SVEUČILIŠTE U ZAGREBU

**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

ZAVRŠNI RAD br. 1913

# Kinematički model čovjeka

Željko Mijočević

Zagreb, lipanj 2011.

Zahvaljujem se mentorici prof. dr. sc. Željki Mihajlović na savjetima i pruženoj pomoći prilikom pisanja ovog rada. Posebno se zahvaljujem roditeljima Marinku i Jelki, te sestri Ani i bratu Ivi na pruženoj podršci tokom dosadašnjeg studija.

## Sadržaj

# 1. Uvod

Korištenjem računalne grafike i animacija u raznim područjima (znanstvena istraživanja, filmska industrija, razvoj igara), došlo je do potrebe za što stvarnijim i preciznijim prikazivanjem ljudskog lika te njegovih kretnji u digitalnom okruženju. U tu svrhu počele su se koristiti kinematičke strukture, koje predstavljaju skup elemenata („kosti“) međusobno povezanih u spojnim točkama („zglobovi“) koji se mogu translatirati i rotirati te na taj način simulirati kretnje. Također se kinematička kontrola, bila ona direktna ili inverzna, pokazala kao uspješna tehnika za interaktivno pozicioniranje i animiranje kompleksnih artikuliranih struktura.

U ovom radu prvo ćemo se pozabaviti općenitim opisom izgradnje kinematičke strukture (hijerarhijsko stablo elemenata – kostiju i zglobova) koja predstavlja grubi kostur čovjeka. Uz to budu predložene te razrađene i razne metode upravljanja strukturom, kako direktne tako i inverzne kinematike. Direktna kinematika se odnosi na zadavanje kutova pokretnih elemenata unutar hijerarhijskog modela preko kojih se određuje položaj elemenata. Inverzna kinematika rješava problem kad je poznata ciljna pozicija, te je na osnovu nje potrebno odrediti kutove elemenata.

Nakon toga ćemo opisati implementaciju kinematičke strukture i nekih metoda kinematike korištenjem programskog jezika C++ i grafičkog standarda OpenGL. Na kraju ćemo uz par primjera objasnit utjecaj određenih parametara i korištenih metoda kinematike na ishod te izgled animiranja kinematičkog modela čovjeka.

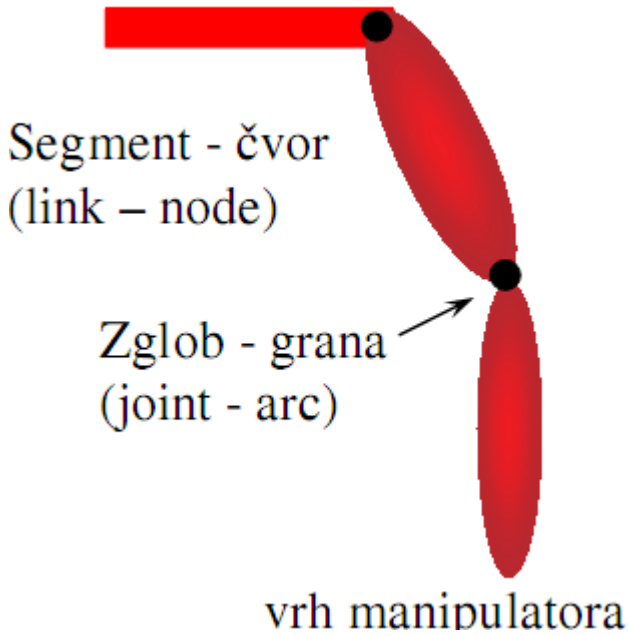
# 2. Kinematička struktura

Kinematika (od grč. *κινεῖν* – *kinein*, hrv. pomicati) prvenstveno predstavlja granu klasične mehanike koja opisuje gibanje tijela (objekata) i sustava (grupe objekata), bez razmatranja sila koje uzrokuju gibanje. Kinematiku ne treba miješati s drugom granom klasične mehanike – analitičkom dinamikom (proučavanje odnosa između gibanja objekata i sila koje su to gibanje prouzrokovale), koja je ponekad podijeljena na kinetiku (proučavanje odnosa između vanjskih sila i gibanja objekta) i statiku (proučavanje odnosa u sustavu u ravnoteži).

Pojam kinematika je preuzet i u terminologiju robotike te preko nje i u terminologiju računalne grafike. Ona predstavlja područje grafike koje se bavi oblikovanjem i kretanjem pokretnih modela, tj. modela koji imaju uz osnovne statičke specifikacije (izgled) definirana i mjesta povezivanja, ograničenja kretnji, stupnjeve slobode itd. što predstavlja kinematičku strukturu.

## 2.1 Kinematički lanci

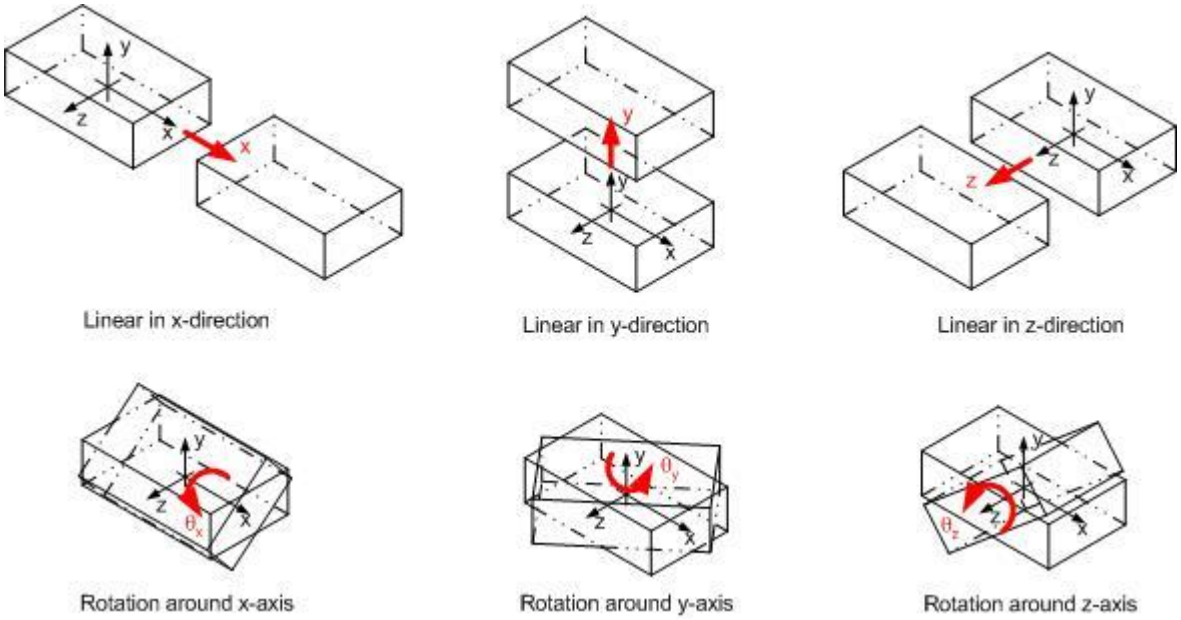
Kinematički lanac predstavlja sustav sačinjen od krutih elemenata (segmenti – čvorovi, eng. link - node), koji su međusobno povezani zglobovima (granama, eng. joint – arc). Pri tome zglobovi mogu imati različite stupnjeve slobode (eng. DOF – „Degrees of freedom“). Kinematički lanac se sastoji od barem tri člana međusobno povezanih zglobovima koji su međusobno u hijerarhijskom odnosu (slika 2.1).




Slika 2.1 Primjer kinematičkog lanca

### 2.1.1 Stupanj slobode (eng. DOF – „Degrees of freedom“)

Svaki zglob ima definiran stupanj slobode. Postoje dvije vrste stupnjeva slobode: rotacijski i translacijski. Kao što sami nazivi i govore rotacijski DOF omogućava rotiranje oko određenih osi, dok translacijski DOF omogućava pomicanje po određenim osima. Kako u Kartezijevom koordinatnom sustavu postoje tri osi (X, Y, Z) moguće je raspodijeliti rotacijske, odnosno translacijske stupnjeve slobode na tri tipa, u ovisnosti o tome u kojem smjeru je rotacija odnosno pokret moguć (slika 2.2).



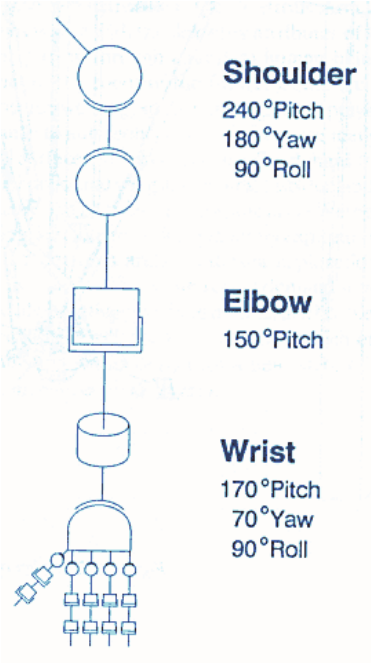

Slika 2.2 Rotacijski i translacijski stupnjevi slobode

Navedeni stupnjevi slobode se mogu opisati na sljedeći način (slika 2.2):

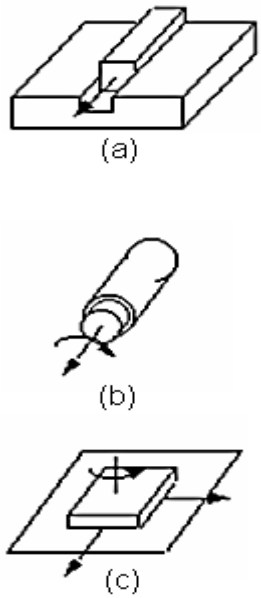
- translacija po x – osi predstavlja pomicanje lijevo – desno (eng. swaying)
- translacija po y – osi predstavlja pomicanje gore – dolje (eng. heaving)
- translacija po z- osi predstavlja pomicanje naprijed – nazad (eng. surging)
- rotacija oko x –osi predstavlja naginjanje naprijed – nazad (eng. pitching)
- rotacija oko y – osi predstavlja zakretanje lijevo – desno (eng. yawning)
- rotacija oko z – osi predstavlja naginjanje u stranu (eng. rolling)

Oni su najjednostavniji oblici te omogućuju rotaciju odnosno translaciju oko samo jedne osi. Zglobovi s jednim stupnjem slobode se zajednički označavaju s 1DOF(eng. „one degree of freedom“). No, povezivanjem više 1DOF zglobova sa segmentima veličine nula, nastaju takozvani *n*-DOF zglobovi. *n* u nazivu,

predstavlja broj omogućenih rotacija i translacija po odabranim osima. Kao primjer možemo uzeti ruku koja sadrži tri zgloba – rame, lakat i ručni zglob (slika 2.3)




Slika 2.3 Primjer rotacijskih n-DOF zglobova

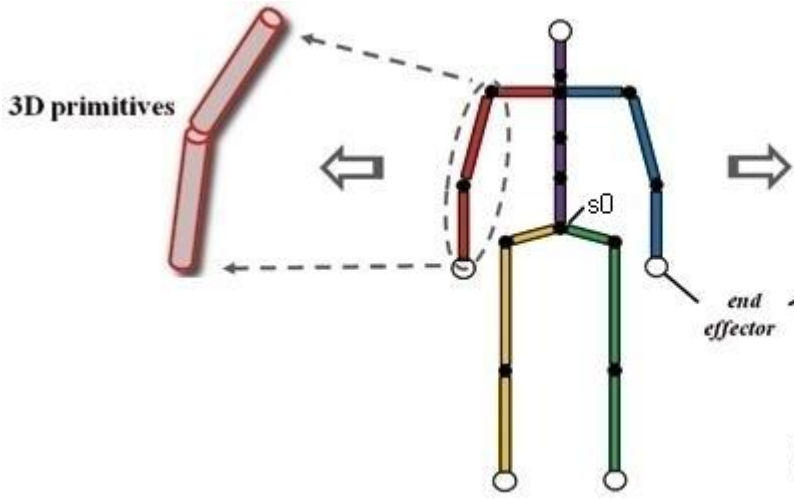



Slika 2.4 Primjer translacijskih n-DOF zglobova

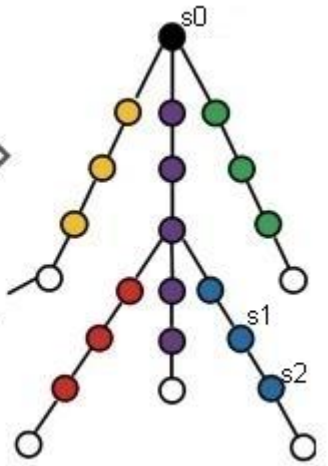
Rame (eng. shoulder) predstavlja rotacijski 3DOF zglob, dok ručni zglob (eng. wrist) predstavlja zglob s 2 rotacijska stupnja slobode, što znači da se mogu rotirati oko sve tri osi (x, y i z) odnosno dvije osi (x i y). Lakat (eng. elbow) predstavlja rotacijski 1DOF zglob, što znači da se može rotirati samo oko jedne osi (x – osi). Na slici 2.4 možemo vidjeti redom primjer translacijskog 1DOF zgloba (a), povezanog translacijskog 1DOF i rotacijskog 1DOF zgloba (b) te povezanog translacijskog 2DOF i rotacijskog 1DOF zgloba (c).

### 2.2 Hijerarhijski kinematički model

Hijerarhijski model se gradi od skupa međusobno povezanih jednostavnih kinematičkih lanaca. U početku izgradnje potrebno je napraviti apstraktnu reprezentaciju objekta koji se želi prikazati (slika 2.5). Apstraktnu reprezentaciju sačinjavaju svi segmenti modela, koji su međusobno povezani sa zglobovima. Iz apstraktnog modela se onda sačinjava graf stablaste strukture (slika 2.6).




Slika 2.5 Apstraktna reprezentacija




Slika 2.6 Stablasta struktura

Graf stablaste strukture se izrađuje na način da se prvo definira korijenski čvor (eng. root). Svi ostali čvorovi se postepeno pozicioniraju u odnosu na korijenski čvor pomoću niza transformacija.

Na slici 2.6 korijenski čvor je označen crnim krugom (*s0*). On predstavlja karlicu (eng. pelvis) čovjeka. Iz njega se izvode drugi segmenti kao npr. segment nadlaktice koji je predstavljen segmentom *s1*. Nadalje se, iz nadlaktice izvodi segment podlaktice koji je predstavljen segmentom *s2*. Važna osobina ovakvog načina povezivanja segmenata je što segmenti izvedeni iz roditeljskog segmenta ovise o poziciji i obliku segmenta roditelja (*s0*). Na taj način segment *s1* ovisi o segmentu *s0*, tj. nadlaktica ovisi o poziciji i obliku karlice, te segment *s2* ovisi o segmentu *s1* te preko njega i o segmentu *s0*, tj. podlaktica ovisi o položaju i obliku segmenata nadlaktice odnosno karlice. Zbog ove pojave je moguće sve čvorove prikazati preko korijenskog čvora uz nizanje transformacija (slika 2.7).

Ako se svaki segment promatra u svom lokalnom koordinatnom sustavu, transformacije za navedene segmente s0, s1 i s2 se mogu zapisati po uzoru na sliku 2.7.

Pozicija segmenta *s0* se nakon translacijskog pozicioniranja $T_0$ može matematički zapisati preko formule

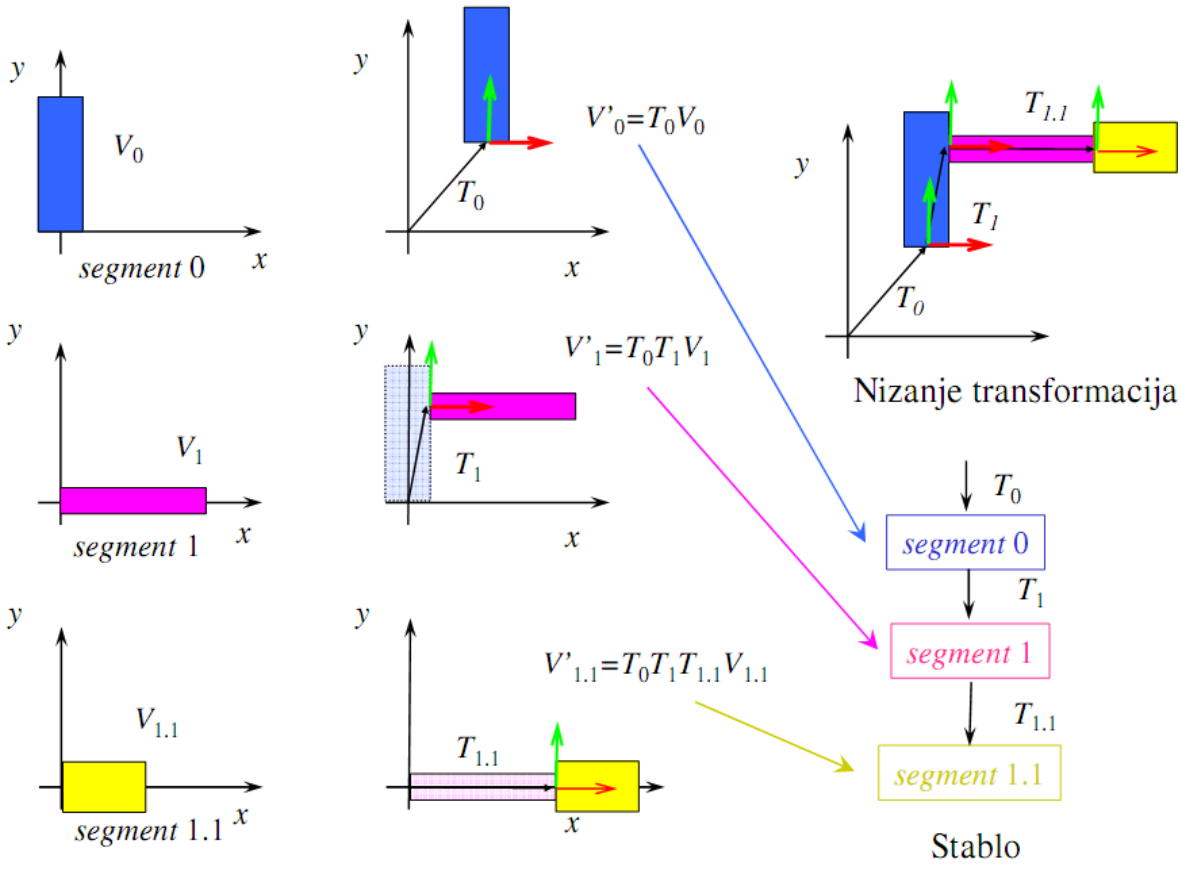
$$V'_0 = T_0 V_0 \quad (1)$$

Poziciju segmenta *s1* određuju translacije nadređenih segmenata iz kojih je izveden tj. translacije segmenta *s0* te vlastita translacija

$$V'_1 = T_0 T_1 V_1 \qquad (2)$$

Na isti način se određuje i pozicija krajnjeg segmenta *s2* koji kao nadređene segmente ima segmente *s0* i *s1* iz čega slijedi formula

$$V'_2 = T_0 T_1 T_{1.1} V_{1.1} \qquad (3)$$

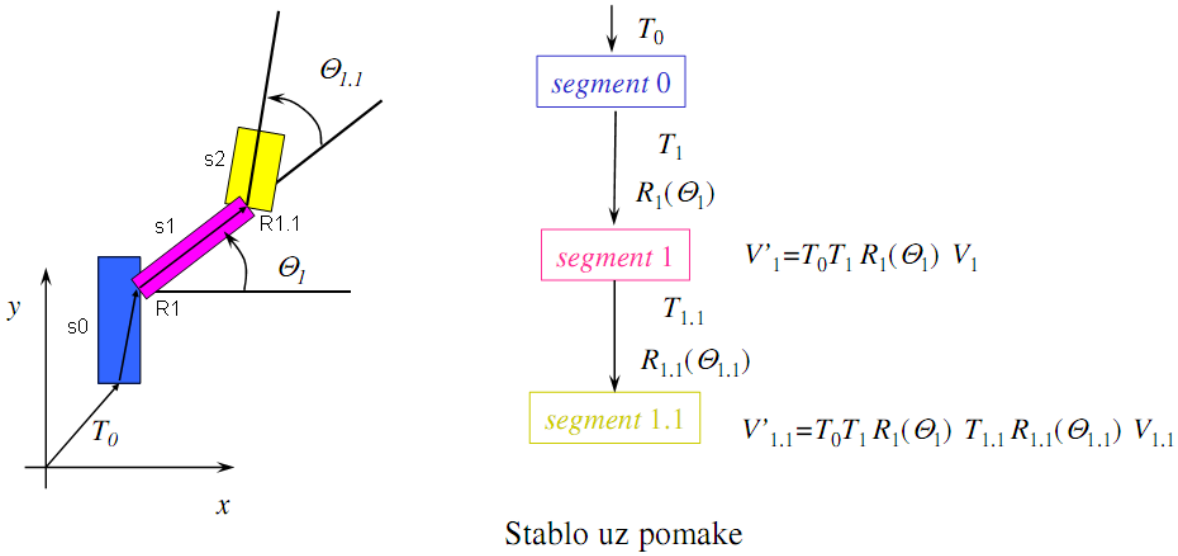



Slika 2.7 Primjer izgradnje stablaste strukture

Hijerarhijski model, načinjen na gore opisan način, predstavlja samo pasivni strukturni model. U njemu se uz translaciju segmenata mogu odrediti i ostale značajke kao što su boja, tekstura, oblik, vidljivost, materijal itd. te hoće li segment – dijete naslijediti te značajke od segmenta – roditelja ili imati svoje vlastite. No, kako bismo mogli ostvariti kretanje strukturnog modela potrebno je definirati tehnike gibanja koje će osigurati „prirodni“ izgled raznih pokreta modela i pojedinih segmenata. To se ostvaruje uvođenjem te povezivanjem tehnika direktne i inverzne kinematike s našim pasivnim strukturnim modelom.

## 2.3 Direktna kinematika

Direktna kinematika je tehnika upravljanja strukturnim modelom u kojem krajnje pozicije segmenata i zglobova ovise o unaprijed definiranim rotacijama i translacijama. Položaj pojedinog segmenta se određuje na osnovi položaja u hijerarhiji njemu nadređenih segmenata. To znači da segment *s1* koji je izveden iz segmenta *s0* ovisi o položaju segmenta *s0*. Isto tako segment *s2* koji je izveden iz segmenta *s1* ovisi o položaju segmenata *s1* i *s0*. Npr. neka *s0* predstavlja trup, *s1* nadlakticu i *s2* podlakticu. Rotacija nadlaktice u području ramena ima utjecaja na položaj podlaktice zato što je podlaktica izvedena iz nje. Nasuprot tome, ako se rotira podlaktica u području lakta, ona nema nikakvog utjecaja na položaj nadlaktice ili trupa zato što su oni na višem položaju u hijerarhijskoj strukturi. Opisani problem se može prikazati sljedećom slikom (slika 2.8):




Slika 2.8 Rotiranje i translatiranje segmenata s unaprijed zadanim kutovima

Položaj segmenta trupa (*s0)* je određen samo translacijom $T_0$, koja se koristi za postavljanje izvorišne pozicije cjelokupnog modela. Na segment nadlaktice (*s1)* utječe više čimbenika. Uz rotaciju ramena R1 za kut $\Theta_1$, utječe i translacija $T_1$ te položaj roditeljskog segmenta *s1*. Položaj segmenta s1 se može matematički zapisati formulom

$$V'_1 = T_0T_1R_1(\Theta_1)V_1 \qquad (4)$$

gdje $T_0$ i $T_1$ označavaju translacije segmenata u statičkom modelu, $R_1(\Theta_1)$ utjecaj rotacije u zglobu R1, te $V_1$ početni položaj segmenta.

Na položaj segmenta podlaktice (*s2)* utječu svi njemu nadređeni segmenti. To znači da, uz osobnu translaciju $T_2$ i rotaciju $R_2(\Theta_2)$, utječu i translacije i rotacije trupa i nadlaktice. To se može prikazati formulom

$$V'_{1.1} = T_0T_1R_1(\Theta_1)T_{1.1}R_{1.1}(\Theta_{1.1})V_{1.1} \qquad (5)$$

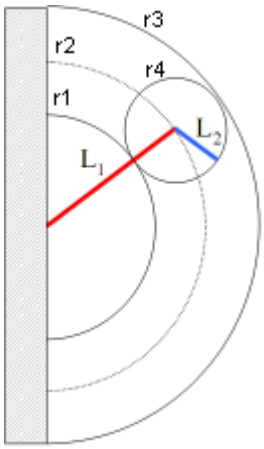
Direktna kinematika je pristupačna za razumjeti, no što se tiče posla zahtijeva puno vremena sa strane programera. Naime, kako bismo mogli iscrtati strukturni model potrebno je definirati translacije i kutove rotacija svih segmenata ručno. To je prihvatljivo kad je potreban model u samo par pozicija. No problem nastaje kada je potrebno ostvariti složenije pokrete. Naime, tada je potrebno namještanje mnoštva kutova što je jako mukotrpan i dugotrajan posao. Zbog toga se umjesto direktne kinematike počela koristiti inverzna kinematika koja nam omogućava prirodnije pokrete uz prethodno zadanu početnu i krajnju poziciju segmenata.

### 2.4 Inverzna kinematika

Inverzna kinematika predstavlja postupak obrnut direktnoj kinematici. Umjesto zadavanja kutova svih segmenata direktno, zadaje se krajnja željena pozicija vrha manipulatora. Na osnovi toga se izračunavaju položaji i kutovi svih segmenata i zglobova od kojih se sastoji manipulator kako bi se on našao u traženoj poziciji.

Dakle, inverzna kinematika nam omogućava da ostvarimo animiranje pokreta uz jednostavno definiranje početne i krajnje (ciljne) točke vrha manipulatora. To nam omogućava jednostavnije animiranje s obzirom da trebamo samo zadati ciljnu točku, nasuprot direktnoj kinematici u kojoj trebamo definirati ručno kutove svakog zgloba i pozicije segmenata kako bi ostvarili jedan pokret koji još k tome ne izgleda prirodno. Zbog toga je inverzna kinematika u većini slučajeva istisnula direktnu kinematiku iz uporabe.

Prije opisivanja algoritma inverzne kinematike objasnit ćemo pojam *dohvatljivog radnog prostora* (eng. reachable workspace). Svaki segment ima definiran svoj dohvatljivi radni prostor. To je prostor u kojem se on može nalaziti i pomicati raznim rotacijama i translacijama. Radni prostor manipulatora koji ima dva zgloba prikazan je na slici 2.9.



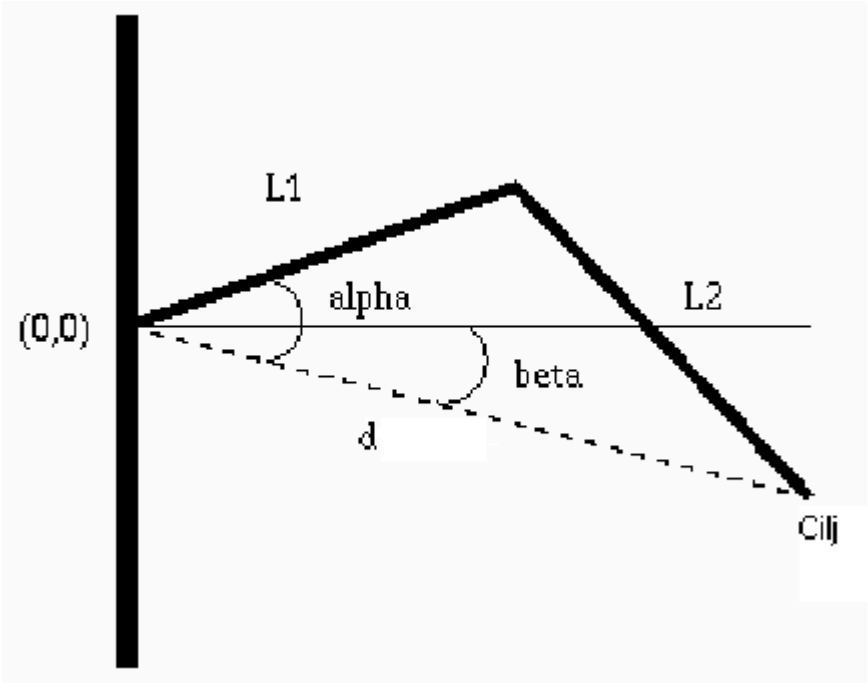

Slika 2.9 Dohvatljivi radni prostor

Polukružnica r2 označava dohvat segmenta $L_1$, dok polukružnice r1 i r3 te kružnica r4 označavaju radni prostor segmenta $L_2$. Dohvatljivi radni prostor cjelokupnog sustava se može matematički opisati formulom

$$L_1 - L_2 \leq \textit{radni prostor} \leq L_1 + L_2 \qquad (6)$$

Lako je primijetiti da na radni prostor utječu duljina segmenata te ograničenja kutova u zglobovima. Na slici 2.9 segment $L_1$ se može pomicati u području od 180°, dok se segment $L_2$ može pomicati u području od 360°. Ograničenja pokretljivosti zglobova su česta pojava u oblikovanju hijerarhijskih kinematičkih modela.

Princip rada inverzne kinematike ćemo opisati koristeći se strukturom od dva segmenta koji su međusobno povezani te zajedno pričvršćeni na čvrstu točku (slika 2.10).




Slika 2.10 Princip inverzne kinematike

Segmenti $L_1$ i $L_2$ su kruti elementi te stoga imaju fiksne duljine. Poznajući koordinate ciljnog položaja možemo odrediti potrebne kutove *alfu* i *betu*.

Kut $\beta$ se može izračunati preko *x* i *y* koordinata ciljne točke

$$\cos\beta = \frac{x}{\sqrt{x^2+y^2}} \quad (7)$$

Koristeći se formulom za kosinusov poučak

$$c^2 = a^2 + b^2 - 2ab\cos(\alpha) \quad (8)$$

možemo izračunati i kut α

$$\cos(\alpha - \beta) = \frac{L_1^2 + x^2 + y^2 + L_2^2}{2L_1L_2} \quad (9)$$

Na ovaj način dobivamo dva rješenja koja su simetrična u odnosu na pravac koji spaja ishodište strukture i ciljnu točku. Povećanjem broja segmenata te ujedno i zglobova, povećava se i broj ukupnih rješenja za određenu ciljnu točku. Stoga je potrebno postaviti ograničenja kretnji u zglobovima tj. definirati minimalni i maksimalni kut koji time predstavljaju dozvoljeni raspon kuta za pojedini zglob.

Tijekom animacije potrebno je konstantno pratiti cilj te mu se iterativnim numeričkim rješavanjem jednadžbi polako približavati. Također, jedan veći problem u ostvarenju inverzne kinematike je teška aproksimacija linearnih pokreta (zbog nelinearnosti funkcija sinus i kosinus koje se koriste u proračunima). Za ostvarenje toga koriste se razne metode koje ćemo opisati u nastavku. Među njima prvenstveno prevladava korištenje Jakobijeve matrice.

### 2.4.1 Jakobijeva matrica

U vektorskom računu, Jakobijeva matrica (Jakobijan) je matrica čiji su elementi sve parcijalne derivacije prvog reda funkcije vektorske ili skalarne vrijednosti u odnosu na drugi vektor. Ona opisuje orijentaciju tangencijalne ravnine zadane funkcije u određenoj točci. Važnost Jakobijana leži u činjenici da ona najbolje predstavlja linearnu aproksimaciju za diferencijabilnu funkciju blizu zadane točke.

Prije nego se uputimo u izračun Jakobijeve matrice za problem inverzne kinematike, definirat ćemo nekoliko vektora radi lakšeg objašnjavanja:

$$X = \begin{bmatrix} \Theta_1 \\ ... \\ \Theta_n \end{bmatrix} \qquad (10)$$

gdje X predstavlja vektor kutova rotacije cijelog kinematičkog lanca. $\Theta_i$ predstavlja kut rotacije zgloba *i* u odnosu na zglob *i-1*, odnosno u odnosu na čvrstu točku ako je i = 1.

Zatim je potreban vektor koji sadrži koordinate pozicije kraja kinematičkog lanca (tj. kraja manipulatora):

$$Y = \begin{bmatrix} x_e \\ y_e \\ z_e \end{bmatrix} \qquad (11)$$

Sada, kad imamo definirane vektore za rotacijske kutove zglobova kinematičkog lanca (10) i vektor pozicije kraja kinematičkog lanca (11) potrebno je definirati vezu između njih tj. funkcije preko kojih se iz vektora X mogu izračunati elementi vektora Y:

$$F = \begin{bmatrix} f_1(X) \\ f_2(X) \\ f_3(X) \end{bmatrix} \quad (12) \qquad Y = F(X) \begin{cases} x_e = f_1(X) \\ y_e = f_2(X) \\ z_e = f_3(X) \end{cases} \quad (13)$$

U unaprijednoj kinematici vrijedi jednakost (13) koja predstavlja izračun ciljne pozicije na osnovu zadanih svih kutova zglobova kinematičkog lanca. Nasuprot tome, u inverznoj kinematici je potreban obrnuti pristup, što opisuje sljedeća jednakost:

$$X = F^{-1}(Y) \qquad (14)$$

Za vrlo male (infinitezimalne) vrijednosti promjene vektora kuta X i vektora pozicije vrha manipulatora Y, vrijedi jednakost:

$$\frac{\Delta Y}{\Delta X} \approx \frac{\partial F}{\partial X} \quad \rightarrow \quad \Delta Y \approx J \Delta X \quad (15)$$

gdje J predstavlja Jakobijevu matricu

$$J = \frac{\partial F}{\partial X} \quad (16)$$

Nakon što smo izračunali Jakobijevu matricu potrebno je pronaći prikladne pomake $\Delta x_e$, $\Delta y_e$, $\Delta z_e$ koji određuju brzinu približavanja ciljnoj poziciji. Oni također određuju veličinu promjene rotacijskih kutova u zglobovima. Ako udaljenost između ciljne pozicije i vrha manipulatora bude unutar definirane granice tolerancije, zaustavlja se algoritam izračunavanja. Inače se algoritam ponavlja, no sada sa kutovima koji su uvećani za izračunate komponente.

**Algoritam inverzne kinematike korištenjem Jakobijeve matrice**

1. Izračunati poziciju vrha manipulatora
2. Odrediti prikladne pomake $\Delta x_e$, $\Delta y_e$, $\Delta z_e$
3. Izračunati Jakobijevu matricu
4. Izračunati pomake rotacijskih kutova $X := X + \Delta X$
5. Izračunati novu poziciju vrha manipulatora

6. Ponavljaj algoritam sve dok udaljenost vrha manipulatora i ciljne pozicije ne bude unutar definirane granice tolerancije

### 2.4.2 Pseudo inverzna Jakobijeva matrica $J^+$

Ponekad Jakobijeva matrica nije kvadratna, te stoga nije moguće odrediti inverznu matricu. Tada se određuje pseudo inverzna matrica $J^+$

$$J^+ = (J^T J)^{-1} J^T \qquad (17)$$

koja zamjenjuje inverznu matricu $J^{-1}$. Pseudo inverzna matrica je jedinstvena, no nestabilna u blizini singulariteta (slika 2.11) te vremenski zahtjevna za izračunati. Također, nije matematički točna kao inverzna (ukoliko ju je moguće izračunati).


Slika 2.11 Primjer singulariteta

### 2.4.3 Transponirana Jakobijeva matrica $J^T$

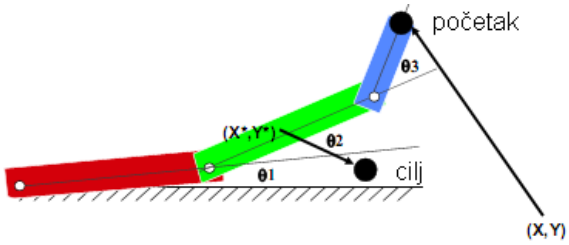
Pseudo inverznu matricu $J^+$ možemo zamijeniti transponiranom matricom $J^T$. Korištenje transponirane matrice nije matematički egzaktno, no teži ka rješenju. Za razliku od pseudo inverzne matrice, računanje nije vremenski zahtjevno te nema problema sa singularitetom. Nedostatak je što promjene u zglobovima nisu ujednačene, tj. obrtni moment je jači dalje od kraja, pa su promjene disproporcionalne.

### 2.4.4 CCD metoda

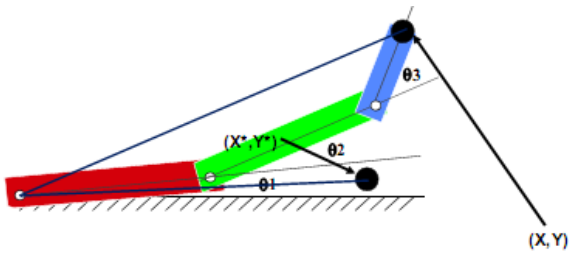
CCD (eng. Cyclic Coordinate Descent) je postupak koji problem inverzne kinematike rješava postupnim rješavanjem 1DOF problema duž lanca. Za svaki zglob se traži promjena koja će minimizirati udaljenost do cilja. Ako određeni zglob ima više stupnjeva slobode, svaki stupanj slobode se obrađuje zasebno, jedan poslije drugoga.

Uzmimo kao primjer kinematički lanac sa slike 2.12. Imamo tri 1DOF zgloba čiji su kutovi rotacija $\Theta_1$, $\Theta_2$ i $\Theta_3$. Ideja je promijeniti kut $\Theta_1$ tako da početna točka bude

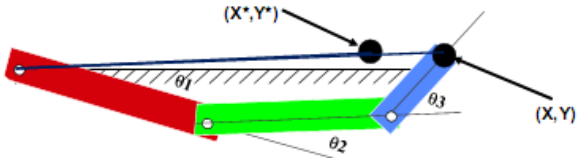
što bliža ciljnoj točci. To se ostvaruje tako da se prvo odrede pravac koji povezuje zglob definiran kutom $\Theta_1$ i kraj kinematičkog lanca te pravac koji povezuje zglob i ciljnu točku (slika 2.13). Zatim se kut $\Theta_1$ promijeni tako da se ta dva pravca preklapaju (slika 2.14). Po istom principu se zatim mijenja kut $\Theta_2$ (slike 2.15 i 2.16), pa $\Theta_3$ (slike 2.17 i 2.18), pa opet $\Theta_1$ itd. sve dok udaljenost između vrha kinematičkog lanca i ciljne pozicije ne bude unutar definirane granice tolerancije (slika 2.19).



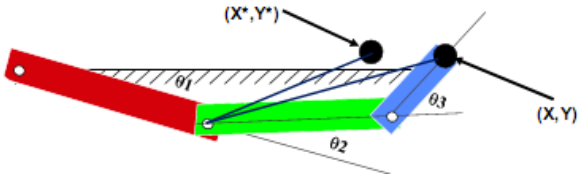

slika 2.12 Početni položaj



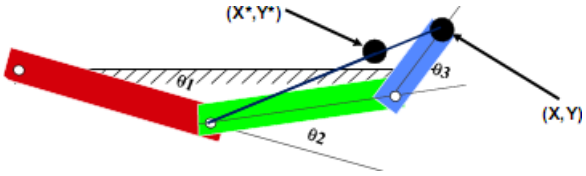

slika 2.13 Određivanje pravaca za $\Theta_1$



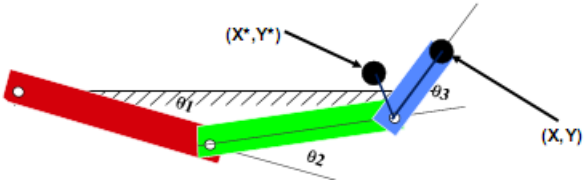

slika 2.14 Preklapanje pravaca za $\Theta_1$



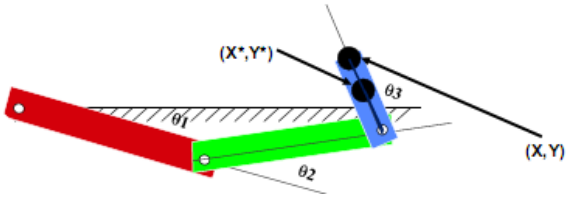

slika 2.15 Određivanje pravaca za $\Theta_2$



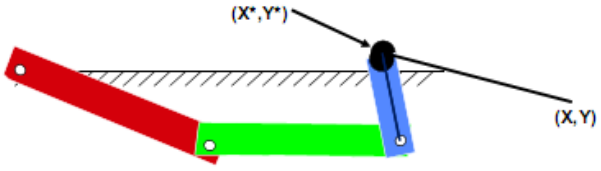

slika 2.16 Preklapanje pravaca za $\Theta_2$




slika 2.17 Određivanje pravaca za $\Theta_3$




slika 2.18 Preklapanje pravaca za $\Theta_3$




slika 2.19 Krajnji položaj

Nedostatak ovog algoritma je neujednačenost promjena u zglobovima. Animirani pokreti često nisu glatki, te se mogu zaglaviti.

# 3. Programsko rješenje

Konačno, nakon općenitog opisa izrade kinematičkog modela čovjeka, dolazimo do njegove implementacije u definiranom programskom okruženju. U ovom poglavlju bit će opisano kako postaviti odgovarajuću radnu okolinu za izradu programskog rješenja te sama izrada tog rješenja. Izrada rješenja se može prikazati u sljedećih par glavnih segmenata:

- postavljanje radne okoline za izradu rješenja korištenjem programskog jezika C++ i grafičkog standarda OpenGL
- izrada statičkog hijerarhijskog modela
- izrada pomoćnih funkcija za izračun pozicija zglobova i kutova
- izgradnja animacije korištenjem direktne kinematike
- programska implementacija algoritama za inverznu kinematiku
- postavljanje kontrola preko tipkovnice

## 3.1 Postavljanje radne okoline

Kao radno okruženje u izradi ovog rada odabran je *Code::Blocks 10.05*. Razlog tomu je što je besplatan te prvenstveno orijentiran prema programskim jezicima *C* i *C++*. Uz to, omogućava jednostavno snalaženje među datotekama projekta te jednostavno organiziranje i pisanje koda.

Kako bi mogli koristiti funkcije iz biblioteka definiranih grafičkim standardom OpenGL potrebno ih je prvo postaviti na mjesta gdje ih *Code::Blocks* može vidjeti. Potrebne datoteke:

- **opengl32.dll**, **glu32.dll**, **glut32.dll** – potrebno postaviti u „system32“ pretinac unutar „Windows“ pretinca (npr. „C:\WINDOWS\system32“)
- **gl.h**, **glu.h**, **glut.h** – potrebno postaviti u „include\GL“ pretinac unutar „Code::Blocks“ pretinca (npr. „C:\Program Files\CodeBlocks\MinGW \include\GL“)
- **libopengl32.a**, **libglu32.a**, **libglut32.a** – potrebno postaviti u „lib“ pretinac unutar „Code::Blocks“ pretinca (npr. „C:\Program Files\CodeBlocks \MinGW\lib“)

Nakon što smo postavili sve potrebne datoteke, potrebno je naš projekt u Code::Blocks povezati (eng. link) s dinamičkim bibliotekama ( .dll – eng. Dynamic

– link library). To učinimo tako da odemo na „Project→Build options“ te u prozoru koji se pojavi odaberemo ime našeg projekta ( ne debug ili release). Nakon toga odaberemo karticu „Linker settings“ te kliknemo na gumb „Add“. U novootvoreni prozor upišemo „opengl32“ te stisnemo „OK“. Isto to ponovimo sa „glu32“, „glut32“ te „gdi32“. Time smo završili postavljanje naše radne okoline. Više informacija u vezi Code::Blocks se može naći na www.codeblocks.org.

### 3.2 Izrada statičkog hijerarhijskog modela

Kako bismo mogli izraditi hijerarhijski model, potrebno je prvo definirati elemente koji ga sačinjavaju. Ti elementi su segmenti i zglobovi. Segmenti predstavljaju kruti element koji povezuje dva zgloba te se može rotirati samo na području zgloba. Elementi su implementirani u obliku razreda *SkeletonLink* (opis segmenta) i *SkeletonJoint* (opis zgloba).

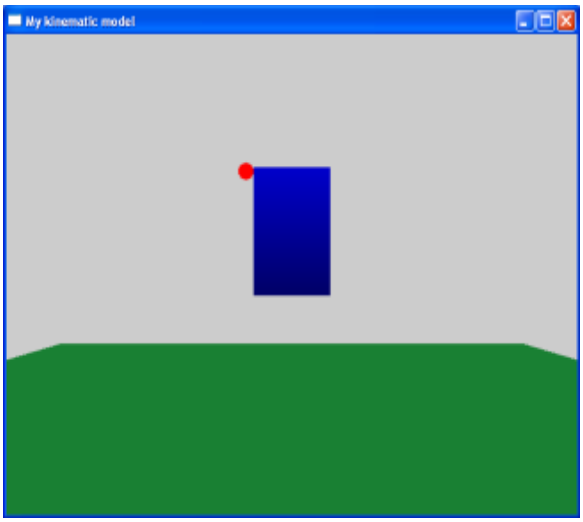
Razred *SkeletonLink* predstavlja opis segmenata (npr. gornji dio ruke, donji dio ruke). On sadrži osnovne podatke vezane za iscrtavanje segmenta te vezu između segmenta i zglobova koji utječu na njega (npr. gornji dio ruke ovisi o ramenu). Također segment sadrži pokazivač na roditeljski segment tj. segment iz kojeg je izveden.

Razred *SkeletonJoint* predstavlja opis zgloba (npr. rame, lakat). Uz ime zgloba sadrži rotacijski kut oko određene osi (npr. *lshoulderRotUD* označava rotacijski kut lijevog ramena oko x osi) te njegova ograničenja. Prilikom pokušaja promjene vrijednosti kuta provjerava se da li je vrijednost unutar određene granice. Ako jest, kut poprima zadanu vrijednost. U protivnom se kut postavlja na prikladnu graničnu vrijednost.

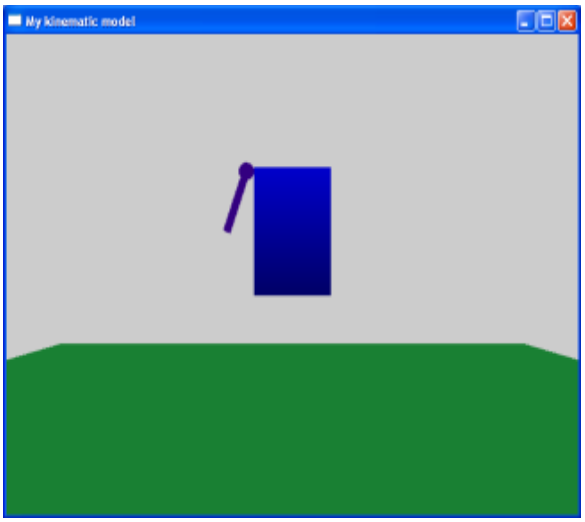
Sada možemo, uz definirane strukture elemenata, implementirati razred koji gradi statički model (razred *Skeleton*). On sadrži pokazivače na sve elemente kostura te metode za iscrtavanje istoga. Prilikom iscrtavanja potrebno je odrediti početnu točku tj. korijenski element preko kojeg ćemo se moći orijentirati. U našem slučaju to je torzo. Uz pomoć funkcija translacija i rotacija možemo se postaviti na potrebne pozicije za iscrtavanje ostalih elemenata. OpenGL unutar globalne pozicijske matrice GL_MODELVIEW_MATRIX sprema trenutnu poziciju na kojoj se nalazimo unutar scene. Ona se prilikom translacije i rotacije mijenja. Kako bi se mogli uvijek orijentirati u odnosu na torzo, prije translacija i rotacija za

iscrtavanje nekog segmenta npr. ruku ili nogu, potrebno je pospremiti pozicijsku matricu na stog. To se ostvaruje korištenjem naredbe glPushMatrix().

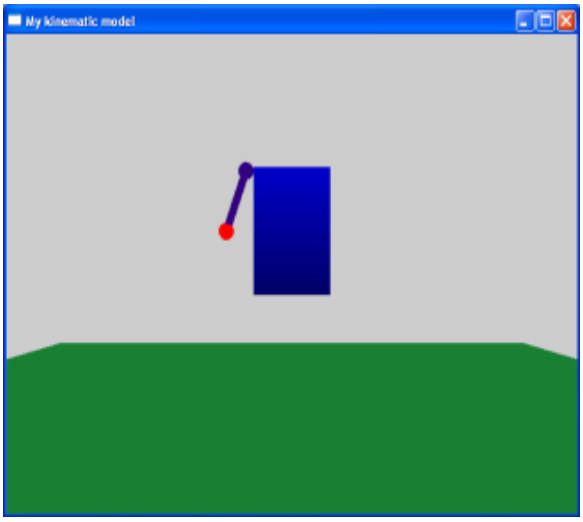
Kad se konačno postavimo na potrebnu poziciju, npr. na poziciju ramena (slika 3.1), slijedi iscrtavanje zgloba koji predstavlja rame te segmenta koji predstavlja nadlakticu (slika 3.2). Daljnjim translatiranjem za duljinu nadlaktice dolazimo do pozicije zgloba koji predstavlja lakat (slika 3.3). Na toj poziciji ponovno iscrtamo zglob te segment koji predstavlja podlakticu i šaku(slika 3.4).



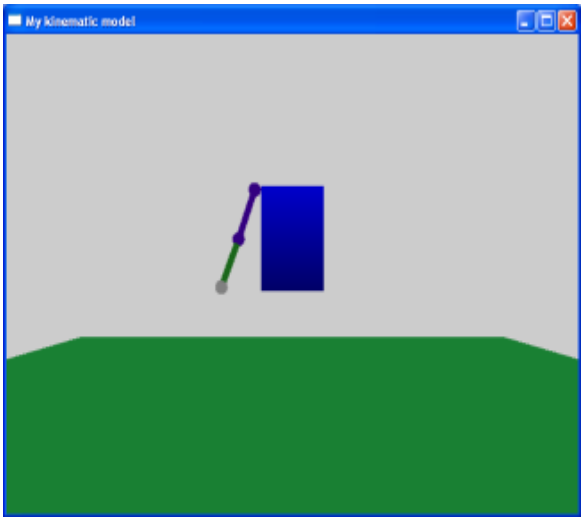

slika 3.1 Pozicioniranje ramena




slika 3.2 Iscrtavanje ramena i nadlaktice



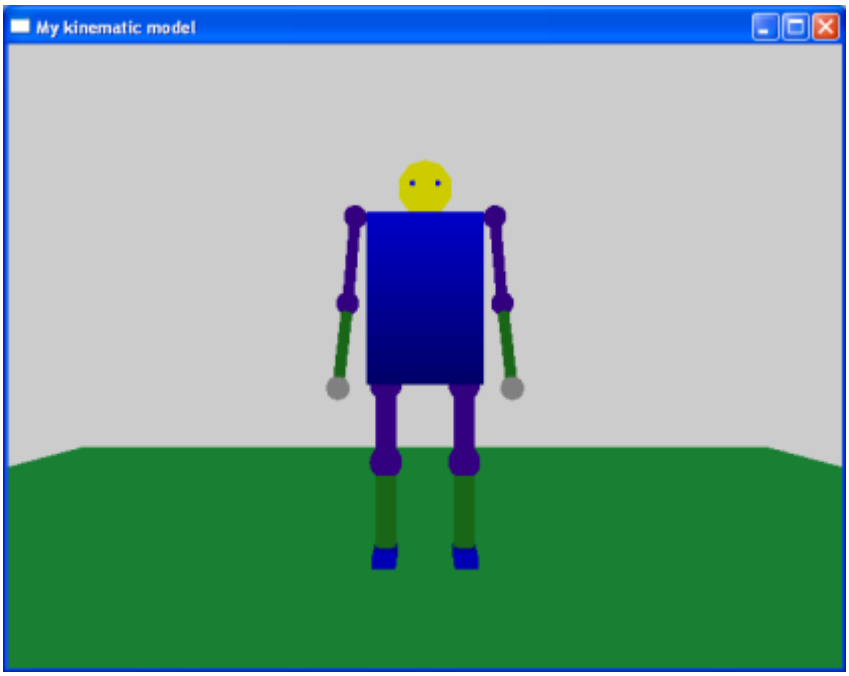

slika 3.3 Pozicioniranje lakta




slika 3.4 Iscrtavanje lakta i podlaktice

Za iscrtavanje segmenata tipa nadlaktice koristimo se ugrađenom funkcijom za iscrtavanje valjka *gluCylinder()*, za glavu i za zglobove se koristimo ugrađenom funkcijom za iscrtavanje kruga *gluSphere()* te se za iscrtavanje torza i stopala koristimo vlastito implementiranom funkcijom za iscrtavanje kvadra.

Nakon što smo iscrtali određeni segment (u gornjem slučaju desnu ruku) potrebno je povratiti prethodno stanje globalne pozicijske matrice. To se ostvaruje preko funkcije glPopMatrix() koja sa stoga uzima prethodno pospremljenu pozicijsku matricu. Korištenjem programskih blokova omeđenih s glPushMatrix() i glPopMatrix() pozicioniramo se te iscrtamo ostale segmente ( lijevu ruku, noge, glavu) slično kao i za lijevu ruku. Nakon iscrtanih svih segmenata dobivamo statički hijerarhijski model čovjeka (slika 3.5).



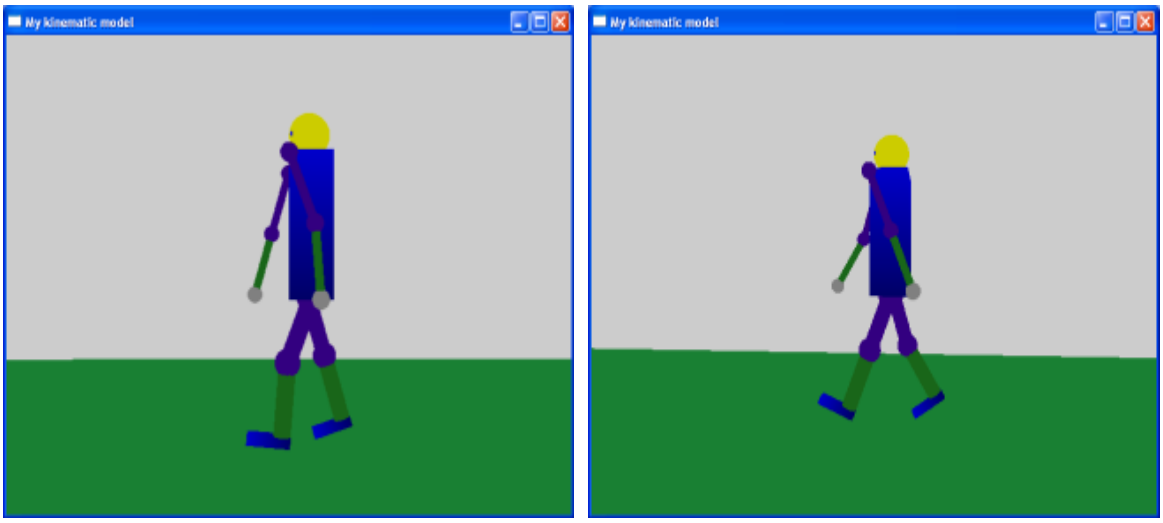

slika 3.5 Statički hijerarhijski model čovjeka

### 3.3 Ostvarenje animacije koristeći princip direktne kinematike

Kao što je već opisano, direktna kinematika se izvodi izravnim zadavanjem kutova svih rotacijskih zglobova prije iscrtavanja. Uzastopnim mijenjanjem tih kutova prilikom izvođenja programa, npr. uzastopnim učitavanjem vrijednosti iz datoteke ili iz alociranog polja u memoriji, možemo ostvariti animaciju našeg modela.

U našem programu direktna kinematika je korištena kako bi ostvarili simulaciju hodanja našeg kinematičkog modela. Simulacija hoda je ostvarena uz pomoć pet pomoćnih kostura modela. Početni kostur predstavlja model u standardnoj poziciji (slika 3.5). Svaki od ostalih pet pomoćnih kostura prikazuje jednu od pozicija u

kojoj se model nalazi prilikom hodanja (npr. desna ruka i lijeva noga naprijed, lijeva ruka i desna noga natrag – slika 3.6 i 3.7)




slika 3.6 i 3.7 Prikaz pomoćnih modela za simulaciju hoda

Pomoćni kosturi su ostvareni tako što smo im definirali sve rotacijske kutove kako bi predstavljali odgovarajuću poziciju. Npr. kod koji definira model sa slike 3.7 je sljedeći:

```
{
    moveSkelet[5] = new Skeleton(bodyC[1]);
    moveSkelet[5]->rhipRotUD->rotation = 107.5;
    moveSkelet[5]->rkneeRotUD->rotation = 15.0;
    moveSkelet[5]->lhipRotUD->rotation = 67.5;
    moveSkelet[5]->lkneeRotUD->rotation = 0.0;
    moveSkelet[5]->rshoulderRotUD->rotation = 72.5;
    moveSkelet[5]->relbowRotUD->rotation = -15.0;
    moveSkelet[5]->lshoulderRotUD->rotation = 112.5;
    moveSkelet[5]->lelbowRotUD->rotation = 0.0;
}
```

gdje je *moveSkelet[5]* pokazivač na objekt tipa *Skeleton* (razreda koji predstavlja hijerarhijski model), dok su redom *rhipRotUD*, *rkneeRotUD* itd. pokazivači na objekte koji predstavljaju zglobove kostura.

Kad smo izgradili sve kosture potrebne za simulaciju hodanja kostura, potrebno je odrediti redoslijed i brzinu izmjene pojedinih kostura prilikom iscrtavanja. Kako se scena u OpenGL osvježava svakih par milisekundi, opcija da se prilikom svakog iscrtavanja promijeni model je neprihvatljiva. Naime, tada bi se

modeli prebrzo izmjenjivali i simulacija hodanja ne bi lijepo izgledala (imali bi super brzo hodanje). Taj problem smo riješili korištenjem globalno definirane varijable *timer* koja se prilikom svakog osvježavanja scene povećava za $\Delta timer = 5$. Kada se vrijednost varijable *timer* poveća za 1000 mijenjamo trenutni model sa sljedećim modelom u nizu.

Također, uz brzinu promjene modela, potrebno je odrediti i putanju po kojoj se model giba. U našem ostvarenju, putanja je pravokutnik. Brzina translacije je definirana u odnosu na brzinu promjene modela tj. $\Delta move = \Delta timer/1000$. Prilikom svakog osvježavanja scene, model se pomiče za $\Delta move$ u određenom smjeru. Kada dođe do vrha pravokutnika koji označava putanju, model se rotira za 90° te nastavlja dalje po zadanoj putanji. Time smo ostvarili simulaciju hodanja našeg modela po definiranoj putanji koristeći princip direktne kinematike.

### 3.4 Ostvarenje animacije koristeći princip inverzne kinematike

Implementacija inverzne kinematike u naš program se svodi na implementiranje algoritama koji će izračunati rotacijske kutove svih zglobova određenog kinematičkog lanca kojemu je zadana ciljna pozicija. Kinematički lanci koji budu animirani principom inverzne kinematike u našem slučaju su lijeva i desna ruka. Ciljna pozicija predstavljena je kuglicom koja lebdi u zraku.

Implementirani algoritmi koriste se dodatno napisanim funkcijama koje nam olakšavaju ostvarenje principa inverzne kinematike. Prvo ćemo opisati te pomoćne funkcije, te onda opisati ostvarenje inverzne kinematike korištenjem dvaju odvojenih pristupa – preko CCD metode te korištenjem Jakobijeve matrice.

#### 3.4.1 Opis pomoćnih funkcija

Prvo ćemo opisati funkcije koje se koriste u oba algoritma. To su funkcije *distance()* i *GetJointPosition()*.

Funkcija *distance()* određuje udaljenost između dviju točaka. Ona nam služi kako bi mogli izračunati udaljenost između vrha manipulatora i ciljne pozicije. Ako je ta udaljenost manja od definirane tolerancije onda se više algoritmi za izračun pomaka kutova ne izvode. Naprotiv, ako je udaljenost veća, nastavlja se s izvođenjem algoritama.

Funkcija *GetJointPosition()* je osnova za svaki algoritam. Ona, uz dano ime zgloba, određuje njegovu poziciju. Osnova izračuna pozicije su rotacijski kutovi zglobova te duljine segmenata koji spajaju te zglobove. Uz korištenje trigonometrijskih funkcija sinus i kosinus određuju se pozicije zglobova. Najveći problem je što pozicija „i-tog“ zgloba ovisi o rotacijskim kutovima svih prethodnih „i-1“ zglobova što dovodi do ogromnih trigonometrijskih izraza. Npr. pozicija lakta ovisi o rotacijskom kutu ramena, dok pozicija šake ovisi o rotacijskim kutovima lakta i šake.

Za primjer ćemo objasniti postupak izračuna pozicije lijeve šake. Potrebno je odrediti pomake $\Delta x$, $\Delta y$ i $\Delta z$ u odnosu na osnovu poziciju lijevog ramena. Kako bismo to ostvarili moramo razmotriti koji pokreti utječu na koje pomake. Na $\Delta x$ utječe pomicanje ramena u stranu (kut $\beta$) te pomicanje lakta gore – dolje (kut $\gamma$). Na $\Delta y$ i $\Delta z$ utječu pomicanja ramena u stranu, pomicanje ramena gore – dolje (kut $\alpha$) te pomicanje lakta gore – dolje. Najjednostavniji način dolaska do traženih formula je odvojeno promatranje utjecaja kutova $\alpha$ i $\beta$ na traženu poziciju. Razlog tome je što se pomaci za ta dva kuta odvijaju u dvije različite ravnine. Za kut $\alpha$ se pomicanje odvija u ravnini y-z, dok se za kut $\beta$ odvija u ravnini x-z. To znači da ćemo, dok promatramo utjecaj kuta $\alpha$ na poziciju šake, kut $\beta$ postaviti na vrijednost 0. S druge strane, dok budemo promatrali utjecaj kuta $\beta$ na poziciju šake, kut $\alpha$ ćemo postaviti na vrijednost 0. Kut $\gamma$ pak, promatramo u oba slučaja zato što on nije povezan sa rotacijom ramena. Iz formula ovisnosti pozicije u odnosu na kut $\alpha$, odnosno $\beta$, jednostavnim povezivanjem dobijemo krajnje formule za određivanje koordinata pozicije šake:

$$x = 1.2 + (1.5 + 1.5 * cos\gamma) * sin\beta \quad (18)$$

$$y = 6.4 - (1.5 + 1.5 * cos\gamma) * cos\beta * \sin\alpha - 1.5 * sin\gamma * cos\alpha \quad (19)$$

$$z = 0.0 + 1.5 * cos\alpha * cos\beta - 1.5 * sin\gamma * sin\alpha + 1.5 * cos\gamma * cos\alpha * cos\beta \quad (20)$$

Sljedeće funkcije se koristi samo pri ostvarenju CCD metode. To su funkcije *GetMoveAngleUD()* i *GetMoveAngleLR()*. Funkcija *GetMoveAngleUD()* služi za izračun kuta između dva pravca u ravnini y-z, dok funkcija *GetMoveAngleLR()* služi za izračun kuta između dva pravca u ravnini x-z. Obje funkcije rade na principu da se izračunaju koeficijenti pravaca te se onda preko formule

$$tg\varphi = \frac{k_2 - k_1}{1 + k_1 k_2} \quad (21)$$

odredi kut $\varphi$ između zadanih pravaca. Funkcije vraćaju iznos kuta u stupnjevima.

### 3.4.2 Izvedba CCD algoritma

Ostvarivanje animacije preko CCD (eng. Cyclic Coordinate Descent) metode u našem programu objasnit ćemo na primjeru lijeve ruke. Ruka se sastoji od dva zgloba – ramena i lakta te dva segmenta – nadlaktice i podlaktice. Rame ima definirana dva rotacijska stupnja slobode dok lakat ima definiran jedan rotacijski stupanj slobode. To znači da sve skupa imamo tri rotacijska kuta koja je potrebno mijenjati kako bi se dostigla ciljna pozicija. Pomicanje nadlaktice gore – dolje označava kut $\alpha$, pomicanje nadlaktice lijevo – desno označava kut $\beta$ te pomicanje podlaktice gore – dolje označava kut $\gamma$. Ciljna pozicija ima koordinate $x_e$, $y_e$ i $z_e$.

Za izračun promjene pojedinog kuta potrebno je odrediti pozicije triju točaka. Uz ciljnu poziciju potrebne su pozicija kraja kinematičkog lanca, odnosno u našem slučaju pozicija šake, te pozicija zgloba koja je definirana tim kutom. Tj. za kut $\alpha$ i $\beta$ potrebna je pozicija ramena, šake te ciljna pozicija dok su za kut $\gamma$ potrebne pozicije lakta, šake i cilja. Te pozicije dobijemo preko naše pomoćne funkcije *GetJointPosition()*. Nakon što smo dobili potrebne pozicije, možemo odrediti potrebne promjene kutova:

- Za kut $\alpha$ to je promjena $\Delta\alpha$ koja je jednaka kutu između pravaca rame-šaka i rame-cilj koji se nalaze u ravnini y-z. Stoga za izračun $\Delta\alpha$ koristimo pomoćnu funkciju *GetMoveAngleUD()*
- Za kut $\beta$ to je promjena $\Delta\beta$ koja je jednaka kutu između pravaca rame-šaka i rame-cilj koji se nalaze u ravnini x-z. Stoga za izračun $\Delta\beta$ koristimo pomoćnu funkciju *GetMoveAngleLR()*
- Za kut $\gamma$ to je promjena $\Delta\gamma$ koja je jednaka kutu između pravaca lakat-šaka i lakat-cilj koji se nalaze u ravnini y-z. Stoga za izračun $\Delta\gamma$ koristimo pomoćnu funkciju *GetMoveAngleUD()*

Nakon što smo izračunali promjene kutova, pribrojimo ih trenutnoj vrijednosti pojedinog kuta tj. kutu $\alpha$ pribrojimo $\Delta\alpha$, kutu $\beta$ pribrojimo $\Delta\beta$ te kutu $\gamma$ pribrojimo $\Delta\gamma$.

Gore navedeni postupak ponavljamo sve dok udaljenost između šake i ciljne pozicije ne bude manja od definirane toleracije (koja je u našem slučaju 0.01).

***Pseudokod za CCD metodu (lijeva ruka):***

```
AnimateSkeletCCD(Skeleton* skelet){

Uzmi koordinate ciljne pozicije
{
Odredi koordinate lijeve šake;
Ako je udaljenost šake od cilja veća od tolerancije nastavi;

Odredi koordinate lijevog ramena;
Izračunaj kut između pravaca rame-cilj i rame-šaka u y-z
ravnini;
Promijeni rotacijski kut za izračunati iznos;
}

{
Ako je udaljenost šake od cilja veća od tolerancije nastavi;

Izračunaj kut između pravaca rame-cilj i rame-šaka u x-z
ravnini;
Promijeni rotacijski kut za izračunati iznos;
}


{
Ako je udaljenost šake od cilja veća od tolerancije nastavi;

Odredi koordinate lijevog lakta
Izračunaj kut između pravaca lakat-cilj i lakat-šaka u y-z
ravnini;
Promijeni rotacijski kut za izračunati iznos;
}

}
```

### 3.4.3 Izvedba inverzne kinematike korištenjem Jakobijeve matrice

Drugi način implementiranja inverzne kinematike ostvaren je korištenjem inverzne Jakobijeve matrice. Kao i za CCD metodu, ostvarenje korištenjem Jakobijana ćemo opisati na primjeru lijeve ruke. Prvo ćemo razmotriti što nam je potrebno za izgradnju Jakobijeve matrice. Zatim ćemo objasniti izračun promjene pojedinih kutova koristeći izgrađenu matricu.

Dakle, definirajmo potrebne elemente za izračun Jakobijeve matrice. Prvo nam je potreban vektor kutova rotacija

$$X = \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix}$$

gdje α predstavlja rotacijski kut za pomicanje nadlaktice gore – dolje, β predstavlja rotacijski kut za pomicanje nadlaktice lijevo – desno, te γ predstavlja rotacijski kut za pomicanje podlaktice gore – dolje. Vrijednostima tih kutova pristupamo preko pokazivača na objekt koji predstavlja naš kostur. Zatim nam je potreban vektor koji sadrži koordinate pozicije kraja kinematičkog lanca tj. u našem slučaju, koordinate lijeve šake

$$Y = \begin{bmatrix} x_e \\ y_e \\ z_e \end{bmatrix}$$

koje dobijemo preko pomoćne funkcije *GetJointPosition()* te konačno vektor funkcija F preko kojih možemo uz zadani X izračunati Y. Taj vektor sadrži trigonometrijske izraze koje koristi i pomoćna funkcija *GetJointPosition()*. Dakle, za lijevu šaku vektor funkcija F je

$$F = \begin{bmatrix} f_1 = 1.2 + (1.5 + 1.5 * cos\gamma) * sin\beta \\ f_2 = 6.4 - (1.5 + 1.5 * cos\gamma) * cos\beta * \sin\alpha - 1.5 * sin\gamma * cos\alpha \\ f_3 = 0.0 + 1.5 * cos\alpha * cos\beta - 1.5 * sin\gamma * sin\alpha + 1.5 * cos\gamma * cos\alpha * cos\beta \end{bmatrix} \quad (22)$$

Po definiciji Jakobijeva matrica je

$$J = \frac{\partial F}{\partial X}$$

koju za našu lijevu šaku raspisujemo na sljedeći način:

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial \alpha} & \frac{\partial f_1}{\partial \beta} & \frac{\partial f_1}{\partial \gamma} \\ \frac{\partial f_2}{\partial \alpha} & \frac{\partial f_2}{\partial \beta} & \frac{\partial f_2}{\partial \gamma} \\ \frac{\partial f_3}{\partial \alpha} & \frac{\partial f_3}{\partial \beta} & \frac{\partial f_3}{\partial \gamma} \end{bmatrix} \quad (23)$$

Matrica je kvadratna te stoga možemo izračunati njenu inverznu matricu. Za njen izračun koristili smo se Cramerovim pravilom

$$J^{-1} = \frac{1}{\det J} \begin{bmatrix} J_{11} & J_{12} & J_{13} \\ J_{21} & J_{22} & J_{23} \\ J_{31} & J_{32} & J_{33} \end{bmatrix}^T \quad (24)$$

gdje detJ predstavlja determinantu matrice J, te elementi $J_{ij}$ predstavljaju algebarski komplement elementa $a_{ij}$ matrice J.

Preostaje još samo definirati prikladne pomake $\Delta x_e$, $\Delta y_e$ i $\Delta z_e$ koji određuju brzinu konvergencije ka ciljnoj poziciji te zatim izračunati promjene kutova $\Delta\alpha$, $\Delta\beta$ i $\Delta\gamma$. Za što ljepši izgled pokreta odabrali smo za pomake slijedeće vrijednosti:

$$\Delta x_e = \frac{(x_c - x_e)}{k} \quad \Delta y_e = \frac{(y_c - y_e)}{k} \quad \Delta z_e = \frac{(z_c - z_e)}{k} \quad (25)$$

gdje $x_c$, $y_c$ i $z_c$ označavaju koordinate ciljne pozicije te 1/k = 10 predstavlja koeficijent konvergencije. Sukladno tome, promjene kutova su definirane s

$$\Delta\alpha = J^{-1}[0][0] * \Delta x_e + J^{-1}[0][1] * \Delta y_e + J^{-1}[0][2] * \Delta z_e \quad (26)$$

$$\Delta\beta = J^{-1}[1][0] * \Delta x_e + J^{-1}[1][1] * \Delta y_e + J^{-1}[1][2] * \Delta z_e \quad (27)$$

$$\Delta\gamma = J^{-1}[2][0] * \Delta x_e + J^{-1}[2][1] * \Delta y_e + J^{-1}[2][2] * \Delta z_e \quad (28)$$

Na kraju kutu $\alpha$ pribrojimo $\Delta\alpha$ te sukladno tome napravimo i s kutovima $\beta$ i $\gamma$. Gore navedeni algoritam ponavljamo sve dok udaljenost šake i ciljne pozicije ne bude manja od definirane tolerancije (koja je u našem slučaju 0.01).

***Pseudokod za animiranje korištenjem Jakobijeve matrice:***

```
AnimateWithJacobian(Skeleton* skelet){

Odredi nalazi li se ciljna pozicija u dohvatu lijeve ili
desne ruke;
Sukladno tome, odredi poziciju lijeve/desne šake;

Ako je udaljenost šake od cilja veća od tolerancije, nastavi;
Uzmi trenutne vrijednosti kutova lijeve/desne ruke;

Izračunaj Jakobijevu matricu;
Izračunaj inverznu Jakobijevu matricu;

Odredi prikladne pomake Δx, Δy i Δz;
Izračunaj promjene kutova Δα, Δβ i Δγ;
Promijeni kutove α, β, γ za vrijednosti Δα, Δβ odnosno Δγ;
}
```

### 3.5 Usporedba rezultata CCD metode i Jakobijeve matrice

Razlika između ove dvije metode je u konvergenciji koordinata vrha manipulatora ka koordinatama cilja. CCD metoda uvijek ima istu brzinu konvergencije zato što nemamo nikakvih ulaznih parametara koje možemo mijenjati. Nasuprot tome, kod korištenja Jakobijeve matrice možemo mijenjati pomake $\Delta x$, $\Delta y$ i $\Delta z$. Postupnim povećavanjem pomaka dolazimo do brže konvergencije i kretnjama sličnima kao pri animaciji CCD metodom.

Nedostatak brže konvergencije te time i same CCD metode je neprirodni izgled pokreta. Također se kod primjene CCD metode može dogoditi da algoritam pri izračunu pomaka zapne na određenoj vrijednosti te nikad ne dođe do ciljne pozicije.

Korištenjem Jakobijeve matrice, uz definirane prikladne pomake $\Delta x$, $\Delta y$ i $\Delta z$, dobivamo prirodnije pokrete te ne postoji mogućnost zapinjanja. Pomaci su definirani formulom (25)

U sljedećoj tablici prikazan je broj potrebnih iteracija za postizanje određene pozicije korištenjem prvo CCD metode te onda korištenjem Jakobijana.

tablica 1. Broj iteracija u ovisnosti o koeficijentu konvergencije

| 1/k | smjer kretnje | Broj iteracija | |
|---|---|---|---|
| | | CCD | Jakobijan |
| 10 | gore/dolje | 41 | 42 |
| 5 | gore/dolje | 41 | 86 |
| 20 | gore/dolje | 41 | 19 |
| 40 | gore/dolje | 41 | 9 |
| 10 | gore/dolje | 11 | 39 |
| 5 | gore/dolje | 11 | 82 |
| 20 | gore/dolje | 11 | 18 |
| 30 | gore/dolje | 11 | 11 |
| 40 | gore/dolje | 11 | 8 |
| 10 | lijevo/desno | 1 | 33 |
| 5 | lijevo/desno | 1 | 69 |
| 20 | lijevo/desno | 1 | 15 |
| 40 | lijevo/desno | 1 | 6 |

Tablica 1. prikazuje broj iteracija u odnosu na koeficijent konvergencije o kojem ovisi samo animacija koja koristi Jakobijana. Prva četiri mjerenja se odnose na kretnje na većoj relaciji. Ostala mjerenja se odnose na kretnje na manjoj relaciji. Iz toga zaključujemo da je za manje promjene efikasnija CCD metoda dok je za veće promjene efikasnija metoda preko Jakobijeve matrice. No, iako je broj iteracija kod Jakobijana veći, ipak je prikladniji za ostvarenje animacija ljudskih kretnji zbog toga što izgledaju prirodnije.

### 3.6 Interakcija korisnika i programa preko tipkovnice

Kako bi se omogućilo lakše praćenje animacije modela, dodane su kontrole za rotiranje pogleda oko x i y osi te pomicanje po z osi. Također su dodane kontrole za pokretanje animacije, pozicioniranje ciljne točke, uključivanje/isključivanje prikaza modela animiranog direktnom kinematikom, promjena načina izvođenja direktne kinematike (CCD metoda/ Jakobijeva matrica).

tablica 2. Prikaz kontrola

| **Tipka** | **Aktivnost** |
|---|---|
| w, a, s, d | rotiranje pogleda oko x i y osi |
| z, u | pomicanje po z osi |
| 2 | promjena načina izvođenja inverzne kinematike (CCD/Jakobijan) |
| strelice | pomicanje ciljne točke gore, dolje, lijevo i desno |
| next, previous | pomicanje ciljne točke naprijed/nazad |
| 1 | prikaz modela animiranog principom direktne kinematike |
| e | izvođenje jednog koraka animacije |
| Esc | prekidanje izvođenja programa |

# 4. Zaključak

Unatoč tome što je izgrađeni model jednostavnog izgleda i daleko od stvarnog prikaza ljudske strukture, bio je dovoljan za osnovni prikaz načina izgradnje statičke hijerarhijske strukture te animiranja istoga preko principa direktne i inverzne kinematike.

Ljepota izgradnje modela koristeći se samo opcijama OpenGL-a omogućava nam bolje razumijevanje kako je sve povezano te na koji način se iscrtaju elementi u određenim pozicijama. No prilikom toga, potrebno je paziti na sve translacije i rotacije u prostoru koje obavljamo kako se ne bi izgubili unutar trenutne scene. Nedostaci su što je za najjednostavniji model potrebna ogromna količina programskog koda te bi stoga trebalo uložiti puno truda kako bi se prikazao donekle realan kostur čovjeka.

Implementacija algoritama za animacije, nakon što je izgrađen model, nije bila pretjerano zahtjevna. Najveću prepreku u ostvarenju cilja bilo je pronalaženje potrebnih trigonometrijskih izraza preko kojih se računaju pozicije pojedinih elemenata modela. Naime, uzastopno rotiranje i translatiranje prilikom iscrtavanja modela otežava nam orijentaciju u sceni. Također, povećanjem složenosti modela, povećava se i složenost trigonometrijskih izraza te ujedno i složenost algoritama. Dodavanjem samo jednog rotacijskog kuta u kinematički lanac potrebno je mijenjati sve trigonometrijske izraze. Također se Jakobijeva matrica poveća za jedan stupanj.

Poznajući osnove izgradnje hijerarhijskog modela i principe animacije, možemo sada izgrađivati složenije modele sa više detalja realnog kostura čovjeka. Sukladno tome, animiranjem tog složenog modela dobit ćemo i pokrete koji više nalikuju na prirodne ljudske pokrete.

## 5. Literatura

[1] Jeff Molofee, OpenGL Tutorials, Lessons 1 - 48 ,
http://nehe.gamedev.net/ , 2011.

[2] Dustin Graves, CSCI 266 – Computer Animation,
http://dgraves.org/coursework/cs266, 2011.

[3] Željka Mihajlović, Direktna i inverzna kinematika
http://www.zemris.fer.hr/predmeti/ra/predavanja/4_kinemat.pdf, 2011.

[4] Mike Bailey, Inverse Kinematics, CS 419G – CS Skills for Simulation and Game Development, Fall 2010.
http://web.engr.oregonstate.edu/~mjb/cs419g/Handouts/inversekinematics.6pp.pdf , 2011.

[5] Kinematics
http://en.wikipedia.org/wiki/Kinematics, 2011.

[6] Roman Filkorn, Marek Kocan, Simulation of Human Body Kinematics,
http://www.cescg.org/CESCG-2000/RFilkorn/, 2011.

[7] Philip Yen, CS 490 Human Motion Simulation, Spring 1998.
http://www.nbb.cornell.edu/neurobio/land/OldStudentProjects/cs490-97to98/yen/, 2011.

# 6. Sažetak

U ovom radu pokušali smo opisati izgradnju jednostavnog hijerarhijskog modela čovjeka te animiranje korištenjem direktne odnosno inverzne kinematike. Služili smo se programskim jezikom C++ i grafičkim standardom OpenGL.

Prvo poglavlje je uvodna riječ rada, te govori o potrebi za razvojem kinematičkih struktura.

Drugo poglavlje daje općenite informacije o kinematičkim strukturama tj. kako se izgrađuju, od čega se sastoje te načine na koje se mogu animirati. Detaljnije su opisane one metode koje se koriste u programskom ostvarenju.

Nakon toga opisuje se ostvarenje programskog rješenja. Prvo se opisuje izgradnja modela i njegovih komponenata. Zatim se osvrćemo na implementiranje algoritama direktne i inverzne kinematike radi ostvarenja animacije. Direktna kinematika je demonstrirana preko modela koji simulira ljudski hod. Inverzna kinematika je demonstrirana preko modela koji rukama dohvaća kuglicu koja lebdi u zraku. Pred kraj je provedena usporedba, te prednosti i nedostaci pojedinih načina animiranja izgrađenog kinematičkog modela čovjeka.

**Ključne riječi:** kinematičke strukture, jednostavni kinematički lanci, DOF – „stupanj slobode“ direktna kinematika, inverzna kinematika, CCD metoda, Jakobijeva matrica, C++, OpenGL

## 7. Abstract

In this paper we have tried to describe the construction of a simple hierarchical human model and animate it using direct and inverse kinematics. The solution is realized using the programming language C++ and the graphic standard OpenGL.

The first chapter is the introductio, and it speaks about the need for kinematic structures.

The second chapter provides general information about kinematic structures, ie how to build them, what they consist of and how to animate them. Methods used in the program realization, are described more in detail.

Then we describe the realization of the software. First we describe the construction of the model and its components. Then we reflect on the implementation of algorithms of direct and inverse kinematics to achieve animation. Direct kinematics is demonstrated through a model that simulates a human walking. Inverse kinematics is demonstrated through a model that retrieves a ball floating in the air. At the end, we compare, and say the advantages and disadvantages of diffrent animation methods for the built kinematic human model.

**Keywords:** kinematic structure, simple kinematic chains, DOF – „Degree of freedom“, direct kinematics, inverse kinematics, CCD method, Jacobian matrix, C++, OpenGL