatno računalo što smanjuje mobilnost. Algoritam za igranje šaha napravljen u sklopu ovog rada daje odlične rezultate, jer uspijeva pobijediti profesionalne šahovske programe na visokim razinama težine, dok bi za najveće razine težine bilo potrebno dodatno razraditi program.

Jedan ovakav rad sadrži dijelove iz gotovo svih grana mehatronike i robotike. Tijekom izrade ovog diplomskog rada moglo se mnogo naučiti iz svih tih područja. Ovim radom pokriveno je konstruiranje, izrada kinematičkog modela, izrada elektroničkog sustava, programiranje mikrokontrolera, stvaranje računalnih aplikacija, učenje neuronske mreže i programiranje mobilnih aplikacija. Sva ova znanja su korisna u budućim poslovima inženjera mehatronike i robotike te mogu uvelike ubrzati razvoj nekih drugih sličnih sustava.

LITERATURA

[1] https://newatlas.com/chess-terminator-robot-takes-on-kramnik-in-match/16996/, 16.6.2021.

[2] https://python-chess.readthedocs.io/en/latest/index.html, 23.6.2021.

[3] https://github.com/ornicar/lila/tree/master/public/piece/cburnett, 24.6.2021.

[4] https://medium.com/the-innovation/the-anatomy-of-a-chess-ai-2087d0d565, 25.6.2021.

[5] https://www.chessprogramming.org/Alpha-Beta, 25.6.2021.

[6] https://chessify.me/blog/nps-what-are-the-nodes-per-second-in-chess-engine-analysis, 25.6.2021.

[7] https://www.chessgames.com/index.html, 25.6.2021.

[8] https://stockfishchess.org/, 25.6.2021.

[9] https://www.tensorflow.org/, 25.6.2021.

[10] D. P. Kingma, J. L. Ba, „Adam: A Method for Stohastic Optimization“, CoRR, abs/1412.6980 (2015)

[11] B. Novaković, D. Majetić, M. Široki, *Umjetne neuronske mreže,* Fakultet strojarstva i brodogradnje, Zagreb, 1998.

[12] https://smartboost.com/blog/deep-learning-vs-neural-network/, 25.6.2021.

PRILOZI

I. CD-R disk
II. Tehničke karakteristike koračnog motora Nema17
III. Tehničke karakteristike koračnog motora Nema23
IV. Tehničke karakteristike servo motora MG995R
V. Tehničke karakteristike drivera za koračni motor TB6600
VI. Tehničke karakteristike mikrokontrolera ESP32
VII. Tehničke karakteristike mikrokontrolera Arduino Nano
VIII. Tehničke karakteristike računala Raspberry Pi 3B+

Tehničke karakteristike koračnog motora Nema17:

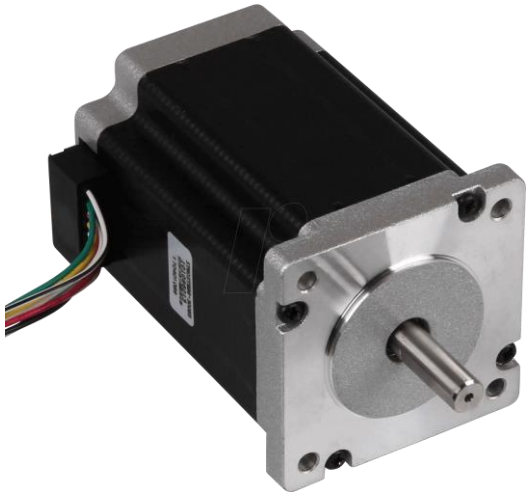
Tehničke karakteristike	
Moment držanja (Ncm)	42
Moment otpora – isključen motor (Ncm)	1,5
Moment inercije rotora (kgcm^2)	0,057
Masa (g)	230
Struja kroz zavojnicu (A)	1,5
Otpor zavojnice (ohm)	2,1
Induktivitet zavojnice (mH)	5,0

Signali i boje žica	
Zavojnica A - početak	Crvena
Zavojnica A - završetak	Plava
Zavojnica B - početak	Zelena
Zavojnica B - završetak	Crna

Više informacija na poveznici:

http://www.autoflexible.com/file_upload/product/attach/NEMA%2017.pdf

Tehničke karakteristike koračnog motora Nema23:



Tehničke karakteristike	
Moment držanja (Ncm)	170
Moment otpora – isključen motor (Ncm)	6,9
Moment inercije rotora ($kgcm^2$)	0,468
Masa (g)	1000
Struja kroz zavojnicu (A)	2,4
Otpor zavojnice (ohm)	1,5
Induktivitet zavojnice (mH)	5,4

Signali i boje žica	
Zavojnica A - početak	Crvena
Zavojnica A - završetak	Plava
Zavojnica B - početak	Zelena
Zavojnica B - završetak	Crna

Više informacija na poveznici:

https://datasheetspdf.com/pdf/1276820/Schneider/NEMA23/1

Tehničke karakteristike servo motora MG995R:

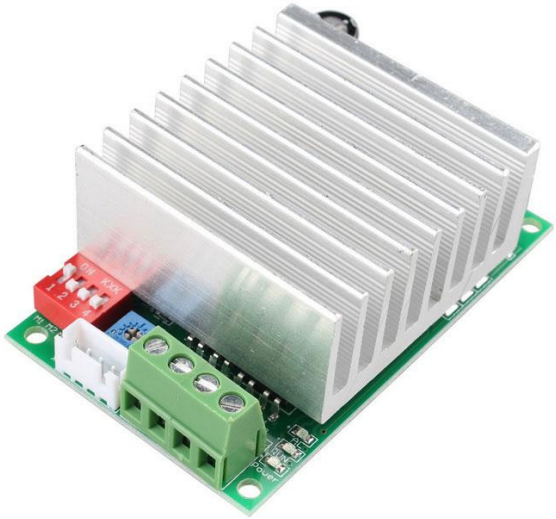
Tehničke karakteristike	
Moment (kgcm)	11,00
Brzina (sec/60°)	0,14
Masa (g)	55
Dimenzije (mm x mm x mm)	40,7 x 19,7 x 42,9
Materijal zupčanika	metal
Ciklus pulsa (ms)	1,0

Signali i boje žica	
Narančasta	PWM
Crvena	+V
Smeđa	GND

Više informacija na poveznici:

https://components101.com/motors/mg996r-servo-motor-datasheet

Tehničke karakteristike drivera za koračni motor TB6600:



Tehničke karakteristike	
Maksimalni napon na ulazu (V)	45
Maksimalna struja na izlazu (A)	4,5
Dimenzije (mm)	50 x 50 x 23
Mogućnost mikro koraka	1/1, 1/2, 1/4, 1/8, 1/16

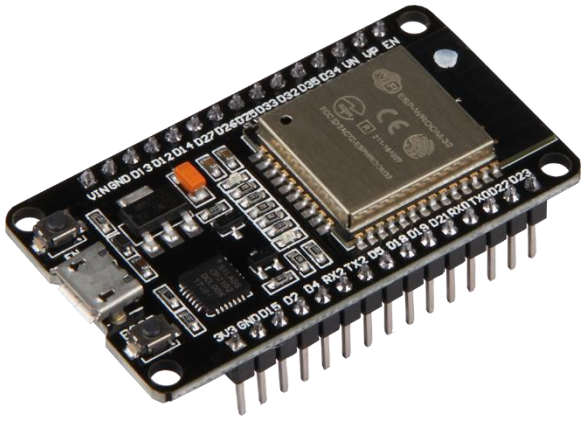
Dodatne karakteristike
Automatska regulacija struje
Automatsko isključivanje sklopa na previsokoj temperaturi
Isključivanje sklopa pri nedostatnom naponu
Zaštita od prevelikih struja
Optoizolacija ulaznih priključaka

Više informacija na poveznici:

https://aws.robu.in/wp-content/uploads/2017/11/TB6600HG_datasheet_en_20160610-1.pdf

https://aws.robu.in/wp-content/uploads/2017/11/TB6600-stepper-motor-Driver-Controller-ROBU.IN_.jpg

Tehničke karakteristike mikrokontrolera ESP32:



Tehničke karakteristike	
Mikroprocesor	Tensilica Xtensa LX6
Maksimalna frekvencija	240 MHz
Radni napon	3,3 V
Broj jezgri	2
Analogni ulazi	16 (12-bitni)
Digitalno-analogni pretvarači	2 (8-bitni)
Digitalni ulazno-izlazni priključci	39
Dopuštena struja na ulazima/izlazima	40 mA
Dopuštena struja na 3,3V priključku	50 mA
SRAM	520 KB
Komunikacijski priključci	SPI (4), I2C (2), I2S (2), CAN, UART (3)
Wi-Fi	802.11 b/g/n
Bluetooth	V4.2 - BLE + Classic

Više informacija na poveznici:

https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

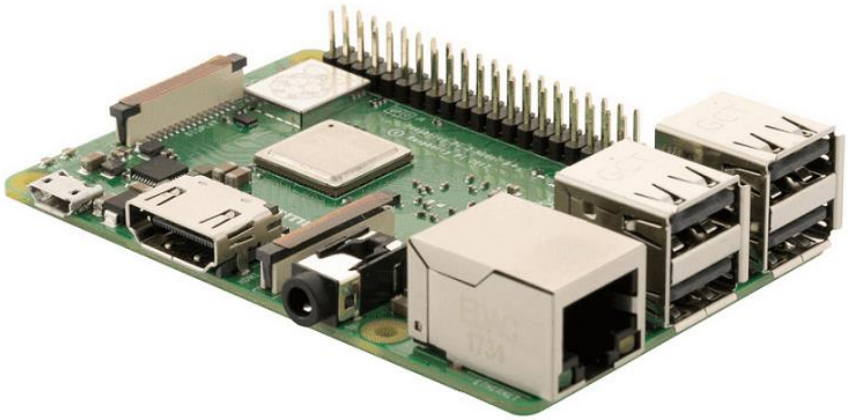
Tehničke karakteristike mikrokontrolera Arduino Nano:

Tehničke karakteristike	
Mikroprocesor	ATmega328P
Maksimalna frekvencija	16 MHz
Radni napon	5 V
Broj jezgri	1
Analogni ulazi	8
Digitalno-analogni pretvarači	0
Digitalni ulazno-izlazni priključci	14
Dopuštena struja na ulazima/izlazima	40 mA
Dopuštena struja na 3,3V priključku	50 mA
SRAM	2 KB
Komunikacijski priključci	SPI, I2C
Wi-Fi	-
Bluetooth	-

Više informacija na poveznici:

https://www.arduino.cc/en/uploads/Main/ArduinoNanoManual23.pdf

Tehničke karakteristike računala Raspberry Pi 3B+:



Tehničke karakteristike	
Mikroprocesor	Broadcom BCM2837B0, Cortex-A53 64-bit SoC
Maksimalna frekvencija	1.4 GHz
Radni napon	3,3 V
Broj jezgri	4
Ulazno-izlazni priključci	40
Dopuštena struja na ulazima/izlazima	16 mA
SRAM	1 GB
Komunikacijski priključci	SPI, I2, UART, Ethernet
Wi-Fi	802.11 b/g/n
Bluetooth	V4.2 - BLE

Više informacija na poveznici:

https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf