Robotska manipulacija s navođenjem pomoću ultrazvučnog senzora

Ćebo, Aldin

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

ROBOTSKA MANIPULACIJA S NAVOĐENJEM POMOĆU ULTRAZVUČNOG SENZORA

Diplomski rad

Aldin Ćebo

Osijek, 2017.

SADRŽAJ

1. UVOD

Robotika je tehnička znanost koja predstavlja spoj stroja i računala. U robotiku su uključena različita znanja kao što su projektiranje strojeva, teorija regulacije i upravljanja, mikroelektronika, programiranje, umjetna inteligencija i mnoga druga znanja. Za robotiku možemo reći da je interdisciplinarna znanost za koju su potrebne četiri krucijalne stvari, a to su: mehanika, elektronika, informatika i automatika. Robotika se prvenstveno bavi proučavanjem strojeva koji mogu zamijeniti čovjeka u izvršavanju zadataka. Razvoj robotike je potaknut pronalaskom zamjene za čovjeka koja bi imala mogućnost oponašanja njegovih svojstava u različitim primjenama, uzimajući u obzir i međudjelovanje s okolinom koja ga okružuje. Početkom 20. stoljeća prvi put se upotrebljava naziv *robot*, uveo ga je češki književnik K. Čapek 1920. godine u svojoj drami "RUR" (*Rossumovi univerzalni roboti*), suvremeni roboti dolaze već 1950-ih u SAD. Robot Institute of America, 1980 godine, definirao je robota kao višefunkcionalnog manipulatora s mogućnošću reprogramiranja, projektiranog za prenošenje materijala, dijelova, alata i posebnih naprava kroz različite programirane pokrete u svrhu obavljanje različitih zadataka. Ova definicija je prilično ograničavajuća, budući da isključuje mobilne robote, koji u današnje vrijeme doživljavaju ekspanziju. Šira i preciznija definicija bi bila da je robot stroj koji posjeduje inteligentnu vezu između percepcije i akcije. Tu se može definirati pojam inteligentnog robota kao stroja sposobnog da prikuplja informacije iz okolnog svijeta i koristeći znanje o okolini uspijeva se uspješno kretati u njoj. Računalno upravljanje robotskim sustavima teži primjeni ekspertnih sustav i umjetne inteligencije u području automatskog upravljanja. Robotski sustav dobiva signale iz okoline preko senzora i djeluju na istu pomoću aktuatora. Veza između opažanja (eng. sensing) i djelovanja (eng. actuation) može biti ostvarena jednostavnom obradom signala ili pak može uključivati složene postupke odlučivanja, interpretaciju cilja i druge aspekte rasuđivanja. [1]

U ovom diplomskom radu opisana je problematika i dano je rješenje za robotski manipulator upravljan ultrazvučnim senzorom. Cijeli programski sustav ovog rada izveden je u programskom paketu Matlab koji je povezan s mikrokontrolerom. Razvijen je kinematički model robota, određeni su DH parametri robotskog sustava te na temelju njih je izvedena direktna i inverzna kinematika robota. U programskom paketu Matlab razvijeno je korisničko sučelje kako bi se moglo vršiti izdavanje naredbi gibanja robota u kartezijskom koordinatnom sustavu te upravljanje po pojedinim zglobovima. Određivanje pozicija objekta koji se nalazi u radnom prostoru robota izvedeno je na temelju mjerenih podataka dobivenih ultrazvučnim senzorom.

2. OPIS SUSTAVA

U ovom poglavlju dan je opis robotskog manipulatora, ultrazvučnog senzora, servo motora, mikrokontrolerske platforme Arduino Mega2560 te programskog paketa Matlab koji se koriste u radu.

2.1. Robotski manipulator

Robotski manipulator korišten u ovom diplomskom radu je namijenjen za edukacijske svrhe, te samim time nema veliku stabilnost i preciznost, slika 2.1.1. Dizajniran je od aluminijskih dijelova koji su povezani sa šest zglobova. Svaki zglob predstavlja jedan servo motor koji je povezan sa sljedećim člankom manipulatora. Direktno preko osovine servo motora vrši se rotacija zgloba. Postolje robotskog manipulatora odnosno baza je dizajnirana od akrilne ploče na kojoj se nalazi aluminijski ležaj.

Sl. 2.1.1 Robotski manipulator s bazom.[13]

Pet servo motora služi za rotaciju robotskog manipulatora dok zadnji šesti motor služi za otvaranje i zatvaranje alata, odnosno hvataljke robotskog manipulatora.

2.2. HC-SR04 – Ultrazvučni senzor

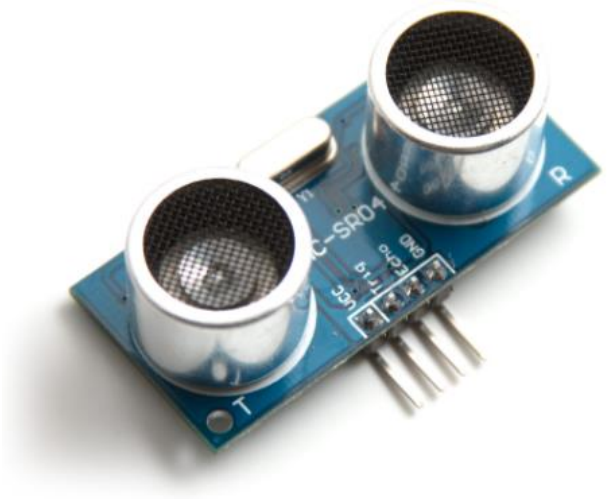
Jedna od osnovnih karakteristika HC-SR04 ultrazvučnog modula je da koristi ultrazvučne valove kako bi odredio udaljenost do predmeta, poput šišmiša ili delfina. Ne utječe na njega sunčeva svijetlost ili crni materijali kao kod svjetlosnog senzora, kao primjerice najpoznatijeg konkurenta Sharp IR-a što je velika prednost ovog tipa ultrazvučnih senzora. Dakle, ovi senzori se koriste za mjerenje udaljenosti između senzora i objekta koji se nalazi ispred njega. Objekt do kojeg se mjeri udaljenost može mirovati ili se kretati. Također, jako je bitno da je senzor kompatibilan s Arduino platformom te ga je vrlo jednostavno povezati. Modul se sastoji od ultrazvučnog predajnika, ultrazvučnog prijemnika i kontrolne elektronike. Karakteristike ovog senzora su:

Tab. 2.2.1 Karakteristike HC-SR04 ultrazvučnog senzora.

Domet	**2-200cm**
Preciznost	3mm
Efektivni kut mjerenja	15°
Napon	5V
Maksimalna mirna struja	2mA
Radna struja	15mA
Ultrazvučna frekvencija	40kHz
Dimenzije	45 x 20 x 15 mm

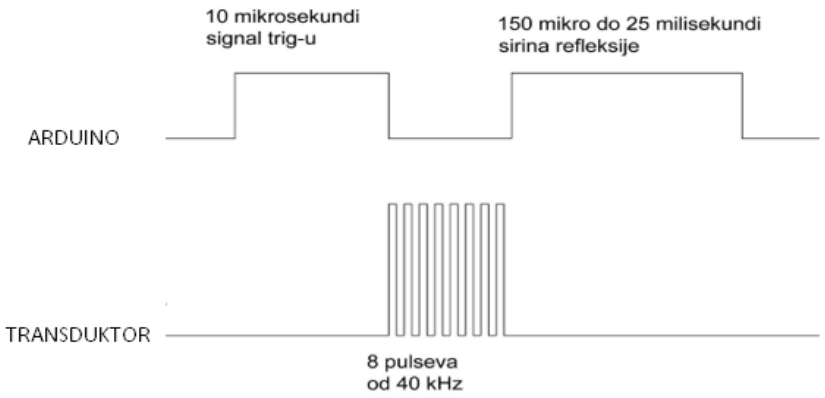
Modul ime sljedeće pinove:[4]

- Pin 1: VCC – napajanje modula, 5V
- Pin 2: Trig – okidanje/aktiviranje mjerenja
- Pin 3: Echo – povratni signal, dužina impulsa proporcionalna razdaljini
- Pin 4: GND



Sl. 2.2.2 Ultrazvučni senzor HC-SR04. [2]

Dva osnovna dijela modula na kojima se temelji princip rada su *trig* (prekidač) i *echo* (refleksija). Mikrokontrolerom (u našem slučaju Arduinom) se šalje 5V na trig pin u trajanju 10 mikro sekundi (minimalno), te se na taj način aktivira ultrazvučni transduktor. Zatim modul automatski šalje 8 impulsa od 40kHz i čeka njihovu refleksiju. Kada detektira povratne ultrazvučne impulse tj. reflektirani impuls, šalje izlazni signal (podatke) mikrokontroleru preko echo pina. Navedeni podaci su zapravo vrijeme trajanja reflektiranog impulsa, od 150 mikro do 25 milisekundi. Ako vrijeme trajanja reflektiranog impulsa traje duže od 35 milisekundi, senzor registrira da je predmet izvan dosega.



Sl. 2.2.3 Princip rada ultrazvučnog senzora HC-SR04. [3]

Dakle, da pojednostavimo, senzor emitira zvučne impulse visoke frekvencije, u slučaju da se ispred senzora nalazi prepreka, impulsi se odbijaju od nje k senzoru. Ukoliko su impulsi detektirani nakon emitiranja, možemo zaključiti da se ispred senzora nalazi prepreka. Val gubi zanemarivo malo energije na svom putu, pa možemo reći da će otići i vratiti se otprilike istom

brzinom. Zato možemo koristiti sljedeću relaciju. Za računanje udaljenosti potrebno je prvo izmjeriti dužinu trajanja povratnog impulsa. A kada znamo dužinu puta koju je impuls prešao, samim tim ćemo znati i rastojanje prepreke od senzora. Udaljenost se može izračunati po formuli:

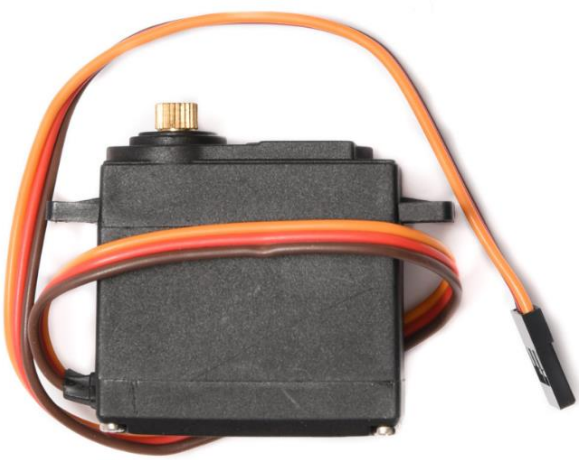
$$Udaljenost = \frac{trajanje\ povratnog\ impulsa\ [sekunde] * brzina\ zvuka\ [340\frac{m}{s}]}{2} \qquad (2.2.\text{-}1)$$

Kao što vidimo, kada dobijemo povratni signal na mikrokontroleru pomoću dužine trajanja tog povratnog impulsa, te već unaprijed poznate brzine zvuka (koja iznosi $340\frac{m}{s}$) vrlo je jednostavno izračunati udaljenost od senzora do željenog predmeta.[3]

2.3. Servo motor

Servo motor si možemo predočiti kao servo volan kojim upravlja robot. Servo motor omogućuje preciznu kontrolu zadanog kuta, brzine okretanja i akceleracije. Servo motori nisu specifična klasa motora, iako se pojam servo motor često koristi za upućivanje na motor prikladan za uporabu u kontrolnom sustavu zatvorene petlje. Servo motori se koriste u aplikacijama kao što su robotika, CNC strojevi ili automatizirana proizvodnja. Servo motori sastoje se od nekoliko glavnih elemenata, a to su DC motor, potenciometar i kontrolna pločica. Jednostavni servo motori koriste otporne potenciometre kao svoje kodere položaja tako što se otpor na potenciometru mijenja kada se motor okreće. Na taj način kontrolna ploča može precizno regulirati okret, brzinu, akceleraciju i smjer okreta. Servo upravljanje postiže se slanjem PWM signala, niza ponavljajućih impulsa promjenjive širine. PMW signal se šalje preko kontrolne žice, pri čemu je potrebno koristiti neke od digitalnih pinova na Arduinu. Budući da je korišten moderni servo motor, ovisno o duljini pulsa, rotor će se okrenuti na željenu poziciju. TowerPro kao i većina servo motora ima mogućnost okreta 90 stupnjeva u oba smjera ukupno 180° stupnjeva. [5]

Servo motor koji je korišten u ovom radu je Towerpro MG996. MG996 je snažniji servo motor koji je sposoban okretati i kontrolirati stvari. Dolazi s više dodataka kako bi ga lakše povezali s vanjskim svijetom, a upravlja se kao svaki standardni servo motor putem PWM signala preko kontrolne žice kako je već ranije objašnjeno. [5]



Sl. 2.3.1 Towerpro MG996 servo motora [5]

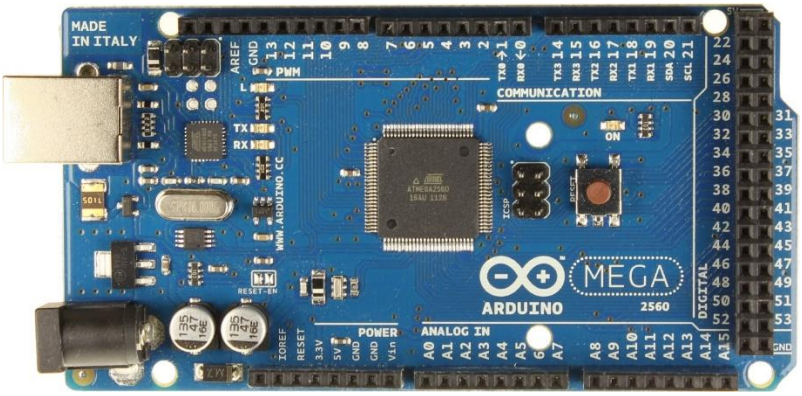
Tab. 2.3.2 Karakteristike Towerpro MG996 servo motora. [5]

Brzina okreta	**0.20s/60 stupnjeva(4.8V),** **0.16s/60 stupnjeva(6V)**
Napon	4.8V - 6.6V
Moment	9.4kg/cm(4.8V) - 11kg/cm(6V)
Kut	60 stupnjeva
Dimenzije	40mm x 20mm x 43mm
Težina	55g

Kako bi se bolje objasnilo što je zapravo moment kod servo motora mora se spomenuti sljedeće: Kada servo motor dobije naredbu za pokret okretati će se do zadane pozicije i zadržati ju. Ako vanjska sila djeluje na motor dok drži poziciju, servo motor će se odupirati pokušavajući zadržati istu. Maksimalna sila koju servo može podnijeti je zapravo moment iz tablice 2.3.2. Dakle, što je moment veći to znači da je sila koju servo može podnijeti veća, te time su karakteristike motora bolje. [6]

2.4. Mikrokontroler Arduino Mega 2560

Arduino Mega2560 je mikrokontrolerska platforma bazirana na Atmel ATMega 2560 mikrokontroleru. Pored mikrokontrolera sadržava potrebno okruženje za njegov rad, komunikaciju i programiranje s PC-om putem USB sučelja preko IC ATmega16U2 . Kompatibilna je s velikim brojem dodatnih ploča Arduino Duemilanove ili Diecimila platformi.[8]



Sl. 2.4.1 Arduino Mega 2560.[7]

Tab. 2.4.2 Tehnički podatci.[9]

Mikrokontroler	**ATmega2560**
Radni napon	5V
Napajanje	7-12V
Ulazni napon (limiti)	6-20V
Digitalni I/O pinovi	54 (14 s PWM izlazom)
Analogni ulazi	16
Flash Memorija	256 KB / 8 KB za bootloader
SRAM	8KB
EEPROM	4KB
Takt	16MHz

- Napajanje

  Mega2560 ima dva načina napajanja, preko USB porta ili vanjskog izvora. Odabir je automatski. Vanjsko napajanje se spaja na 2,1 mm utičnicu ili na pinove. Preporučeno je napajanje od 7 do 12 V. Može izdržati od 5 do 20 V, ali ako je napajanje manje od 7 V Arduino može postati nestabilan. Za napajana koja su preko 12 V može doći do pregrijavanja i kvara stabilizatora. [10]

- Memorija

  Mikrokontroler ATmega2560 ima 256 KB flash memorije za spremanje programskog koda od čega je 8 KB za bootloader, 8 KB SRAM i 4 KB EEPROM. EEPROM-u se pristupa preko EEPROM.h biblioteka. [10]

- Ulazi i Izlazi

  Svaki od 54 digitalna pina može se koristiti kao ulaz ili izlaz. Oni rade na 5 V i mogu dati ili primiti 40 mA. Neki pinovi imaju specijalne funkcije kao što su serijska komunikacija:

  1. Serial 0: pin 0 (RX) i pin 1 (TX),
  2. Serial 1: pin 19 (RX) i pin 18 (TX),
  3. Serial 2: pin 17 (RX) i pin 16 (TX),
  4. Serial 3: pin 15 (RX) i pin 14 (TX).

  Koriste se za primanje (RX) i slanje (TX) serijske komunikacije TTL razina. Također Arduino ima šest vanjskih prekidnih rutina:

  1. pin 2 (interrupt0),
  2. pin 3 (interrupt 1),
  3. pin 18 (interrupt 5),
  4. pin 19 (interrupt 4),
  5. pin 20 (interrupt 3),
  6. pin 21 (interrupt2)).

  Ovi pinovi mogu biti podešeni na okidanje na niskoj razini, rastući ili padajući rub ili promjenu napona. Za rad s prekidima koristi se attachInterrupt() funkcija. PWM (eng. Pulse Width Modulation), pinovi su od 0 do 13. Arduino Mega2560 ima 16 analognih ulaza koji se očitavaju 10 bitnom rezolucijom (moguće 1024 različite razine). Zadano je mjerenje od 0 do 5 V. [10]

- Komunikacija

Arduino Mega2560 posjeduje brojne načine za komunikaciju s vanjskim svijetom. Na primjer PC, drugi Arduino ili neki drugi mikrokontroler. ATmega2560 posjeduje četiri sučelja za serijsku komunikaciju na TTL razinama. USB kontroler ATmega16U2 daje virtualni serijski port preko USB komunikacije.[10]

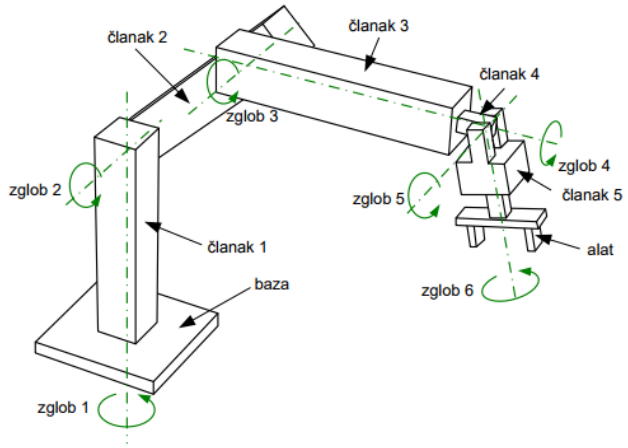
2.5. Programski paket MATLAB

MATLAB je programski jezik visoke razine. Pomoću MATLAB-a možemo raditi analizu podataka, izraditi algoritme i matematičke modele, te kreirati aplikacije. On nam omogućuju brži rad nego s tradicionalnim programskim jezicima, kao što su C/C++ ili Java.

U ovom radu MATLAB se koristio za računanje direktne kinematike, inverzne kinematike, te je bio u serijskoj komunikaciji s Arduino Mega 2560. U samom programskom sučelju MATLABA je kreirana aplikacija tj. Grafičko korisničko sučelje za upravljanje robotskog manipulatora i računanje kinematike.

3. GRAĐA ROBOTA I OSNOVNE STRUKTURE MANIPULATORA

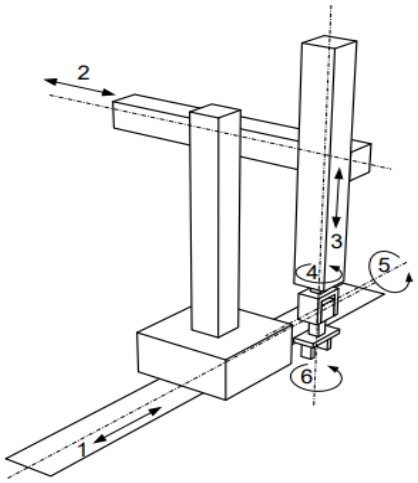
Robot je stroj programiran računalom koji je sposoban za automatsko provođenje složenih nizova radnji. Robot je programibilni, višenamjenski mehanički stroj dizajniran za pomicanje materijala, dijelova, alata ili specijaliziranih uređaja po različitim putanjama pomoću odgovarajućih senzora. Putanje po kojima robot pomiče predmete često su precizno definirane od strane korisnika ali također može ih upravljački sustav robota automatski generirati na temelju specifikacija različitih zadataka. Glavni dijelovi robota su članci (engl. links) i zglobovi (engl. joints). Članci su kruta tijela te se robot sastoji od više članaka koji su međusobno povezani pokretljivim zglobovima. Robot može biti napravljen u različite svrhe, te ovisno o njegovoj svrsi može biti opremljen još nekim dijelovima, kao npr. robot može biti opremljen hvataljkom ukoliko je njegova svrha manipulacija fizičkim objektima, tj. njihovo premještanje s jednog mjesta na drugo, sastavljanje odnosno rastavljanje. U tom slučaju potrebna mu je hvataljka da uhvati objekt od interesa i drži ga dok se giba u prostoru. Također, još jedna svrha robota može biti gibanje odgovarajućeg alata koji služi za obradu neke površine, bušenje, zavarivanje, bojenje, inspekciju ili sl. U oba slučaja, robot predstavlja uređaj koji svoju zadaću ostvaruje gibanjem nekog završnog mehanizma za koji ćemo koristiti općeniti termin alat, te mu je u oba slučaja potrebna hvataljka. Danas se u praktičnoj primjeni najčešće koriste robotski manipulatori koji po funkciji, a često i po izgledu, odgovaraju ljudskoj ruci. Jedan primjer robotskog manipulatora prikazan je na Sl. 3.1. [11]



Sl. 3.1 Robotski manipulator [11]

Kao što možemo vidjeti na slici, svaki članak ovog robotskog manipulatora je povezan s druga dva susjedna članka, osim prvog i zadnjeg članka koji su povezani samo s jednim susjednim člankom. Prvi članak je jednim krajem povezan s susjednim člankom, a drugim

krajem je pričvršćen na pomičnu ili nepomičnu bazu, dok zadnji članak predstavlja alat. Takva mehanička struktura naziva se kinematičkim lancem. Radi jednostavnosti pretpostavit ćemo da je baza nepokretna, iako ona može biti montirana na pokretnu platformu. Promjenu orijentacije odnosno pozicije svakog članka u odnosu na njemu susjedne članke omogućuje to što su članci međusobno povezani pokretljivim zglobovima kao što je već ranije spomenuto. Time se omogućuje i promjena položaja zadnjeg članka u kinematičkom lancu alata. Zglobovi koji su prikazani na Sl.3.1 su rotacijski, no postoje i translacijski zglobovi koji su prikazani na Sl. 3.2. Za zglobove robota se u literaturi još koristi i naziv osi, tako da se npr. robotski manipulator sa šest zglobova naziva i šesto osni manipulator. [11]



Sl. 3.2. Robotski manipulator s translacijskim i rotacijskim zglobovima. Zglobovi 1, 2 i 3 su translacijski, dok su zglobovi 4, 5 i 6 rotacijski. [11]

Kod ovog robotskog manipulatora je specifično to, da bi mogao ostvariti gibanje zadnjeg članka (alata), robot mora imati pogon koji djelovanjem sila pokreće članke. Da bi to ispravno funkcioniralo te da bi se ostvarilo precizno pozicioniranje alata, u svaki zglob robota ugrađuje se mjerni uređaj koji mjeri relativnu orijentaciju odnosno poziciju susjednih članka povezanih tim zglobom. Ukoliko robot mora ostvariti pozicioniranje alata u odnosu na neki okolni objekt čiji je položaj u odnosu na bazu robota promjenljiv, tada robot mora biti opremljen percepcijskim senzorima, koji omogućuju određivanje položaja objekta od interesa u odnosu na robota. [11]

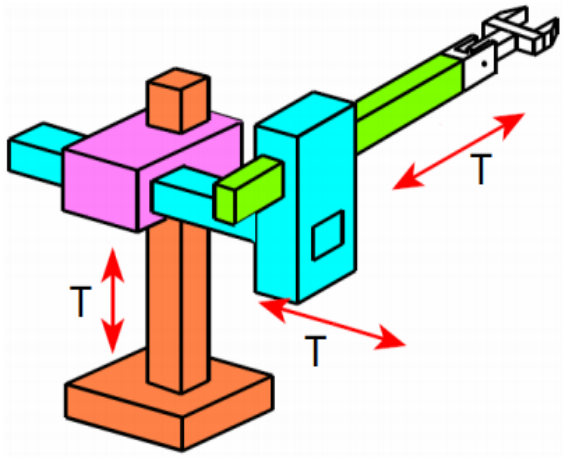
3.1. Osnovne konfiguracije robota

Kretanje robota odvija se u trodimenzionalnom prostoru, prve tri osi najčešće se koriste za određivanje pozicije ručnog zgloba, dok preostale osi određuju orijentaciju vrha manipulatora.

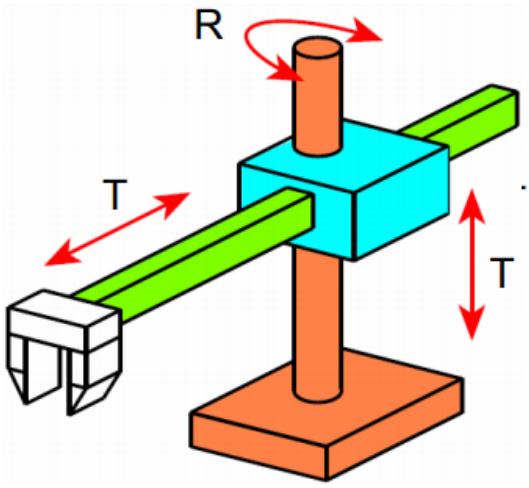
Općeniti manipulator ima šest osi te može dovesti vrh manipulatora u bilo koju poziciju i orijentaciju unutar radnog prostora. Radni prostor robota predstavlja skup točaka u trodimenzionalnom prostoru koje se mogu dohvatiti vrhom manipulatora. Oblik i zapremina radnog prostora ovise o strukturi manipulatora, kao i prisutnim ograničenjima mehaničkih zglobova. Danas se najviše susreću slijedeće četiri osnovne strukture manipulatora:[1]

- pravokutna (eng. *Cartesian or rectangular*) ili TTT,
- cilindrična (eng. *cylindrical*) ili RTT,
- sferna (eng. *spherical*) ili RRT,
- rotacijska (eng. *articulated*) ili RRR

Pravokutna ili TTT konfiguracija robota ima tri zgloba koja su translacijska te su njoj osi međusobno okomite (Sl. 3.3.). Zbog jednostavnosti geometrije, svaki stupanj pokretljivosti se podudara sa stupnjem slobode u Cartesianovom prostoru. Možemo reći da se tu radi o pravocrtnom kretanju, te takva struktura pokazuje dobru mehaničku čvrstoću. Točnost pozicioniranja ručnog zgloba je jednaka u cijelom radnom prostoru. Ovakav sustav ima visoku točnost, ali zato ima slabu pokretljivost, jer su svi zglobovi translacijski. Prostor u kojemu robot izvršava radnje je prizma. Pravokutna konfiguracija robota pristupa objektu sa strane. Ukoliko želimo objektu pristupiti s gornje strane, trebali bi ovaj manipulator realizirati u obliku stalka. Pravokutna konfiguracija robota omogućuje postizanje radnog prostora velikih dimenzija i manipuliranje glomaznim objektima. Zbog toga se najčešće primjenjuje u rukovanju materijalima i montaži. Motori za pokretanje zglobova manipulatora su najčešće električni, a rijetko kada pneumatski. Ako se prvi zglob kod pravokutne strukture zamijeni rotacijskim zglobom, tada se dobiva robot cilindrične konfiguracije, prikazan na Sl. 3.4. Radni prostor takvog robota je volumen između dva vertikalna koncentrična plašta valjka (zbog ograničenog translacijskog kretanja). Cilindrični manipulator pokazuje dobru mehaničku čvrstoću, ali se točnost pozicioniranja ručnog zgloba smanjuje s povećanjem horizontalnog hoda. Za prijenos se najčešće upotrebljavaju hidraulički motori zbog prijenosa objekata koji su većih dimenzija.[1]

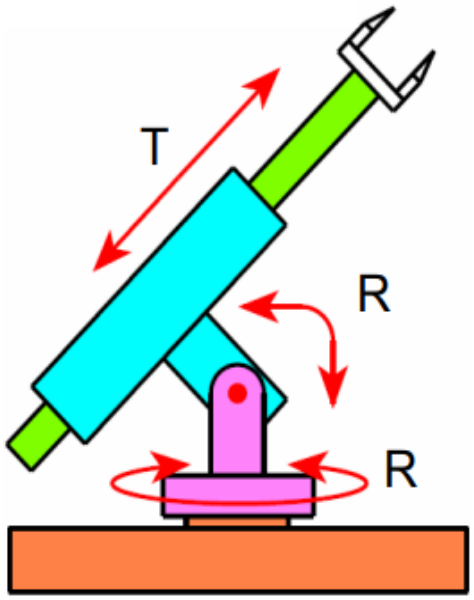


Sl. 3.3. Pravokutna konfiguracija robota[16]

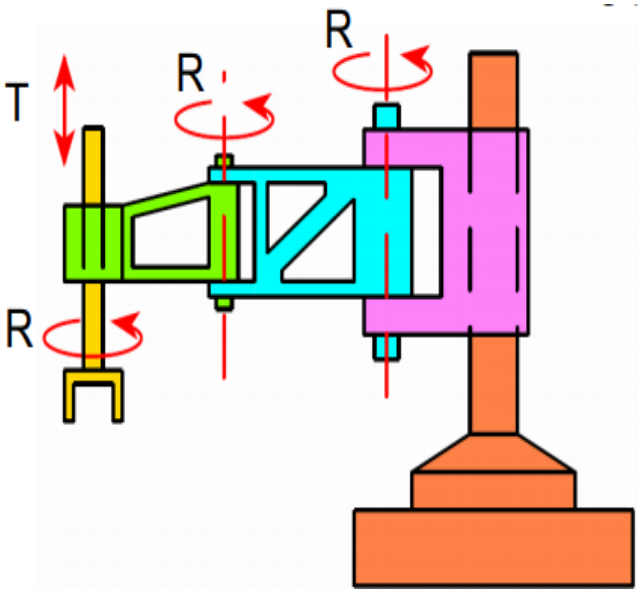


Sl. 3.4. Cilindrična konfiguracija robota[16]

Zamjenom drugog zgloba cilindrične konfiguracije robota rotacijskim zglobom dobiva se robot sferne konfiguracije (Sl. 3.5.). Ako postoji ograničenje translacijskog kretanja, tada je radni prostor tog tipa robota volumen između dvije koncentrične sfere, a uz ograničenje svih kretanja, radni prostor je dio volumena između dvije koncentrične sfere. Mehanička čvrstoća je manja u odnosu na prethodne strukture zbog složenije geometrijske i mehaničke konstrukcije. Točnost pozicioniranja se smanjuje s porastom radijalnog hoda. Sferični manipulator se uglavnom koristi u strojarskoj industriji. Obično se koriste električni motori za pokretanje zglobova manipulatora. Robot tipa SCARA (eng. Selective Compliance Assembly Robot Arm) također ima dva rotacijska i jedan translacijski zglob, kao što je prikazano na Sl. 3.6. Kod ovog tipa robota su sve tri osi vertikalne. SCARA manipulator karakterizira visoka čvrstoća za opterećenja na vertikalnoj osi i popustljivost za opterećenja u horizontalnoj osi. Zbog toga se SCARA koristi za zadatke montiranja po vertikalnoj osi. Točnost pozicioniranja se smanjuje s porastom udaljenosti između ručnog zgloba i osi prvog zgloba. [1]

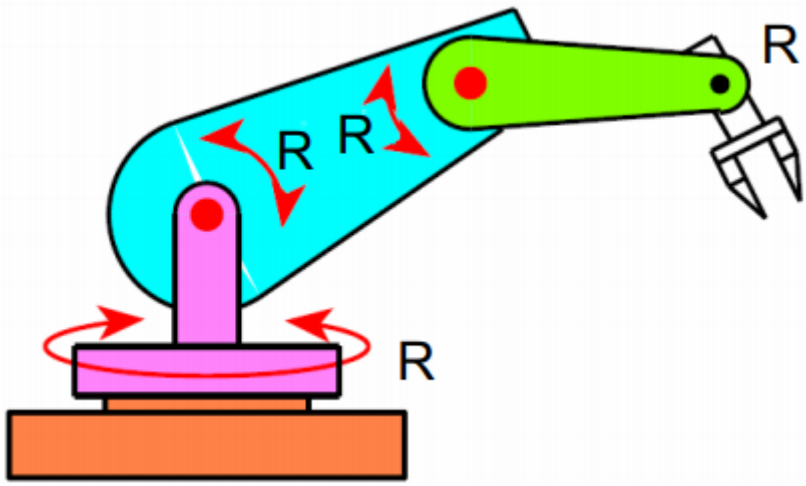


Sl. 3.5. Sferna konfiguracija robota[16]



Sl. 3.6. SCARA robot[16]

Ako su upotrjebljena sva tri rotacijska zgloba dobiva se rotacijska struktura manipulatora, koja se još naziva laktasta ili zglobna (Sl. 3.7). Osi rotacije drugog i trećeg zgloba su paralelne i okomite na os rotacije prvog zgloba. Ako ne postoje ograničenja rotacijskih kretanja, tada je radni prostor tog robota kugla, a uz ograničenja to je dio kugle složenog oblika čiji je presjek sa strane najčešće u obliku polumjeseca. Zbog sličnosti sa čovjekovom rukom, drugi zglob se zove vratni zglob, a treći zglob lakta jer povezuje gornji dio ruke s podlakticom. Za pogon zglobova antropomorfne strukture koriste se električni motori. Područje primjene je jako široko.



Sl. 3.7. Rotacijska konfiguracija robota[16]

Navedene strukture manipulatora dobivene su na osnovu zahtjeva na poziciju ručnog zgloba i orijentaciju vrha manipulatora. Ako se želi postići odgovarajuća orijentacija u trodimenzionalnom prostoru, ručni zglob mora posjedovati najmanje tri stupnja pokretljivosti

ostvarenih rotacijskim zglobovima. Budući da ručni zglob čini krajnji dio manipulatora on može biti stisnut (zbijen), što će imati za posljedicu kompliciranu mehaničku izvedbu. Bez ulaženja u konstrukcijske detalje, realizacija ručnog zgloba s najvećom okretljivošću je ona gdje se osi sva tri rotacijska zgloba sijeku u jednoj tački. Ovaj zglob se zove sferni. Glavna osobina sfernog zgloba je razdvajanje pozicije i orijentacije vrha manipulatora. Ruka je zadužena za zadatke pozicioniranja gornje točke presjeka, dok je ručni zglob zadužen za određivanje orijentacije vrha manipulatora. Realizacije u kojima zglob nije sferni su jednostavne s mehaničke točke gledišta, ali su pozicija i orijentacija sjedinjene i to komplicira koordinaciju između kretanja ruke i obavljanja zadatka od strane ručnog zgloba. Vrh manipulatora je određen u skladu sa zadatkom kojeg robot treba izvršiti. Za rukovanje materijalom, vrh manipulatora je sačinjen u obliku hvataljke određenog oblika i dimenzija koje ovise o objektu koji se hvata. Za zadatke montiranja, vrh manipulatora je alat(oruđe) ili određena sprava, kao npr. zavarivač, glodalica, bušilica, uređaj za zašarafljivanje. Izbor robota je uvjetovan primjenom koju ograničava oblik i dimenzije radnog prostora, maksimalan iznos tereta, točnost pozicioniranja i dinamičke performanse manipulatora.[1]

3.2. Vrste pogona

Pomicanje dijelova robotskog manipulatora omogućeno je upotrebom pogonskog mehanizma robota. Pogonski mehanizam određuje brzinu pomicanja ruke, jakost i dinamička svojstva manipulatora. Vrste pogonskih mehanizama određuje područja primjene robotskog manipulatora. Najzastupljenija su tri slijedeća pogona:

1. Električni motor
2. Hidraulički motor
3. Pneumatski motor.

Većina robotskih manipulatora danas koristi električni motori za pogon. Najčešće su to istosmjerni, izmjenični i koračni, zbog svoje niske cijene, ne zauzimaju puno prostora, postižu veliku brzinu i točnost, te je kod njih moguća primjena složenih algoritama upravljanja. Međutim, kod specifičnih primjena (npr. rukovanje užarenim čelikom ili sastavljanje dijelova automobila), kada se zahtijeva manipulacija velikim teretima, češće se koriste roboti s hidrauličkim motorom. Hidraulički motor osim velike brzine (veća nego kod električnog motora) i snage, omogućuje mirno održavanje pozicije zbog nestlačivosti ulja. Koriste se kod robota većih dimenzija. Glavni nedostaci ovih motora su njihove visoke cijene i zagađivanje okolice zbog buke i mogućeg istjecanja ulja. Pneumatski motori primjenjuju se kod malih

robota. Prednost im je relativno niska cijena, velika brzina rada i nezagađivanje okoline, te su zbog toga pogodni za laboratorijski rad. Takvi motori nisu pogodni za rad s velikim teretima, jer je zbog stlačivosti zraka nemoguće mirno održavati željenu poziciju. Uz to je prisutna buka te je potrebno dodatno filtriranje i sušenje zraka zbog nepoželjne prašine i vlage. Ako se zahtijeva samo otvaranje i zatvaranje hvataljke (vrh manipulatora), tada se u završnom mehanizmu koristi pneumatski motor da se grubim stiskom ne bi oštetio lomljivi predmet.[1]

3.3. Način upravljanja

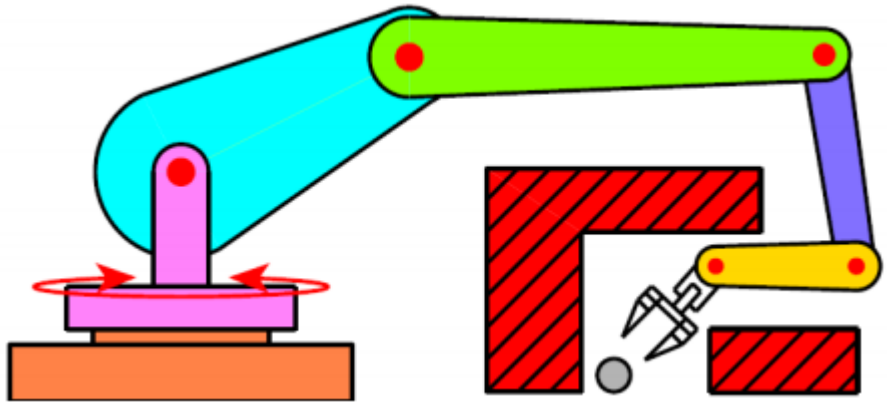
Postoje dva načina upravljanja kretanja vrha manipulatora, to su:

- od točke do točke (eng. Point-to-point motion),
- kontinuirano kretanje po putanji (eng. Continuous path).

Kod kretanja od točke do točke, vrh manipulatora se kreće po diskretnim točkama u radnom prostoru i pri tome nije bitna putanja između točaka, ali je važna točnost pozicioniranja. Takav način kretanja koristi se za diskretne operacije kao što su: točkasto zavarivanje te podizanje i spuštanje predmeta. Pri kontinuiranom kretanju po putanji vrh manipulatora mora se kretati po unaprijed određenoj putanji u trodimenzionalnom prostoru i pri tome su bitne trajektorija i točnost pozicioniranja. Roboti kod kojih se upravlja trajektorijom kretanja mogu se koristiti za bojenje, šavno zavarivanje ili lijepljenje. Osim vrsta pogona, geometrije radnog prostora i načina upravljanja kretanjem postoji niz dodatnih karakteristika robota: broj osi, maksimalna masa tereta, maksimalna brzina, dohvat, hod, orijentacija alata, ponovljivost, preciznost, točnost te radna okolina.[1]

3.4. Stupanj slobode gibanja

Svakog robota karakterizira broj stupnjeva slobode gibanja tj. broj osi za rotacijsko ili translacijsko kretanje njegovih segmenata. Kretanje robota odvija u trodimenzionalnom prostoru, prve tri osi najčešće se koriste za određivanje pozicije ručnog zgloba, dok preostale osi određuju orijentaciju vrha manipulatora. Manipulator najčešće ima šest osi te može dovesti vrh manipulatora u bilo koju poziciju i orijentaciju unutar radnog prostora. Pri tome se mehanizam otvaranja i zatvaranja hvataljke alata ne smatra nezavisnom osi jer ne utječe niti na poziciju niti na orijentaciju hvataljke. Ako manipulator ima više od šest osi, tada se redundantne (dodatne osi) mogu koristiti za izbjegavanje prepreka unutar radnog prostora, slika 3.4.1.[1]



Sl. 3.4.1 Redundantni roboti [16]

4. MATEMATIČKI MODEL ROBOTSKOG MANIPULATORA

4.1. Direktna kinematika

Robotski manipulator može se promatrati kao lanac krutih članaka koji su povezani zglobovima. Da bi se mogla odrediti točka u prostoru u koju je postavljen vrh alata s obzirom na bazu robota, treba odrediti vezu između varijabli zglobova i pozicije odnosno orijentacije alata. Rješavanje tog problema naziva se rješavanje direktnog kinematičkog problema. Robotski manipulator promatra se kao kinematički lanac koji se sastoji od niza članaka povezanih zglobovima, pri čemu je baza robota nulti članak, a alat *n*-ti članak, gdje je n broj osi odnosno zglobova robota. *i*-ti zglob povezuje članak *i*-1 s *i*-tim člankom, kao što je prikazano na Sl. 3.1. Kao što je već navedeno, zglobovi robota mogu biti ili rotacijski ili translacijski. Rotacijski zglob, kao što sam naziv kaže rotira odgovarajući članak robota u odnosu na njemu prethodni članak oko osi zgloba te tako utječe na položaj alata. Dok translacijski pomiče odgovarajući članak u odnosu na njemu prethodni članak oko osi zgloba, te tako utječe na položaj alata. Uvodimo novi pojam varijabla zgloba, to je kut rotacijskog odnosno pomak translacijskog zgloba. Iz praktičnih će se razloga u nastavku varijabla *i*-tog zgloba označavati s q_i, što će biti jedinstvena oznaka kako za kut rotacijskog zgloba tako i za pomak translacijskog zgloba. Sve varijable zglobova nekog robotskog manipulatora mogu se prikazati vektorom [11]

$$q = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{bmatrix} \tag{3.1.-1}$$

Ovaj vektor se naziva vektor varijabli zglobova. Prostor svih vektora varijabli zglobova se naziva prostorom zglobova (engl. joint space). To je n-dimenzionalni prostor, gdje je n broj zglobova robota, čija je svaka točka jedna moguća kombinacija vrijednosti varijabli zglobova. Položaj alata u odnosu na bazni koordinatni sustav može se predstaviti točkom u šestodimenzionalnom prostoru, koji se naziva prostor konfiguracije alata. Koristeći tu terminologiju direktna kinematika se može promatrati kao preslikavanje iz prostora zglobova u prostor konfiguracije alata. Iz činjenice da proizvoljan položaj ima šest stupnjeva slobode proizlazi da je minimalni broj zglobova robotskog manipulatora, kojim se alat može postaviti u proizvoljan položaj u radnom prostoru robota, šest. Kako se robot može modelirati kao otvoreni kinematički lanac, za određivanje kinematičkih parametara može se primijeniti Denavit-Hartenbergova metoda. Metoda se sastoji od pridruživanja desno orijentiranih ortonormiranih

koordinatnih sustava svakom članku kinematičkog lanca, te iz odnosa koordinatnih sustava mogu se odrediti parametri članaka i zglobova. [11]

4.1.1 Denavit-Hartenbergova metoda

Da bi prikazali određene geometrijske i fizičke veličine vezane za kretanje elemenata robota, potrebno je definirati određene referentne koordinatne sustave u kojima će se sve navedene veličine prikazivati i računati. Pomoću Denavit–Hartenbergove notacije određuje se veza između članaka robota. Denavit–Hartenbergovi parametri se određuju prema sljedećim koracima. [12]

1. Potrebno je odrediti osi svih zglobova i pridružiti im redne brojeve tako da su zglobovi i i $i+1$ povezani člankom i, gdje je $i = 1, 2, \ldots, n-1$, a n je broj zglobova.
2. Bazi robota pridružiti koordinatni sustav S_0. Os Z_0 postaviti u smjeru zgloba 1
3. Odrediti normalu B_i na osi zglobova i i $i+1$. Normala B_i predstavlja pravac koji je okomit na osi zglobova i i $i+1$ i siječe te osi. $(i = 1, 2, \ldots, n-1)$
4. Postaviti ishodište koordinatnog sustava S_i u sjecište pravca B_i s osi zgloba $i+1$. $(i = 1, 2, \ldots, n-1)$
5. Os Z_i postaviti u smjeru zgloba $i+1$. $(i = 1, 2, \ldots, n-1)$.
6. Os X_i postaviti tako da leži na normali B_i i usmjerena je od Z_{i-1} prema Z_i. $(i = 1, 2, \ldots, n-1)$
7. Os Y_i postaviti tako da se dobije desno orijentirani ortonormirani koordinatni sustav. $(i = 1, 2, \ldots, n-1)$
8. Ishodište koordinatnog sustava S_n postaviti u vrh alata. Os Z_n postaviti u smjeru osi n-tog zgloba, a os X_n postaviti tako da leži na normali na osi Z_n i Z_{n-1} i usmjerena je od Z_{n-1} prema Z_n .
9. Odrediti parametar θ_i kao kut rotacije od osi X_{i-1} prema osi X_i mjeren oko osi Z_{n-1}. $(i = 1, 2, \ldots, n)$
10. Točku presjeka osi X_i i osi Z_{i-1} označiti sa C_i. $(i = 1, 2, \ldots, n)$
11. Odrediti parametar d_i kao udaljenost od ishodišta koordinatnog sustava S_{i-1} do točke C_i, mjereno u smjeru osi Z_{i-1}. $(i = 1, 2, \ldots, n)$
12. Odrediti parametar a_i kao udaljenost od točke C_i do ishodišta koordinatnog sustava S_i, mjereno u smjeru osi X_i. $(i = 1, 2, \ldots, n)$
13. Odrediti parametar α_i kao kut rotacije od osi Z_{i-1} prema osi Z_i mjeren oko osi X_i. $(i = 1, 2, \ldots, n)$

Denavit-Hartenbergovim postupkom određuju se četiri kinematička parametra robotskog manipulatora za svaki zglob. To su: θ_i, d_i, a_i i α_i. Jedan od ta četiri parametra je varijabla zgloba. Kada se radi o rotacijskom zglobu to je θ_i, a ako je zglob translacijski, tada je d_i, varijabla zgloba. Ostala tri parametra su konstante. Robotski manipulator koji se koristi u ovom diplomskom radu nema translacijski zglob, već su svi zglobovi isključivo rotacijski. [12]

Denavit-Hartenbergovi parametri i njihove vrijednosti za robotski manipulator korišteni u diplomskom radu su navedeni u tablici 4.1.1.1.

Tab. 4.1.1.1 Denavit-Hartenbergovi parametri.

***Članak* i**	**θ_i[rad]**	**d_i[mm]**	**a_i[mm]**	**α_i[rad]**
1	θ_1	60	0	$\frac{\pi}{2}$
2	θ_2	0	105	0
3	θ_3	0	96	π
4	θ_4	0	30	$\frac{\pi}{2}$
5	θ_5	135	0	0

Određivanje kinematičkih parametara nekog robotskog manipulatora omogućuje sustavno rješenje problema direktne kinematike. Naime, može se pokazati da vrijedi

$$ {}^{i-1}T_i = \begin{bmatrix} c\theta_i & -s\theta_i c\alpha_i & s\theta_i s\alpha_i & a_i c\theta_i \\ s\theta_i & c\theta_i c\alpha_i & -c\theta_i s\alpha_i & a_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1.1-1} $$

$$ {}^{0}T_n = {}^{0}T_1 * {}^{1}T_2 * {}^{2}T_3 \ldots {}^{n-1}T_n \tag{4.1.1-2} $$

Izračunavanjem matrica ${}^{i-1}T_i$ za svaki $i = 1, 2, \ldots, n$ pomoću matrice (4.1.1-1) te uvrštavanjem dobivenih matrica u (4.1.1 - 2) može se izračunati matrica ${}^{0}T_5$ kojom je definiran položaj alata u odnosu na bazu robota, čime je riješen problem direktne kinematike.

$$ {}^{0}T_5 = {}^{0}T_1 * {}^{1}T_2 * {}^{2}T_3 * {}^{3}T_4 * {}^{4}T_5 \tag{4.1.1-3} $$

4.1.2 Programsko rješenje direktne kineamtike

Za izračun direktne kinematike koristili smo formulu (4.1.1-1) koju smo prethodno objasnili. Prilikom izračuna koristili smo Denavit-Hartenbergovi parametre iz tablice 4.1.1.1 i kutove zglobova q_1, q_2, q_3 koje smo dobili izračunom direktne kinematike koju ćemo objasniti u nastavku. U drugoj, osmoj i četrnaestoj liniji koda formirana je homogena matrica transformacije ${}^0H_1,\ {}^1H_2,\ {}^2H_3$.

```
1.  % Direct kinematics
2.  H01 = [cos(q1) -sin(q1)*cos(alfa1) sin(q1)*sin(alfa1)  a1*cos(q1);
3.        sin(q1)  cos(q1)*cos(alfa1)  -cos(q1)*sin(alfa1) a1*sin(q1);
4.        0        sin(alfa1)          cos(alfa1)          d1;
5.        0        0                   0                   1
6.  ];
7.
8.  H12 = [cos(q2) -sin(q2)*cos(alfa2) sin(q2)*sin(alfa2)  a2*cos(q2);
9.        sin(q2)  cos(q2)*cos(alfa2)  -cos(q2)*sin(alfa2) a2*sin(q2);
10.       0        sin(alfa2)          cos(alfa2)          d2;
11.       0        0                   0                 1
12. ];
13.
14. H23 = [cos(q3) -sin(q3)*cos(alfa3) sin(q3)*sin(alfa3)  a3*cos(q3);
15.       sin(q3)  cos(q3)*cos(alfa3)  -cos(q3)*sin(alfa3) a3*sin(q3);
16.       0        sin(alfa3)          cos(alfa3)          d3;
17.       0        0                   0                   1
18. ];
19.
20. H03=H01*H12*H23;
21. R03=H03(1:3,1:3);
22.
23. beta = pi/2;
24. R06=[cos(beta)  -sin(beta)  0;
25.      sin(beta)  cos(beta)   0
26.      0          0          -1];
27. R36=R03'*R06;
```

U 20. liniji koda izračunata je homogena matrica transformacije ${}^0H_3$ s formulom:

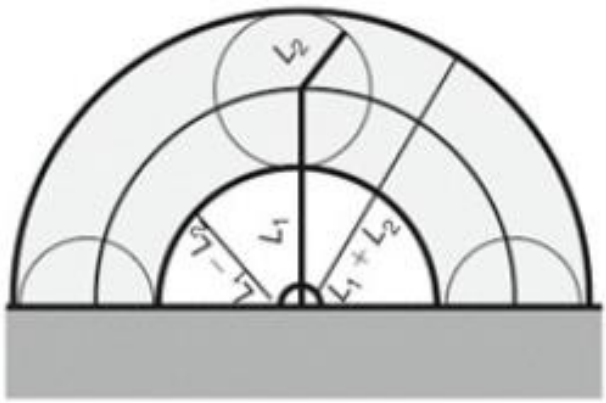
$$ {}^0H_3 = {}^0H_1 * {}^1H_2 * {}^2H_3 \qquad (4.1.1\text{-}3) $$

Nakon toga iz homogene matrice transformacije uzeti rotacijski dio i spremiti u matricu ${}^0R_3$. Pomoću rotacijskog dijela može se dobiti kut pod kojim hvatamo predmet s robotskim manipulatorom tj. orijentaciju alata. Prije samog izračuna kuta za orijentaciju alata potrebno je izračunati matricu rotacije ${}^0R_3$.

4.2. Inverzna kinematika

Inverzna kinematika rješava problem obrnut od direktne kinematike. Za zadanu krajnju točku manipulatora inverzna kinematika pronalazi položaje i kutove svih zglobova takve da manipulator bude u traženoj točki, ako je ta točka dohvatljiva. To je složen problem, redovito s

beskonačno mnogo rješenja. Kao što vidimo inverzna kinematika je puno upotrebljivija i zanimljivija od direktne kinematike. Za shvaćanje principa inverzne kinematike prvo ćemo objasniti pojam dohvatljivog radnog prostora (eng. *reachable workspace*). Na slici 4.2.1 vidimo jednostavan primjer, s označenim radnim prostorom. [14]



Sl. 4.2.1 Dohvatljivi radni prostor (eng. *reachable workspace*)[15]

Tanka puna linije označavaju doseg segmenta L_1, dok puna deblja linija predstavlja doseg segmenta L_2. Matematički bi radni prostor mogli opisati sljedećom formulom:

$$L_1 - L_2 \leq dohvatljivi\ radni\ prostor \leq L_1 + L_2$$

Često u praksi neki zglobovi imaju ograničenja u pokretima. Sa slike 4.2.1 možemo vidjeti da zglob kod prvog članka L_1 ima ograničenje od 180°, a L_2 ima tek ograničenje u rubnom dijelu dohvatljivog radnog prostora.

4.2.1 Pieperovo rješenje

Pieperovo rješenje se još zove „321 kinematička struktura“ i ona se odnosi na robotske manipulatore sa šest zglobova. Ovakvo rješenje je razvio Donald L. Pieper i ono se koristi u mnogim komercijalnim robotskim manipulatorima. Ovo je analitičko rješenje problema inverzne kinematike. Kao što smo rekli takvo rješenje se odnosi na manipulator sa šest zglobova, pri čemu su zadnja tri zgloba rotacijska, a njihove se osi sijeku u jednoj točki. Za položaj alata zadan matricom homogene transformacije

$$^0T_6 = \begin{bmatrix} ^0R_6 & ^0t_6 \\ 0 & 1 \end{bmatrix} \qquad (4.2.1\text{-}1)$$

vrijednosti varijabli zglobova q mogu se izračunati sljedećim postupkom:

1. Potrebno je odrediti poziciju $p = [x \quad y \quad z]^T$ sjecišta osi tri zadnja zgloba, pomoću izraza

$$\begin{bmatrix} x \\ x \\ z \\ 1 \end{bmatrix} = {}^0T_6 \begin{bmatrix} 0 \\ 0 \\ -d_6 \\ 1 \end{bmatrix} \tag{4.2.1-2}$$

2. Odrediti vrijednosti varijabli prva tri zgloba rješavajući sustav jednadžbi

$$x = c_1 g_1 - s_1 g_2 \tag{4.2.1-3}$$

$$y = s_1 g_1 - c_1 g_2 \tag{4.2.1-4}$$

$$z = s * \alpha_1 (f_1 s_2 + f_2 c_2) + c\alpha_1 (f_3 + d_2) + d_1 \tag{4.2.1-5}$$

$$g_1 = c_2 f_1 - s_2 f_2 + a_1 \tag{4.2.1-6}$$

$$g_2 = c\alpha_1 (f_1 s_2 + f_2 c_2) - s\alpha_1 (f_3 + d_2) \tag{4.2.1-7}$$

$$f_1 = d_4 s\alpha_3 s_3 + a_3 c_3 + a_2 \tag{4.2.1-8}$$

$$f_2 = a_3 c\alpha_2 s_3 - d_4 c\alpha_2 s\alpha_3 c_3 - d_4 s\alpha_2 c\alpha_3 - s\alpha_2 d_3 \tag{4.2.1-9}$$

$$f_3 = a_3 s\alpha_2 s_3 - d_4 s\alpha_2 s\alpha_3 c_3 + d_4 c\alpha_2 c\alpha_3 + c\alpha_2 d_3 \tag{4.2.1-10}$$

Gdje je $c_i = cos\theta_i, s_i = sin\theta_i, c\alpha_i = cos\alpha_i, s\alpha_i = sin\alpha_i, i = 1,2, \dots, n$

3. Pomoću vrijednosti varijabli prva tri zgloba izračunatih u koraku 2. odrediti rotacijsku matricu ${}^0R_3$
4. ${}^0R_6 = ({}^0R_3)^T \, {}^0R_6$
5. Odrediti varijable zadnja tri zgloba rješavanjem sustava jednadžbi dobivenog izjednačavanjem elemenata matrice ${}^0R_3$ izračunate u koraku 4. s matricom

${}^0R_6(\theta_4, \theta_5, \theta_6)$

$$= \begin{bmatrix} c_4 & -s_4 c\alpha_4 & s_4 s\alpha_4 \\ s_4 & c_4 c\alpha_4 & -c_4 s\alpha_4 \\ 0 & s\alpha_4 & c\alpha_4 \end{bmatrix} \begin{bmatrix} c_5 & -s_5 c\alpha_5 & s_5 s\alpha_5 \\ s_5 & c_5 c\alpha_5 & -c_5 s\alpha_5 \\ 0 & s\alpha_5 & c\alpha_5 \end{bmatrix} \begin{bmatrix} c_6 & -s_6 c\alpha_6 & s_6 s\alpha_6 \\ s_6 & c_6 c\alpha_6 & -c_6 s\alpha_6 \\ 0 & s\alpha_6 & c\alpha_6 \end{bmatrix} \tag{4.2.1-11}$$

U nastavku su dane upute za rješavanje sustava jednadžbi od (4.2.1-3) do (4.2.1-10) za rotacijsku konfiguraciju robota. Varijable zglobova su $\theta_1, \theta_2 \; i \; \theta_3$.

Iz (4.2.1-3) do (4.2.1-7) slijedi:

$$x^2 + y^2 + z^2 = 2a_1 (f_1 c_2 - f_2 s_2) + 2d_1 z + k_1 \tag{4.2.1-12}$$

$$z = s\alpha_1 (f_1 s_2 + f_2 c_2) + k_2 \tag{4.2.1-13}$$

$$k_1 = f_1^2 + f_2^2 + f_3^2 + a_1^2 - d_1^2 + d_2^2 + 2d_2 f_3 \tag{4.2.1-14}$$

$$k_2 = c\alpha_1(f_3 + d_2) + d_1 \tag{4.2.1-15}$$

Iz (4.2.1-12) i (4.2.1-13) slijedi

$$\frac{(x^2 + y^2 + z^2 - 2d_1 z - k_1)}{4a_1^2} + \frac{(z + k_1)^2}{s^2\alpha_1} = f_1^2 + f_2^2 \tag{4.2.1-16}$$

Uvrštavanjem (4.2.1-14) i (4.2.1-15) u (4.2.1-16) te uvrštavanjem (4.2.1–8), (4.2.1-9) i (4.2.1-10) u dobivenu jednadžbu, dobiva se jednadžba četvrtog stupnja u kojoj je jedina nepoznanica θ_3. Rješavanjem te jednadžbe može se izračunati vrijednost kuta θ_3. U slučaju da je $a_1 = 0$, jednadžba (4.2.1-12) prelazi u:

$$x^2 + y^2 + z^2 = 2d_1 z + k_1 \tag{4.2.1-17}$$

pa se umjesto rješavanjem jednadžbe (4.2.1-16), θ_3 se izračunava rješavanjem jednadžbe (4.2.1-16). U slučaju da je $\alpha_1 = 0$, jednadžba (4.2.1-13) prelazi u:

$$z = k_2 \tag{4.2.1-18}$$

pa se umjesto rješavanjem jednadžbe (4.2.1-16), θ_3 se izračunava rješavanjem jednadžbe (4.2.1-18). Jednadžbe (4.2.1-16), (4.2.1-17) odnosno (4.2.1-18) mogu poslužiti za izračunavanje θ_1, θ_2 i θ_3 primjenom sljedećeg postupka:

1. Ako je $a_1 = 0$, uvrstimo (4.2.1-14) u (4.2.1-17) te u dobivenu jednadžbu uvrstimo u (4.2.1-8), (4.2.1-9) i (4.2.1-10). Dobiva se jednadžba u kojoj je jedina nepoznanica θ_3. Ta se jednadžba može riješiti uvođenjem sljedećih supstitucija:

   $$cos\theta_3 = \frac{1 - u^2}{1 + u^2}, sin\theta_3 = \frac{2u}{1 + u^2} \tag{4.2.1-19}$$

   te rješavanjem jednadžbe po u. Nakon toga se θ_3 izračunava iz u na sljedeći način:

   $$\theta_3 = 2\arctan u \tag{4.2.1-20}$$

   Ako je $\alpha_1 = 0$, uvrštavamo (4.2.1-15) u (4.2.1-18) te u dobivenu jednadžbu uvrštavamo u (4.2.1-8), (4.2.1-9) i (4.2.1-10). Dobiva se jednadžba u kojoj je jedina nepoznanica θ_3. Ta se jednadžba također može riješiti pomoću supstitucija (4.2.1-19).

   Ako je $a_1 \neq 0$ i $\alpha_1 \neq 0$, uvrstiti (4.2.1-14) i (4.2.1-15) u (4.2.1-16) te u dobivenu jednadžbu uvrstiti (4.2.1-8), (4.2.1-9) i (4.2.1-10). Dobiva se jednadžba u kojoj je jedina nepoznanica θ_3. Dobivenu jednadžbu rješavamo pomoću supstitucije (4.2.1-19). Pri rješavanju jednadžbi (4.2.1-16), (4.2.1-17) odnosno (4.2.1-18), korisno je znati sljedeću relaciju:

$$f_1^2 + f_2^2 + f_3^2 = a_3^2 + d_4^2 + d_3^2 + a_2^2 + 2d_4 c\alpha_3 d_3 + 2a_2 a_3 c_3 + 2a_2 d_4 s\alpha_3 s_3 \quad (4.2.1\text{-}21)$$

2. Uvrštavanjem vrijednosti kuta θ_3 izračunate u koraku 1. u jednadžbe (4.2.1-8), (4.2.1-9) i (4.2.1-10) dobijemo $f_1, f_2\ i\ f_3$.
3. Uvrštavanjem dobivenih vrijednosti $f_1, f_2\ i\ f_3$ u (4.2.1-12) ili (4.2.1-13) može se dobiti jednadžba čijim se rješavanjem dolazi do θ_2. Ta se jednadžba može riješiti supstitucijama (4.2.1-19).
4. Uvrštavanjem vrijednosti kuta θ_2 izračunate u koraku 3 u jednadžbe (4.2.1-6) i (4.2.1-7) izračunati $g_1\ i\ g_2$.
5. Uvrštavanjem dobivenih vrijednosti $g_1\ i\ g_2$ u (3) i (4) dobiva se sustav jednadžbi čijim se rješavanjem dolazi do $c_1\ i\ s_1$. Iz ovih se vrijednosti može izračunati θ_1 kao:

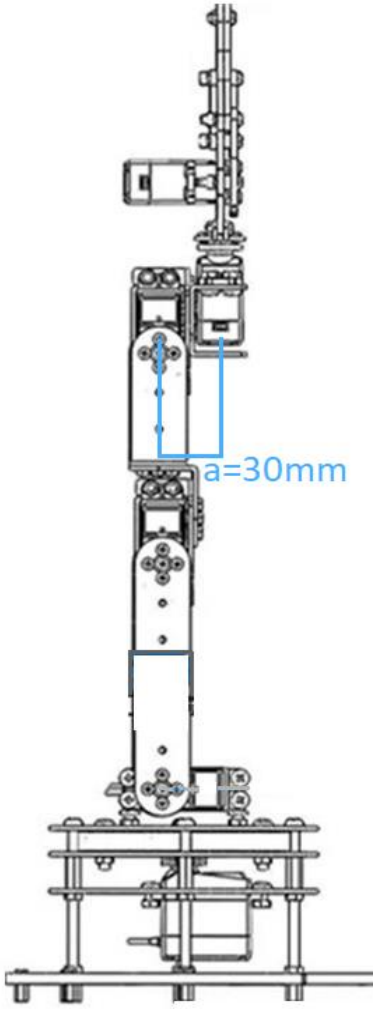
$$\theta_1 = atan2(s_1, c_1) \quad (4.2.1\text{-}22)$$

4.2.2 Programsko rješenje inverzne kinematike

Kod programskog rješenja inverzne kinematike je u potpunosti riješen u programskom jeziku MATLAB. Potrebno je bilo prvo validirati unos korisnika, tj. provjeriti je li korisnik unio sve potrebne podatke za izračun inverzne kinematike. Za izračun nam je potrebna točka u prostoru s koordinatama x, y, z. Prilikom unosa te tri točke i pritiskom korisnika na tipku „Calculate“ poziva se funkcija *calculate_Callback()*. Funkcija na samom početku (od 2. do 4. linije koda) dohvaća vrijednosti iz polja za unos i pretvara ih iz tekstualnog oblika (eng. *string*) u broj (eng. number). Od 5. do 11. linije koda provjeravamo je li korisnik unio ispravno koordinate neke točke u prostoru koja je dohvatljiva robotskom manipulatoru. U slučaju da je korisnik unio krivo bilo koju točku, prikazat će mu se skočni prozor s odgovarajućom porukom. A ako su sve točke u odgovarajućem rasponu, nastaviti će se s izvršavanjem koda te će se definirati varijable vezane za DH parametre (linije koda do 13. do 15).

```
1.  function calculate_Callback(hObject, eventdata, handles)
2.  x = str2num(char(get(handles.x,'String')));
3.  y = str2num(char(get(handles.y,'String')));
4.  z = str2num(char(get(handles.z,'String')));
5.  if isempty(x) || x<-217 || x>217
6.      msgbox('The value X must be between -217 and 217','Error','error');
7.  elseif isempty(y) || y<-217 || y>0
8.      msgbox('The value Y must be between -217 and 0','Error','error');
9.  elseif isempty(z) || z<0 || z>40
10.     msgbox('The value Z must be between 0 and 40','Error','error');
11. else
12.     %% Denavit-Hartenberg parameters
13.     d1=75;      d2=0;    d3=0;
14.     a1=0;       a2=105;  a3=96;
15.     alfa1=pi/2; alfa2=0; alfa3=pi;
```

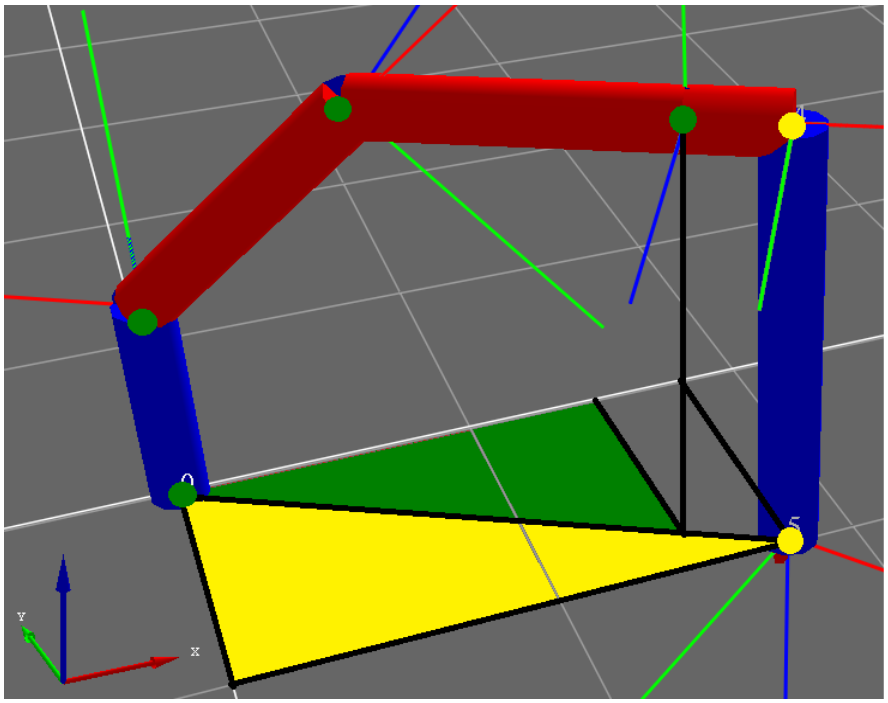
Od 17. do 31. linije koda računamo Pitagorin poučak kako bi dobili poziciju u kojoj će se nalaziti završetak trećeg članka. To radimo zbog pomaka (eng. *displacement*) koji se nalazi na četvrtom članku, te je on prikazan na slici 4.2.2.1



Sl. 4.2.2.1 Bokocrt robotskog manipulatora

Odgovarajućim projektiranjem manipulatora može se značajno olakšati rješavanje inverzne kinematike. Dovoljno je da se osi orijentacije sijeku u jednoj točki (ili da su neke tri osi paralelne). Ovim se problem može razdvojiti na IKP[1] pozicioniranja i IKP orijentacije. Da bi lakše objasnili programsko rješenje prikazat ćemo ga slikom 4.2.2.2 modela robotskog manipulatora.

[1] IKP-inverzni kinematički problem



Sl. 4.2.2.2 Model robotskog manipulatora

Kao što vidimo na slici se nalazi model robotske ruke sa zelenim i žutim točkama na svakom zglobu. Zelene točke predstavljaju onaj dio koji rješavamo s IK pozicioniranja, dok žute točke predstavljaju dio koji rješavamo s IK[2] orijentacije. Prilikom rješavanja IK pozicioniranja krećemo od završetka trećeg članka robotskog manipulatora. Završetak trećeg članka tj. pozicija računamo preko Pitagore, kako je obilježeno na slici sa žutim i zelenim trokutom.

```
%% Pythagoras theorem
    a=abs(x); b=abs(y);
    c=sqrt(a^2+b^2);
    alpha=atand(a/b);
    c_=c-30;
    a_=sind(alpha)*c_;
    b_=sqrt(c_^2-a_^2);
    if x>0
        x=a_;
        y=-b_;
        z=z+135;
    else
        x=-a_;
        y=-b_;
        z=z+135;
    end
```

Nakon što smo dobili vrijednosti $x, y\ i\ z$ na trećem članku, možemo izračunati kut θ_3. Slijedeći postupak koji smo opisali u poglavlju 4.2.1 za Pieperovo rješenje dobivamo sljedeće formule za rješenje kuta θ_3.

[2] IK-inverzna kinematika

$$c_3 = \frac{(1-u_3^2)}{(1+u_3^2)}, \tag{4.2.2-1}$$

$$s_3 = \frac{2 * u_3^2}{(1+u_3^2)}, \tag{4.2.2-2}$$

$$r = x^2 + y^2 + z^2, \tag{4.2.2-3}$$

$$f_1 = a_3 * c_3 + a_2 \tag{4.2.2-4}$$

$$f_2 = a_3 * s_3, \tag{4.2.2-5}$$

$$k_1 = f_1^2 + f_2^2 - d_1^2, \tag{4.2.2-6}$$

$$\theta_3 = 2 * \text{atan}(u_3) \tag{4.2.2-6}$$

U 34. liniji koda uvodi se simbolička varijablu *u3*, koja je realan broj. Nakon što smo raspisali sve jednadžbe rješavamo ih kao jednadžbe s jednom nepoznanicom te dobijemo dva rezultata. I na samom kraju pomoću funkcije $2 * \text{atan}$ računamo kut.

```
32.     %% Inverse kinematics
33.     % theta3
34.     syms u3 'real';
35.     % substitution equations
36.     c3 = (1-u3^2)/(1+u3^2);
37.     s3 = 2 * u3 / (1+u3^2);
38.
39.     r = x^2+y^2+z^2;
40.     f1=a3*c3+a2;
41.     f2=a3*s3;
42.     k1=f1^2+f2^2-d1^2;
43.     %k1=a3^2+a2^2+2*a2*a3*c3-d1^2;
44.     u3 = double(solve(r-2*d1*z-k1));
45.     q33 = 2*atan(u3);
46.     q3=q33(2) ;
```

Od 47. do 62. linije koda računamo θ_2. U 48. liniji koda također kao i kod θ_3 uvodimo simboličku varijablu *u2*. U nastavku su ispisane formule koje se koriste za rješavanje kuta θ_2.

$$c_2 = \frac{(1-u_2^2)}{(1+u_2^2)}, \tag{4.2.2-6}$$

$$s_2 = \frac{2 * u_2^2}{(1+u_2^2)}, \tag{4.2.2-7}$$

$$c_3 = \cos(\theta_3), \tag{4.2.2-8}$$

$$s_3 = \sin(\theta_3), \tag{4.2.2-9}$$

$$k_2 = d_1 \tag{4.2.2-10}$$

$$\theta_2 = 2 * \text{atan}(u_2) \tag{4.2.2-11}$$

```
%% theta2
    syms u2 'real';
    % substitution equations
    c2 = (1-u2^2)/(1+u2^2);
    s2 = 2 * u2 / (1+u2^2);

    c3 = cos(q3);
    s3 = sin(q3);
    f1=a3*c3+a2;
    f2=a3*s3;
    k1=f1^2+f2^2-d1^2;
    k2=d1;

    u2 = double(solve(f1*s2+f2*c2+d1-z));
    q22 = 2*atan(u2);
    q2=q22(2);
```

Zadnji kut koji se računa za IK pozicioniranja je θ_1. U nastavku su dane formule korištene u programskom rješenju:

$$c_2 = \cos(\theta_2), \tag{4.2.2-12}$$

$$s_2 = \sin(\theta_2), \tag{4.2.2-13}$$

$$g_1 = c_2 f_1 - s_2 f_2 + a_1 \tag{4.2.2-14}$$

$$s_1 = \frac{y}{g_1} \tag{4.2.2-15}$$

$$c_1 = \frac{x}{g_1} \tag{4.2.2-16}$$

$$\theta_1 = atan2(s_1, c_1) \tag{4.2.2-16}$$

```
    %% theta1
    c2 = cos(q2);
    s2 = sin(q2);
    g1 = (c2*f1)-(s2*f2)+a1;
    s1= y/g1;
    c1= x/g1;
    q1=atan2(s1,c1);
```

Sljedeći dio koji se treba izračunati je vezan za direktnu kinematiku. Taj dio je objašnjen u poglavlju „*4.1.2 Programsko rješenje direktne kinematike*“. Nakon toga slijedi inverzna kinematika orijentacije. Potrebno je izračunati kut pod kojim će alat tj. grabilica robotskog manipulatora dohvatiti objekt. U našem slučaju kod inverzne kinematike orijentacije potrebno je izračunati θ_4 i θ_5.

```
% Position and angle of tool
    beta = pi/2;
    R06=[cos(beta)  -sin(beta)  0;
         sin(beta)  cos(beta)   0
         0          0          -1];
    R36=R03'*R06;

    q4=atan2(R36(1,3), -R36(2,3));
    q5=atan2(R36(3,1), -R36(3,2));

    q=[q1 q2 q3 q4 q5]';

    handles.q1 = q1;
    handles.q2 = q2;
    handles.q3 = q3;
    handles.q4 = q4;
    handles.q5 = q5;

    guidata(hObject,handles);  % save the updated handles structure
    set(handles.solution,'String',num2str(radtodeg(q)));
```

5. MAPIRANJE 2D PROSTORA S ULTRAZVUČNIM SENZOROM HC-SR04

Za dobivanje mape 2D prostora s ultrazvučnim senzorom potrebno je napraviti nekoliko mjerenja. To se može postići na način da senzor rotiramo oko njegove osi i prilikom rotacije senzor vrši mjerenja za svaki stupanj za koji je pomaknut, slika 5.1.

Sl. 5.1 Ultrazvučni senzor na servo motoru.[17]

Senzor sam po sebi je dosta neprecizan i ima kut širine ultrazvučnog signala od 15°. Kako bi smanjili pogrešku raditi će se više mjerenja i uzimati srednju vrijednost iz dobivenih mjerenja. Dakle, da pojednostavimo, servo motor će se gibati od 0° do 180°. Prilikom okretanja

servo motora senzor će na svakom stupnju raditi mjerenje i zapisivati to mjerenje u tablicu. Nakon što dođe do 180° vraćati će se u početni položaj 0° i također raditi mjerenje. Prilikom drugog mjerenja računati će srednju vrijednost prvog i drugog mjerenja te to spremati u tablicu.

5.1. Programsko rješenje za ultrazvučni senzor HC-SR04

Na samom početku potrebno je definirati nekoliko stvari koje se koriste u ovom programskom rješenju. To su:

- *a* - globalna varijabla u kojoj se nalazi objekt koji se stvara prilikom spajanja na Arduino,
- *Sensor* - globalna varijabla koja je vezana za biblioteku (eng. librarie) „JRodrigoTech/HCSR04“
- *servo7* - globalna varijabla koja je vezana za servo motor, te s njom možemo upravljati tim servo motorom.

```
1.  global a;
2.  global Sensor;
3.  global servo7;
4.  muveServo(0.5,0.45,0,0,0.5,0.7);
```

Ultrazvučni senzor u ovom radu je postavljen ispred robotske ruke. Prilikom mjerenja moramo osigurati da robotsku ruku postavimo u neutralan položaj u kojemu neće smetati senzoru. Zato smo u 4. liniji koda prvo pomaknuli robotsku ruku u neutralnu poziciju prije samog mjerenja. Nakon toga imamo dvije *for* petlje. U prvoj for petlji imamo brojač koji se iterativno pomjera od 0° do 180° za korak iteracije od 1° (6. linija koda). U svakoj iteraciji se pomjeri servo motor za 1 otkucaj (eng. *tick*) te napravi dva mjerenja u razmaku od 100 ms. Ta dva mjerenja se zbrajaju te se dijele s dva kako bi dobili udaljenost do predmeta. Dobiveni rezultat spremamo u polje *table* (14. linija koda).

```
5.  i = 1;
6.  table = zeros(180,2);
7.  for theta = 0 : 1/180 : 1
8.      writePosition(servo7, theta);
9.      dist1 = readTravelTime(Sensor)*170;
10.     pause(.01);
11.     dist2 = readTravelTime(Sensor)*170;
12.     dist = (dist1+dist2)/2;
13.     table(i,1) = (i-1);
14.     table(i,2) = round(dist * 100,2);
15.     i = i + 1;
16. end
```

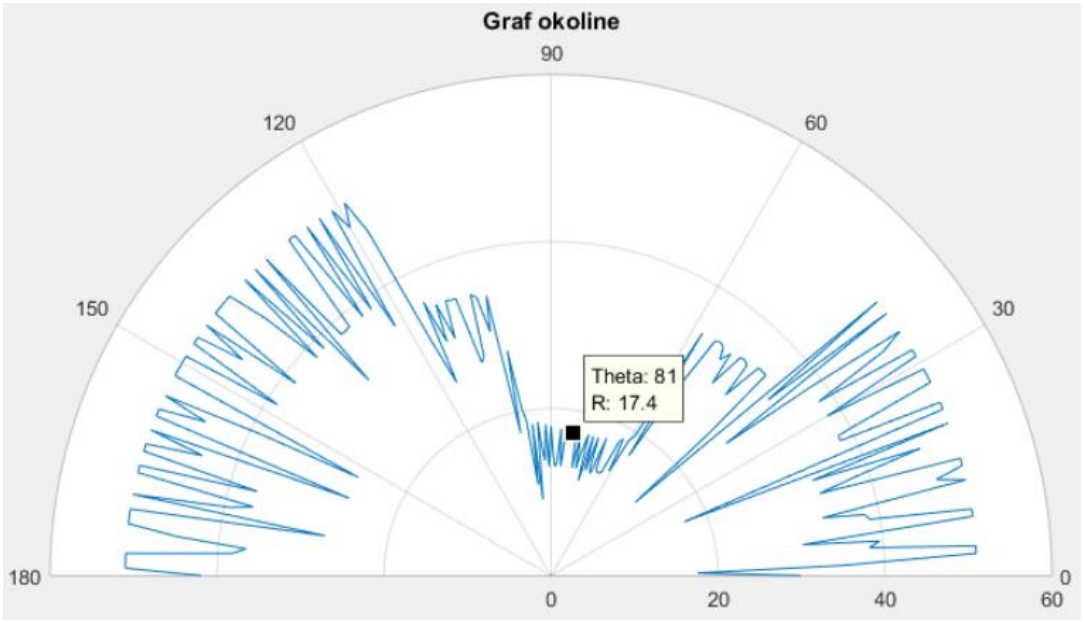
U drugoj for petlji radimo sličnu stvar samo što se servo motor sada pomjera u obrnutom smjeru tj. od 180 do 0. Također imamo dva mjerenja, ali razlika je prilikom spremanja rezultata.

Prilikom spremanja rezultata zbrajamo trenutni rezultat s rezultatom iz prošle *for* petlje i dijelimo s dva kako bi dobili srednju vrijednost (24. linija koda).

```
17. j = 1;
18. for theta = 1 : -1/180 : 0
19.     writePosition(servo7, theta);
20.     dist1 = readTravelTime(Sensor)*170;
21.     pause(.01);
22.     dist2 = readTravelTime(Sensor)*170;
23.     dist = (dist1+dist2)/2;
24.     table(i-j,2) = (table(i-j,2) + round(dist * 100,2))/2;
25.     j = j + 1;
26. end
```

Na samom kraju prikazujemo mjerenje korisniku s funkcijom polarplot() (27. linija koda). U 29. liniji koda smo funkciji rekli da nam prikaže rezultate u dva kvadranta. Razlog je taj što je mjerenje rađeno od 0° do 180° te pokrivamo tim mjerenjem samo dva kvadranta. Slika 5.1.1 prikazuje graf mjerenja udaljenost objekta od senzora.

```
27. polarplot (table(:,1)*pi/180, table (:,2));
28. title('Graf okoline');
29. thetalim([0 180]);
30. grid on;
```



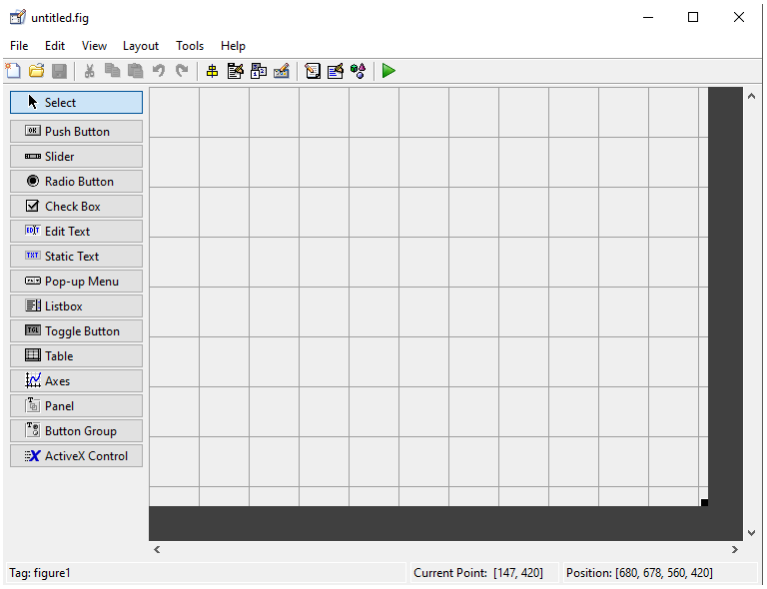
Sl. 5.1.1 Primjer mjerenja i prikaza rezultata

Na slici 5.1.1 je prikazan primjer mjerenja udaljenost objekta od senzora

6. GRAFIČKO KORISNIČKO SUČELJE I TESTIRANJE MANIPULATORA

6.1. MATLAB GUIDE razvojno okruženje

Grafičko sučelje za upravljanje robotskim manipulatorom je izrađeno u MATLAB-ovom alatu GUIDE razvojno okruženje (eng. *GUIDE development environment*). GUIDE pruža razne alate za dizajniranje korisničkog sučelja aplikacije. Jedan od takvih alata je GUIDE Layout Editor. Nakon što smo kreirali sučelje GUIDE automatski generira MATLAB kod za izradu korisničkog sučelja, što možemo lagano izmijeniti i doraditi za programiranje aplikacije. Za veću kontrolu nad projektiranjem i razvojem aplikacije možemo stvoriti MATLAB kod koji definira sva svojstva i ponašanja komponenti. MATLAB sadrži ugrađene funkcije koje omogućuju kreiranje grafičkog sučelja za dijaloške okvire, gumbove, klizače i polja za unos ili ispis i mnoge druge stvari.

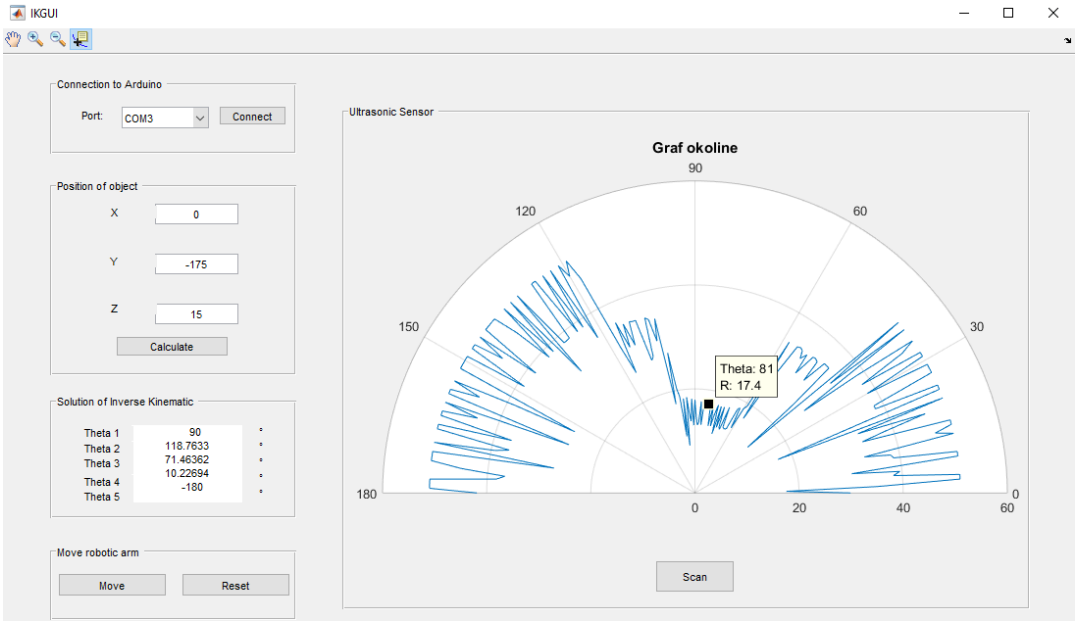


Sl. 6.1 Sučelje GUIDE

6.2. Programsko rješenje MATLAB GUIDE

Programsko rješenje grafičkog sučelja aplikacije (slika 6.2.1) omogućuje korisniku kontrolu nad robotskom rukom te prikaz rezultata. Aplikacija dozvoljava korisniku sljedeće stvari:

- spajanja na Arduino,
- polja za unos x, y *i* z kordinata,
- prozor za ispis rješenja inverzne kinematike,
- tipke za pomicanje i vraćanja ruke u početni položaj,
- tipka za aktiviranje senzora
- prozor za prikaz mjerenja sa senzora.



Sl. 6.2.1 Grafičkog sučelja aplikacije

6.2.1 Spajanje na Arduino

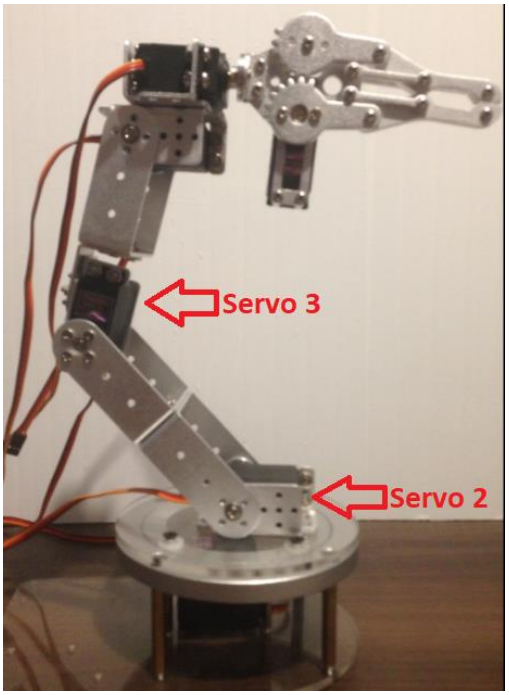
Spajanje na Arduino se vrši kroz padajući izbornik koji nam omogućava odabir USB ulaza (eng. *port*). Nakon što korisnik odabere ulaz i stisne tipku „*Connect*“ poziva se funkcija *connect_button_Callback.*

```
function connect_button_Callback(hObject, eventdata, handles)
popStrings = handles.connect_popupmenu.String; % All the strings in the menu.
selectedIndex = handles.connect_popupmenu.Value;
selectedString = popStrings{selectedIndex};

delete(instrfind({'PORT'},{selectedString}));
clear a;
clear servo1; clear servo2; clear servo3;
global a;
clc;
a = arduino(selectedString, 'Mega2560', 'Libraries', {'JRodrigoTech/HCSR04', 'Servo'
});
global servo1;
servo1 = servo(a, 'D2');
global servo2;
servo2 = servo(a, 'D3');
global servo3;
servo3 = servo(a, 'D4');
global servo4;
servo4 = servo(a, 'D5');
global servo5;
servo5 = servo(a, 'D6');
global servo6;
servo6 = servo(a, 'D7');

global servo7;
servo7 = servo(a, 'D11');
global Sensor;
Sensor = addon(a, 'JRodrigoTech/HCSR04', 'D12', 'D13');
msgbox('Arduino successfully connected','Connected');
```

Od 2. do 4. linije koda dohvaćamo vrijednost koju je korisnik odabrao u padajućem izborniku prije nego što je pritisnuo tipku „*Connect*“. Od 6. do 10. brišemo sve vrijednosti iz varijabli i osiguravamo nesmetano spajanje na Arduino. Nakon toga možemo napraviti povezivanje s Arduinom. Prilikom spajanja na Arduino kreiramo objekt *a* (11. linija koda). Od 11. do 26. linije koda kreiramo objekte koji služe za kontrolu servo motora, te svakom motoru dodjeljujemo odgovarajući digitalni pin s Arduina. Slična stvar se radi i s ultrazvučnim senzorom u 28. liniji koda. Razlika je ta što kod senzora moramo dati digitalni pin za echo odnosno trig.

6.3. Testiranje robotskog manipulatora

Kao što je na samom početku ovog rada naglašeno, robotski manipulator korišten u ovom radu je za edukacijske svrhe te nema veliku preciznost. Najveći problem predstavljaju servo motori. Točnije servo motor na drugom i trećem zglobu (slika 6.3.1) rade najveću grešku zbog opterećenja koja djeluju na njih.



Sl. 6.3.1 Robotski manipulator

Ta greška je uzrokovana zbog zračnosti koja se pojavljuje na osovini servo motora, te dozvoljava robotskoj ruci hod od desetak stupnjeva ovisno u kojem položaju se ruka nalazi, te se ona gomila kroz cijeli kinetički lanac. Da bi kompenzirali tu grešku nakon što izračunamo inverznu kinematiku, oduzimamo 10° od proračuna te tu vrijednost predajemo servo motoru

dva i tri. To možemo vidjeti u 4. i 12. liniji koda. Time smo kompenzirali grešku kada robotski manipulator dohvaća neki predmet.

```
1.  %Servo2
2.  if isfield(handles,'q2')
3.      q2 = handles.q2;
4.      q2=radtodeg(q2)-10;
5.      q2=180/q2;
6.      q2=1/q2;
7.  end
8.
9.  %Servo3
10. if isfield(handles,'q3')
11.     q3 = handles.q3;
12.     q3=radtodeg(q3)-10;
13.     q3=180/q3;
14.     q3=1/q3;
15. end
```

7. ZAKLJUČAK

Kroz izradu rada stečena su mnoga znanja u području robotike i primjenu matematičkih modela, računanja kinematike i geometrije, elektronike, korištenja mikroprocesora sa senzorima i servo motorima, te mnoge druge stvari. Robotika je vrlo kompleksno područje te iziskuje mnogo istraživanja i širokog znanja u mnogim sferama. Cilj ovog diplomskog rada je bio sastaviti robotski manipulator, riješiti inverznu kinematiku, napisati algoritam za upravljanje te koristiti ultrazvučni senzor za mapiranje 2D prostora. Za potrebe ovog rada korišten je robotski manipulator s pet stupnjeva slobode gibanja. Kinematički problem je riješen s inverznom i direktnom kinematikom. Kod direktne kinematike korištena je Denavit-Hartenbergova metoda. Problem inverzne kinematike je riješen s Pieperovim rješenjem. Programsko rješenje cjelokupnog rada je rađeno u programskom paketu MATLAB koji je komunicirao s Arduinom mega 2560. Prilikom upravljanja robotskom rukom uočeni su mnogi problemi. Jedan od njih je bio i zračnost u zupčanicima servo motora. Ultrazvučni senzor u ovom radu je korišten za 2D mapiranje prostora. Cilj je bio s ultrazvučnim senzorom koji je pričvršćen na servo motor dobiti približno poziciju objekta kojeg robotski manipulator treba dohvatiti. Senzor se u radu pokazao dosta neprecizan zbog velikog kuta bacanja ultrazvučnog signala. Mjesta za poboljšanja svakako ima, pogotovo u području ultrazvučnog senzora. Servo motori korišteni u ovom radu su se pokazali zadovoljavajući ali svakako korištenje boljih bi dalo puno bolje rezultate.

LITERATURA

[1] http://people.etf.unsa.ba/~jvelagic/laras/dok/Robotika_uvod.pdf

[2] http://rees52.com/75-large_default/ultrasonic-sensor-module-hc-sr-04.jpg

[3] https://e-radionica.com/hr/blog/2015/08/19/kkm-ultrazvucni-modul-hc-sr04/

[4] www.otpornik.com/elektronika/komponente/hc-sr04-ultrazvucni-senzor-rastojanja.html

[4]http://www.otpornik.com/elektronika/komponente/hc-sr04-ultrazvucni-senzor-rastojanja.html

[5] https://e-radionica.com/hr/servo-motor-towerpro-mg995.html

[6] https://e-radionica.com/hr/blog/2016/06/23/kkm-servo-motor-tower-pro-sg90/

[7] https://en.wikipedia.org/wiki/Arduino

[8] https://www.arduino.cc/en/Guide/Introduction

[9] https://www.arduino.cc/en/Main/arduinoBoardMega

[10] https://bib.irb.hr/datoteka/776579.diplomski_Zavrna_verzija_DinoCarFESB.pdf

[11] https://loomen.carnet.hr/pluginfile.php/312925/mod_resource/content/2/PRED/OIRS_PR.pdf

[12] https://loomen.carnet.hr/pluginfile.php/312928/mod_resource/content/1/PRED/OR_PR_03_public.pdf

[13] http://imgs.inkfrog.com/pix/thanksbuyer/27247-1_0004.jpg

[14] https://bib.irb.hr/datoteka/586104.InteraktivniInverzniKinematickiModel.pdf

[15] http://slideplayer.com/slide/6850449/

[16] http://www.ss-strukovna-djurdjevac.skole.hr/dokumenti?dm_document_id=746&dm_dnl=1

[17] https://www.robotistan.com/ultrasonic-sensor-mount-device-a-type-compatible-with-servo-10510-44-O.jpg

SAŽETAK

Naslov: Robotska manipulacija s navođenjem pomoću ultrazvučnog senzora

Sažetak: U ovom radu opisan je sustav koji se sastoji od robotskog manipulatora, mikroprocesora i senzora. Riješen je matematički model robotskog manipulatora kroz direktnu odnosno inverznu kinematiku. Direktna kinematika je riješena preko Denavit-Hartenbergove metode. Inverzna kinematika rješena je Pieperovim rješenjem. Također su prikazana programska rješenja za direktnu i inverznu kinematiku. Korišten je ultrazvučni senzor za mapiranje 2D prostora te je opisana problematika i prikazano programsko rješenje. Cijeli programski kod je napisan u programskom jeziku MATLAB. U radu je također opisan i korišten mikroprocesor Arduino Mega 2560. Grafičko sučelje je napravljeno kroz GUIDE pomoću MATLABA te je sve spojeno s mikroprocesorom koji omogućava upravljanje robotskim manipulatorom.

Ključne riječi: robotski manipulator, direktna kinematika, Denavit-Hartenbergove metoda, inverzna kinematika, Pieperovo rješenje, MATLAB, Arduino Mega 2560, ultrazvučni senzor.

ABSTRACT

Title: Robot manipulation guided by ultrasonic sensor

Abstract: A system consisting of a robot manipulator, microprocessor and sensor is described. The mathematical model of the robotic manipulators was solved using direct and inverse kinematics. Direct kinematics has been solved using the Denavit-Hartenberg method and inverse kinematics by Pieper's solution. Program solutions for direct and inverse kinematics are also described. An ultrasonic sensor is used for 2D mapping of the robot's environment. The problem of 2D mapping by ultrasonic sensor is described and a program solution si presented. The full program code is written in MATLAB programming language. The Arduino Mega 2560 microprocessor is also described and used in this work. A graphical interface is created using GUIDE tool of MATLAB and it is connected to the microprocessor which controls the robot manipulator.

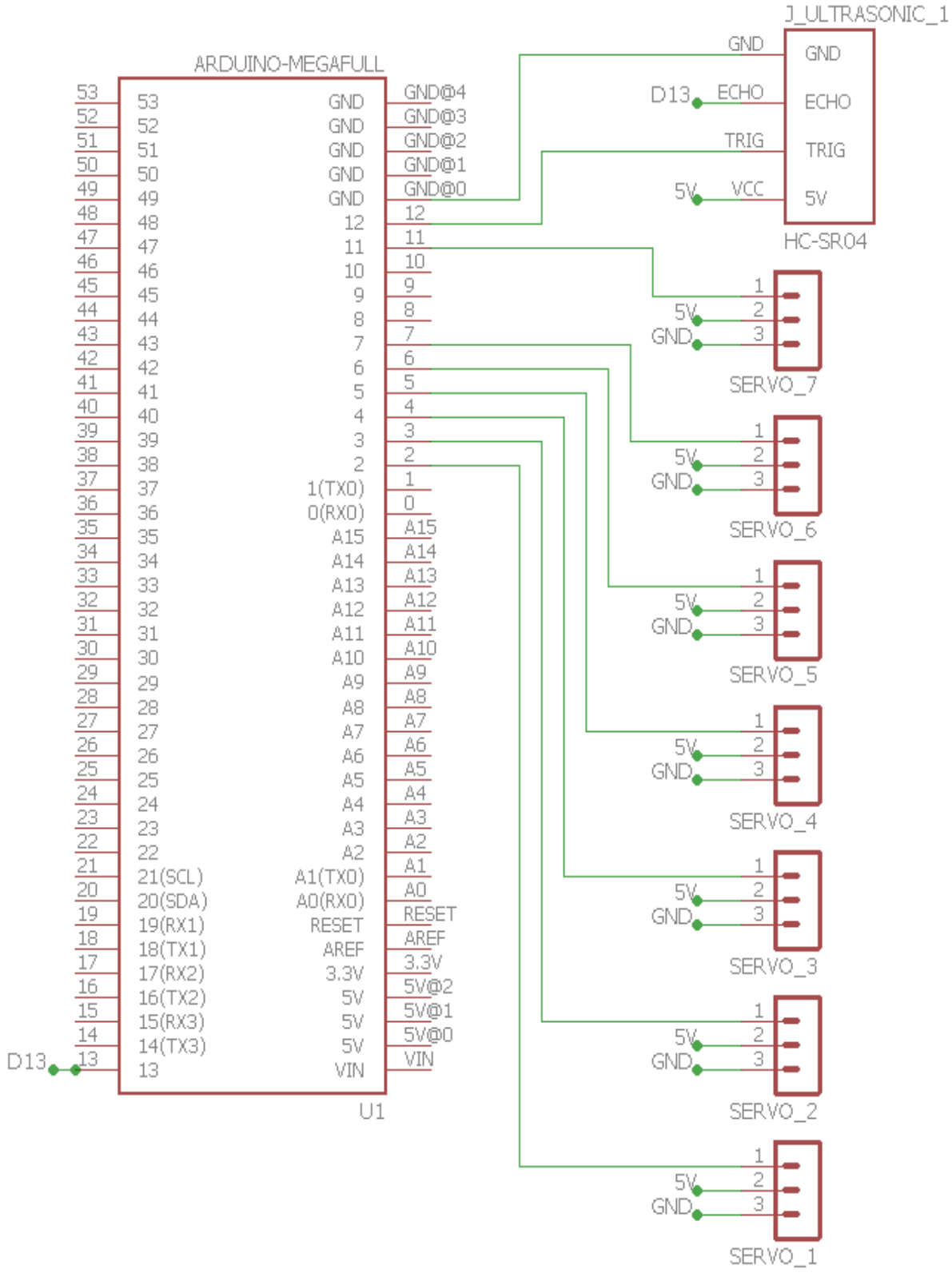
Keywords: robot manipulator, direct kinematics, Denavit-Hartenberg method, inverse kinematics, Pieper's solution, MATLAB, Arduino Mega 2560, ultrasonic sensor

ŽIVOTOPIS

Aldin Ćebo rođen je 21.04.1991. u Sarajevu. Živi u Osijeku. Osnovnu školu polazio je u Dardi nakon koje upisuje srednju školu u Osijeku smjer Tehničar za mehatroniku. Srednjoškolsko obrazovanje završava 2010. i iste godine upisuje Elektrotehnički fakultet u Osijeku, Stručni studij smjer informatika koji završava 2014. Trenutno je student 2. godine diplomskog studija računalno inženjerstvo na Fakultetu elektrotehnike računarstva i informacijskih tehnologija u Osijeku. Od stranih jezika koristi se engleskim.

PRILOZI

Shema sklopa:



```
function varargout = IKGUI(varargin)
% IKGUI MATLAB code for IKGUI.fig
%      IKGUI, by itself, creates a new IKGUI or raises the existing
%      singleton*.
%
%      H = IKGUI returns the handle to a new IKGUI or the handle to
%      the existing singleton*.
%
%      IKGUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in IKGUI.M with the given input arguments.
%
%      IKGUI('Property','Value',...) creates a new IKGUI or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before IKGUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to IKGUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help IKGUI

% Last Modified by GUIDE v2.5 02-Nov-2017 18:55:11

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @IKGUI_OpeningFcn, ...
                   'gui_OutputFcn',  @IKGUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before IKGUI is made visible.
function IKGUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to IKGUI (see VARARGIN)

% Choose default command line output for IKGUI
handles.output = hObject;

% Update handles structure
```

```
guidata(hObject, handles);

% UIWAIT makes IKGUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = IKGUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function x_Callback(hObject, eventdata, handles)
% hObject    handle to x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of x as text
%        str2double(get(hObject,'String')) returns contents of x as a
double


% --- Executes during object creation, after setting all properties.
function x_CreateFcn(hObject, eventdata, handles)
% hObject    handle to x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function y_Callback(hObject, eventdata, handles)
% hObject    handle to y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of y as text
%        str2double(get(hObject,'String')) returns contents of y as a
double


% --- Executes during object creation, after setting all properties.
function y_CreateFcn(hObject, eventdata, handles)
% hObject    handle to y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function z_Callback(hObject, eventdata, handles)
% hObject    handle to z (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of z as text
%        str2double(get(hObject,'String')) returns contents of z as a
double


% --- Executes during object creation, after setting all properties.
function z_CreateFcn(hObject, eventdata, handles)
% hObject    handle to z (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in calculate.
function calculate_Callback(hObject, eventdata, handles)
% hObject    handle to calculate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
x = str2num(char(get(handles.x,'String')));
y = str2num(char(get(handles.y,'String')));
z = str2num(char(get(handles.z,'String')));
if isempty(x) || x<-217 || x>217
    msgbox('The value X must be between -217 and 217','Error','error');
elseif isempty(y) || y<-217 || y>0
    msgbox('The value Y must be between -217 and 0','Error','error');
elseif isempty(z) || z<0 || z>40
    msgbox('The value Z must be between 0 and 40','Error','error');
else
    %% Denavit-Hartenberg parameters
    d1=75;      d2=0;    d3=0;
    a1=0;       a2=105;  a3=96;
    alfa1=pi/2; alfa2=0; alfa3=pi;
    %% Pythagoras theorem
    a=abs(x); b=abs(y);
    c=sqrt(a^2+b^2);
    alpha=atand(a/b);
    c_=c-30;
    a_=sind(alpha)*c_;
    b_=sqrt(c_^2-a_^2);
    if x>0
```

```
    x=a_;
    y=-b_;
    z=z+135;
else
    x=-a_;
    y=-b_;
    z=z+135;
end
%% Inverse kinematics
% theta3
syms u3 'real';
% substitution equations
c3 = (1-u3^2)/(1+u3^2);
s3 = 2 * u3 / (1+u3^2);

r = x^2+y^2+z^2;
f1=a3*c3+a2;
f2=a3*s3;
k1=f1^2+f2^2-d1^2;
%k1=a3^2+a2^2+2*a2*a3*c3-d1^2;
u3 = double(solve(r-2*d1*z-k1));
q33 = 2*atan(u3);
q3=q33(2) ;

%% theta2
syms u2 'real';
% substitution equations
c2 = (1-u2^2)/(1+u2^2);
s2 = 2 * u2 / (1+u2^2);

c3 = cos(q3);
s3 = sin(q3);
f1=a3*c3+a2;
f2=a3*s3;
k1=f1^2+f2^2-d1^2;
k2=d1;

u2 = double(solve(f1*s2+f2*c2+d1-z));
q22 = 2*atan(u2);
q2=q22(2);

%% theta1
c2 = cos(q2);
s2 = sin(q2);
g1 = (c2*f1)-(s2*f2)+a1;
s1= y/g1;
c1= x/g1;
q1=atan2(s1,c1);

%%
% Direct kinematics
H01 = [cos(q1) -sin(q1)*cos(alfa1) sin(q1)*sin(alfa1)  a1*cos(q1);
       sin(q1)  cos(q1)*cos(alfa1) -cos(q1)*sin(alfa1) a1*sin(q1);
       0        sin(alfa1)         cos(alfa1)          d1;
       0        0                  0                   1
];

H12 = [cos(q2) -sin(q2)*cos(alfa2) sin(q2)*sin(alfa2)  a2*cos(q2);
       sin(q2)  cos(q2)*cos(alfa2) -cos(q2)*sin(alfa2) a2*sin(q2);
       0        sin(alfa2)         cos(alfa2)          d2;
```

```
          0        0                   0                   1
    ];

    H23 = [cos(q3) -sin(q3)*cos(alfa3) sin(q3)*sin(alfa3)  a3*cos(q3);
          sin(q3)  cos(q3)*cos(alfa3)  -cos(q3)*sin(alfa3) a3*sin(q3);
          0        sin(alfa3)          cos(alfa3)          d3;
          0        0                   0                   1
    ];

    H03=H01*H12*H23;
    R03=H03(1:3,1:3);

    % Position and angle of tool
    beta = pi/2;
    R06=[cos(beta)  -sin(beta)  0;
         sin(beta)  cos(beta)   0
         0          0          -1];
    R36=R03'*R06;

    q4=atan2(R36(1,3), -R36(2,3));
    q5=atan2(R36(3,1), -R36(3,2));

    q=[q1 q2 q3 q4 q5]';

    handles.q1 = q1;
    handles.q2 = q2;
    handles.q3 = q3;
    handles.q4 = q4;
    handles.q5 = q5;

    guidata(hObject,handles);  % save the updated handles structure
    set(handles.solution,'String',num2str(radtodeg(q)));
end

function solution_Callback(hObject, eventdata, handles)
% hObject    handle to solution (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of solution as text
%        str2double(get(hObject,'String')) returns contents of solution as
a double


% --- Executes during object creation, after setting all properties.
function solution_CreateFcn(hObject, eventdata, handles)
% hObject    handle to solution (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in move_robotic_arm.
```

```
function move_robotic_arm_Callback(hObject, eventdata, handles)
% hObject    handle to move_robotic_arm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Servo3
if isfield(handles,'q3')
    q3 = handles.q3;
    q3=radtodeg(q3)-10;
    q3=180/q3;
    q3=1/q3;
end

%Servo4
if isfield(handles,'q4')
    q4 = handles.q4;
    q4=radtodeg(q4);
    q4=180/q4;
    q4=1/q4;
    if q4<0
        q4=0;
    end
end

%Servo5
if isfield(handles,'q5')
    q5 = handles.q5;
    q5=radtodeg(q5);
    q5=180/q5;
    q5=1/q5;
    if q5<0
        q5=0;
    end
end

%Servo6
q6 = 0.7;
q6=radtodeg(q6);
q6=180/q6;
q6=1/q6;

% Servo 1.
if isfield(handles,'q1')
    q1 = handles.q1;
    q1=radtodeg(q1);
    q1=180/q1;
    q1=1/q1;
end

%Servo2
if isfield(handles,'q2')
    q2 = handles.q2;
    q2=radtodeg(q2)-10;
    q2=180/q2;
    q2=1/q2;
end
muveServo(q1,q2,q3,q4,q5,q6);

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes1


% --- Executes on button press in reset_button.
function reset_button_Callback(hObject, eventdata, handles)
% hObject    handle to reset_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

muveServo(0.5,0.45,0.4,0,0.5,0.5);

% --- Executes on button press in scan_button.
function scan_button_Callback(hObject, eventdata, handles)
% hObject    handle to scan_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global a;
global Sensor;
global servo7;
muveServo(0.5,0.45,0,0,0.5,0.7);

i = 1;
table = zeros(180,2);
for theta = 0 : 1/180 : 1
    writePosition(servo7, theta);
    dist1 = readTravelTime(Sensor)*170;
    pause(.01);
    dist2 = readTravelTime(Sensor)*170;
    dist = (dist1+dist2)/2;
    table(i,1) = (i-1);
    table(i,2) = round(dist * 100,2);
    i = i + 1;
end

j = 1;
for theta = 1 : -1/180 : 0
    writePosition(servo7, theta);
    dist1 = readTravelTime(Sensor)*170;
    pause(.01);
    dist2 = readTravelTime(Sensor)*170;
    dist = (dist1+dist2)/2;
    table(i-j,2) = (table(i-j,2) + round(dist * 100,2))/2;
    j = j + 1;
end

polarplot (table(:,1)*pi/180, table (:,2));
title('Graf okoline');
thetalim([0 180]);
grid on;

% --- Executes on button press in connect_popupmenu.
function connect_popupmenu_Callback(hObject, eventdata, handles)
% hObject    handle to connect_popupmenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% --- Executes during object creation, after setting all properties.
function connect_popupmenu_CreateFcn(hObject, eventdata, handles)
% hObject    handle to connect_popupmenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes on button press in connect_button.
function connect_button_Callback(hObject, eventdata, handles)
% hObject    handle to connect_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Arduino connection
popStrings = handles.connect_popupmenu.String; % All the strings in the
menu.
selectedIndex = handles.connect_popupmenu.Value;
selectedString = popStrings{selectedIndex};

delete(instrfind({'PORT'},{selectedString}));
clear a;
clear servo1; clear servo2; clear servo3;
global a;
clc;
a = arduino(selectedString, 'Mega2560', 'Libraries',
{'JRodrigoTech/HCSR04', 'Servo'});
global servo1;
servo1 = servo(a, 'D2');
global servo2;
servo2 = servo(a, 'D3');
global servo3;
servo3 = servo(a, 'D4');
global servo4;
servo4 = servo(a, 'D5');
global servo5;
servo5 = servo(a, 'D6');
global servo6;
servo6 = servo(a, 'D7');

global servo7;
servo7 = servo(a, 'D11');
global Sensor;
Sensor = addon(a, 'JRodrigoTech/HCSR04', 'D12', 'D13');
msgbox('Arduino successfully connected','Connected');

function muveServo(q1, q2, q3, q4, q5, q6);
global a;
global Sensor;
global servo7;
global servo1; global servo2; global servo3; global servo4; global servo5;
global servo6;

%Servo3
postionServo3 = readPosition(servo3);
if postionServo3 < q3
    for theta = postionServo3 : 1/180 : q3
        writePosition(servo3, theta);
        pause(.04);
    end
```

```
elseif postionServo3 > q3
    for theta = postionServo3 : -1/180 : q3
        writePosition(servo3, theta);
        pause(.04);
    end
end
writePosition(servo3, q3);

%Servo4
postionServo4 = readPosition(servo4);
if postionServo4 < q4
    for theta = postionServo4 : 1/180 : q4
        writePosition(servo4, theta);
        pause(.04);
    end
elseif postionServo4 > q4
    for theta = postionServo4 : -1/180 : q4
        writePosition(servo4, theta);
        pause(.04);
    end
end
writePosition(servo4, q4);

%Servo5
postionServo5 = readPosition(servo5);
if postionServo5 < q5
    for theta = postionServo5 : 1/180 : q5
        writePosition(servo5, theta);
        pause(.04);
    end
elseif postionServo5 > q5
    for theta = postionServo5 : -1/180 : q5
        writePosition(servo5, theta);
        pause(.04);
    end
end
writePosition(servo5, q5);

%Servo6
postionServo6 = readPosition(servo6);
if postionServo6 < q6
    for theta = postionServo6 : 1/180 : q6
        writePosition(servo6, theta);
        pause(.04);
    end
elseif postionServo6 > q6
    for theta = postionServo6 : -1/180 : q6
        writePosition(servo6, theta);
        pause(.04);
    end
end
writePosition(servo6, q6);

% Servo 1.
postionServo1 = readPosition(servo1);
if postionServo1 < q1
    for theta = postionServo1 : 1/180 : q1
        writePosition(servo1, theta);
        pause(.04);
    end
elseif postionServo1 > q1
```

```
    for theta = postionServo1 : -1/180 : q1
        writePosition(servo1, theta);
        pause(.04);
    end
end
writePosition(servo1, q1);

%Servo2
postionServo2 = readPosition(servo2);
if postionServo2 < q2
    for theta = postionServo2 : 1/180 : q2
        writePosition(servo2, theta);
        pause(.04);
    end
elseif postionServo2 > q2
    for theta = postionServo2 : -1/180 : q2
        writePosition(servo2, theta);
        pause(.04);
    end
end
writePosition(servo2, q2);
```