SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 866

INVERZNA KINEMATIKA OSTVARENA FABRIK METODOM

Fran Ogrinšak

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 866

INVERZNA KINEMATIKA OSTVARENA FABRIK METODOM

Fran Ogrinšak

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 10. ožujka 2023.

ZAVRŠNI ZADATAK br. 866

Pristupnik:	**Fran Ogrinšak (0036523945)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentorica:	prof. dr. sc. Željka Mihajlović

Zadatak: **Inverzna kinematika ostvarena FABRIK metodom**

Opis zadatka:

Proučiti postupke implementacije inverzne kinematike. Posebice obratiti pažnju na metodu FABRIK. Izraditi realistične animacije pomicanja kinematičkog lanca inverznom kinematikom uz implementaciju metode FABRIK. Diskutirati utjecaj različitih parametara. Načiniti ocjenu rezultata i implementiranih algoritama. Izraditi odgovarajući programski proizvod. Koristiti grafičko programsko okruženje Unity. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 9. lipnja 2023.

Sadržaj

Uvod

Računalna animacija u današnjem svijetu ima široke primjene. Animiranje ljudskih modela ili bilo čega što se može opisati kinematičkim lancima, nije uvijek jednostavno te ako se primjenjuje direktna kinematika može biti i dugotrajno. Tu ulazi inverzna kinematika koja olakšava cjelokupni proces animiranja.

Problem inverzne kinematike nije jednostavan te ga se vrlo često ne može analitički riješiti. Vrlo su zastupljene heurističke metode koje jednostavnim algoritmima pokušavaju postići realistične poze animiranih modela.

Cilj ovog rada jest predstaviti probleme direktne i inverzne kinematike te predstaviti neke od najčešće korištenih metoda. U sklopu ovog rada su implementirane metode CCD i FABRIK te demonstrirane na jednom kinematičkom lancu.

FABRIK metoda obećava prirodne poze te mogućnost primjene na vrlo širok raspon modela. Također ima jako dobre vremenske performanse naspram konkurentnih metode, te je vrlo jednostavna za implementirati.[2]

1. Kinematika

Kinematika je grana fizike koja se bavi proučavanjem gibanja točke ili skupa točaka kroz prostor ne uzimajući u obzir sile koje utječu na njihovo gibanje. U ovom radu će biti fokus na proučavanju kretanja kinematičkog lanca.

1.1. Kinematički lanac

U stvarnom životu objekte kao što su ljudi ili roboti se mogu modelirati jednim ili više kinematičkih lanaca. Sam pojam kinematičkog lanca se pojavljuje u robotici te se definira kao niz međusobno spojenih zglobova. Na zglob može ili ne mora biti spojen segment koji ima zglob na kraju. Na kraju segmenta se obično nalazi drugi zglob ali ako je odsutan onda se smatra krajnjim manipulatorom. Na primjer na vrhu robotske ruke bi se nalazila kliješta ili neki drugi alat s kojim robot manipulira okolinu. U računalnoj animaciji često je zanimljiva pozicija krajnjeg manipulatora.

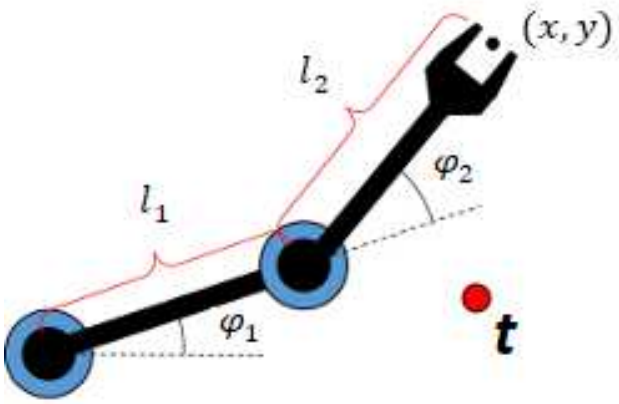
Postoji puno vrsta različitih zglobova kao što su rotacijski i prizmatični, a u sklopu ovog rada ćemo se baviti rotacijskim zglobovima koji imaju 3 stupnja slobode.

1.2. Direktna kinematika

U problemu direktne kinematike zadani su parametri kinematičkog lanca, najčešće su to relativni kutovi i duljine segmenta svakog manipulatora. Direktna kinematika se koristi u situacijama kada je poznato kako se lanac mora ponašati. Na primjer u animaciji za mahanje ruke bi lagano bilo animirati koristeći direktnu kinematiku ili bilo kakve animacije za čije su pokreti uvijek isti. Konkretno za slučaj lanca koji se sastoji od dva manipulatora u xy ravnini želimo naći preslikavanje sljedećeg oblika[3][4]:

$$\mathbf{P}(\varphi_1, \varphi_2) = \boldsymbol{p} = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} l_1 \cos(\varphi_1) + l_2 \cos(\varphi_1 + \varphi_2) \\ l_1 \sin(\varphi_1) + l_2 \sin(\varphi_1 + \varphi_2) \end{bmatrix} \tag{1.1}$$

Gdje $\boldsymbol{p}$ predstavlja vektor pozicije vrha krajnjeg manipulatora. Kut φ_1 je kut prvog manipulator a φ_2 krajnjeg, te su l_1 i l_2 pripadne dužine segmenata. Situacija je prikazana na slici (Sl. 1.1).



Sl. 1.1 Planarni maniuplator [4]

Za opći slučaj od **m** zglobova u **n** dimenzija:

$$\mathbf{P}(\varphi_1, \dots, \varphi_n) = [x_1, \dots, x_m] \tag{1.2}$$

Ukupnu transformaciju koja preslikava točku iz koordinatnog sustava vrha krajnjeg manipulatora u globalni je sljedeća:

$$\mathbf{T} = \prod_{i=1}^{n-1} (T_i R_i) T_n \tag{1.3}$$

Gdje je T_i translacijska matrica za i-ti zglob a R_i rotacijska.

1.3. Inverzna kinematika

Problem inverzne kinematike je suprotan od direktne. Umjesto da se računa pozicija krajnjeg manipulatora u globalnom koordinatnom sustavu cilj je pronaći konfiguraciju koja će postaviti krajnji manipulator u traženu poziciju ***t***. IK se koristi kada je potrebno animirati kretanje koje je dinamički generirano, na primjer u računalnim igrama kada model igrača hoda po neravnoj površini te poziciju stopala treba dinamički promijeniti da bude prirodno priljubljeno uz površinu. Također se može koristiti u robotici za pozicioniranje robotske ruke u dinamičnoj okolini.

Preslikavanje je inverzno s obzirom na (1) te su rješenja složenija:

$$\mathbf{P^{-1}}(x, y) = \boldsymbol{p} = \begin{bmatrix} \varphi_1 \\ \varphi_2 \end{bmatrix}$$

(1.4)

Rješenja predstavljena u [1]:

$$\varphi_1 = \cos^{-1}\left(\frac{l_1^2 + x^2 + y^2 - l_2^2}{2l_1\sqrt{x^2 + y^2}}\right)$$

(1.5)

$$\varphi_2 = \cos^{-1}\left(\frac{l_1^2 + l_2^2 - (x^2 + y^2)}{2l_1 l_2}\right)$$

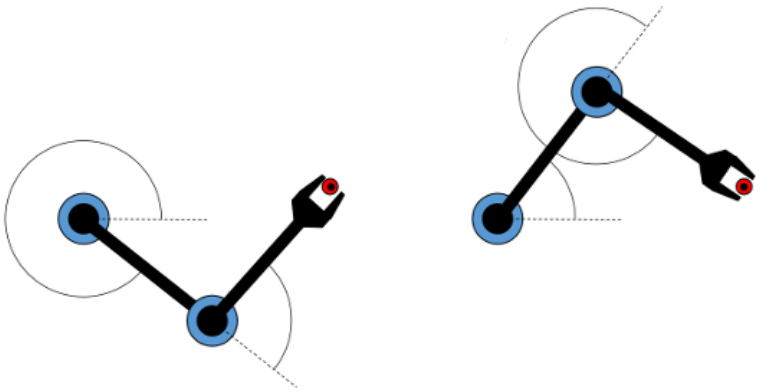
(1.6)

Analogno kao u formuli (1.2) opći slučaj je slično definiran:

$$\mathbf{P^{-1}}(x_1, \dots, x_m) = [\boldsymbol{\varphi_1}, \dots, \boldsymbol{\varphi_n}]$$

(1.7)

Kao što se vidi u (1.1) u izrazu se sadrži funkcije sinus i kosinus koje su nelinearne te je $\mathbf{P^{-1}}$ zbog tog razloga vrlo teško izračunati. Također neko rješenje ne mora biti jedinstveno za neki ulaz, na slici (Sl. 1.2) su prikazana dva moguća rješenja za istu traženu poziciju.



Sl. 1.2 Dva rješenja za istu poziciju [4]

1.3.1. Radni prostor IK

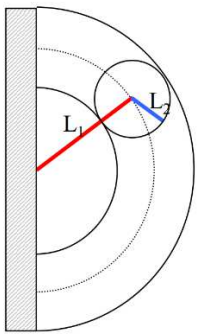
Jednadžba (1.2) nema uvijek rješenja, konkretno kada su x i y koordinate izvan radnog radnoga prostora. Što znači da su ukupne duljine segmenta previše male ili je prvi segment previše dugačak. Radni prostor se definira kao skup ulaza za IK problem za koji postoji rješenje. Konkretno za primjer iz 1.2 radni prostor čine x i y koji zadovoljavaju sljedeću nejednakost:

$$l_1 - l_2 \leq x^2 + y^2 \leq l_1 + l_2 \tag{1.8}$$

Za opći slučaj gdje je $\boldsymbol{t}$ radij vektor mete:

$$l_1 - \sum_{j=2}^{n} l_i \leq \|\boldsymbol{t}\| \leq \sum_{j=1}^{n} l_i \tag{1.9}$$

Naravno taj slučaj ne vrijedi kada su pokreti svakog zgloba ograničeni, onda je u općem slučaju teško odrediti radni prostor. Na slici (Sl. 1.3) je vizualiziran slučaj za n = 2.



Sl. 1.3 Radni prostor planarnog manipulatora [3]

2. Metode u inverznoj kinematici

Postoje različite metode rješavanja problema IK. Idealni način je pronaći neku zatvorenu formulu u koju je potrebno unijeti parametre. Takva analitička rješenja su egzaktna i brza računalu za izvesti, ali za složene kinematičke lance analitičko rješenje ne postoji ili ga je jako teško pronaći. Time su istraživači prispjeli numeričkim i heurističkim metodama koje su iterativne i time sporije od analitičkih, ali su zato široko primjenjive i često jednostavne za implementaciju. U nastavku su predstavljene jedne od najpopularnijih.[1]

2.1. Jacobijeve metode

Jacobijeve metode koriste Jacobijevu matricu iterativno kako bi pronašle konfiguraciju koja je dovoljno blizu željenom rješenju. Ideja metode je slična kao gradijentni spust, gledaju se promjene kuta svakog zgloba te se na temelju te promjene računa udaljenost od cilja koja se nastoji minimizirati. Sve kreće od Jacobijeve matrice koja sadrži parcijalne derivacije pozicije vrha manipulatora koje ovise o kutovima zglobova. Matrica se pojavljuje u derivaciji formule (2):

$$\mathbf{P}(\mathbf{\Phi})' = \mathbf{J}(\mathbf{\Phi})\mathbf{\Phi}'$$

(2.1)

Gdje $\mathbf{\Phi}$ predstavlja vektor konfiguracije lanca, a $'$ predstavlja derivaciju s obzirom na vrijeme. Označimo $\mathbf{p}$ s radij vektorom manipulatora. Svaki element matrice se definira kao:

$$\mathbf{J}(\Phi)_{ij} = \frac{\partial \mathbf{p}_i}{\partial \mathbf{\Phi}_j}$$

(2.2)

Da se dobiju kutovi za koje se treba pomaknuti koristi se aproksimacija:

$$\Delta\mathbf{p} \approx \mathbf{J}(\mathbf{\Phi})\Delta\mathbf{\Phi}$$

(2.3)

Iz (2.3) direktno slijedi množenjem inverza matrice slijeva:

$$\Delta\mathbf{\Phi} \approx \mathbf{J}(\mathbf{\Phi})^{-\mathbf{1}}\Delta\mathbf{p}$$

(2.3)

Metode koje koriste Jacobijevu matricu imaju problem s pronalaženjem inverza zato što matrica ne mora nužno biti invertibilna te se pojavljuju singulariteti kada kraj manipulatora ne može doći do cilja promjenom kutova zglobova. Također su sporije od heurističkih.[1][3]

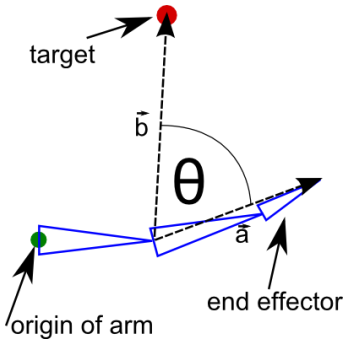
2.2. Ciklički koordinatni spust

Jedna od češće korištenih heurističkih metode za IK je ciklički koordinatni spust. Algoritam rješava problem tako da se računa kut između vrha manipulatora i cilja s obzirom na trenutno promatrani zglob. Slika (Sl. 2.1) to demonstrira. Pošto je algoritam heuristički mora imati zadan neki korisničko zadani parametar, konkretno u ovom algoritmu je to udaljenost između vrha manipulatora i mete. Označimo s `joints` listu svih zglobova te neka se na zadnjem mjestu nalazi vrh manipulatora. U nastavku je priložen pseudo kod jedne iteracije algoritma:

```
For i = n - 1 To 1
      toEnd =  endEffector – joints[i]
      toTarget = target – joints[i]
      rotation = getRotation(toEnd, toTarget)
      rotate(joints[i], rotation)
EndFor
```

Kôd 2.1 – Pseudokod iteracije CCD algoritma

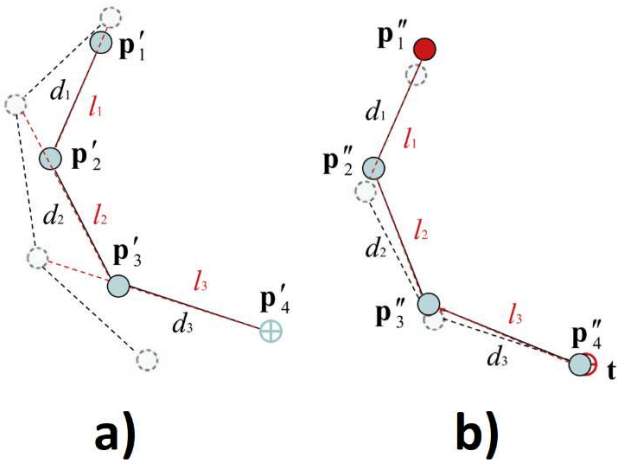
Kao što vidimo jedna iteracija je vrlo jednostavna, nakon što se odrede vektori `toEnd` i `toTarget` potrebno je samo rotirati zglob, sam kut rotacije se može odrediti koristeći skalarni produkt. `getrotation` vraća transformaciju koju je potrebno primijeniti na i-ti zglob tako da se kut između prethodna dva vektora bude 0. `rotate` samo promjeni tu rotaciju. Na kraju iteracije je potrebno provjeriti je li udaljenost između mete i vrha manipulatora ispod prethodno definiranog praga, ako jest algoritam se zaustavlja. Često se uvede i ograničenje na broj iteracija ako je prag previše malen. CCD metoda može proizvesti neprirodne konfiguracije lanca što može biti problem ako se koristi za postavljanje pozicija i pokreta ljudskih modela.[1][3]



Sl. 2.1 Prikaz kuta koji CCD mora minimizirati [5]

2.3. FABRIK algoritam

FABRIK(eng. *Forward And Backward Reaching Inverse Kinematics*) algoritam također spada u skupni heurističkih metoda te koristi isti iterativni princip kao CCD gdje je udaljenost od cilja nakon svake iteracije vrlo često manja od prošle. Kao parametri se isto kao i kod CDD-a zadaju tolerancija i broj iteracija. Jedna iteracija algoritma se sastoji od dva dijela. FABRIK ne barata s kutovima, nego samo s pozicijama zglobova. U prvom dijelu rezultat su nove pozicije svakog zgloba, to uključuje i bazni zglob koji inače treba biti fiksan, stoga su ove pozicije privremene. Rezultat te iteracije je prikazan na slici (Sl. 2.2 a). U drugom koraku se svaki zglob orijentira prema poziciji koje su nastale u prošlom djelu, prikazano na slici (Sl. 2.2 b).[2]



Sl. 2.2 a) Rezultat prvog djela FABRIK iteracije b) rezultat drugog dijela [2]

Kôd 2.2 je pseoudokod jedne iteracije. Cache lista predstavlja pohranu privremenih pozicija

```
cache[n - 1] = target
For i = n - 1 To 1
      toNextLength = magnitude(cache[i + 1] - joints[i])
```

```
        scalar = lengths[i] /toNextLength
        cache[i] = (1 – scalar)*cache[i+1] + scalar*joints[i]
EndFor
For i = 1 To n - 1
        toCached = cache[i + 1] -  joints[i]
        segment = joints[i + 1] – joints[i]
        rotation = getRotation(segment, toCached)
        rotate(joints[i], rotation)
EndFor
```

Kôd 2.2 – Pseudokod iteracije FABRIK algoritma

U prvom dijelu se prvo kao novi kraj manipulatora stavi pozicija mete. Privremene pozicije se prate u listi `cache` Nakon toga se kreće od zadnjeg zgloba prema prvom. Pri svakom koraku se na spojnicu trenutnog i prethodno izračunate privremene pozicije zgloba postavlja nova pozicija na udaljenost koja odgovora duljini segmenta za taj zglob. U drugom dijelu se obavlja sličan posao kao i kod CCD, samo se smanjuje kut između smjera segmenta i smjera prema odgovarajućoj poziciji iz prvog dijela.

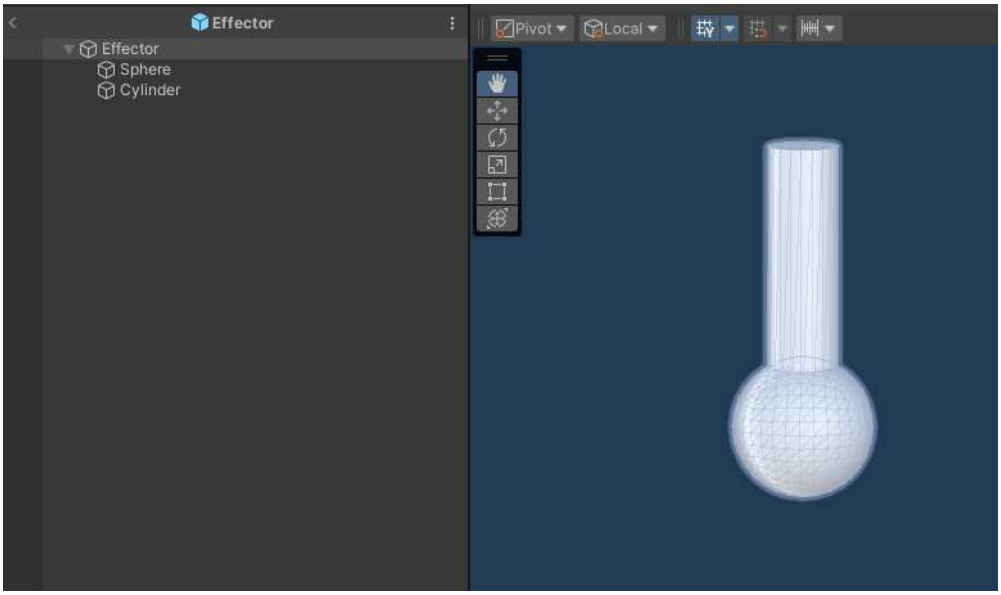
FABRIK vrlo brzo nalazi zadovoljavajuću konfiguraciju te najbitnije daje prirodne pokrete te nema problema ako se nova meta nalazi na lancu. Također za razliku od CCD-a može se prilagoditi da funkcionira za složene lance, gdje postoje ciklusi i postoji više krajnjih manipulatora, na primjer artikulirani model ljudskog tijela. Moguće je implementirati rotacijska i prostorna ograničenja za svaki zglob.[2]

3. Implementacija

Simulacija kinematičkog lanca je implementirana u grafičkom pogonu Unity. Sav kod je pisan u jeziku C# koji Unity podržava.

3.1. Objekti kinematičkog lanca

Kinematički lanac se promatra kao niz čvrstih segmenata i zglobova. Zglob i segment su predstavljeni `Effector` prefabom. Sastoji se od sfere i cilindra predodređene duljine. Algoritmi koji mijenjaju stanje lanca samo mijenjaju rotation svojstvo svakog `Effector` objekta. Svaki `Effector` objekt sadrži u sebi drugi `Effector` objekt, i to vrijedi za svaki objekt osim za posljednji koji predstavlja vrh manipulatora i modeliran je samo sferom. Posljedica ovakve stablaste konstrukcije jest da se ne mora mijenjati lokalna pozicija svakog zgloba, nego promjena rotacije jednog zgloba, mijenja pozicije ostalih koji se nastavljaju na trenutni. Prikaz `Effector` objekta je na slici (Sl. 3.1).



Sl. 3.1 Effector prefab

3.2. Implementacija algoritma

U sklopu ovog završnog rada implementirani su algoritmi FABRIK i CCD. Klasa `Chain` sadrži algoritme koji rješavaju problem IK, također klasa sadrži parametre kao što su maksimalni broj iteracija i tolerancija. Lista `effectors` sadrži listu svih `Effector`

objekata, te njihova pozicija odgovara poziciji jednog zgloba. Oba dva algoritma su jako slična, te imaju isti kod za provjeru je li meta unutar radnog prostora. U nastavku je prikazan kod za konfiguriranje lanca kad je meta izvan radnog prostora, slučaj kada je manipulator prekratak:

```
float totalLength = 0f;
for (int i = 0; i < effectors.Count; ++i)
totalLength += effectors[i].GetComponent<Effector>().length;

float distance = (targetPosition.position -
startJoint.position).magnitude;
if (distance > totalLength)
{
      for (int i = 0; i < effectors.Count - 1; ++i)
      {
            var joint = effectors[i];
            var rotateTo = target.transform.position -
            joint.transform.position;
            joint.transform.rotation =
            Quaternion.FromToRotation(Vector3.up, rotateTo);
      }
}
```

Kôd 3.1 – Slučaj izvan radnog prostora

Prvo se računa ukupna duljina lanca `distance`, te se ta duljina koristi za provjeru dohvatljivosti mete. Ako je udaljenost vrha manipulatora veća od ukupne duljine potrebno je poravnati svaki zglob da pokazuje u smjeru mete. To se postiže klasom `Quaternion` i metodom `FromtoRotation`. Ako je meta unutar radnog prostora onda se za CCD izvodi sljedeći isječak `PerfromCCD` metode:

```
for (int i = effectors.Count - 2; i >= 0; --i)
{
      var joint = effectors[i].transform;
      var toTarget = target.transform.position -
      joint.position;
      var toEnd = effectors[effectors.Count -
      1].transform.position - joint.position;
      var rotation = Quaternion.FromToRotation(toEnd,
      toTarget);
      var targetRotation = rotation * joint.rotation;
      var startRot = joint.rotation;
```

```
        if (difference > 0.1 && !continuous)
        yield return InterpolateRotation(joint, startRot,
        targetRotation);
        else
        joint.rotation = rotation * joint.rotation;
}
```

Kôd 3.2 – Implementirana iteracija CCD algoritma

Obavljanje jedne iteracije počinje s određivanjem vektora upisanih u poglavlju 2.2. Nakon toga je potrebno izračunati tu rotaciju, no ta rotacija nije nova konfiguracija trenutnog zgloba, nego je potrebno pomnožiti trenutnu rotaciju s novom.

Ovisno o tome je li korisnik zatražio kontinuirano izvođenje algoritma ili nije pozivaju se metode `InterpolateRotation` ili se odmah promjeni u novu konfiguraciju. U interpolaciji primjena svake rotacije se korisniku prikazuje usporeno.

U nastavku je isječak jedne FABRIK iteracije:

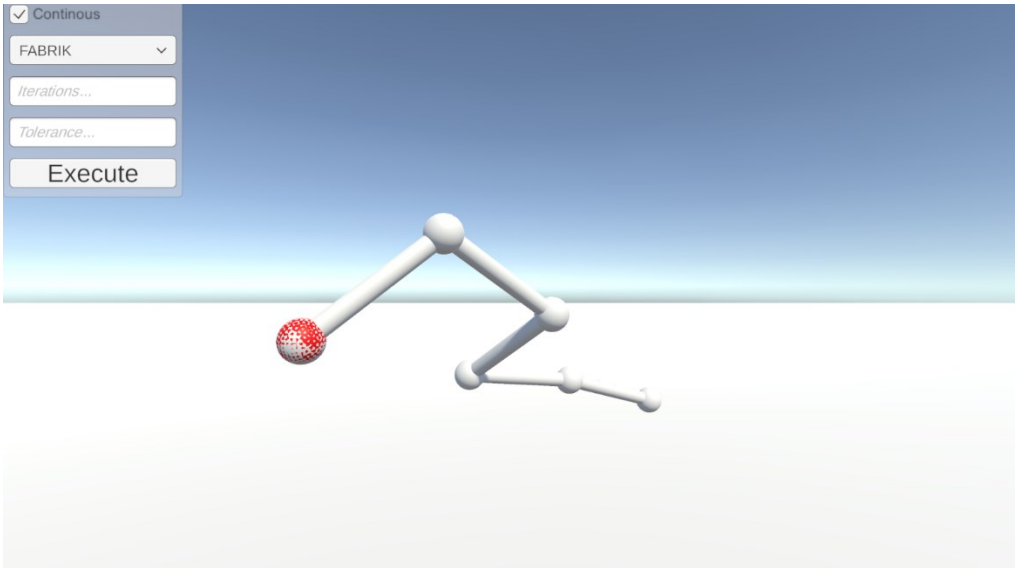
```
cache[effectors.Count - 1] = targetPosition.position;
var forwardsPositions = cache;
forwardsPositions[effectors.Count - 1] =
targetPosition.position;
for (int i = effectors.Count - 2; i >= 0; --i)
{
        float newPositionDistance = (forwardsPositions[i + 1]
        - effectors[i].transform.position).magnitude;
        float scalar =
        effectors[i].GetComponent<Effector>().length /
        (float)newPositionDistance;
        forwardsPositions[i] = (1 - scalar) * effectors[i +
        1].transform.position +
        scalar * effectors[i].transform.position;
}

if (difference > 0.1 && !continuous)
        yield return
        StartCoroutine(updatePositionsInterpolated());
else
        UpdatePositions();
```

Kôd 3.3 – Implementirana iteracija CCD algoritma

Pri stvaranju `Chain` objekta inicijalizira se `cache` lista za pohranu privremenih lokacija zglobova. Na početku svake iteracije na kraj liste se postavlja pozicija mete. Prvi dio se izvodi od zadnjeg zgloba do prvog, isto kao u kôdu 2.3. Nakon što su izračunate privremene pozicije drugi dio algoritma se može izvesti odmah ili interpolirano kako bi se ilustrirala primjena svih rotacija u svakoj iteraciji.

4. Rezultati

Rezultat je simulacija kinematičkog lanca koji se sastoji od 5 zglobova i vrha manipulatora. U Unity uređivaču se može podesiti lanac koristeći `SceneManager` objekt koristeći njegov kontekstni meni. Na slici (Sl. 4.1) je prikazana inicijalna konfiguracija scene.



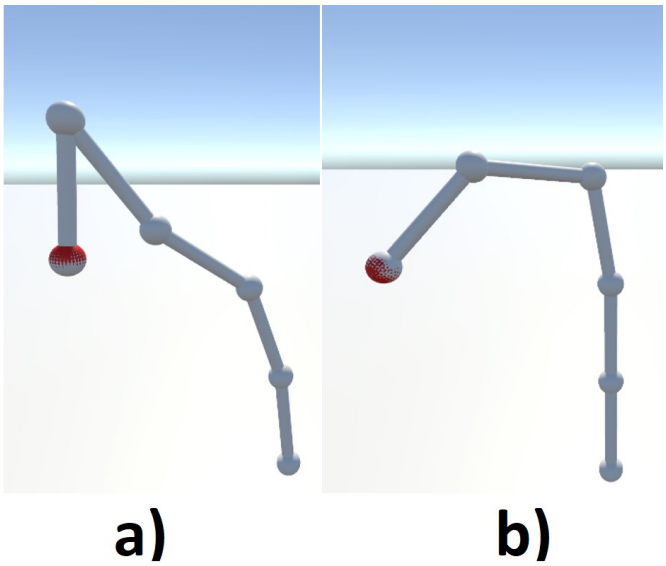
Sl. 4.1 Inicijalni prikaz

Korisnik pritiskom tipke C može postaviti lanac u ispruženu poziciju u +y smjeru, pritiskom tipke R postavlja metu u početnu poziciju i pritiskom tipke V postavlja kameru u početnu poziciju. Micanje mete se ostvaruje držanjem lijevog klika miša te micanjem kotačića se udaljava ili približava od kamere. Držanjem desnog klika se ostvaruje rotacija kamere.

Postoji korisničko sučelje u gornjem lijevom kutu ekrana gdje korisnik može birati koji algoritam se koristi, te podesiti broj iteracija i toleranciju. Sučelje je prikazano na slici (Sl. 4.2). Gumb Execute pokreće odabrani algoritam, a zastavica Continous određuje je li se algoritam izvede svaki put kada korisnik pomakne metu ako je kvačica postavljena, ili se izvede interpolirano samo kad korisnik pritisne gumb Execute.

Sl. 4.2 Korisničko sučelje za promjenu parametara

Pri kontinuiranom praćenju mete FABRIK daje prirodnije konfiguracije lanca od CCD metode. Također FABRIK se jako dobro nosi sa situacijama kada se meta nalazi na lancu ili je cijeli lanac ili dio lanca gotovo ispružen. Lanac je ispružen kada su smjerovi y osi svakog vrha gotovo kolinearne u globalnom sustavu scene. CCD često ima u problema s takvim situacijama zato što dosegne maksimalni broj iteracija prije nego što je zadovoljio toleranciju, ako je razumno mala. Ispod 0.05 za konkretni lanac u programu. Broj iteracija je dovoljno postaviti na 20 za zadovoljavajući rezultat, a za interpolirani način rada može potrebno biti više, ovisno o prethodnoj konfiguraciji. Na slici (Sl. 4.3) su prikazane usporedbe konfiguracija za istu situaciju, prijašnja pozicija je uspravni lanac.



Sl. 4.3 a) Rezultat CCD algoritma b) Rezultat FABRIK algoritma

4.1. Poboljšanja

Ovakva implementacija IK se rijetko koristi u praksi. Gotovo uvijek je potrebno zadati rotacijska i prostorna ograničenja. Korisničko sučelje bi se moglo nadograditi tako da se doda mogućnost zadavanja proizvoljnog lanca i ograničenja. FABRIK algoritam ima sposobnost rada sa složenim artikuliranim figurama koje nisu samo lanci, tako da je implementacija toga moguće poboljšanje. Konačno kôd za implementirane IK algoritme u ovom radu bi se mogao poopćiti zato što imaju slične ili iste dijelove.

Zaključak

Inverzna kinematika je težak i bitan problem u računalnoj animaciji. U sklopu ovog rada predstavljene su poznate metode rješavanja IK. Algoritmi koji su implementirani su heuristički te uspješno rješavaju IK za jedan kinematički lanac. FABRIK se iskazao kao boljom metodom CCD metode uglavnom zbog prirodnih poza ali ima i boje vremenske performanse. Uspješno su implementirane metode koje korisniku kroz jednostavno sučelje prikazuju kontinuirano ili interpolirano praćenje mete.

Literatura

[1] Aristidou, A., Lasenby, J., Chrysanthou, Y., & Shamir, A. Inverse Kinematics Techniques in Computer Graphics: A Survey. U Computer Graphics Forum. Wiley. (2017). Sv. 37, Issue 6. str. 35–58. http://www.andreasaristidou.com/publications/papers/IK_survey.pdf

[2] Aristidou, A., & Lasenby, J. FABRIK: A fast, iterative solver for the Inverse Kinematics problem. U Graphical Models. Elsevier BV. (2011). Sv. 73, Issue 5. str. 243–260. http://www.andreasaristidou.com/FABRIK.html

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

[4] Kris Hauser, Chapter 6. Inverse Kinematics, 28.12.2020, *Robotic Systems*, https://motion.cs.illinois.edu/RoboticSystems/InverseKinematics.html

[5] Inverse Kinematics For Virtual Robot Arm, https://jak-o-shadows.users.sourceforge.net/python/robot/armik.html

Sažetak

INVERZNA KINEMATIKA OSTVARENA FABRIK METODOM

U sklopu ovog završnog rada opisani su problemi inverzne i direktne kinematike. Predstavljene su poznate metode za rješavanje inverzne kinematike. Opisana je implementacija kinematičkog lanca koji je animiran FABRIK i CCD metodama. Za izradu rada korišten je grafički pogon Unity te su skripte za algoritme napisane koristeći programski jezik C#.

Ključne riječi: inverzna kinematika, Unity, FABRIK, CCD

Summary

INVERSE KINEMATICS USING THE FABRIK METHOD

In this final paper, the problems of inverse and direct kinematics are described. Known methods for solving inverse kinematics are presented. Implementation of the kinematic chain which is animated using the FABRIK and CCD methods is described. Graphics engine unity was used to create this paper and algorithms were written using the programming language C#.

Keywords: inverse kinematics, Unity, FABRIK, CCD

Skraćenice

IK *Inverzna kinematika*

FABRIK *Forward And Backward Reaching Inverse Kinematics*

CCD *Cyclic coordinate descent*

Privitak

Poveznica na izvorni kod:

https://gitlab.com/FranGT/zr