Elektrotehnički fakultet u Sarajevu
Odsjek za Automatiku i elektroniku
Predmet: Digitalni računari i organizacija softvera
Nastavnik: Prof. dr Dušanka Bošković

SEMINARSKI RAD

Student: Mirza Balta

Naslov rada**: Optimizacijski pristup i primjena BFGS algoritma u rješavanju problema inverzne kinematike robotskog manipulatora**

Sarajevo, januar 2021. godine

Uvod

U ovom radu izložen je optimizacijski pristup u rješavanju problema inverzne kinematike robotskog manipulatora i implementacija u programskom jeziku C++ jedne od metoda rješavanja tog problema. Kod ovog pristupa problem inverzne kinematike se posmatra kako problem optimizacije sa ograničenjima, gdje su ograničenja određena donjom i gornjom granicom pokretljivosti zglobova, i kao takva data ograničenja su linearna, a skup dozvoljnenih vrijednosti je konveksan skup.

Izbor funkcije cilja čiju optimalnu, u ovom slučaju minimalnu vrijednost tražimo, je ključan. Ona je izabrana u obliku pozitivno definitne kvadratne forme koju figurišu greška pozicije i greška orijentacije vrha manipulatora, te matrica težinskih kojeficijenata koje određuju doprinose komponenti greški po koordinatama funkciji kriterija. Greška pozicije (vektor) je određena kao razlika vektora željene i trenutne pozicije vrha manipulatora. Greška orijentacije je određena relativno složenije i moguće je izabrati na više načina. U ovom radu ona je određena kao matrica koja opisuje rotaciju koja je potrebna da se željena orijentacija vrha manipulatora podudari sa trenutnom orijentacijom. Ova matrica koja određuje grešku orijentacije je prevedena u formalizam kvaterniona, pa je prema tome i u vezi toga izloženim razmatranjima greška orijentacije određena vektorskim dijelom tog kvaterniona. Matrica, odnosno matrice težinskih kojeficijenata su izabrane kao (pozitivno definitne) dijagonalne matrice čime greška pozicije i greška orijentacije uzimaju oblik Euklidske norme (rastojanja).

Ovdje su trenutna pozicija i trenutna orijentacija vrha manipulatora funkcije vektora zglobnih varijabli. Prema tome one su određene jednačinom direktne kinematike, pa je zato u prvom poglavlju nastojano da se dovoljno detaljno opiše jednačina direktne kinematike i standardni način njenog određivanja preko Denevit-Hartembergovih parametara.

Jednodimenzionalno pretraživanje po pravcu, kojim se nalazi optimalna dužina koraka, je sastavni dio nekih od najefikasnijih metoda pretraživanja, pa su u drugom dijelu rada navedene neke od metoda jednodimenzionalnog pretraživanja. Da bi se stekao uvid u izbor pravca jednodimenzionalnog pretraživanja kod ovih metoda, izložene su neke osobine koje se odnose na pravac pretraživanja, koje slijede iz Taylorove teoreme. Time se na izbor različitih metoda pretraživanja od kojih su neke opisanje, može gledati kao na izbor pravca pretraživanja.

BFGS (Broyden-Fletcher-Goldberg-Shanno) kao iterativni metod nelinearne optimizacije bez ograničenja je ključan za teorijski i praktični dio rada pa je ovaj metod detaljno opisan i izveden. Ovaj metod određuje pravac pretraživanja preduslovljavajući gradijent kriterija (a time i pravac pretraživanja) u tekućoj iteraciji sa zakrivljenošću mjerenu tokom prethodnih iteracija. BFGS je izveden opet kao rješenje optimizacijskog problema: *da je minimalna* Frobenijusova matrična norma razlike simetrične matrice koja zadovoljava jednačinu sekante i matrice aproksimacije inverznog Hessiana za tekuću iteraciju. Kao rješenje ovog problema dobije se formula za ažuriranje matrice aproksimacije inverznog Hessiana. Ova matrica umjesto da se nanovo računa kao u Newtonovoj metodi, računa se na jednostavniji način uzimajući u obzir zakrivljenost mjerenu tokom prethodnih iteracija i uz to se još izbjegava računanje inverzne matrice. Kao generalizacija BFGS metoda, navedena je jedna šira (Broydenova) klasa kojoj pripada ovaj i srodne metode (DFP). Navedene su teoreme koje obezbjeđuju dovoljne uslove konvergencije i brzine konvergencije niza kojeg generiše ovaj metod.

U praktičnom dijelu rada primijenjen je BFGS algoritam u rješavanju problema inverzne kinematike robota. Ovaj algoritam je implementiran u programskom jeziku C++, u obliku S-funkcije koja je kao blok inverzne kinematike primijenjena u simulaciji koja je rađena u Matlab Simulink-u. U ovoj simulaciji je problem inverzne kinematike tretiran kao problem optimizacije bez ograničenja, čime je omogućena direktna primjena BFGS metode u rješavanju ovog problema, dok je u realnim situacijama potrebno uzeti ograničenja pokretljivosti zglobova robota. Stoga je u dodatku B naveden algoritam koji je kombinacija BFGS metoda i metoda projekcije gradijenta, a koji uzima u obzir i ova ograničenja.

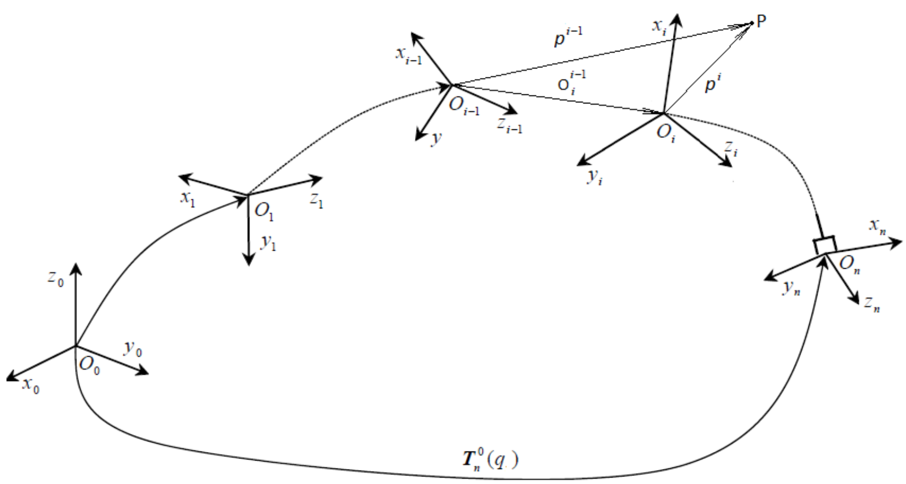
1. Kinematika robota

1.1 Funkcija direktne kinematike

Svaki zadatak koji robot obavlja može se svesti na ispravno pozicioniranje i orijentaciju vrha manipulatora, dok izvršenje zadatka obavljaju aktuatori koji pokreću zglobove pri tome mijenjajući varijable zglobova. Funkcija direktne kinematike izražava zavisnost pozicije i orijentacije vrha manipulatora od varijabli zglobova. Pozicija vrha manipulatora u odnosu na koordinatni sistem baze je određena vektorom $\boldsymbol{p}(\boldsymbol{q})$, gdje je $\boldsymbol{q} = \left[q_1\ q_2 \ldots q_n\right]^T$ *vektor zglobnih varijabli*. Orijentacija vrha manipulatora u odnosu na koordinatni sistem baze je određena matricom $\boldsymbol{R}_n^0(\boldsymbol{q}) = \left[\boldsymbol{n}(\boldsymbol{q})\ \boldsymbol{s}(\boldsymbol{q})\ \boldsymbol{a}(\boldsymbol{q})\right]$, gdje su $\boldsymbol{n}$, $\boldsymbol{s}$, $\boldsymbol{a}$ vektori kolone matrice $\boldsymbol{R}_n^0$ i predstavljaju jedinične vektore koordinatnog sistema pridruženog vrhu manipulatora. *Funkcija direktne kinematike* se formalno izražava preko matrice homogene transformacije, jednačinom:

$$\boldsymbol{T}_n^0\left(\boldsymbol{q}\right) = \begin{bmatrix} \boldsymbol{R}_n^0(\boldsymbol{q}) & \boldsymbol{p}(\boldsymbol{q}) \\ \boldsymbol{0}^T & 1 \end{bmatrix} \tag{1.1}$$

U definisanju značenja i konkretnog analitičkog izraza za ovu funkciju, robot se modelira sistemom krutih tijela koje zovemo *segmenti robota* (manipulatora), koji su povezani preko rotacijskih ili translacijskih zglobova. Cijela struktura formira *kinematički lanac*, za kojeg kažemo da je *otvoren* ako postoji samo jedan slijed segmenata koji povezuju dva kraja kinematičkog lanca. Svakom segmentu je pridružen odgovarajući koordinatni sistem.



Slika 1.1 Kinematička struktura robota posmatrana kao otvoreni kinematički lanac i tačka P posmatrana u odnosu na koordinatne sisteme pridružene i-1 –vom i i -tom segmentu

Posmatrajmo i-1-vi i i-ti segment kojima su pridruženi koordinatni sistemi $O_{i-1} - x_{i-1}y_{i-1}z_{i-1}$ $O_i - x_i y_i z_i$ respektivno. Neka i -ti segment u odnosu na i-1-vi segment ima trenutni položaj orijentaciju. Na slici 1.1. je $\boldsymbol{p}^{i-1}$ -vektor položaja tačke P izražen u koordinatama $O_{i-1} - x_{i-1}y_{i-1}z_{i-1}$ koordinatnog sistema, $\boldsymbol{p}^i$ - *vektor položaja* tačke P izražen u koordinatama $O_i - x_i y_i z_i$, $\boldsymbol{o}_i^{i-1}$ -vektor položaja ishodišta O_i izražen u koordinatama $O_{i-1}-x_{i-1}y_{i-1}z_{i-1}$. Za specijalni slučaj kada se ishodišta O_{i-1} i O_i poklapaju imamo da je :

$$\boldsymbol{p}^{i-1}=\boldsymbol{p}_x^i\boldsymbol{x}_i^{i-1}+\boldsymbol{p}_y^i\boldsymbol{y}_i^{i-1}+\boldsymbol{p}_z^i\boldsymbol{z}_i^{i-1}=\left[\boldsymbol{x}_i^{i-1}\ \boldsymbol{y}_i^{i-1}\ \boldsymbol{z}_i^{i-1}\right]\boldsymbol{p}^i=\boldsymbol{R}_i^{i-1}\boldsymbol{p}^i$$

$$\boldsymbol{p}^{i-1}=\boldsymbol{R}_i^{i-1}\boldsymbol{p}^i \tag{1.2}$$

$$\boldsymbol{p}^i=\left(\boldsymbol{R}_i^{i-1}\right)^{-1}\boldsymbol{p}^{i-1}=\left(\boldsymbol{R}_i^{i-1}\right)^T\boldsymbol{p}^{i-1} \tag{1.3}$$

Gdje su:

- $\boldsymbol{x}_i^{i-1},\boldsymbol{y}_i^{i-1},\boldsymbol{z}_i^{i-1}$ jedinični vektori sistema $O_i-x_iy_iz_i$ izreženi u sistemu $O_{i-1}-x_{i-1}y_{i-1}z_{i-1}$,
- $\boldsymbol{p}^i=\left[\boldsymbol{p}_x^i\ \boldsymbol{p}_y^i\ \boldsymbol{p}_z^i\right]^T$ vektor pozicije tačke P čije su koordinate izražene u koordinatnom sistemu $O_i-x_iy_iz_i$,
- $\boldsymbol{R}_i^{i-1}$ matrica koja prevodi koordinate tačke P izražene u $O_i-x_iy_iz_i$ koordinatnom sistemu u koordinate iste tačke izražene u sistemu $O_{i-1}-x_{i-1}y_{i-1}z_{i-1}$, ona ima osobinu ortogonalnosti $\left(\boldsymbol{R}_i^{i-1}\right)^{-1}=\left(\boldsymbol{R}_i^{i-1}\right)^T$ i predstavlja *matricu rotacije.*

U slučaju da se O_{i-1} i O_i ne poklapaju, prema slici 1.1 imamo da je:

$$\boldsymbol{p}^{i-1}=\boldsymbol{o}_i^{i-1}+\boldsymbol{R}_i^{i-1}\boldsymbol{p}^i \tag{1.4}$$

Ova jednačina predstavlja transformaciju koordinata (translacija i rotacija) iz koordinatnog sistema $O_i-x_iy_iz_i$ u koordinatni sistem $O_{i-1}-x_{i-1}y_{i-1}z_{i-1}$. Da bi se postigla kompaktnost u notaciji uvodi se vektor $\tilde{\boldsymbol{p}}^i$ i *matrica homogene transformacije* $\boldsymbol{A}_i^{i-1}$:

$$\tilde{\boldsymbol{p}}^i=\begin{bmatrix}\boldsymbol{p}^i\\ 1\end{bmatrix}\quad,\qquad \boldsymbol{A}_i^{i-1}=\begin{bmatrix}\boldsymbol{R}_i^{i-1} & \boldsymbol{p}\\ \boldsymbol{0}^T & 1\end{bmatrix} \tag{1.5}$$

prema tome, transformacija koordinata (1.4) se može pisati u homogenom obliku:

$$\tilde{\boldsymbol{p}}^{i-1}=\boldsymbol{A}_i^{i-1}\tilde{\boldsymbol{p}}^i \tag{1.6}$$

Za razliku od matrice rotacije, matrica homogene transformacije ima osobinu:

$$\left(\boldsymbol{A}_i^{i-1}\right)^{-1}\neq\left(\boldsymbol{A}_i^{i-1}\right)^T \tag{1.7}$$

Primjenjujući jednačinu (1.6) na svaki segment, imamo:

$$\tilde{\boldsymbol{p}}^{n-1}=\boldsymbol{A}_n^{n-1}\tilde{\boldsymbol{p}}^n\quad,\tilde{\boldsymbol{p}}^{n-2}=\boldsymbol{A}_{n-1}^{n-2}\tilde{\boldsymbol{p}}^{n-1}\quad,\dots,\ \tilde{\boldsymbol{p}}^0=\boldsymbol{A}_1^0\tilde{\boldsymbol{p}}^1 \tag{1.8}$$

Sukcesivnim smjenama u gornjim jednačinama, dolazimo do izraza za složenu homogenu transformaciju:

$$\tilde{\boldsymbol{p}}^0=\boldsymbol{A}_1^0\boldsymbol{A}_2^1\dots\boldsymbol{A}_n^{n-1}\tilde{\boldsymbol{p}}^n \tag{1.9}$$

Ovo znaći da kompozicija pojedinačnih rotacija i translacija se može prikazati kao umnožak matrica osnovne homogene transformacije. Pri tome je $\boldsymbol{A}_i^{i-1}=\boldsymbol{A}_i^{i-1}(\boldsymbol{q}_i),\ \mathrm{i}=1,2,\dots,\mathrm{n}$, jer svaki pojedinačni zglob je translacijski ili rotacioni. Za translacijski (prizmatični) zglob je $q_i=d_i$, za rotacioni zglob je $q_i=\theta_i$.
Svaka od homogenih transformacija $\boldsymbol{A}_i^{i-1}(\boldsymbol{q}_i),\ \mathrm{i}=1,2,\dots,\mathrm{n}$ predstavlja položaj i orijentaciju koordinatnog sistema pridruženog i-tom segmentu u odnosu na koordinatni sistem pridružen i-1 –vom segmentu, pa prema tome kompozicija $\boldsymbol{A}_1^0(q_1)\boldsymbol{A}_2^1(q_2)\dots\boldsymbol{A}_n^{n-1}(q_n)$ predstavlja položaj i orijentaciju koordinatnog sistema pridruženog vrhu manipulatora u odnosu na koordinatni sistem baze. Dakle, vrijedi:

$$\boldsymbol{T}_n^0(\boldsymbol{q})=\boldsymbol{A}_1^0(q_1)\boldsymbol{A}_2^1(q_2)\dots\boldsymbol{A}_n^{n-1}(q_n) \tag{1.10}$$

Matrica rotacije $\boldsymbol{R}_i^{i-1}$ opisuje orijentaciju jednog koordinatnog sistema (a time i krutog tijela kome je on pridružen) u odnosu na drugi, i sastoji se od devet elemenata koji nisu međusobno nezavisni.

Obzirom na ograničenja i osobinu ortogonalnosti matrice rotacije, može se dokazati da su tri parametra dovoljna za opis orijentacije. Reprezentacija orijentacije u terminima od tri nezavisna parametra sačinjava *minimalnu reprezentaciju*, koja se može dobiti koristeči skup od tri ugla $\boldsymbol{\phi} = \begin{bmatrix} \varphi & \theta & \psi \end{bmatrix}^T$ koji se zovu *Eulerovi uglovi*. Postoji više načina na koje se orijentacija korespondira sa ova tri ugla. Jedni od tih načina su *ZYZ i ZYX (ili RPY)* uglovi. Ako je orijentacija data u ovoj reprezentaciji tada je moguće analitički odrediti matricu rotacije $\mathbf{R} = \mathbf{R}(\varphi, \theta, \psi)$, inverzno - ako je poznata matrica rotacije $\mathbf{R}$ tada je (uz određena ogreničenja za ugao θ) moguće odrediti Eulerove uglove.

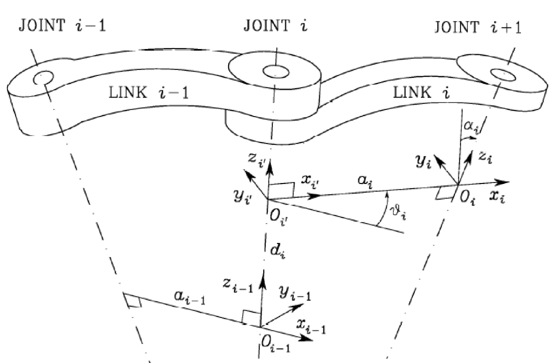
Ako je orijentacija vrha manipulatora zadana u obliku minimalne reprezentacije (Eulerovih uglova) $\boldsymbol{\phi}_e$, tada $(m \times 1)$ vektor:

$$\mathbf{x}_e = \begin{bmatrix} \mathbf{p}_e \\ \boldsymbol{\phi}_e \end{bmatrix} \tag{1.12}$$

$(m \leq n)$ gdje vektor $\mathbf{p}_e$ opisuje poziciju, a vektor $\boldsymbol{\phi}_e$ orijentaciju vrha manipulatora. $\mathbf{x}_e$ se zove *vektor vanjskih koordinata* i definisan je u prostoru u kome je zadatak manipulatoru specificiran. Ovaj prostor se zove *operacijski prostor*. *Zglobni prostor (konfiguracijski prostor*) se definiše kao područje u kojem je definisan $(n \times 1)$ vektor zglobnih varijabli $\mathbf{q} = \begin{bmatrix} q_1 & q_2 & \dots & q_n \end{bmatrix}^T$.*Radni prostor* predstavlja područje dostupno vrhu manipulatora. Za manipulator sa n stepeni slobode kretanja dostupni radni prostor je geometrijsko mjesto tačaka koje mogu biti dohvatljive na osnovu jednačine direktne kinematičke i *ograničenja varijabli zglobova* $q_{i\max} \leq q_i \leq q_{i\max}$, $\mathrm{i}=1, \dots, \mathrm{n}$, gdje $q_{i\min}$, $\mathrm{q}_{i\max}$ minimalan i maksimalan iznos ograničenja za i-ti zglob.

1.2 Denavit-Hartembergova konvencija

Standardni način pridruživanja koordinatnih sistema segmentima i određivanje pozicije i orijentacije koordinatnog sistema i u odnosu na sistem $i-1$ (tj. matrice homogene transformacije A_i^{i-1}) , jeste Denavit-Hartembergov postupak. U cilju određivanja funkcije direktne kinematike svim segmentima manipulatora pridružujemo desno orijentirane ortonormirane koordinatne sisteme, zatim određujemo matrice homogene transformacije $\mathbf{A}_i^{i-1}$. Denavit-Hartembergov postupak predstavlja konvencionalan i sistematičan način pridruživanja koordinatnih sistema segmentima manipulatora i određivanja relativne pozicije i orijentacije koordinatnih sistema dva susjedna segmenta, odnosno određivanja matrice homogene transformacije $\mathbf{A}_i^{i-1}$.



Slika 1.2. Denavit-Hartembergovi kinematički patametri

Na slici 1.2.a) su predstavljena dva susjedna segmenta i-1 -vi i i -ti, i njima pridruženi koordinatni sistemi i odgovarajući DH parametri. Po Denavit-Hartembergovoj konvenciji i-tom segmentu je pridružen koordinatni sistem tako da je:

- z_i osa je izabrana duž ose i+1 –vog zgloba,
- ishodište O_i je locirano na presjeku z_i ose i zajedničke normale z_{i-1} i z_i osa,
- x_i osa je izabrana duž ove zajedničke normale i usmjerena je od zgloba i ka zglobu i+1,
- y_i osa je izabrana u skladu sa desnom orijentacijom.

Za ovako određene koordinatne sisteme dva susjedna segmenta pozicija i orijentacija koordinatnog sistema i u odnosu na koordinatni sistem i-1 u potpunosti je određena sljedećim parametrima:

- a_i - udaljenost između O_i i $O_i^{'}$
- d_i - koordinata od $O_i^{'}$ duž osi z_{i-1}
- α_i - ugao rotacije oko osi x_i koji je mjeren od osi z_{i-1} prema osi z z_i,
- θ_i - ugao rotacije oko osi z_{i-1} koji je mjeren od osi x_{i-1} prema osi x_i .

Uglovi a_i i θ_i imaju pozitivan predznak u slučaju da se rotacija vrši u smjeru obrnutom od smjera kretanja kazaljke na satu.Parametr a_i i α_i *su* uvijek konstante i zavise samo od geometrije segmenata i zglobova. Kod translacijskih (prizmatičnih) zglobova varijabla je d_i, a kod rotacijskih θ_i.

Matricom $\mathbf{A}_i^{i-1}$ je opisana transformacija koordinata iz sistema pridruženog segmentu i u koordinate sistema pridruženog segmentu i-1 i i obavlja se na sljedeći način :

1. koordinatni sistem $O_{i-1} - x_{i-1}y_{i-1}z_{i-1}$ translatirati za dužinu d_i duž koordinatne ose z_{i-1},a nakon toga koordinatni sistem zarotirati oko iste osi za ugao θ_i,
2. katim koordinatni sistem translatirati za dužinu a_i duž koordinarne ose x_i , a nakon toga zarotirati za ugao α_i

nakon čega se koordinatni sistem $O_{i-1} - x_{i-1}y_{i-1}z_{i-1}$ i $O_i - x_i y_i z_i$ podudaraju. Može se reći da je ovim postupkom transformacija koordinata podijeljena na elementarne transformacije: rotacija oko jedne koordinatne osi i translacija duž te koordinatne osi. Možemo kratko pisati:

$$\mathbf{A}_i^{i-1} = Trans_{z_{i-1}}(d_i) \cdot Rot_{z_{i-1}}(\theta_i) \cdot Trans_{x_i}(a_i) \cdot Rot_{x_i}(\alpha_i) \quad (1.13)$$

Par transformacija translacija-rotacija duž/oko iste ose je određen matricom homogene transformacije . Množenjem ovih dviju matrica dobijamo matricu homogene transformacije:

$$\mathbf{A}_i^{i-1} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\theta_i \cos\alpha_i & -\cos\theta_i \sin\alpha_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.14)$$

2. Rješavanje problema inverzne kinematike robota

2.1 Problem inverzne kinematike

Neka je dat n–segmentni manipulator koji predstavlja otvoreni kinematički lanac za koji su dati DH parametri u skladu sa segmentima pridruženim koordinatnim sistemima. Jednačina direktne kinematike manipulatora uspostavlja funkcionalnu vezu izmedju vektora zglobnih varijabli $\boldsymbol{q} = \left[q_1 \; q_2 \ldots \; q_n\right]^T$ i pozicije i orijentacije vrha manipulatora. Iz jednačina 1.10 i 1.14 vidimo da za dati manipulator sa poznatim DH parametrima, možemo na jedinstven način odrediti poziciju i orijentaciju vrha manipulatora na osnovu poznavanja vektora **q**.

Problem *inverzne kinematike* se sastoji u odredjivanju vektora zglobnih varijabli na osnovu date pozicije i orijentacije vrha manipulatora. Rješavanje ovog problema ima fundementalni značaj i mnogo je kompleksnije od problema direktne kinematike. Razlozi za ovu osobinu problema inverzne kinematike su sljedeći:

- U opštem slučaju jednačine koje se u ovom problemu rješavaju su nelinearne, i nije ih moguće riješiti u zatvorenoj formi.
- Nejedinstvenost rješenja ovih jednačina.
- Egzistencija beskonačno mnogo rješenja za slučaj *kinematički redudantnog* manipulatora.
- Postojanje neprihvatljivih rješenja u pogledu kinematičke strukture manipulatora.

Postojanje rješenja je garantovano samo ako se zahtjevane pozicija i orijentacija dostupne vrhu manipulatora, tj.ako se nalaze u radnom prostoru. Nejedinstvenost rješenja ne zavisi samo od stepena slobode kretanja vrha manipulatora, nego i od broja nenultih DH parametara, tako da broj ne nultih DH parametara utiče na broj prihvatljivih rješenja. Postojanje ograničenja varijabli zglobova $q_{i\max} \le q_i \le q_{i\max}$, $\mathrm{i}=1$,...,n eventualno može redukovati broj ovih rješenja. Sveukupno postoji pet klasa načina pristupa u rješavanju problema inverzne kinematike. Neki od njih su analitički pristup gdje se koristi matrica geometrijski ili analitički jakobijan manipulatora, i optimizacijski pristup gdje se koriste optimizacijske metode.

2.2 Optimizacijski pristup u rješavanju problema inverzne kinematike

Opšti *problem optimizacije bez ograničenja* se sastoji u pronalaženju takve vrijednosti $\boldsymbol{v}^*$ za koju vrijedi:

$$\boldsymbol{v}^* = \min_{\boldsymbol{x}\in\mathbb{R}^n} f\left(\boldsymbol{x}\right) \tag{2.1a}$$

odnosno takvog potencijalnog rješenja (ili potencijalnih rješenja) za koje vrijedi:

$$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}\in\mathbb{R}^n} f\left(\boldsymbol{x}\right) \tag{2.1b}$$

tj. za koje se postiže minimalna vrijednost kriterija $f\left(\boldsymbol{x}\right)$. Funkciju f zovemo *kriterij optimalnosti* ili *funkcija cilja*. Vidimo da u problemu optimizacije bez ograničenja promjenjiva $\boldsymbol{x}$ uzima vrijednosti iz $\mathbb{R}^n$ za koje je funkcija f definisana, a kojeg nazivamo *problemski prostor*.

Kod problema optimizacije sa ograničenjima:

$$\boldsymbol{v}^* = \min_{\boldsymbol{x}\in\Omega} f\left(\boldsymbol{x}\right) \tag{2.2a}$$

problemski prostor Ω je zadan sistemom jednakosti i/ili sistemom nejednakosti:

$$\Omega = \left\{\boldsymbol{x} \in \mathbb{R}^n \mid \; g_i(\boldsymbol{x}) = 0, \; i = 1, \; \ldots, l; \; g_i(\boldsymbol{x}) \le 0, \; i = l+1, \; \ldots, m; \; m \le n\right\} \tag{2.2b}$$

gdje su $g_i : \mathbb{R}^n \to \mathbb{R}, \; i = 1, \; \ldots, m$. Ovaj skup nazivamo *skupom dozvoljenih vrijednosti*.

U optimizacijskom pristupu rješavanja problema inverzne kinematike problem inverzne kinematike se posmatra kao *problem optimizacije sa ograničenjima*, čijim rješavanjem se vektor zglobnih varijabli $\boldsymbol{q}^* = \left[q_1^* \; q_2^* \ldots \; q_n^*\right]^T$ dobije kao rezultat rješavanja ovog problema, tj. :

$$\boldsymbol{q}^* = \arg\min_{\boldsymbol{q}\in\Omega} f\left(\boldsymbol{q}\right) \tag{2.3a}$$

gdje je Ω skup ograničenja definisan kao:

$$\Omega = \left\{\boldsymbol{q} = \left[q_1 \; q_2 \; \ldots q_n\right] \in \mathbb{R}^n \mid \; q_{i \; \min} \le q_i \le q_{i \; \max}, \; i = 1, \; 2, \; \ldots, n\right\} \tag{2.3b}$$

gdje je n-broj segmenata manipulatora, $q_{i \; \min}$, $q_{i \; \max}$ donja i gornja granica pokretljivosti i-tog zgloba. f je funkcija kriterija optimalnosti koju je potrebno izabrati.

Definicija: *Za kvadratnu matricu $\boldsymbol{Q}$ kažemo da je pozitivno definitna ako vrijedi $\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} > 0$ za $\forall \boldsymbol{x} \in \mathbb{R}^n \setminus \boldsymbol{0}$. Skalarnu funkciju $g\left(\boldsymbol{x}\right) = \boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x}$ nazivamo kvadratna forma i kažemo da je pozitivno definitna ako je matrica $\boldsymbol{Q}$ pozitivno definitna.*

Funkcija u problemu (2.3) se obično bira kao kvadrat greške pozicije i orijentacije vrha manipulatora, tj. kao kvadratna foma:

$$f\left(\boldsymbol{q}\right) = \frac{1}{2}\boldsymbol{e}^T\left(\boldsymbol{q}\right)\boldsymbol{M}\boldsymbol{e}\left(\boldsymbol{q}\right) \tag{2.4}$$

gdje je $\boldsymbol{M}$ (6×6) *matrica težinskih koeficijenata* koja je izabrana tako da određuje doprinose pojedinih komponenti vektora $\boldsymbol{e}\left(\boldsymbol{q}\right)$ funkciji kriterija f . $\boldsymbol{e}\left(\boldsymbol{q}\right)$ je (6×1) vektor greške pozicije i orijentacije i može se izabrati u obliku:

$$\boldsymbol{e}\left(\boldsymbol{q}\right) = \begin{bmatrix} \boldsymbol{e}_P(\boldsymbol{q}) \\ \boldsymbol{e}_O(\boldsymbol{q}) \end{bmatrix} \tag{2.5}$$

$\boldsymbol{e}_P(\boldsymbol{q})$ je (3×1) *vektor greške pozicije* vrha manipulatora, i on je izabran kao razlika vektora $\boldsymbol{p}_d$ željene pozicije vrha manipulatora i vektora $\boldsymbol{p}_e(\boldsymbol{q})$ trenutne-aktuelne pozicije vrha manipulatora, tj. :

$$\boldsymbol{e}_P(\boldsymbol{q}) = \boldsymbol{p}_d - \boldsymbol{p}_e(\boldsymbol{q}) \tag{2.6}$$

$\boldsymbol{p}_e(\boldsymbol{q})$ kao vektor trenutne pozicije vrha manipulatora je prema tome određen elementima prva tri reda četvrte kolone matrice (1.1) , odnosno (1.10) . Vektor $\boldsymbol{p}_d$ je generisan nekim od algoritama planiranja trajektorije.

$\boldsymbol{e}_O(\boldsymbol{q})$ je (3×1) *vektor greške orijentacije* vrha manipulatora. Ovaj vektor je moguće izabrati na više načina. Prvi način je da se izabere kao vektor čiji su elementi jednaki razlici Eulerovih uglova željene i trenutne orijentacije, pri tome se koristi relacija koja transformira matricu rotacije (orijentacije) vrha manipulatora u Eulerove uglove. Kod ovog načina definisanja vektora greške orijentacije prisutan je jedan nepoželjan efekat, to je tzv. *gimbal lock*. Drugi način, koji je ovdje izabran, je da $\boldsymbol{e}_O(\boldsymbol{q})$ u određenom smislu bude određena matricom:

$$\boldsymbol{R}_d\boldsymbol{R}_e^T(\boldsymbol{q}) \tag{2.7}$$

koja opisuje rotaciju koja je potrebna da se željena orijentacija vrha manipulatora koja je određena matricom $\boldsymbol{R}_d$ podudari sa trenutnom orijentacijom koja je određena matricom $\boldsymbol{R}_e(\boldsymbol{q})$. Matrica $\boldsymbol{R}_e(\boldsymbol{q})$ je određena matricom rotacije u jednačini direktne kinematike (1.1), odnosno (1.10). $\boldsymbol{R}_d$ je generisana algoritmom planiranja trajektorije. Matricama $\boldsymbol{R}_d$ i $\boldsymbol{R}_e(\boldsymbol{q})$ se mogu asocirati *jedinični kvaternioni* $Q_d = \{\eta_d,\ \boldsymbol{\varepsilon}_d\}$ i $Q_e = \{\eta_e,\ \boldsymbol{\varepsilon}_e\}$.

Transformacija matrice $\boldsymbol{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$ u jedinični kvaternion $Q = \{\eta,\ \boldsymbol{\varepsilon}\} = \eta + i\varepsilon_x + j\varepsilon_y + k\varepsilon_z$ je određena relacijama:

$$\eta = \frac{1}{2}\sqrt{r_{11} + r_{22} + r_{33} + 1} \tag{2.8a}$$

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \end{bmatrix} = \frac{1}{2}\begin{bmatrix} \operatorname{sgn}(r_{32} - r_{23})\sqrt{r_{11} - r_{22} - r_{33} + 1} \\ \operatorname{sgn}(r_{13} - r_{31})\sqrt{r_{22} - r_{33} - r_{11} + 1} \\ \operatorname{sgn}(r_{21} - r_{12})\sqrt{r_{33} - r_{11} - r_{22} + 1} \end{bmatrix} \tag{2.8b}$$

Kvaternion $Q = \{\eta,\ \boldsymbol{\varepsilon}\} = \eta + i\varepsilon_x + j\varepsilon_y + k\varepsilon_z$ je *jedinični* pa prema tome mora biti zadovoljena jednakost: $\eta^2 + \boldsymbol{\varepsilon}^T\boldsymbol{\varepsilon} = 1$. η se zove *skalarni dio*, a $\boldsymbol{\varepsilon} = \left[\varepsilon_x, \varepsilon_y, \varepsilon_z\right]^T = i\varepsilon_x + j\varepsilon_y + k\varepsilon_z$ *vektorski dio* kvaterniona. Matrica (2.7) može biti izražena u terminima kvaterniona kao:

$$\Delta Q = Q_d * Q_e^{-1} \tag{2.9}$$

gdje je $*$ *Hamiltonov produkt*, i obzirom na osobine ovog produkta i osobine kvaterniona imamo:

$$\Delta Q = Q_d * Q_e^{-1} = \{\Delta\eta,\ \Delta\boldsymbol{\varepsilon}\} = \{\eta_d\eta_e + \boldsymbol{\varepsilon}_d^T\boldsymbol{\varepsilon}_e,\ \eta_d\boldsymbol{\varepsilon}_e - \eta_e\boldsymbol{\varepsilon}_d - \boldsymbol{\varepsilon}_d \times \boldsymbol{\varepsilon}_e\} \tag{2.10}$$

gdje $\times$ označava vektorski proizvod. Kada se orijentacije određene matricama $\boldsymbol{R}_d$ i $\boldsymbol{R}_e(\boldsymbol{q})$ podudare, tada je $\Delta Q = \{1,\ \boldsymbol{0}\}$ i $\eta_d\boldsymbol{\varepsilon}_e - \eta_e\boldsymbol{\varepsilon}_d = \boldsymbol{\varepsilon}_d \times \boldsymbol{\varepsilon}_e$. Lijeva i desna strana posljednje jednakosti su međusobno okomiti vektori, pa je ova jednakost zadovoljena samo u slučaju kada su lijeva i desna strana nul-vektori, na osnovu čega imamo:

$$\boldsymbol{\varepsilon}_e = \eta_e\left(\eta_e / \eta_d\right)\boldsymbol{\varepsilon}_d \tag{2.11}$$

Prema tome, dovoljno je grešku orijentacije definisati vektorskim dijelom kvaterniona ΔQ , tj.:

$$\boldsymbol{e}_P = \Delta\boldsymbol{\varepsilon} = \eta_d \boldsymbol{\varepsilon}_e - \eta_e \boldsymbol{\varepsilon}_d - \boldsymbol{\varepsilon}_d \times \boldsymbol{\varepsilon}_e \tag{2.12}$$

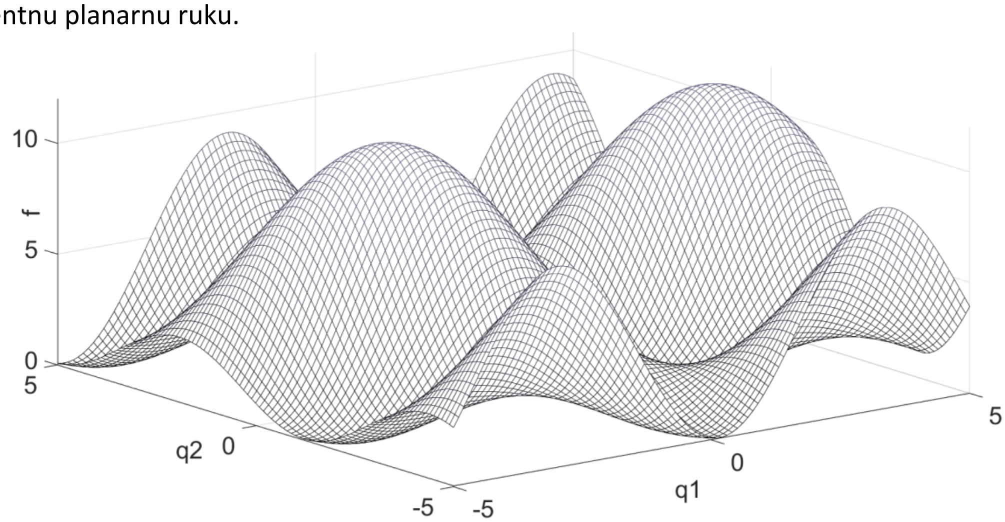
(2.4) možemo pisati i u obliku:

$$f(\boldsymbol{q}) = \frac{1}{2}\boldsymbol{e}_P^T(\boldsymbol{q})\boldsymbol{M}_P\boldsymbol{e}_P(\boldsymbol{q}) + \frac{1}{2}\boldsymbol{e}_O^T(\boldsymbol{q})\boldsymbol{M}_O\boldsymbol{e}_O(\boldsymbol{q}) \tag{2.13}$$

gdje su $\boldsymbol{e}_P(\boldsymbol{q})$ i $\boldsymbol{e}_O(\boldsymbol{q})$ određene relacijama (2.6) i (2.12), respektivno. Kao i kod matrice $\boldsymbol{M}$, izborom elemenata matrica $\boldsymbol{M}_P$ i $\boldsymbol{M}_O$ određuju se doprinosi pojedinih komponenti greške pozicije i orijentacije funkciji kriterija. Tako npr. stavimo da je $\boldsymbol{M}_O$ nula-matrica, vektor zglobnih varijabli u problemu optimizacije po će biti tako određen da će greška pozicije težiti nuli sa određenom tolerancijom, a orijentacija će biti proizvoljna. U općem slučaju vidimo da za ovako definisan kriterij optimalnosti u idealnom slučaju će biti:

$$\min_{\boldsymbol{q}\in\Omega} f(\boldsymbol{q}) = 0 \tag{2.14}$$

za one $\boldsymbol{p}_d$ i $\boldsymbol{R}_d$ koje se nalaze unutar radnog prostora robota, a koji su iz jednačine direktne kinematike (1.10) određeni na osnovu $\boldsymbol{q}$ kojeg smo dobili rješavanjem problema (2.3), što je intuitivno jasno. Da bi se stekao uvid u geometrijsku predstavu funkcije kriterija, na slici (1.3) prikazan je njen grafik (2.13) za dvosegmentnu planarnu ruku.



Slika 1.3: Funkcija kriterija (2.13) za jedan dvodimenzionalan slučaj $\boldsymbol{q} = [q_1 \; q_2]^T$ (dvosegmentnu planarnu ruku) za $\boldsymbol{p}_d = [1 \; 1]^T$

2.3 Jednodimenzionalno pretraživanje

U nastojanju da izlaganje u ovom radu bude kompletnije i konzistentnije ovdje su navedene definicije i stavovi korišteni u izlaganju koje slijedi. Masnim slovima su označavane matrice i vektori-kolone uz koje su zbog preglednosti relacija upotrebljavani i donji i gornji indeksi, pri tome je njihova upotreba ravnopravna te imaju isto značenje. Npr., $\boldsymbol{q}^k$ označava vrijednost vektora-kolone zglobnih variabli u k-toj iteraciji, isto tako $\boldsymbol{p}_k$ označava vektor-kolonu koja određuje pravac u k-tok iteraciji.

Jednodimenzionalno pretraživanje u prostoru $\mathbb{R}^n$ predstavlja sastavni dio nekih od najefikasnijih metoda pretraživanja. Pod jednodimenzionalnim pretraživanjem u n-dimenzionalnom prostoru se podrazumijeva provođenje pretraživanja po pravoj u $\mathbb{R}^n$. Jednačinu prave u ovom prostoru možemo definisati kroz dvije poznate tačke, ili kroz jednu poznatu tačku uz zadani vektor pravca. Jednačina prave kroz tačku $\boldsymbol{x}^a \in \mathbb{R}^n$ uz definisani vektor pravca $\boldsymbol{p}$ je data izrazom:

$$\boldsymbol{x}(\alpha) = \boldsymbol{x}^a + \alpha \boldsymbol{p} \tag{2.15}$$

Posmatrajmo tačke na pravoj datom prethodnim izrazom i vrijednost kriterija $f(\boldsymbol{x})$ za ove tačke. Očigledno je da na ovoj pravoj kriterij postaje funkcija parametra α, tako da problem jednodimenzionalnog pretraživanja možemo formalno iskazati u obliku:

$$\min_{\alpha>0} f\left(\boldsymbol{x}^a + \alpha \boldsymbol{p}\right) \tag{2.16}$$

Definicija: *Funkciji* $f:\mathbb{R}^n \to \mathbb{R}$ *koja je definisana u okolini tačke* $\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^T$ *:*

- *pridružujemo* $(1\times n)$ *vektor:*

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} \dfrac{\partial f(\boldsymbol{x})}{\partial x_1} & \dfrac{\partial f(\boldsymbol{x})}{\partial x_2} & \cdots & \dfrac{\partial f(\boldsymbol{x})}{\partial x_n} \end{bmatrix}^T \tag{2.17}$$

kojeg nazivamo gradijent funkcije f *u tački* $\boldsymbol{x}$ *,*

- *pridružujemo* $(n\times n)$ *matricu:*

$$\nabla^2 f(\boldsymbol{x}) = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1 \partial x_n} \\ \dfrac{\partial^2 f}{\partial x_2 \partial x_1} & \dfrac{\partial^2 f}{\partial x_1^2} & \cdots & \dfrac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & & \ddots & \vdots \\ \dfrac{\partial^2 f}{\partial x_n \partial x_1} & \dfrac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \tag{2.18}$$

koju nazivamo Hessian funkcije f *u tački* $\boldsymbol{x}$ *.*

Definicija: *Neka je funkcija* $f:\mathbb{R}^n \to \mathbb{R}$ *definisana u okolini tačke* $\boldsymbol{x}^*$. *Ako je zadovoljen uslov* $\nabla f\left(\boldsymbol{x}^*\right) = 0$ *tada se tačka* $\boldsymbol{x}^*$ *naziva stacionarnom tačkom funkcije* f *.*

Naredna teorema predstavlja potreban uslov da bi funkcija imala lokalni minimum u $\boldsymbol{x}^*$.

Teorema: *Ako je* $\boldsymbol{x}^*$ *lokalni minimum i funkcija* f *je neprekidno diferencijabilna u otvorenoj okolini tačke* $\boldsymbol{x}^*$ *, tada je* $\boldsymbol{x}^*$ *stacionarna tačka funkcije* f *.*

Naredna teorema predstavlja dovoljan uslov da bi funkcija imala lokalni minimum u $\boldsymbol{x}^*$.

Teorema: *Ako je* $\nabla^2 f$ *neprekidna matrica u otvorenoj okolini tačke* $\boldsymbol{x}^*$ *i ako je* $\nabla f\left(\boldsymbol{x}^*\right) = 0$ *i matrica* $\nabla^2 f\left(\boldsymbol{x}^*\right)$ *je pozitivno definitna, tada je* $\boldsymbol{x}^*$ *lokalni minimum (u užem smislu) funkcije* f *.*

Da bismo odredili stacionarne tačke funkcije f po parametru α , potražimo prvi izvod po α :

$$\frac{d}{d\alpha} f\left(\boldsymbol{x}^a + \alpha \boldsymbol{p}\right) = \boldsymbol{p}^T \cdot \nabla f\left(\boldsymbol{x}^a + \alpha \boldsymbol{p}\right) \tag{2.19}$$

Prirodu stacionarne možemo odrediti analizom vrijednosti drugih izvoda:

$$\frac{d^2}{d\alpha^2} f\left(\boldsymbol{x}^a + \alpha \boldsymbol{p}\right) = \boldsymbol{p}^T \cdot \nabla^2 f \cdot \boldsymbol{p} \tag{2.20}$$

Umjesto rješavanjem jednačina, stacionarne tačke na pravcu se najčešće određuju nekom od metoda za jednodimenzionalno pretraživanje. Jedna od tih metoda je *klasična Newtonova metoda*. Za funkciju

$$\varphi(\alpha) = f\left(\boldsymbol{x}^a + \alpha \boldsymbol{p}\right) \tag{2.21}$$

uzimajući u obzir početnu tačku α_0 ova metoda je određena rekurzivnom relacijom:

$$\alpha_{k+1} = \alpha_k - \frac{\dot{\varphi}(\alpha_k)}{\ddot{\varphi}(\alpha_k)} \tag{2.22}$$

Definicija: *Neka niz* $\left\{\boldsymbol{x}^k\right\}$ *,* $\boldsymbol{x}^k \in \mathbb{R}^n$, $(k = 0,\ 1,\ 2,\ \ldots)$ *konvergira ka* $\boldsymbol{x}^* \in \mathbb{R}^n$ *. Za dati niz kažemo da konvergira ka* $\boldsymbol{x}^*$ *brzinom konvergencije* q *, ako za pozitivnu konstatntu* $M > 0$ *vrijedi uslov:*

$$\lim_{k\to\infty} \frac{\left\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^*\right\|}{\left\|\boldsymbol{x}^k - \boldsymbol{x}^*\right\|^q} < M, \qquad q \geq 1. \tag{2.23}$$

Za $q = 1$ *imamo linearnu konvergenciju,* $q = 2$ *kvadratnu konvergenciju. Ako je za* $q = 1$ *ova granična vrijednost jednaka 1 tada kažemo da dati niz konvergira sublinearno ka* $\boldsymbol{x}^*$ *. Ako je za* $q = 1$ *ova granična vrijednost jednaka 0 tada kažemo da dati niz konvergira superlinearno ka* $\boldsymbol{x}^*$ *.*

Ova metoda konvergira kvadratno ka stacionarnoj tački funkcije φ . Newtonova metoda može i da divergira ili oscilira pa se koriste druge iterativne metode koje obuhvataju znatno širu klasu funkcija za koje metod konvergira. Jedna od tih metoda je tzv. *two-point Newtonova metoda*. Uz dvije početne tačke α_0 i α_1 , ova metoda za nalaženje stacionarne tačke funkcije φ je određena relacijom:

$$\alpha_{k+1} = \alpha_{k-1} - \frac{\alpha_{k-1} - \alpha_k}{1 - \dfrac{\dot{\varphi}(\alpha_k)}{\dot{\varphi}(\alpha_{k-1})} \cdot \dfrac{\dfrac{\dot{\varphi}(\alpha_k) - \dot{\varphi}(\alpha_{k-1})}{\alpha_k - \alpha_{k-1}}}{\ddot{\varphi}(\alpha_k)}} \tag{2.24}$$

Ovaj metod super-kvadratno ($q = 2.414$) konvergira ka stacionarnoj tački , i za razliku od klasičnog Newtonovog metoda on ima osobinu da konvergira i u slučaju kada $\ddot{\varphi}(\alpha_k) \to 0$, što se može analitički pokazati, pa je ovaj metod korišten u ovom radu . Kao tipični uslovi zaustavljanja za oba metoda se koriste:

- dostizanje zadanog broja iteracija,
- postizanje zadane minimalne vrijednosti izvoda,
- postizanje zadane minimalne promjene problemske varijable.

Obe ove metode spadaju u egzaktne metode jednodimenzionalnog pretraživanja, jer je njihov cilj tačno određivanje stacionarne tačke, odnosno optimalne dužine koraka u svakoj iteraciji. Često nije potrebno tačno određivanje ovog koraka pa postoje i neegzaktne metode koje približno određuju optimalnu dužinu koraka. Neke od približnih metoda su zasnovane na tzv. *Wolfe-ovim uslovima*:

$$\begin{cases} f\left(\boldsymbol{x}^k+\alpha_k \boldsymbol{p}_k\right) \le f\left(\boldsymbol{x}^k\right)+\delta_1 \alpha_k \nabla f_k^T \boldsymbol{p}_k & (2.25a) \\ \nabla f\left(\boldsymbol{x}^k+\alpha_k \boldsymbol{p}_k\right)^T \boldsymbol{p}_k \ge \delta_2 \nabla f_k^T \boldsymbol{p}_k & (2.25b) \end{cases}, \quad 0<\delta_1<\delta_2$$

Uslov (2.25a) određuje dovoljnu dekremantaciju funkcije f, koji nije sam po sebi dovoljan da algoritam napravi razuman progres jer je ovaj uslov uvijek zadovoljen za dovoljno malo α. Uslov (2.25b) je tzv. *uslov zakrivljenosti*, čija je lijeva strana izvod funkcije $\varphi(\alpha_k)=f\left(\boldsymbol{x}^k+\alpha_k \boldsymbol{p}_k\right)$ po α_k, i ovaj uslov osigurava da nagib funkcije φ u α_k je veći δ_2 puta od početnog nagiba $\varphi'(0)$. Tako ako je nagib $\varphi'(\alpha)$ strogo negativan ovim uslovom imamo indikaciju da možemo očekivati da ćemo postići značajnu dekrementaciju funkcije f po ovom pravcu.

2.4 Pravac pretraživanja, metode pretraživanja

Da bi se stekao uvid u izbor pravca jednodimenzionalnog pretraživanja kod BFGS metoda ovdje ćemo izložiti neke osobine koje se odnose na pravac pretraživanja, koje slijede iz Taylorove teoreme. Time se na izbor različitih metoda pretraživanja može gledati kao na izbor pravca pretraživanja.

***Teorema** (Taylorova teorema): Neka je funkcija $f:\mathbb{R}^n \to \mathbb{R}$ neprekidno diferencijabilna na svome domenu i neka $\boldsymbol{p}\in\mathbb{R}^n$. Tada za neko $t\in(0,\ 1)$ vrijedi:*

$$f(\boldsymbol{x}+\boldsymbol{p})=f(\boldsymbol{x})+\nabla f(\boldsymbol{x}+t\boldsymbol{p})^T \boldsymbol{p} \qquad \textit{(2.26)}$$

Ako je f dvostruko neprekidno diferencijabilna, za neko $t\in(0,\ 1)$ imamo da je:

$$\nabla f(\boldsymbol{x}+\boldsymbol{p})=\nabla f(\boldsymbol{x})+\int_0^1 \nabla^2 f(\boldsymbol{x}+t\boldsymbol{p})\,\boldsymbol{p}\,dt \qquad \textit{(2.27)}$$

takođe vrijedi:

$$f(\boldsymbol{x}+\boldsymbol{p})=f(\boldsymbol{x})+\nabla f(\boldsymbol{x})^T \boldsymbol{p}+\frac{1}{2}\boldsymbol{p}^T\nabla^2 f(\boldsymbol{x}+t\boldsymbol{p})\,\boldsymbol{p} \qquad \textit{(2.28)}$$

2.4.1 Metoda najbržeg pada

Na osnovu (2.28) za funkciju f koja nam govori da za bilo koji pravac pretraživanja $\boldsymbol{p}$ i dužinu koraka α, imamo:

$$f\left(\boldsymbol{x}^k+\alpha\boldsymbol{p}\right)=f\left(\boldsymbol{x}^k\right)+\alpha\boldsymbol{p}^T\nabla f_k+\frac{1}{2}\alpha^2\boldsymbol{p}^T\nabla^2 f\left(\boldsymbol{x}^k+t\boldsymbol{p}\right)\boldsymbol{p}, \quad \boldsymbol{t}\in(0,\ \alpha) \qquad (2.29)$$

gdje je $f_k=f\left(\boldsymbol{x}^k\right)$. Iznos promjene funkcije f duž pravca $\boldsymbol{p}$ u tački $\boldsymbol{x}^k$ jednak je kojeficijentu uz α, tj. $\boldsymbol{p}^T\nabla f_k$. Jedinični vektor pravca $\boldsymbol{p}$ najbržeg pada može se dobiti kao rješenje problema:

$$\min_{\boldsymbol{p}} \boldsymbol{p}^T\nabla f_k, \text{ uz ograničenje } \|\boldsymbol{p}\|=1 \qquad (2.30)$$

Obzirom da je $\boldsymbol{p}^T\nabla f_k=\|\boldsymbol{p}\|\|\nabla f_k\|\cos\theta=\|\nabla f_k\|\cos\theta$,što postiže minimum kada je $\cos\theta=-1$,pa je:

$$\boldsymbol{p} = -\nabla \boldsymbol{f}_k / \|\nabla \boldsymbol{f}_k\| \tag{2.31}$$

Dakle pravac pretraživanja u tački $\boldsymbol{x}^k$ je određen gradijentom funkcije u toj tački isa obzirom na predstavlja pravac najbržeg pada funkcije u $\boldsymbol{x}^k$, pa zato metoda kod koje je pravac pretraživanja u svakoj iteraciji određen gradijentom, se zove *metoda najbržen pada*.

2.4.2 Newtonova metoda

Pravac pretraživanja *Newtonove metode pretraživanja u prostoru* $\mathbb{R}^n$ možemo dobiti polazeći od (2.28):

$$f\left(\boldsymbol{x}^k + \boldsymbol{p}\right) = f\left(\boldsymbol{x}^k\right) + \boldsymbol{p}^T \nabla f_k + \frac{1}{2} \boldsymbol{p}^T \nabla^2 f_k \boldsymbol{p} \overset{def}{=} m_k\left(\boldsymbol{p}\right) \tag{2.32}$$

Obzirom da je $\nabla^2 f_k$ pozitivno definitna matrica, Newtonov pravac dobijamo nalazeći vektor $\boldsymbol{p}$ koji minimizira $m_k\left(\boldsymbol{p}\right)$. Izjednačavanjem gradijenta od $m_k\left(\boldsymbol{p}\right)$ sa nulom iz te jednačine dobijamo:

$$\boldsymbol{p}_k = -\left(\nabla f_k\right)^{-1} \nabla f_k \tag{2.33}$$

Ovaj metod konvergira kvadratno i pouzdan je kada nije velika razlika između $f\left(\boldsymbol{x}^k + \boldsymbol{p}\right)$ i njenog kvadratnog modela $m_k\left(\boldsymbol{p}\right)$. Zahtjeva računanje drugih izvoda i inverznog Hessiana, što je numerički zahtjevno, pa su konstruisani drugi metodi kojim je to izbjegnuto.

2.4.3 Kvazi-Newtonove metode

Kvazi-Newtonove metode pretraživanja (u prostoru $\mathbb{R}^n$) su pogodna alternativa Newtonovoj metodi jer oni ne zahtijevaju računanje matrice Hessiana, umjesto kojeg koriste njegovu aproksimaciju $\boldsymbol{B}_k$. Na osnovu (2.27) imamo:

$$\nabla f\left(\boldsymbol{x} + \boldsymbol{p}\right) = \nabla f\left(\boldsymbol{x}\right) + \nabla^2 f\left(\boldsymbol{x}\right)\boldsymbol{p} + \int_0^1 \left[\nabla^2 f\left(\boldsymbol{x} + t\boldsymbol{p}\right) - \nabla^2 f\left(\boldsymbol{x}\right)\right]\boldsymbol{p}\,dt \tag{2.34}$$

Postavljanjem $\boldsymbol{x} = \boldsymbol{x}^k$ i $\boldsymbol{p} = \boldsymbol{x}^{k+1} - \boldsymbol{x}^k$, dobijamo:

$$\nabla f_{k+1} = \nabla f_k + \nabla^2 f_{k+1}\left(\boldsymbol{x}^{k+1} - \boldsymbol{x}^k\right) + o\left(\left\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^k\right\|\right) \tag{2.35}$$

Kada $\boldsymbol{x}^k$ i $\boldsymbol{x}^{k+1}$ leže u oblasti bliskoj rješenju $\boldsymbol{x}^*$, pri čemu je $\nabla^2 f$ pozitivno definitna matrica posljednji član izraza (2.35) možemo zanemariti, tako da vrijedi:

$$\nabla^2 f_k\left(\boldsymbol{x}^{k+1} - \boldsymbol{x}^k\right) \approx \nabla f_{k+1} - \nabla f_k \tag{2.36}$$

Može se izabrati aproksimacija Hessiana $\nabla^2 f$ koju ćemo označiti sa $\boldsymbol{B}_{k+1}$ koja približno zadovoljava relaciju (2.36), tako da na osnovu (2.36), imamo:

$$\boldsymbol{B}_{k+1}\boldsymbol{s}_k = \boldsymbol{y}_k \tag{2.37}$$

gdje je $\boldsymbol{s}_k = \boldsymbol{x}^{k+1} - \boldsymbol{x}^k$, $\boldsymbol{y}_k = \nabla f_{k+1} - \nabla f_k$. Obično se za matricu $\boldsymbol{B}_{k+1}$ postavlja uslov simetričnosti jer tu osobinu ima i Hessian. Na osnovu gornjih razmatranja pravac pretraživanja kvazi-Newtonovih metoda pretraživanja je određen relacijom:

$$\boldsymbol{p}_k = -\boldsymbol{B}_k^{-1}\nabla f_k \tag{2.38}$$

Postoji više kvazi-Newtonovih metoda sa različitim rekurzivnim relacijama za određivanje matrice $\boldsymbol{B}_{k+1}$. Najefikasnije od tih metoda su *DFP (Davidon-Flatcher-Powell)* i *BFGS* metod koji je opisan i primijenjen u ovom radu.

2.5 Broyden-Fletcher-Goldberg-Shanno (BFGS) metoda nelinearne optimizacije

Neka imamo opšti problem optimizacije bez ograničenja dat sa (2.1). U izvođenju BFGS metode rješavanja ovog problema koristit ćemo algoritamsku strategiju poznatu kao *trust region*, kod koje su informacije o funkiciji f upotrijebljene za konstrukciju modela m_k. Kao model m_k dakle ima ponašanje u okolini tekuće tačke $\boldsymbol{x}_k$ slično ponašanju kriterijalne funkcije f. m_k može da ne bude dobra aproksimacija funkcije f kada je $\boldsymbol{x}$ daleko od $\boldsymbol{x}^k$, u tom slučaju ćemo suziti pretraživanje minimuma od m_k na neki region oko $\boldsymbol{x}^k$. Drugim riječima, uzet ćemo korak $\mathbf{p}$ tako što ćemo aproksimativno riješiti problem:

$$\min_{\boldsymbol{p}} m_k\left(\boldsymbol{x}^k + \boldsymbol{p}\right) \tag{2.39}$$

gdje $\boldsymbol{x}^k + \boldsymbol{p}$ leži u trust regionu. Ako rješenje ne proizvodi dovoljno smanjenje u f, zaključujemo da je trust region preširok, tada ovaj region sužavamo i ponovo rješavamo gornji problem. Trust region se obično de finiše kao oblast (lopta) $\|\boldsymbol{p}\|_2 \le \Delta$, gdje skalar $\Delta > 0$ se zove *trust region radijus*. Obično m_k se bira u formi:

$$m_k\left(\boldsymbol{x}^k + \boldsymbol{p}\right) = f_k + \nabla f_k^T \boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^T \boldsymbol{B}_k \boldsymbol{p} \tag{2.40}$$

gdje su f_k, ∇f_k, $\boldsymbol{B}_k$ skalar, vektor i matrica, respektivno. Trust region i linijsko pretraživanje se razlikuju po pristupu izbora pravca i koraka do sljedeće iteracije. Linijsko pretraživanje starta fiksiranjem pravca $\boldsymbol{p}^k$ a zatim nekom od metoda jednodimenzionalnog pretraživanja nalazi optimalnu dožinu koraka α_k . Kod trust region metode, prvo se odabere trust region radijus Δ_k, a zatim pronalazi pravac i korak kojim se postiže najbolje moguće poboljšanje kriterija. Ako je ovaj korak nezadovoljavajuči, reduciramo mjeru rastojanja Δ_k i pokušavamo ponovo. Kako je (2.40) kvadratična (time je i konveksna) funkcija, to je njena stacionarna tačka ujedno i njen minimum, i računajući njen gradijent po argumentu $\boldsymbol{p}$ i izjednačujući sa nulom imamo njen minimum:

$$\boldsymbol{p}_k = -\boldsymbol{B}_k^{-1}\nabla f_k \tag{2.41}$$

i koji je upotrijebljen kao pravac pretraživanja u iteraciji:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha_k \boldsymbol{p}_k \tag{2.42}$$

gdje je α_k dužina koraka k -te iteracije (izabrana je u skladu sa Wolfe-ovim uslovima). Ovdje se vidi sličnost sa Newtonovom metodom, gdje je za razliku od Newtonove metode matrica $\boldsymbol{B}_k$ *aproksimira matricu Hessian*. Omjesto da se $\boldsymbol{B}_k$ računa iznova u svakoj iteraciji, ova matrica se može ažurirati na jednostavniji način uzimajući u obzir zakrivljenost mjerenu tokom prethodnih iteracija.

Neka imamo novu iteraciju $\boldsymbol{x}^{k+1}$, i neka imamo novi kvadratični model za ovu iteraciju:

$$m_{k+1}\left(\boldsymbol{p}\right) = f_{k+1} + \nabla f_{k+1}^T \boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^T \boldsymbol{B}_{k+1} \boldsymbol{p} \tag{2.43}$$

Zahtjev koji se postavlja jeste da je gradijent od m_{k+1} treba da se podudara sa gradijentom funkcije f u posljednje dvije iteracije $\boldsymbol{x}^k$ i $\boldsymbol{x}^{k+1}$, te nakon nalaženja gradijenta obje strane jednačine (2.43) imamo:

$$\nabla m_{k+1}\left(-\alpha_k \boldsymbol{p}_k\right) = \nabla f_{k+1} - \alpha_k \boldsymbol{B}_{k+1} \boldsymbol{p}_k = \nabla f_k \tag{2.44}$$

odakle imamo:

$$\boldsymbol{B}_{k+1} \alpha_k \boldsymbol{p}_k = \nabla f_{k+1} - \nabla f_k \tag{2.45}$$

Radi pojednostavljenja relacije (2.45) definišemo vektore:

$$\boldsymbol{s}_k = \boldsymbol{x}^{k+1} - \boldsymbol{x}^k = \alpha_k \boldsymbol{p}_k, \qquad \boldsymbol{y}_k = \nabla f_{k+1} - \nabla f_k \tag{2.46}$$

tako da relacija (2.45) postaje :

$$\boldsymbol{B}_{k+1} \boldsymbol{s}_k = \boldsymbol{y}_k \tag{2.47}$$

i ova relacija se naziva *jednačina sekante*. Ovom jednačinom je određeno preslikavanje $\boldsymbol{s}^k$ u $\boldsymbol{y}^k$. Matrica $\boldsymbol{B}_{k+1}$ je pozitivno definitna ako je zadovoljen *uslov zakrivljenosti*:

$$\boldsymbol{s}_k^T \boldsymbol{y}_k > 0 \tag{2.48}$$

što se lako provjerava lijevo množeći obje strane jednačine (2.47) sa $\boldsymbol{s}_k^T$. Kada je ispunjen ovaj uslov jednačina (2.47) dopušta beskonačno mnogo rješenja. Zahtjev za pozitivnom definitnosti nameće n dodatnih ograničenja - svi glavni minori moraju biti pozitivni. Da bi odredili izraz za matricu koja aproksimira inverzni Hessian u BFGS algoritmu, postavljamo slučne uslove za maticu inverznu matrici $\boldsymbol{B}_{k+1}$, pa iz jednačine sekanate (2.47) imamo :

$$\boldsymbol{H}_{k+1} \boldsymbol{y}_k = \boldsymbol{s}_k \tag{2.49}$$

gdje matrica $\boldsymbol{H}_{k+1}$ mora biti simetrična i pozitivno definitna i zadovoljava jednačinu sekante koja je sada napisana u obliku (2.49). Sve matrice koje zadovaljavaju gore naveden uslove, u određenom smuslu moraju biti bliske tekućoj matrici $\boldsymbol{H}_k$ tako da vrijedi:

$$\min_H \left\| \boldsymbol{H} - \boldsymbol{H}_k \right\| \tag{2.50a}$$

i zadovoljava ograničenja:

$$\boldsymbol{H} = \boldsymbol{H}^T, \qquad \boldsymbol{H} \boldsymbol{y}_k = \boldsymbol{s}_k \tag{2.50b}$$

Za matričnu normu $\|.\|$ koja omogućava jednostavno rješenje gornjeg optimizacijskog problema (2.50) može se uzeti *Frobeniusova matrična norma*:

$$\left\| \boldsymbol{A} \right\|_W = \left\| \boldsymbol{W}^{1/2} \boldsymbol{A} \boldsymbol{W}^{1/2} \right\|_F \tag{2.51}$$

gdje je $\|.\|$ definisano sa:

$$\|\boldsymbol{C}\|_F^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}^2 \tag{2.52}$$

Za težinsku matricu $\boldsymbol{W}$ može biti uzeta bilo koja matrica koja zadovoljava uslov $\boldsymbol{W}\boldsymbol{s}_k = \boldsymbol{y}_k$, konkretno uzeto je da je $\boldsymbol{W} = \overline{\boldsymbol{G}}_k^{-1}$, gdje je matrica $\overline{\boldsymbol{G}}_k$ *usrednjeni Hessian* definisan sa:

$$\overline{\boldsymbol{G}}_k = \left[\int_0^1 \nabla^2 f\left(\boldsymbol{x}^k + \tau\alpha_k \boldsymbol{p}_k\right) d\tau\right] \tag{2.53}$$

Koristeći Taylorovu teoremu može se dokazati da vrijedi:

$$\boldsymbol{y}_k = \overline{\boldsymbol{G}}_k \alpha_k \boldsymbol{p}_k = \overline{\boldsymbol{G}}_k \boldsymbol{s}_k \tag{2.54}$$

Postoji jedinstveno rješenje $\boldsymbol{H}_{k+1}$ optimizacijskog problema (2.50) koje zadovojava navedena ograničenja i ono je dato sa:

$$\boldsymbol{H}_{k+1} = \left(\boldsymbol{I} - \rho_k \boldsymbol{s}_k \boldsymbol{y}_k^T\right)\boldsymbol{H}_k\left(\boldsymbol{I} - \rho_k \boldsymbol{y}_k \boldsymbol{s}_k^T\right) + \rho_k \boldsymbol{s}_k \boldsymbol{s}_k^T \tag{2.55}$$

gdje je: $\rho_k = \dfrac{1}{\boldsymbol{y}_k^T \boldsymbol{s}_k}$. Ako se ρ_k uvrsti u (2.55) i uzimajući u obzir da su $\boldsymbol{y}_k^T \boldsymbol{H}_k \boldsymbol{y}_k$ i $\boldsymbol{s}_k^T \boldsymbol{y}_k$ skalari, iz (2.55) slijedi:

$$\boldsymbol{H}_{k+1} = \boldsymbol{H}_k + \frac{\left(\boldsymbol{s}_k^T \boldsymbol{y}_k + \boldsymbol{y}_k^T \boldsymbol{H}_k \boldsymbol{y}_k\right)\left(\boldsymbol{s}_k \boldsymbol{s}_k^T\right)}{\left(\boldsymbol{s}_k^T \boldsymbol{y}_k\right)^2} - \frac{\boldsymbol{H}_k \boldsymbol{y}_k \boldsymbol{s}_k^T + \boldsymbol{s}_k \boldsymbol{y}_k^T \boldsymbol{H}_k}{\boldsymbol{s}_k^T \boldsymbol{y}_k} \tag{2.56}$$

gdje se obično uzima da je $\boldsymbol{H}_0 = \boldsymbol{I}$. Ako na jednačinu (2.55) primjenimo formulu (*Sherman-Morrison-Woodbury*):

$$\widehat{\boldsymbol{A}}^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{U}\left(\boldsymbol{I} - \boldsymbol{V}^{\mathrm{T}}\boldsymbol{A}^{-1}\boldsymbol{U}\right)^{-1}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{A}^{-1} \quad \text{gdje je } \widehat{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{U}\boldsymbol{V}^{\mathrm{T}} \tag{2.57}$$

imamo:

$$\boldsymbol{B}_{k+1} = \boldsymbol{B}_k - \frac{\boldsymbol{B}_k \boldsymbol{s}_k \boldsymbol{s}_k^T \boldsymbol{B}_k}{\boldsymbol{s}_k^T \boldsymbol{B}_k \boldsymbol{s}_k} + \frac{\boldsymbol{y}_k \boldsymbol{y}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k} \tag{2.58}$$

Na osnovu gornjih razmatranja za rješavanje problema optimizacije (2.1) imamo algoritam:

Korak 1. Izabrati početnu aproksimaciju $\boldsymbol{x}^0$ i početnu pozitivno definitnu matricu $\boldsymbol{H}_0$ (često se uzima $\boldsymbol{H}_0 = \boldsymbol{I}$). Izračunati gradijent funkcije f u tački $\boldsymbol{x}^0$. Odrediti kriterij zaustavljanja, npr. stagnacija norme gradijenta, tj. kada norma gradijenta funkcije f padne ispod neke unaprijed date proizvoljno male vrijednosti $\varepsilon > 0$, tj.:

$$\|\nabla f_k\| < \varepsilon \tag{2.59}$$

Staviti $k = 0$.

Korak 2. Izračunati pravac pretraživanja funkcije f iz tačke $\boldsymbol{x}^k$

$$\boldsymbol{p}_k = -\boldsymbol{H}_k \nabla f_k \tag{2.60}$$

Korak 3. Naći $\alpha_k > 0$ kao rešanje problema jednodimenzionalne optimizacije u pravcu $\boldsymbol{p}_k$:

$$\min_{\alpha>0} f\left(\boldsymbol{x}^k + \alpha \boldsymbol{p}_k\right) \tag{2.61}$$

Korak 4. Računati novu aproksimaciju:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha_k \boldsymbol{p}_k \tag{2.62}$$

$$\text{i } \boldsymbol{s}_k = \boldsymbol{x}^{k+1} - \boldsymbol{x}^k \tag{2.63a}$$

$$\boldsymbol{y}_k = \nabla f_{k+1} - \nabla f_k \tag{2.63b}$$

Korak 5. Računati $\boldsymbol{H}_{k+1}$ na osnovu (2.56), staviti $k = k+1$. Provjeriti kriterij zaustavljanja. Ako je kriterij zaustavljanja zadovoljen optimalno rješenje problema je $\boldsymbol{x}^* = \boldsymbol{x}^k$. Ako kriterij zaustavljanja nije zadovoljen preći na Korak 2.

2.6 Broydenova klasa

Za matricu B_{k+1} u jednačini sekante (2.47) koja zadovoljava navedene uslove, možemo postaviti analogan problem kao u prethodnom naslovu:

$$\min_{B} \left\| \boldsymbol{B} - \boldsymbol{B}_k \right\| \tag{2.64a}$$

uz ogrančenja:

$$\boldsymbol{B} = \boldsymbol{B}^T, \quad \boldsymbol{B}\boldsymbol{s}_k = \boldsymbol{y}_k \tag{2.64b}$$

Ovaj optimizacijski problem ima jedinstveno rješenje i ono je određeno sa:

$$\boldsymbol{B}_{k+1} = \left(\boldsymbol{I} - \rho_k \boldsymbol{y}_k \boldsymbol{s}_k^T\right) \boldsymbol{B}_k \left(\boldsymbol{I} - \rho_k \boldsymbol{s}_k \boldsymbol{y}_k^T\right) + \rho_k \boldsymbol{y}_k \boldsymbol{y}_k^T \tag{2.65}$$

Primjenjujući (2.57) na (2.65), dobijamo:

$$\boldsymbol{H}_{k+1} = \boldsymbol{H}_k - \frac{\boldsymbol{H}_k \boldsymbol{s}_k \boldsymbol{s}_k^T \boldsymbol{H}_k}{\boldsymbol{s}_k^T \boldsymbol{H}_k \boldsymbol{s}_k} + \frac{\boldsymbol{s}_k \boldsymbol{s}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k} \tag{2.66}$$

gdje je $\boldsymbol{H}_k = \boldsymbol{B}_k^{-1}$. Jednačina (2.66) predstavlja jednačinu za približno računanje matrice Hessijana *DFP (Davidon-Fletcher-Powell)* algoritma, i ona uz formulu (2.58) BFGS algoritma predstavlja jednu od mnogih koje pripadaju tzv. *Broydenovoj klasi* koja je definisana sljedećom općom formulom:

$$\boldsymbol{B}_{k+1} = \boldsymbol{B}_k - \frac{\boldsymbol{B}_k \boldsymbol{s}_k \boldsymbol{s}_k^T \boldsymbol{B}_k}{\boldsymbol{s}_k^T \boldsymbol{B}_k \boldsymbol{s}_k} + \frac{\boldsymbol{y}_k \boldsymbol{y}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k} + \phi_k \left(\boldsymbol{s}_k^T \boldsymbol{B}_k \boldsymbol{s}_k\right) \boldsymbol{v}_k \boldsymbol{v}_k^T \tag{2.67}$$

gdje je ϕ_k skalarni parametar i

$$\boldsymbol{v}_k = \left[\frac{\boldsymbol{y}_k}{\boldsymbol{y}_k^T \boldsymbol{s}_k} - \frac{\boldsymbol{B}_k \boldsymbol{s}_k}{\boldsymbol{s}_k^T \boldsymbol{B}_k \boldsymbol{s}_k}\right] \tag{2.68}$$

Formula (2.58) za ažuriranje u matrice $\boldsymbol{B}_k$ u BFGS metodi se dobija kao specijalan slučaj formule (2.67) za $\phi_k = 0$, a formula (2.65) za ažuriranje matrice $\boldsymbol{B}_k$ u DFP algoritmu se dobije iz sa za $\phi_k = 1$, tako da sa možemo napisati kao linearnu kombinaciju ovih formula (2.58) i (2.65)za BFGS i DFP algoritma:

$$B_{k+1}=\left(1-\phi_k\right)B_{k+1}^{BFGS}+\phi_k B_{k+1}^{DFP} \tag{2.69}$$

Iz ove jednačine se vidi da svi članovi Broydenove klase zadovoljavaju jednačinu sekante (2.47), dok BFGS i DFP matrice same zadovojavaju ovu jednačinu i čuvaju pozotivnu definitnost aproksimacije Hessiana kada je zadovoljen uslov $s_k^T y_k > 0$. Ovu osobinu imaju i članovi Broydenove klase za $0 \le \phi_k \le 1$.

2.7 Globalna konvergencija i brzina konvergencije BFGS algoritma

Ovdje ćemo bez dokaza navesti teoreme koje osiguravaju dovoljne uslove za konvergenciju niza kojeg generira BFGS algoritam , odnosno za brzinu konvergencije ovog niza.

***Teorema**: Ako su zadovoljeni sljedeći uslovi:*

(i) *Kriterijalna funkcija f je dvostruko neprekidno diferencijabilna.*

(ii) *Skup $\mathcal{L}=\left\{x \in \mathbb{R}^n \;\middle|\; f\left(x\right) \le f\left(x^0\right)\right\}$ gdje je x^0 početna tačka, je konveksan i postoje pozitivne konstante m i M tako da vrijedi*

$$m\|z\|^2 \le z^T \nabla^2 f\left(x\right) z \le M\|z\|^2 \text{ , } \forall z \in \mathbb{R}^n, \quad x \in \mathcal{L} \tag{2.70}$$

(iii) *H_0 simetrična i pozitivno definitna početna matrica.*

Tada niz $\left\{x^k\right\}$ kojeg generira BFGS algoritam konvergira tački minimuma x^* kriterijalne funkcije f .

***Teorema**: Neka je kriterijalna funkcija f dvostruko neprekidno diferencijabilna i neka niz $\left\{x^k\right\}$ generiran BFGS algoritnom konvergira tački minimuma x^* kriterijalne funkcije f , tako da je matrica Hessijan funkcije f Lipschitz neprekidna u toj tački, tj. vrijedi:*

$$\left\|\nabla^2 f\left(x\right)-\nabla^2 f\left(x^0\right)\right\| \le L\left\|x-x^*\right\| \text{ , } \quad \forall x \in O_\delta\left(x^0\right) \text{ , } \quad \forall L \in \mathbb{R}^+ \tag{2.71}$$

i neka vrijedi:

$$\sum_{k=1}^{+\infty}\left\|x^k-x^*\right\| < \infty \tag{2.72}$$

Tada niz $\left\{x^k\right\}$ konvergira ka x^* superlinerno.

3. Implementacija algoritma inverzne kinematike u C++ programskom jeziku i simulacija u Matlab Simulink-u

U ovom dijelu navedeni su rezultati praktičnog dijela rada: sadržaj kreiranih fajlova i kreiranih biblioteka sa izvornim kodom koji definiše S-funkciju koja definira algoritam inverzne kinematike manipulatora, te simulacija kreirane S-funkcije koja predstavlja blok inverzne kinematike, i rezultati simulacije u Matlab R2020a Simulink-u za zadanu trajektoriju vrha manipulatora i zadane parametre. Za ogledni robot u simulaciji na kome je testiran rad implementiranog algoritma inverzne kinematike jeste *KUKA LBR iiwa 14 R820* (u kinematičkom smislu) *redudantni* robot (sa 7 stepeni pokretljivosti). U implementaciji algoritma inverzne kinematike korišten je algoritam izveden na osnovu BFGS metoda nelinearne optimizacije, koji je naveden u 2.5. Biblioteke kao i dijelovi S-funkcije su pisani u Dev C++ razvojnom okruženju. U header fajlovima `linesearch.h`, `bfgsagoritkm.h`, `matrixlibr.h` se nalaze deklaracije (prototipovi) funkcija i ovdje nije naveden njihov sadržaj kao i sadržaj još nekih fajlova koji su automatski generirani uz pomoć alata *S-function Builder*. Izvorni kod koji definira funkcije iz biblioteke `matrixlibr` u kojoj su definisane operacije nad matricama i vektorima, je naveden u dodatku A.

3.1 Putne tačke i trajektorija vrha manipulatora

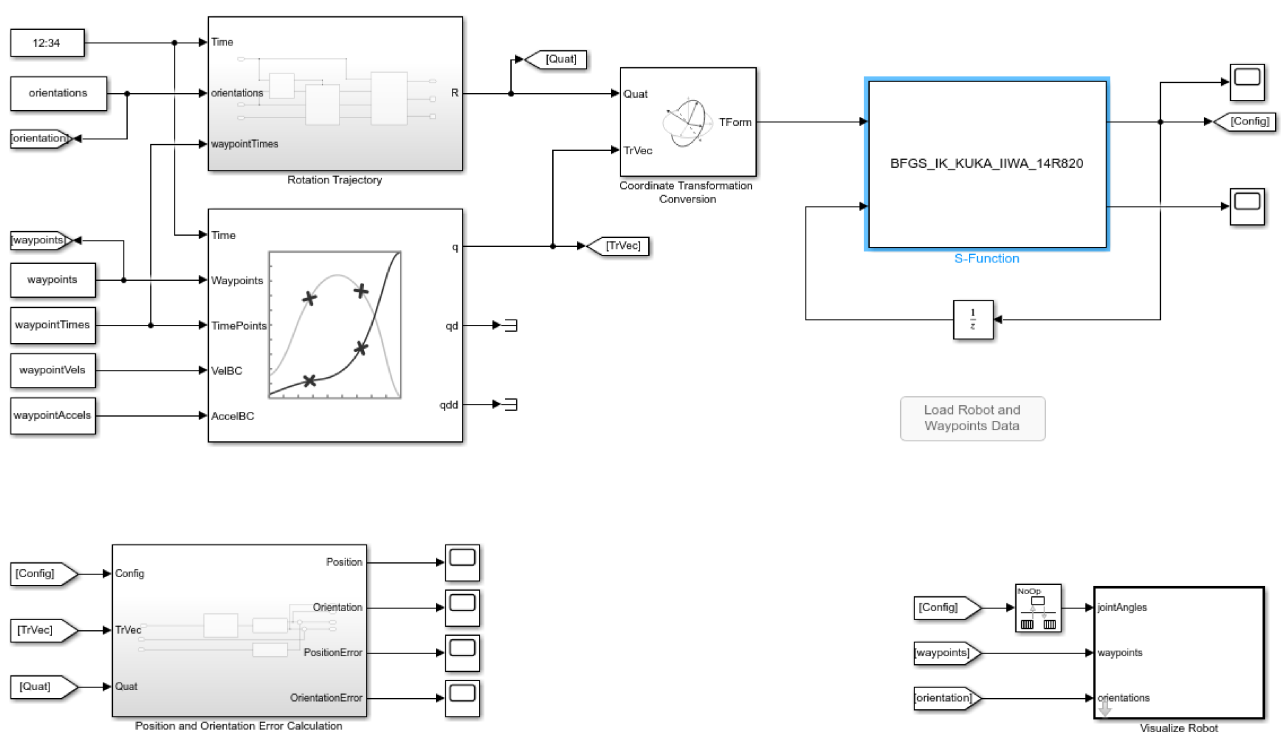
Trajektorija vrha manipulatora u ovoj simulaciji čiji je blok dijagram prikazan na slici 1.4, je zadana u operacijskom prostoru. Pritiskom na dugme `Load Robot and Waypoints Data` u Simulink blok dijagramu na slici 1.4 pokreće se izvršavanje M-fajla:

```
%Imprtovanje parametara KUKA LBR iiwa 14 R820 robotskog manipulatora
 %iz  .urdf fajla
robotArm=importrobot('iiwa14.urdf');
robotArm.DataFormat='column';
numJoints = numel(robotArm.homeConfiguration);
initConfig=robotArm.homeConfiguration;
maxWaypoints =7;
%Perioda generisanja uzoraka trajektorije
Ts=0.1;
%Pocetna putna tacka
homeConfigHomTr=getTransform(robotArm,robotArm.homeConfiguration,'world','iiwa_link_ee_
kuka');
%Putne tacke
waypoints =  [-homeConfigHomTr(1:3,4)';   %T1
               0.5  -0.5   0.3;           %T2
               0.5  -0.5   0.3;           %T2'
               0.5  -0.5   0.7;           %T3
               0.5  -0.5   0.7;           %T3'
               0   -0.75   0.7;           %T4
              -0.5  -0.5   0.7]';         %T5

%Orijentacija vrha manipulatora u putnim tackama
orientations = [0     0     0;
                0     pi/3  0;
                0     0     pi/2;
                0     0     pi/2;
                0     pi/8  pi/8;
                0     0     pi/4;
                0    -pi/8 -pi/8]';
%Vremena u kojima vrh manipulatora pristiže u putne tacke
```

```
waypointTimes = [0 5 6.5 9.5 11 13 15];
%Brzine vrha manipulatora u putnim tackama
waypointVels = 0.1 *[0    0    0;
                     0    0    0;
                     0    0    0;
                     0    0    0;
                     0    0    0;
                    -3    0    0
                     0    0    0]';

%Ubrzanja vrha manipulatora u putnim tackama
waypointAccels = zeros(size(waypointVels));
```

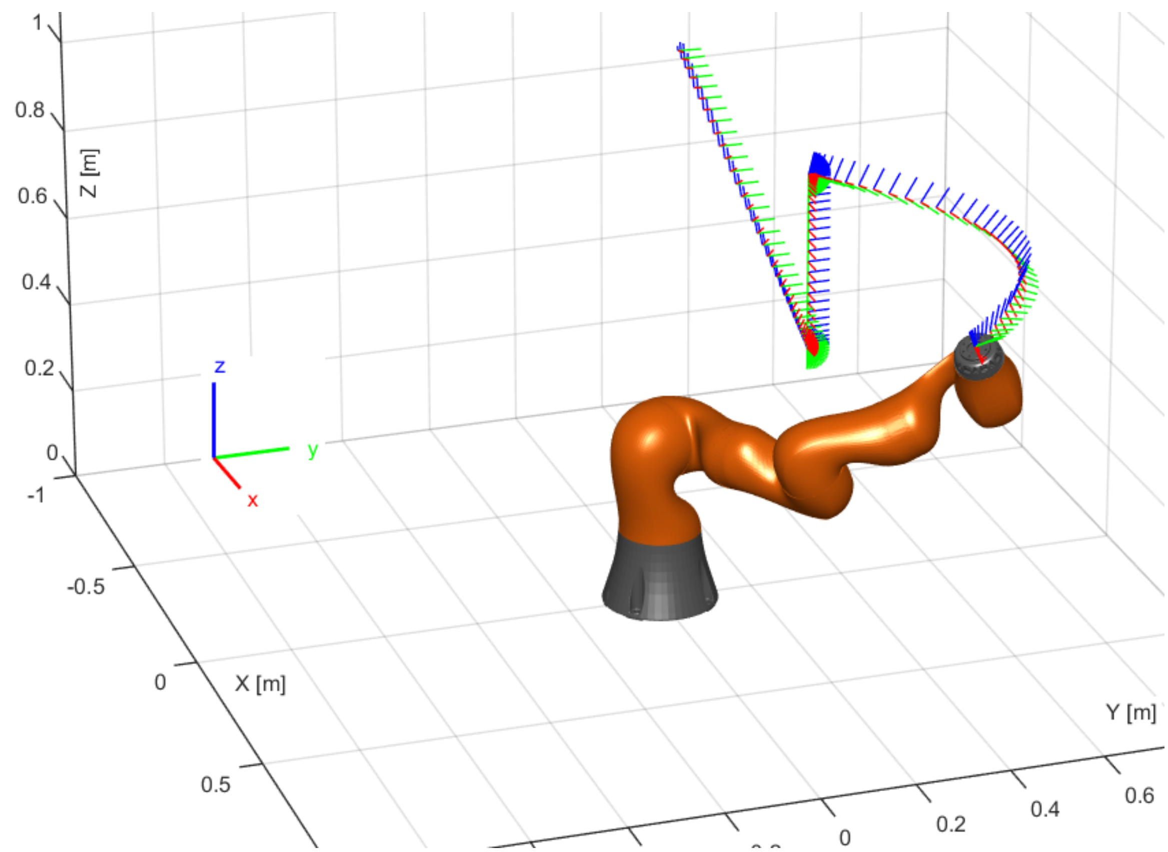
Pored učitavanja parametara modela robota korištenog u simulaciji, izvršavanjem ovog fajla učitavaju se *koordinate putnih tačaka*, orijentacija vrha manipulatora u putnim tačkama, vremena pristizanja vrha manipulatora u putne tačke, komponente *vektora brzine i ubrzanja* u putnim tačkama. Trajektorija u operacijskom prostoru je u ovom radu specificirana sa ovih pet putnih tački. Izabran je polinom petog reda koji ih interpolira i kod kojeg je pored specificirane brzine moguće specificirati i ubrzanje u putnim tačkama, a kojeficijente interpolacionog polinoma usloviti neprekidnošću ubrzanja u putnim tačkama. Za putnu tačku T_1 je uzeta *home konfiguracija* manipulatora. Komandom getTransform dobijamo matricu homogene transformacije u koordinatama baznog koordinatnog sistema za navedenu konfiguraciju. Putne tačke T_2 i T_3 su zadane tako da u njima vrh manipulatora mijenja samo orijentaciju, dok za vrijeme promjene orijentacije u istim, vrh manipulatora zadržava zadanu fiksnu poziciju. Ovo ima za cilj da se demonstrira rad algoritma inverzne kinematike koji se odnosi na orijentaciju u funkciji kriterija (2.13). Orijentacija u putnim tačkama je specificirana ZYX Eulerovim uglovima.

3.2 Simulacija kreirane S-funkcije kao bloka inverzne kinematike i rezultati simulacije

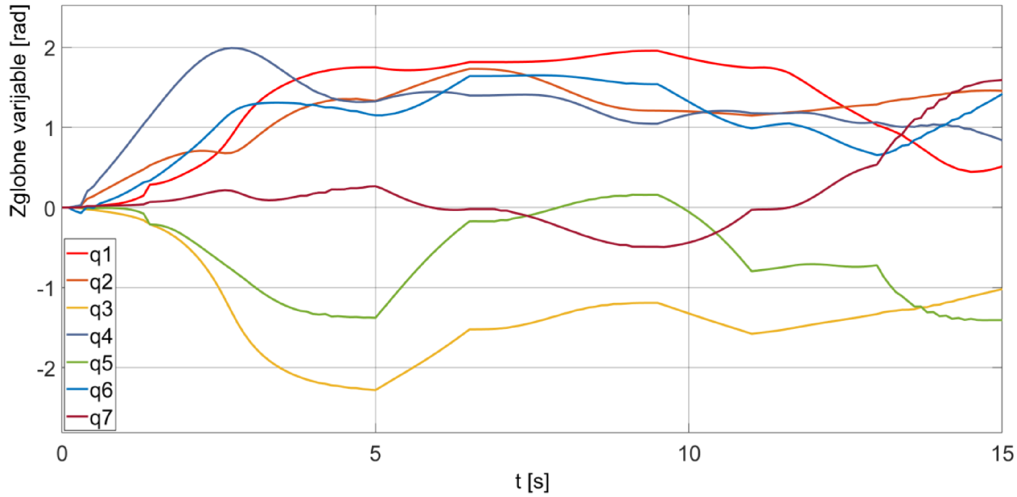


Slika 1. 4: Simulink blok dijagram kompletne simulacije

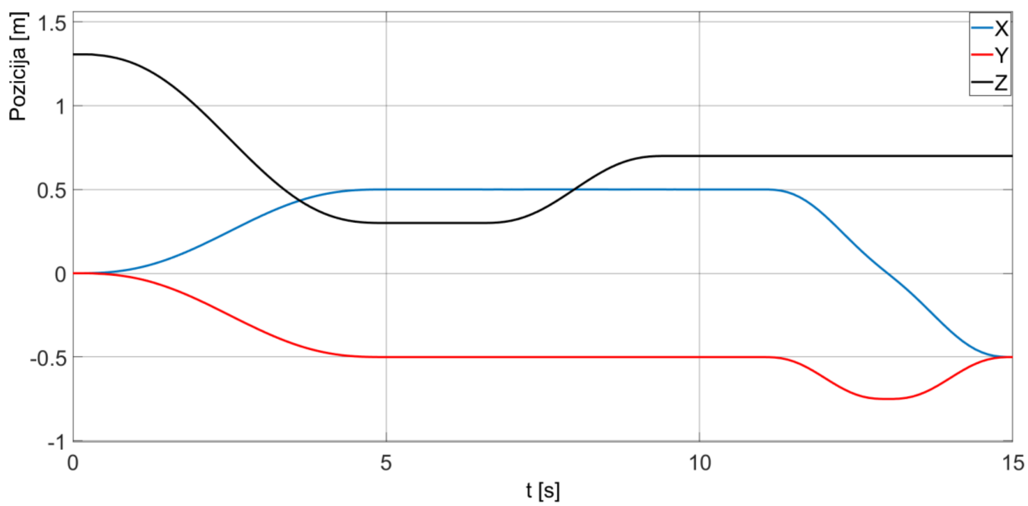
Ovdje je predstavljena simulacija i njeni rezultati čija je Simulink blok-struktura predstavljena na slici 1.4, sa S-funkcijom koja u simulaciji predstavlja blok inverzne kinematike, čija je funkcionalnost definisana izvornim kodom fajlova i biblioteka navedenih u naslovima 3.3 i 3.4. Subsistem Rotational Trajectory predstavljen blok dijagramom na slici 1.4 koji generiše rotacijsku trajektoriju u i između putnih tački je preuzet iz primjera sa github.com, kao i blok Visualize Robot (fajlovi trajExampleUtils, visualizeRobot). Izlaz iz bloka Coordinate Transformation Conversion je (diskretno-vremenski zavisna) matrica homogene transformacije, koja određuje željenu-zadanu pozu (poziciju i orijentaciju) vrha manipulatora. Ova matrica je ulaz u S-funkciju koja predstavlja blok inverzne kinematike, što je prestavljeno na slici 1.4. Drugi ulaz u blok naše S-funkcije je inicijalna vrijednost, tj. početna aproksimacija algoritma inverzne kinematike, koja je jednaka vrijednosti vektora zglobnih varijabli (uglova) zakašnjelih za jedan period, što je u stvari jednako tom vektoru iz prethodne iteracije. Ovim je znatno olakšano pretraživanje algoritma, zbok jako složene konfiguracije hiperpovrši čija je jednačina određena funkcijom kriterija (2.13). Konfiguracije hiperpovrši donekle se može naslutiti imajući u vidu dvodimenzionalni slučaj predstavljen na slici 1.3. Izlaz iz Bloka S-funkcije je vektor zglobnih varijabli, izračunat algoritmom inverzne kinematike na osnovu ulaza u blok. To su uglovi koje zglobovi manipulatora treba da zauzmu određenom brzinom da bi se u datom trenutku ostvario zadani položaj i orijentacija, odnosno zadano kretanje vrha manipulatora.



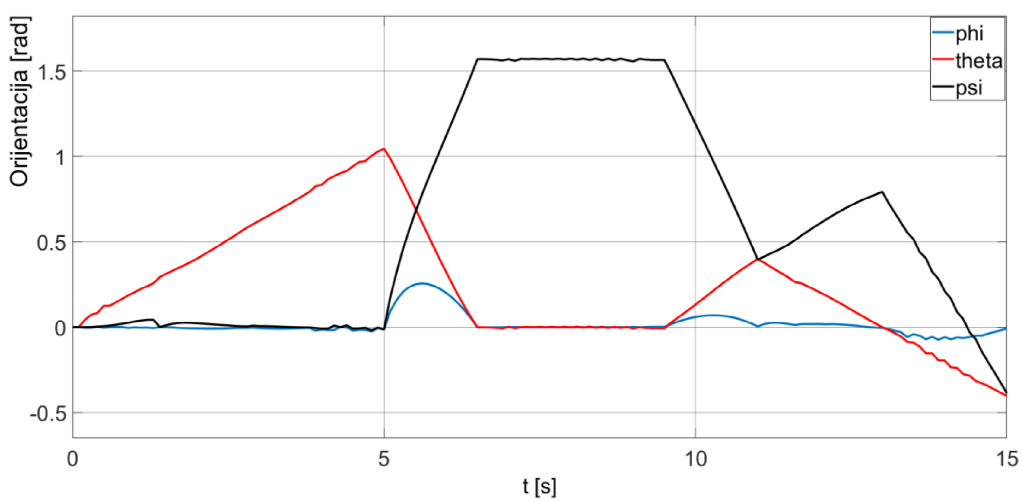
Slika 1. 5: Uzorci orijentacije vrha manipulatora (KUKA LBR iiwa 14 R820) duž njegove putanje za T_s=0.1s, za zadane vrijednosti matrica težinskih kojeficijenata u (2.13) i zadane vrijednosti kriterija zaustavljanja algoritma pretraživanja po pravcu i BFGS algoritma



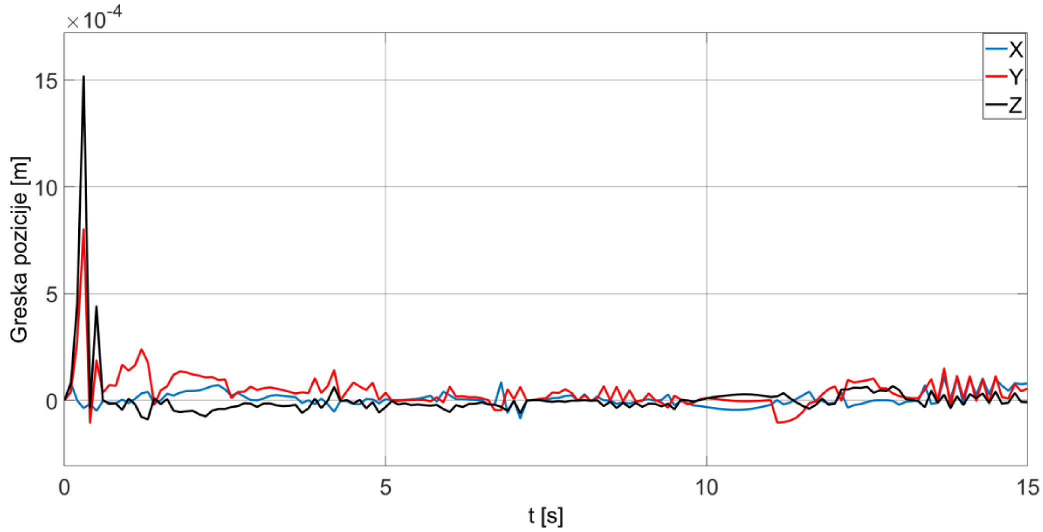
Slika 1.6 : Vremenski dijagram zglobnih varijabli



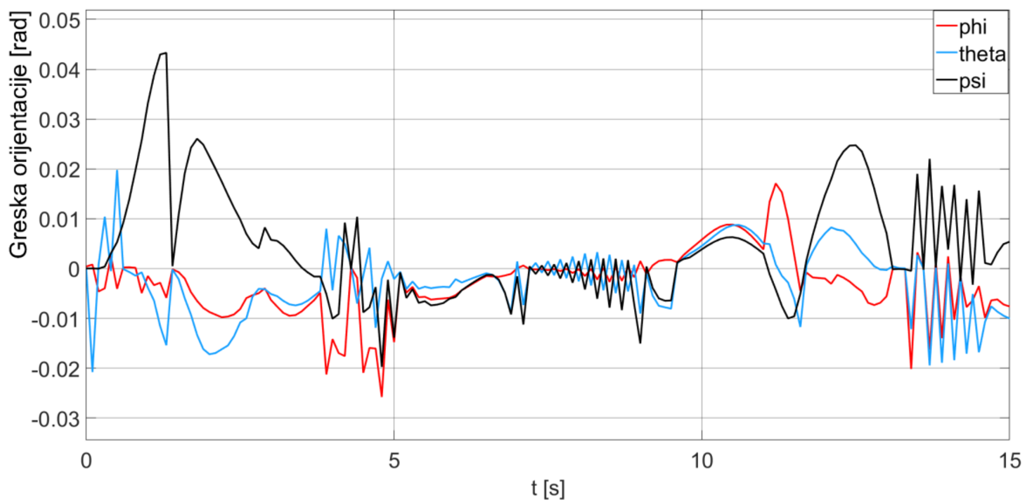
Slika 1.7: Vremenski dijagram koordinata pozicije vrha manipulatora



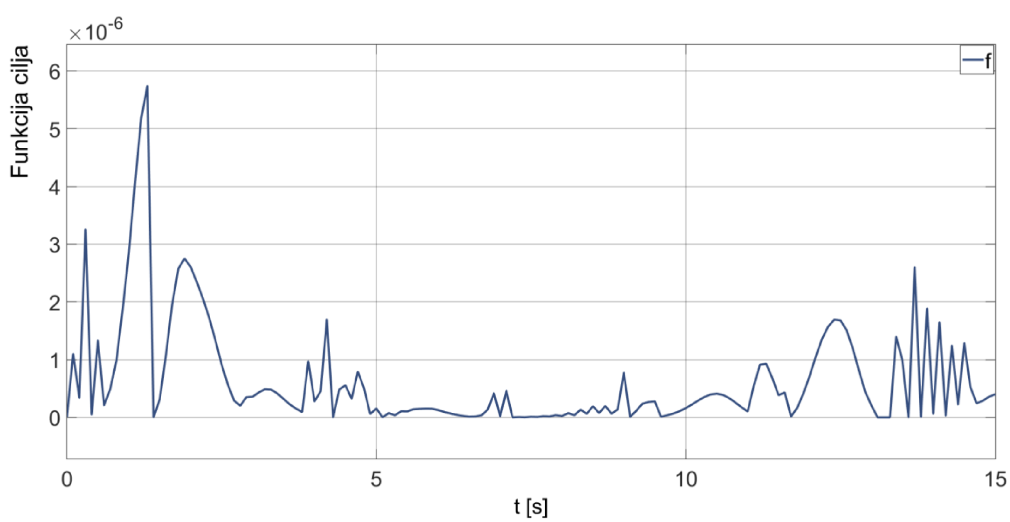
Slika 1. 8: Vremenski dijagram ZYX Eulerovh uglova koji predstavljaju orijentaciju vrha manipulatora



Slika 1.9: Vremenski dijagram koordinata greške orijentacije vrha manipulatora



Slika 1.10: Vremenski dijagram ZYX Eulerovih uglova greške orijentacije vrha manipulatora



Slika 1.11: Vremenski dijagram funkcije kriterija

3.3 Sadržaj fajlova koji definišu funkcije korištenih biblioteka

Ovdje je dat izvorni kod biblioteka `linesearch` i `bfgsalgorithm`, dok izvorni kod biblioteke `matrixlibr` je zbog preglednosti rada dat u dodatku A. Zbog mogučnosti proslijeđivanja cijelog vektora/matrice funkciji deklarisana je struktura `Variable`, i njena deklaracija se nalazi u `matrixlibr.cpp` u dodatku A , kao i deklaracija strukture `Quaternion`. Algoritam naveden u 2.5 čija je implementacija data u biblioteci `bfgsalgorithm` za tačku optimuma pronalazi tačku (vektor zglobnih varijabli) lokalnog minimuma.

Pri računanju izvoda uzeto je da je su svi priraštaji i svi parcijalni priraštaji jednaki $h = 10^{-5}$. U linijama 42. i 116. uzete su konkretne brojne vrijednosti koje definišu kriterij zaustavljanja algoritma. Ovdje treba napomenuti da je moguće kreirati masku bloka S-funkcije u cilju da se omogući proizvoljno zadavanje ovih parametara kao parametara bloka. Slično vrijedi za matrice težinskih kojeficijenata funkcije kriterija iz (2.13) u linijama 85. i 86.

3.3.1 Biblioteka `linesearch`

```
1.  #include "linesearch.h"
2.
     /* ================================================================   *
      * Linija  odredjena  sa  (2.15)  po  kojoj  vrsimo  jednodimenzionalno  *
      * pretrazivanje po parametru alpha.                                    *
      * ================================================================   */
3.  struct Variable line (struct Variable point, struct Variable direction, double
    alpha){
4.      struct Variable linea=initialize_vector_column(point.X.size());
5.      for (int i=0; i<point.X.size(); i++){
6.          linea.X[i][0] = point.X[i][0] + alpha * direction.X[i][0];
7.      }
8.      return linea;
9.  }
10.
11. /* Izvod funkcije f na pravoj odredjenom tackom 'point' i pravcem 'direction',
    za parametar t. */
12.
13. double derivative_on_line (double( *f)(struct Variable, struct Variable, struct
    Variable), struct Variable dh, struct Variable dpc, struct Variable point,
    struct Variable direction, double t){
14.
15.     double  df  =  f(dh,  dpc,  line(point,  direction,  t+h))  -  f(dh,  dpc,
    line(point, direction, t-h));
16.     return df/(2.0*h);
17. }
18.
19. /*  Drugi  izvod  funkcije  f  na  pravoj  odredjenom  tackom  'point'  i  pravcem
    'direction', za parametar t. */
20.
21. double second_derivative  (double( *f)(struct Variable, struct Variable, struct
    Variable), struct Variable dh, struct Variable dpc, struct Variable point,
    struct Variable direction, double t){
22.
23.     double  ddf  =  derivative_on_line  (f,  dh,  dpc,  point,  direction,  t+h)  -
    derivative_on_line (f, dh, dpc, point, direction, t-h);
24.     return ddf/(2.0*h);
25. }
26.
```

```
    /* =========================================================== *
     * Jednodimenzionalno  pretrazivanje   za   odredjivanje   optimalnog *
     * koraka prema (2.16) odnosno (2.61). odredjeno sa (2.24) po pravoj *
     * kroz tacku 'point' sa pravcem 'direction'. Kriterij zaustavljanja *
     * (eps) je dostizanje minimalne promjene problemske varijable i ta *
     * promjena je jednaka 0.0001 (linija 42.). Pocetna vrijednost x1k je *
     * uzeta           proizvoljno            (linija            34.). *
     * =========================================================== */
27.  struct Variable newton_two_points (double( *f)(struct Variable, struct
   Variable, struct Variable), struct Variable dh, struct Variable dpc, struct
   Variable point, struct Variable direction){
28.     int i;
29.     double xk1;
30.     double eps;
31.     struct Variable XK=point;
32.     struct Variable optimal_step=initialize_vector_column(1);
33.     double xk=0.0;
34.     double x1k=0.2;
35.     do{
36.         double dxk = xk-x1k;
37.         double d_derivative_on_line_k = derivative_on_line( f, dh, dpc, XK,
   direction, xk) - derivative_on_line( f, dh, dpc, XK, direction, x1k);
38.         xk1=x1k+(dxk/(1-((derivative_on_line(f,  dh,  dpc,  XK,  direction,
   xk)/derivative_on_line(f, dh, dpc, XK, direction, x1k))*(d_derivative_on_line_k
   /(dxk*second_derivative( f, dh, dpc, XK, direction, xk))))));
39.         eps=fabs(xk1-xk);
40.         x1k=xk;
41.         xk=xk1;
42.     }while (eps>0.0001);
43.     optimal_step.X[0][0]=xk1;
44.     return optimal_step;
45. }
```

3.3.2 Biblioteka bfgsalgorithm

```
#include "bfgsalgorithm.h"

  /* =========================================================== *
   * Matrica  (4x4)  homogene  transformacije  iz  jednacine  (1.14) *
   * =========================================================== */
struct Variable h_t_matrix (struct Variable A, int i){
struct Variable B=initialize_matrix(4, 4);
B.X[0][0]=cos(A.X[i][3]);          B.X[0][1]=-sin(A.X[i][3])*cos(A.X[i][1]);
B.X[1][0]=sin(A.X[i][3]);          B.X[1][1]= cos(A.X[i][3])*cos(A.X[i][1]);
B.X[2][0]=0.0;                     B.X[2][1]= sin(A.X[i][1]);
B.X[3][0]=0.0;                     B.X[3][1]= 0.0;

B.X[0][2]= sin(A.X[i][3])*sin(A.X[i][1]); B.X[0][3]=A.X[i][0]*cos(A.X[i][3]);
B.X[1][2]=-cos(A.X[i][3])*sin(A.X[i][1]); B.X[1][3]=A.X[i][0]*sin(A.X[i][3]);
B.X[2][2]= cos(A.X[i][1]);               B.X[2][3]=A.X[i][2];
B.X[3][2]= 0.0;                          B.X[3][3]=1.0;
return B;
}

```

```
 /* ================================================================   *
  *  Matrica (4x4) koja odredjuje jednacinu direktne kinematike koja se  *
  *  dobije    mnozenjem    matrica    A    prema    jednacini    (1.10).  *
  *  ================================================================   */
struct Variable direct_cinematics_matrix( struct Variable A){
struct Variable M, N;
N = h_t_matrix(A, 0);
for (int i=1; ; ){
    M = product(N, h_t_matrix(A, i));
    i++;
    if(i>=A.X.size()) break;
    N = product(M, h_t_matrix(A, i));
    i++;
    if(i>=A.X.size()) break;
    }
    if(A.X.size()%2==0) {return M;}
    else                {return N;}
}

 /* ================================================================   *
  *  Transformacija matrice rotacije A u kvaternion prema relaciji (2.8),  *
  *  uz            tretiranje            specijalnih            slucajeva.  *
  *  ================================================================   */
struct Quaternion rotm_to_quats (struct Variable A){
struct Quaternion B;
double trace = A.X[0][0]+A.X[1][1]+A.X[2][2];
if (trace>h) {
        double s = 0.5/sqrt(trace+1.0);
        B.s = 0.25/s;
        B.v[0] = (A.X[2][1] - A.X[1][2]) * s;
        B.v[1] = (A.X[0][2] - A.X[2][0]) * s;
        B.v[2] = (A.X[1][0] - A.X[0][1]) * s;
     } else {
        if (A.X[0][0] - A.X[1][1] >h && A.X[0][0] - A.X[2][2] >h){
            double s = 2.0*sqrt(1.0 + A.X[0][0] - A.X[1][1] - A.X[2][2]);
             B.s = (A.X[2][1] - A.X[1][2])/s;
             B.v[0] = 0.25 * s;
             B.v[1] = (A.X[0][1] + A.X[1][0]) / s;
             B.v[2] = (A.X[0][2] + A.X[2][0]) / s;
         } else if (A.X[1][1] - A.X[2][2]>h){
            double s = 2.0 * sqrt(1.0 + A.X[1][1] - A.X[0][0] - A.X[2][2]);
             B.s = (A.X[0][2] - A.X[2][0]) / s;
             B.v[0] = (A.X[0][1] + A.X[1][0]) / s;
             B.v[1] = 0.25 * s;
             B.v[2] = (A.X[1][2] + A.X[2][1]) / s;
         } else {
            double s = 2.0 * sqrt(1.0 + A.X[2][2] - A.X[0][0] - A.X[1][1]);
             B.s = (A.X[1][0] - A.X[0][1]) / s;
             B.v[0] = (A.X[0][2] + A.X[2][0]) / s;
             B.v[1] = (A.X[1][2] + A.X[2][1]) / s;
             B.v[2] = 0.25 * s;
         }
     }
    return B;
 }
```

```
 /* ================================================================  *
  *  Racunanje greske orijentacije definisanom u terminima kvaterniona    *
  *  koja je odredjena sa (2.12).                                         *
  *  dp_htm - desired pose homogene transform matrix,                     *
  *  ap_htm - actual  pose homogene transform matrix .                    *
  *  ================================================================  */
struct Quaternion orientation_error (struct Variable dp_htm, struct Variable ap_htm){
struct Quaternion D = rotm_to_quats(dp_htm);
struct Quaternion E = rotm_to_quats(ap_htm);
struct Quaternion delta;
delta.s = 1.0;
delta.v[0] = E.s * D.v[0] - D.s * E.v[0] - (-D.v[2] * E.v[1] +D.v[1]*E.v[2]);
delta.v[1] = E.s * D.v[1] - D.s * E.v[1] - ( D.v[2] * E.v[0] -D.v[0]*E.v[2]);
delta.v[2] = E.s * D.v[2] - D.s * E.v[2] - (-D.v[1] * E.v[0] +D.v[0]*E.v[1]);
return delta;
}

 /* ================================================================  *
  *  Funkcija   cilja   'criterion',   var.X[i][0]   (i=1,2,...,n)   su  *
  *  varijable ove funkcije i one predstavljaju zglobne promjenjive q1,  *
  *  q2,  ...,qn.  Matrica  tezinskih  koeficijenata  Mp  iz  (2.13)  je  *
  *  dijagonalna  i  njeni  elementi  na  glavnoj  dijagonali  su  uzeti  da  su  *
  *  jednaki  2,  a  elementi  glavne  dijagonale  matrice  Mo  su  uzeti  da  su  *
  *  jednaki 0.02.                                                        *
  *  ================================================================  */
double criterion (struct Variable dh, struct Variable dp_htm, struct Variable var){
for (int i=0; i<dh.X.size(); i++){
    dh.X[i][3] = var.X[i][0];
}
struct Variable dcm = direct_cinematics_matrix(dh);
struct Quaternion delta = orientation_error (dp_htm, dcm);
double position_part=0.0;
double orientation_part=0.0;
for(int i=0; i<3; i++){
   position_part += pow(dp_htm.X[i][3] - dcm.X[i][3], 2);
   orientation_part += 0.01 * pow(delta.v[i], 2);
};
return position_part + orientation_part;
}

 /* ================================================================  *
  *  Gradijent funkcije f po promjenjivoj 'var', i on predstavlja vektor-  *
  *  kolonu.                                                              *
  *  dpc=dp_htm                                                           *
  *  ================================================================  */
struct Variable gradient (double(*f)(struct Variable, struct Variable, struct Variable), struct Variable dh, struct Variable dpc, struct Variable var){
int i;
struct Variable var_1, var_2;
struct Variable partial_derivatives;
partial_derivatives=initialize_vector_column(var.X.size());
for(i=0; i<var.X.size(); i++){
var_1 = var;
var_2 = var;
var_1.X[i][0] += h;
```

```
var_2.X[i][0] -= h;
partial_derivatives.X[i][0] = ( f(dh, dpc, var_1) - f(dh, dpc, var_2) )/(2.0*h);
}
return partial_derivatives;
}

 /* ======================================================================= *
  *  Implementacija BFGS algoritma opisanog u 2.5.                          *
  *  f - funkcija kriterija data sa (2.13),                                 *
  *  dh - matrica DH  parametara robota,                                    *
  *  start_point-pocetna aproksimacija,                                     *
  *   dp_htm-matr. homogene transformacije zeljene pozicije i orijentacije. *
  *  ====================================================================== */
struct Variable BFGS ( double( *f)(struct Variable, struct Variable, struct Variable), struct Variable dh, struct Variable dp_htm, struct Variable start_point){
struct Variable Xk, Xk1, DXk;
Variable Gk, Gk1, DGk;
Variable step, Rk;
Variable Hk, Hk1;
Variable Mk, Nk;
double gamma, beta;
Xk = start_point;
Hk = eye(start_point.X.size());
/* Kriterij zaustavljanja dat sa (2.59). */
while (vector_norm(gradient(f, dh, dp_htm, Xk))>0.0001){
Gk    = gradient(f, dh, dp_htm, Xk);
/* Pravac pretrazivanja dat sa (2.60). */
Rk    = c(product(Hk, Gk), -1);
/* Jednodimenzionalno pretrazivanje (2.61). */
step  = newton_two_points (f, dh, dp_htm, Xk, Rk);
/* Racunanje nove aproksimacije (2.62). */
Xk1   = sum(Xk, product(Rk, step));
Gk1   = gradient(f, dh, dp_htm, Xk1);
/* Razlika data sa(2.63a). */
DXk   = razlika(Xk1, Xk);
/* Razlika data sa(2.63b). */
DGk   = razlika(Gk1, Gk);
gamma = (product( transpose(DXk), DGk)).X[0][0];
beta  = (product( product( transpose(DGk), Hk), DGk)).X[0][0];
Mk    = c(product(DXk, transpose(DXk)), (gamma+beta)/pow(gamma,2));
Nk        = c( sum( product( product(Hk, DGk), transpose(DXk)), product( product(DXk, transpose(DGk)), Hk)), -1.0/gamma);
/* Racunanje aproksimacije inverznog Hessiana H(k+1) prema (2.56). */
Hk1   = sum( sum(Hk, Mk), Nk);
Hk    = Hk1;
Xk    = Xk1;
}
return Xk1;
}
```

3.4 Sadržaj fajlova koji definišu funkcionalnost S-funkcije

U generisanju dijelova izvornog koda fajlova `BFGS_IK_KUKA_IIWA_14R820.cpp` i `BFGS_IK_KUKA_IIWA_14R820_wrapper.cpp` korišten je alat S-function Builder. Ovaj alat je generisao i komentar koda koji zbog umanjenja tekstualnog obima ovdje nije naveden. Izvorni kod ovih fajlova moguće je generisati koristeći gotov S-function template.

Treba napomenuti da je moguće kreirati masku za blok S-funkcije sa ciljem da se broj zglobova robota kao i DH parametri zadaju kao parametri bloka ili iz workspace-a. Zbog obima rada ovdje to nije učinjeno.

3.4.1 BFGS_IK_KUKA_IIWA_14R820.cpp

```
#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME BFGS_IK_KUKA_IIWA_14R820
#define NUM_INPUTS              2
/* Input Port  0 – ulaz u blok S-funkcije je (4 x 4) matrica homogene transformacije koja odredjuje zeljenu pozu (poziciju i orijentaciju)vrha manipulatora. */
#define IN_PORT_0_NAME          Pose
#define INPUT_0_WIDTH           4
#define INPUT_DIMS_0_COL        4
#define INPUT_0_DTYPE           real_T
#define INPUT_0_COMPLEX         COMPLEX_NO
#define IN_0_FRAME_BASED        FRAME_NO
#define IN_0_BUS_BASED          0
#define IN_0_BUS_NAME
#define IN_0_DIMS               2-D
#define INPUT_0_FEEDTHROUGH     1
#define IN_0_ISSIGNED           0
#define IN_0_WORDLENGTH         8
#define IN_0_FIXPOINTSCALING    1
#define IN_0_FRACTIONLENGTH     9
#define IN_0_BIAS               0
#define IN_0_SLOPE              0.125
/* Input Port  1 -  ulaz u blok S-funkcije jeste pocetna aproksimacija za tekucu  iteraciju koja je jednaka (7 x 1)vektoru zglobnih varijabli iz prethodne iteracije.*/
#define IN_PORT_1_NAME          Init
#define INPUT_1_WIDTH           7
#define INPUT_DIMS_1_COL        1
#define INPUT_1_DTYPE           real_T
#define INPUT_1_COMPLEX         COMPLEX_NO
#define IN_1_FRAME_BASED        FRAME_NO
#define IN_1_BUS_BASED          0
#define IN_1_BUS_NAME
#define IN_1_DIMS               1-D
#define INPUT_1_FEEDTHROUGH     1
#define IN_1_ISSIGNED           0
#define IN_1_WORDLENGTH         8
#define IN_1_FIXPOINTSCALING    1
#define IN_1_FRACTIONLENGTH     9
#define IN_1_BIAS               0
#define IN_1_SLOPE              0.125

#define NUM_OUTPUTS             2
/* Output Port  0 – izlaz iz bloka S-funkcije jeste (7 x 1)vektor zglobnih varijabli.*/
#define OUT_PORT_0_NAME         Config
#define OUTPUT_0_WIDTH          7
```

```
#define OUTPUT_DIMS_0_COL     1
#define OUTPUT_0_DTYPE        real_T
#define OUTPUT_0_COMPLEX      COMPLEX_NO
#define OUT_0_FRAME_BASED     FRAME_NO
#define OUT_0_BUS_BASED       0
#define OUT_0_BUS_NAME
#define OUT_0_DIMS            1-D
#define OUT_0_ISSIGNED        1
#define OUT_0_WORDLENGTH      8
#define OUT_0_FIXPOINTSCALING 1
#define OUT_0_FRACTIONLENGTH  3
#define OUT_0_BIAS            0
#define OUT_0_SLOPE           0.125
/* Output Port  1 - izlaz iz blika S-funkcije jeste vrijednost funkcije
kriterija za izracunati vektor zglobnih varijabli.*/
#define OUT_PORT_1_NAME       CriterionValue
#define OUTPUT_1_WIDTH        1
#define OUTPUT_DIMS_1_COL     1
#define OUTPUT_1_DTYPE        real_T
#define OUTPUT_1_COMPLEX      COMPLEX_NO
#define OUT_1_FRAME_BASED     FRAME_NO
#define OUT_1_BUS_BASED       0
#define OUT_1_BUS_NAME
#define OUT_1_DIMS            1-D
#define OUT_1_ISSIGNED        1
#define OUT_1_WORDLENGTH      8
#define OUT_1_FIXPOINTSCALING 1
#define OUT_1_FRACTIONLENGTH  3
#define OUT_1_BIAS            0
#define OUT_1_SLOPE           0.125
#define NPARAMS               0
#define SAMPLE_TIME_0         INHERITED_SAMPLE_TIME
#define NUM_DISC_STATES       0
#define DISC_STATES_IC        [0]
#define NUM_CONT_STATES       0
#define CONT_STATES_IC        [0]
#define SFUNWIZ_GENERATE_TLC  1
#define SOURCEFILES           "__SFB__"
#define PANELINDEX            8
#define USE_SIMSTRUCT         0
#define SHOW_COMPILE_STEPS    0
#define CREATE_DEBUG_MEXFILE  0
#define SAVE_CODE_ONLY        0
#define SFUNWIZ_REVISION      3.0

#include "simstruc.h"

extern void BFGS_IK_KUKA_IIWA_14R820_Outputs_wrapper(const real_T *Pose,
const real_T *Init, real_T *Config, real_T *CriterionValue);
/*===============================*
Data Transposition Routines *
*===============================*/
int linear_idx(const int srcdims[], const int numdims, const int dstk){
int idxCurDim;
int i, j;
int srck = 0, dstk_remainingDims = dstk;
for(i=0,j=numdims-1; i<numdims; i++, j--){
```

```c
    idxCurDim = dstk_remainingDims % srcdims[j];
    srck = srck * srcdims[j] + idxCurDim;
    dstk_remainingDims = dstk_remainingDims / srcdims[j];
}
return srck;
}

void NDTransposeBySrcSpecs(void *dst, const void *src, const int srcdims[],
const int numdims, const int elsize){
int w = srcdims[0];
int k;
for (k = 1; k < numdims; k ++) {
    w *= srcdims[k];
}

for (k = 0; k < w; k ++) {
    int sk = linear_idx(srcdims, numdims, k);
    int offset = k * elsize;
    memcpy((char*)dst + k * elsize, (const char*)src + sk * elsize, elsize);
}
}
void NDTransposeByDstSpecs(void *dst, const void *src, const int dstdims[],
const int numdims, const int elsize) {
int w = dstdims[0];
int k;
for (k = 1; k < numdims; k ++) {
    w *= dstdims[k];
}

for (k = 0; k < w; k ++) {
    int dk = linear_idx(dstdims, numdims, k);
    int offset = k * elsize;
    memcpy((char*)dst + dk * elsize, (const char*)src + k * elsize, elsize);
}
}
/*====================*
S-function methods *
*====================*/
static void mdlInitializeSizes(SimStruct *S){
DECL_AND_INIT_DIMSINFO(inputDimsInfo);
DECL_AND_INIT_DIMSINFO(outputDimsInfo);
ssSetNumSFcnParams(S, NPARAMS);
if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
  return; /* Parameter mismatch will be reported by Simulink */
}

ssSetArrayLayoutForCodeGen(S, SS_ROW_MAJOR);

ssSetOperatingPointCompliance(S, USE_DEFAULT_OPERATING_POINT);

ssSetNumContStates(S, NUM_CONT_STATES);
ssSetNumDiscStates(S, NUM_DISC_STATES);


if (!ssSetNumInputPorts(S, NUM_INPUTS)) return;
/* Input Port 0 */
inputDimsInfo.width = INPUT_0_WIDTH;
```

```
ssSetInputPortDimensionInfo(S, 0, &inputDimsInfo);
ssSetInputPortMatrixDimensions(S, 0, INPUT_0_WIDTH, INPUT_DIMS_0_COL);
ssSetInputPortFrameData(S, 0, IN_0_FRAME_BASED);
ssSetInputPortDataType(S, 0, SS_DOUBLE);
ssSetInputPortComplexSignal(S, 0, INPUT_0_COMPLEX);
ssSetInputPortDirectFeedThrough(S, 0, INPUT_0_FEEDTHROUGH);
ssSetInputPortRequiredContiguous(S, 0, 1); /*direct input signal access*/

/* Input Port 1 */
ssSetInputPortWidth(S, 1, INPUT_1_WIDTH);
ssSetInputPortDataType(S, 1, SS_DOUBLE);
ssSetInputPortComplexSignal(S, 1, INPUT_1_COMPLEX);
ssSetInputPortDirectFeedThrough(S, 1, INPUT_1_FEEDTHROUGH);
ssSetInputPortRequiredContiguous(S, 1, 1); /*direct input signal access*/


if (!ssSetNumOutputPorts(S, NUM_OUTPUTS)) return;
/* Output Port 0 */
ssSetOutputPortWidth(S, 0, OUTPUT_0_WIDTH);
ssSetOutputPortDataType(S, 0, SS_DOUBLE);
ssSetOutputPortComplexSignal(S, 0, OUTPUT_0_COMPLEX);
/* Output Port 1 */
ssSetOutputPortWidth(S, 1, OUTPUT_1_WIDTH);
ssSetOutputPortDataType(S, 1, SS_DOUBLE);
ssSetOutputPortComplexSignal(S, 1, OUTPUT_1_COMPLEX);
if (!ssSetNumDWork(S, 4)) return;

/*
Configure the dwork 0 (Pose_t)
*/
ssSetDWorkDataType(S, 0, ssGetInputPortDataType(S, 0));
ssSetDWorkUsageType(S, 0, SS_DWORK_USED_AS_SCRATCH);
ssSetDWorkName(S, 0, "Pose_t");
ssSetDWorkWidth(S, 0, ssGetInputPortWidth(S, 0));
ssSetDWorkComplexSignal(S, 0, COMPLEX_NO);
/*
Configure the dwork 1 (Init_t)
*/
ssSetDWorkDataType(S, 1, ssGetInputPortDataType(S, 1));
ssSetDWorkUsageType(S, 1, SS_DWORK_USED_AS_SCRATCH);
ssSetDWorkName(S, 1, "Init_t");
ssSetDWorkWidth(S, 1, ssGetInputPortWidth(S, 1));
ssSetDWorkComplexSignal(S, 1, COMPLEX_NO);
/*
Configure the dwork 2 (Config_t)
*/
ssSetDWorkDataType(S, 2, ssGetOutputPortDataType(S, 0));
ssSetDWorkUsageType(S, 2, SS_DWORK_USED_AS_SCRATCH);
ssSetDWorkName(S, 2, "Config_t");
ssSetDWorkWidth(S, 2, ssGetOutputPortWidth(S, 0));
ssSetDWorkComplexSignal(S, 2, COMPLEX_NO);
/*
Configure the dwork 3 (CriterionValue_t)
*/
ssSetDWorkDataType(S, 3, ssGetOutputPortDataType(S, 1));
ssSetDWorkUsageType(S, 3, SS_DWORK_USED_AS_SCRATCH);
ssSetDWorkName(S, 3, "CriterionValue_t");
```

```
ssSetDWorkWidth(S, 3, ssGetOutputPortWidth(S, 1));
ssSetDWorkComplexSignal(S, 3, COMPLEX_NO);
ssSetNumPWork(S, 0);

ssSetNumSampleTimes(S, 1);
ssSetNumRWork(S, 0);
ssSetNumIWork(S, 0);
ssSetNumModes(S, 0);
ssSetNumNonsampledZCs(S, 0);

ssSetSimulinkVersionGeneratedIn(S, "10.1");

/* Take care when specifying exception free code - see sfuntmpl_doc.c */
ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}
/* Function: mdlInitializeSampleTimes ==========================================
Abstract:
Specifiy  the sample time.
*/
static void mdlInitializeSampleTimes(SimStruct *S)
{
ssSetSampleTime(S, 0, SAMPLE_TIME_0);
ssSetModelReferenceSampleTimeDefaultInheritance(S);
ssSetOffsetTime(S, 0, 0.0);
}
#define MDL_SET_INPUT_PORT_DATA_TYPE
static void mdlSetInputPortDataType(SimStruct *S, int port, DTypeId dType)
{
ssSetInputPortDataType(S, 0, dType);
}

#define MDL_SET_OUTPUT_PORT_DATA_TYPE
static void mdlSetOutputPortDataType(SimStruct *S, int port, DTypeId dType)
{
ssSetOutputPortDataType(S, 0, dType);
}

#define MDL_SET_DEFAULT_PORT_DATA_TYPES
static void mdlSetDefaultPortDataTypes(SimStruct *S)
{
ssSetInputPortDataType(S, 0, SS_DOUBLE);
ssSetOutputPortDataType(S, 0, SS_DOUBLE);
}

#define MDL_START  /* Change to #undef to remove function */
#if defined(MDL_START)
/* Function: mdlStart =======================================================
Abstract:
This function is called once at start of model execution. If you
have states that should be initialized once, this is the place
to do it.
*/
static void mdlStart(SimStruct *S)
{
}
#endif /*  MDL_START */
```

```
/* Function: mdlOutputs =======================================================
*
*/
static void mdlOutputs(SimStruct *S, int_T tid)
{
const real_T *Pose = (real_T *) ssGetInputPortRealSignal(S, 0);
const real_T *Init = (real_T *) ssGetInputPortRealSignal(S, 1);
real_T *Config = (real_T *) ssGetOutputPortRealSignal(S, 0);
real_T *CriterionValue = (real_T *) ssGetOutputPortRealSignal(S, 1);

/* S-Function Builder Row Major Support has been enabled for custom
code, a transposed copy will be created for any array signals.
*/
real_T *Pose_t = (real_T *)ssGetDWork(S, 0);
real_T *Init_t = (real_T *)ssGetDWork(S, 1);
real_T *Config_t = (real_T *)ssGetDWork(S, 2);
real_T *CriterionValue_t = (real_T *)ssGetDWork(S, 3);

NDTransposeBySrcSpecs((void*)Pose_t, (const void*)Pose,
ssGetInputPortDimensions(S, 0), ssGetInputPortNumDimensions(S, 0),
sizeof(real_T));
NDTransposeBySrcSpecs((void*)Init_t, (const void*)Init,
ssGetInputPortDimensions(S, 1), ssGetInputPortNumDimensions(S, 1),
sizeof(real_T));
BFGS_IK_KUKA_IIWA_14R820_Outputs_wrapper(Pose_t, Init_t, Config_t,
CriterionValue_t);
NDTransposeByDstSpecs((void*)Config, (const void*)Config_t,
ssGetOutputPortDimensions(S, 0), ssGetOutputPortNumDimensions(S, 0),
sizeof(real_T));
NDTransposeByDstSpecs((void*)CriterionValue, (const void*)CriterionValue_t,
ssGetOutputPortDimensions(S, 1), ssGetOutputPortNumDimensions(S, 1),
sizeof(real_T));

}

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

3.4.2 BFGS_IK_KUKA_IIWA_14R820_wrapper.cpp

```
#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif
#include <math.h>
#include <vector>
#include "matrixlibr.h"
#include "matrixlibr.cpp"
#include "linesearch.h"
#include "linesearch.cpp"
#include "bfgsalgorithm.h"
#include "bfgsalgorithm.cpp"
#define u_width 4
#define y_width 1

void BFGS_IK_KUKA_IIWA_14R820_Outputs_wrapper(const real_T *Pose, const real_T *Init, real_T *Config, real_T *CriterionValue)
{
using namespace std;
struct Variable start_point, desired_pose_htm;
struct Variable A, joint_variables_vector;
desired_pose_htm = initialize_matrix(4, 4);
start_point = initialize_vector_column(7);
joint_variables_vector = initialize_vector_column(7);
A = initialize_matrix(7, 4);

 /*  ===================================================================   *
  *  DH parametri robota KUKA iiwa 14 R820.                                *
  *  ===================================================================   */
double a1=0;
double a2=0;
double a3=0;
double a4=0;
double a5=0;
double a6=0;
double a7=0;
double alpaha1=pi/2.0;
double alpaha2=-pi/2.0;
double alpaha3=-pi/2.0;
double alpaha4=pi/2.0;
double alpaha5=pi/2.0;
double alpaha6=-pi/2.0;
double alpaha7=0.0;
double d1=0.360;
double d2=0;
double d3=0.420;
double d4=0;
double d5=0.400;
double d6=0;
double d7=0.126;


 /*  ===================================================================   *
```

```
  *  Matrica DH parametara.                                                *
  *  ====================================================================  */
A.X[0][0]=a1;  A.X[0][1]=alpaha1;          A.X[0][2]=d1;
A.X[1][0]=a2;  A.X[1][1]=alpaha2;          A.X[1][2]=d2;
A.X[2][0]=a3;  A.X[2][1]=alpaha3;          A.X[2][2]=d3;
A.X[3][0]=a4;  A.X[3][1]=alpaha4;          A.X[3][2]=d4;
A.X[4][0]=a5;  A.X[4][1]=alpaha5;          A.X[4][2]=d5;
A.X[5][0]=a6;  A.X[5][1]=alpaha6;          A.X[5][2]=d6;
A.X[6][0]=a7;  A.X[6][1]=alpaha7;          A.X[6][2]=d7;
int i, j;
int k=0;

 /* ====================================================================  *
  *  desired_pose_htm - matrica homogene transformacije koja odredjuje     *
  *  zeljenu poziciju i orijentaciju robota i ona je jednaka ulazu u blok  *
  *  inverzne kinematike (Pose).                                           *
  *  ====================================================================  */
for (i=0; i<4; i++){
    for (j=0; j<4; j++){
        desired_pose_htm.X[i][j]=Pose[i+j+k];
    }
k+=3;
};

 /* ====================================================================  *
  *  start_point - ulaz u blok S-funkcije. To je pocetna aproksimacija u   *
  *  odredjivanju vektora zglobnih varijabli i jednaka je vektoru zglobnih *
  *  varijbli iz prethodne iteracije.                                      *
  *  ====================================================================  */
for (i=0; i<7; i++){
    start_point.X[i][0]=Init[i]+0.0001;
};

 /* <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<   *
  *  joint_variables_vector - izlaz iz bloka S-funkcije.                   *
  *  Racunanje  vektora  zglobnih  varijabli  'joint_variables_vector'za   *
  *  odabranu funkciju kriterija datu sa (2.13), za robotski manipulator   *
  *  cija je kinematika oderedjena matricom DH parametara 'A', za zeljenu  *
  *  poziciju  i  orijentaciju  koje  su  odredjene  matricom  homogene    *
  *  transformacije   'desired_pose_htm'   i   pocetnu   aproksimaciju     *
  *  'start_point' koja je jednaka vektoru zglobnih varijabli iz prethodne *
  *  iteracije.                                                            *
  *   <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<  */
joint_variables_vector = BFGS(criterion, A, desired_pose_htm, start_point);
for (i=0; i<7; i++){
    Config[i]=joint_variables_vector.X[i][0];
};

 /* ====================================================================  *
  *   CriterionValue - izlaz iz bloka S-funkcije.                          *
  *   Vrijednost funkcije kriterija za robot sa matricom DH parametara A,  *
  *  za  zeljenu  poziciju  i  orijentaciju  odredjenu  matricom  homogene *
  *  transformacije  desired_pose_htm,  i  za  BFGS  algoritmom  izracunati *
  *  vektor      zglobnih      varijabli      joint_variables      vector. */
     ====================================================================
CriterionValue[0]=criterion(A, desired_pose_htm, joint_variables_vector);
}
```

Dodatak A: Biblioteka `matrixlibr`

```
#include <math.h>
#include "matrixlibr.h"
#define pi 3.141592653
const double h=10e-5;

using namespace std;

/* Zbog mogucnosti proslijedjivanja cijelog vektora/matrice funkciji deklarisana je opsta struktura 'Variable'. */
struct Variable{
       vector< vector<double> >X;
};
/* s - skalarni dio kvaterniona, v - vektorski dio kvaterniona. */
struct Quaternion {
        double s;
        double v[3];
};
/* Inicijalizacija matrice dimenzija (m x n). */
struct Variable initialize_matrix (int m, int n){
struct Variable matrix;
for (int i=0; i<m; i++){
     vector<double> row;
     for (int j=0; j<n; j++){
         row.push_back(0.0);
     }
     matrix.X.push_back(row);
}
return matrix;
}
/* Inicijalizacija vektor-kolone dimenzija (n x 1). */
struct Variable initialize_vector_column (int n){
struct Variable matrix;
int i;
for (i=0; i<n; i++){
     vector<double> column;
     column.push_back(0.0);
     matrix.X.push_back(column);
}
return matrix;
}
/* Inicijalizacija vektora dimenzija (1 x n). */
struct Variable initialize_vector_row (int n){
struct Variable matrix;
int i;
vector<double> row;
for (i=0; i<n; i++){
    row.push_back(0.0);
}
matrix.X.push_back(row);
return matrix;
}
/* Transponovanje matrice ili vektora. */
struct Variable transpose (struct Variable A){
struct Variable B;
int i, j;
```

```
if (A.X.size()>1 && A.X[0].size()==1){
B = initialize_vector_row(A.X.size());
for (i=0; i<A.X.size(); i++){
     B.X[0][i] = A.X[i][0];
}
} else if (A.X.size()>1 && A.X[0].size()>1){
B = initialize_matrix(A.X.size(), A.X[0].size());
for (i=0; i<A.X.size(); i++){
     for (j=0; j<A.X[0].size(); j++){
        B.X[j][i] = A.X[i][j];
     }
}
} else if (A.X.size()==1 && A.X[0].size()>1){
B = initialize_vector_column(A.X[0].size());
for (i=0; i<A.X[0].size(); i++){
     B.X[i][0] = B.X[0][i];
}
}  else return A;
return B;
}
/* Suma matrica A i B. Zbog prirode ostatka koda nije ispitivana jednakost dimenzija A i B. */
struct Variable sum (struct Variable A, struct Variable B){
struct Variable C = initialize_matrix(A.X.size(), A.X[0].size());
for(int i=0; i<A.X.size(); i++){
    for(int j=0; j<A.X[0].size(); j++){
        C.X[i][j] = A.X[i][j] + B.X[i][j];
    }
}
return C;
}
/* Razlika matrica A i B. */
struct Variable razlika (struct Variable A, struct Variable B){
struct Variable C = initialize_matrix(A.X.size(), A.X[0].size());
for(int i=0; i<A.X.size(); i++){
    for(int j=0; j<A.X[0].size(); j++){
        C.X[i][j] = A.X[i][j] - B.X[i][j];
    }
}
return C;
}
/* Mnozenje vektora/matrice A skalarom. */
struct Variable c (struct Variable A, double scalar){
struct Variable B=initialize_matrix(A.X.size(), A.X[0].size());
for (int i=0; i<A.X.size(); i++){
     for (int j=0; j<A.X[0].size(); j++){
          B.X[i][j] = scalar * A.X[i][j];
     }
}
return B;
}
/* Jedinicna matrica. */
struct Variable eye (int order){
struct Variable A=initialize_matrix(order, order);
for (int i=0; i<order; i++){
     for (int j=0; j<order; j++){
          if(i==j){ A.X[i][j] = 1.0; }
```

```
          else    { A.X[i][j] = 0.0; }
     }
}
return A;
}
/* Proizvod matrica A i B. */
struct Variable product (struct Variable A, struct Variable B){
int i,j,k;
int m = A.X.size();
int n = B.X.size();
int p = B.X[0].size();
struct Variable C = initialize_matrix(m, p);
double s;
for (i=0; i<m; i++){
    for (k=0; k<p; k++){
         s=0;
         for (j=0; j<n; j++){
             s += (A.X[i][j]) * (B.X[j][k]);
         }
         C.X[i][k] = s;
    }
}
return C;
}
/* Submatrica matrice A koja se dobije izostavljanjem p-te vrste i q-te kolone od A. */
struct Variable submatrix ( struct Variable A, int p, int q){
int i=0, j=0;
int n=A.X.size();
struct Variable matrix = initialize_matrix(n-1, n-1);;
for(int row=0; row<n; row++){
    for(int col=0; col<n; col++){
       if(row !=p && col !=q){
       matrix.X[i][j++] = A.X[row][col];
       if(j==n-1) {j=0; i++;}
       }
    }
}
return matrix;
}
/* Rekurzivna fukcija koja racuna determinantu matrice A. */
double determinant (struct Variable A, int n){
double D=0;
if(n==1)
return A.X[0][0];
int sign=1;
for(int f=0;f<n;f++) {
    D += sign * A.X[0][f] * determinant( submatrix(A,0,f), n-1);
    sign = -sign;
}
return D;
}
/* Racunanje adjungirane  matrice od A. */
struct Variable adjung (struct Variable A){
int i, j;
int n = A.X.size();
struct Variable B = initialize_matrix(n, n);
```

```
for(i=0;i<n;i++){
    for(j=0;j<n;j++){
        B.X[j][i] = pow(-1,i+j) * determinant( submatrix(A, i, j), n-1);
    }
}
return B;
}
/* Racunanje inverzne matrice od A. */
struct Variable inv (struct Variable A){
int n=A.X.size();
struct Variable B;
struct Variable C = initialize_matrix(n, n);
B=adjung(A);
if(determinant(A, n)!=0){
for (int i=0; i<n; i++){
     for (int j=0; j<n; j++){
          C.X[i][j] = B.X[i][j] / determinant(A, n);
     }
}
return C;
}
}
/* Euklidska norma vektora 'vec'. */
double vector_norm (struct Variable vec){
double norm=0.0;
for (int i=0; i<vec.X.size(); i++)
     norm += vec.X[i][0] * vec.X[i][0];
return sqrt(norm);
}
```

Dodatak B: Algoritam inverzne kinematike koji tretira i ograničenja

Neka je dat n-segmentni robotski manipulator koji predstavlja otvoreni kinematički lanac, i neka je svakom segmentu manipulatora pridružen odgovarajući koordinatni sistem u skladu sa DH konvencijom za koje su dati odgovarajući DH parametri. U naslovu 2.2 smo problem inverzne kinematike robota postavili kao problem optimizacije sa ograničenjima (2.3). Ovaj algoritam je konbinacija BFGS metoda i metoda projekcije gradijenta, i pronalazi stacionarnu tačku, tj. tačku u kojoj je zadovoljen potreban uslov lokalnog minimuma funkcije kriterija (2.13).

Ako lijeve nejednakosti u sistemu nejednakosti (2.3b) napišemo u formi $-q_i \le -q_{i\ min}$, tada ovaj sistem nejednakosti možemo napisati u sljedećoj formi:

$$\boldsymbol{a}_i^T \boldsymbol{q} \le \boldsymbol{b}_i , \qquad i = 1,\ 2,\ \dots,\ 2n \tag{b-1}$$

Prema tome skup dozvoljenih vrijednosti je određen skupom:

$$\Omega = \left\{ \boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a}_j^T \boldsymbol{x} \le \boldsymbol{b}_j;\ j = 1,\ \dots, m;\ m \le n \right\} \tag{b-2}$$

gdje su $\boldsymbol{a}_i$ vektor-kolone. Ovdje u našem slučaju obzirom na sistem nejednakosti u (2.3b) to su jedinični vektor-kolone koji su linearno nezavisni. Za vektor zglobnih varijabli $\boldsymbol{q}$ kažemo da je *izvodiv* ako zadovoljava sva postavljena ograničenja (2.88). Za i-to ograničenje kažemo da je *aktivno ograničenje* za vektor $\boldsymbol{q}$, ako vrijedi $\boldsymbol{a}_i^T \boldsymbol{q} = \boldsymbol{b}_i$. Definišimo matricu dimenzija $(n \times p)$ sastavljenu od p vektora $\boldsymbol{a}_i$ koji odgovaraju aktivnim ograničenjima:

$$\boldsymbol{A}_p = \left[\boldsymbol{a}_{i_1} \ \boldsymbol{a}_{i_2} \ \cdots \ \boldsymbol{a}_{i_p} \right] \tag{b-3}$$

gdje je indeks i asociran sa i-tom iteracijom.

Korak 1. Neka je $\boldsymbol{q}^0$ vektor početnih zglobnih varijabli, $\boldsymbol{H}_0^0$ je početna $(n \times n)$ pozitivno definitna simetrična matrica. Pretpostavimo da je p ograničenja aktivno u $\boldsymbol{q}^0$. $\mathbf{A}_p$ je sastavljena od p vektora $\boldsymbol{a}_i$, za $\boldsymbol{a}_i$ u ograničenjima koja su aktivna u $\boldsymbol{q}^0$. $\boldsymbol{H}_p^0$ dobijemo koristeći (b-10).

Korak 2. Za dato $\boldsymbol{q}^i$, $\boldsymbol{g}_i = \nabla f_i = \nabla f\left(\boldsymbol{q}^i\right)$ i $\boldsymbol{H}_\mathbf{p}^\mathbf{i}$ računamo $\boldsymbol{H}_p^i \boldsymbol{g}_i$

$$\upsilon = \left(\boldsymbol{A}_p^T \boldsymbol{A}_p\right)^{-1} \boldsymbol{A}_p^T \boldsymbol{g}_i \tag{b-4}$$

Ako je $\boldsymbol{H}_p^i \boldsymbol{g}_i = 0$ i $\upsilon_j \leq 0$, $j = 1,\ 2,\ ...,\ p$, zaustavi algoritam. $\boldsymbol{q}^i$ je stacionarna tačka.

Korak 3. Ako algoritam nije zaustavljen u koraku 2. ili ako je $\left\|\boldsymbol{H}_p^i \boldsymbol{g}_i\right\| > \max\left\{0,\ \frac{1}{2}\upsilon_p a_{pp}^{-1/2}\right\}$ ili $\left\|\boldsymbol{H}_p^i \boldsymbol{g}_i\right\| \leq \frac{1}{2}\upsilon_p a_{pp}^{-1/2}$ gdje je a_{pp} p-ti dijagonalni element matrice $\left(\boldsymbol{A}_p^T \boldsymbol{A}_p\right)^{-1}$ -tada preći na korak 4.

U protivnom slučaju izbaci p -to ograničenje iz $\boldsymbol{A}_p$ i izračunaj $\boldsymbol{H}_{p-1}^i$ iz:

$$\boldsymbol{H}_{p-1}^i = \boldsymbol{H}_p^i + \frac{\boldsymbol{P}_{p-1} \boldsymbol{a}_{i_p} \boldsymbol{a}_{i_p}^T \boldsymbol{P}_{p-1}}{\boldsymbol{a}_{i_p}^T \boldsymbol{P}_{p-1} \boldsymbol{a}_{i_p}} \tag{b-5}$$

gdje je : $\boldsymbol{P}_{p-1} = \boldsymbol{I} - \boldsymbol{A}_{p-1}\left(\boldsymbol{A}_{p-1}^T \boldsymbol{A}_{p-1}\right)^{-1} \boldsymbol{A}_{p-1}^T$ matrica projekcije, $\boldsymbol{a}_{i_p}$ je p-ta kolona matrice $\boldsymbol{A}_p$ i $\boldsymbol{A}_{p-1}$ je $\left(n \times (p-1)\right)$ matrica dobijena uklanjanjem p-te kolone matrice $\boldsymbol{A}_p$. Evaluirati $p \leftarrow p-1$ i preći na korak 2.

Korak 4. Uzeti za pravac pretraživanja $\boldsymbol{p}_i = -\boldsymbol{H}_p^i \boldsymbol{g}_i$ i izračunati:

$$\lambda_j = \frac{\boldsymbol{b}_j - \boldsymbol{a}_j^T \boldsymbol{q}^j}{\boldsymbol{a}_j^T \boldsymbol{p}_i}\ , \qquad j = p+1,\ p+2,\ ...,\ k \tag{b-6}$$

$$\lambda_i = \min_j \left\{\lambda_j > 0\right\} \tag{b-7}$$

Tehnikom jednodimenzionalnog pretraživanja izračunati najviše moguće α_i tako da vrijedi:

$$0 < \alpha_i \leq \min\left\{1,\ \lambda_i\right\} \quad \text{i} \quad \begin{cases} f\left(\boldsymbol{q}^i + \alpha_i \boldsymbol{p}_i\right) \leq f\left(\boldsymbol{q}^i\right) + \delta_1 \alpha_i \boldsymbol{g}_i^T \boldsymbol{p}_i & \text{(b-8a)} \\ \boldsymbol{g}\left(\boldsymbol{q}^i + \alpha_i \boldsymbol{p}_i\right)^T \boldsymbol{p}_i \geq \delta_2 \boldsymbol{g}_i^T \boldsymbol{p}_i & \text{(b-8b)} \end{cases}$$

gdje je $0 < \delta_1 < \delta_2 < 1$ i $\delta_1 < 0.5$, kao u (2.25).
Računamo vektor zglobnih varijable za sljedeću iteraciju:

$$\boldsymbol{q}^{i+1} = \boldsymbol{q}^i + \alpha_i \boldsymbol{p}_i \tag{b-9}$$

Korak 5. Ako je $\alpha_i = \lambda_i$,matrici $\boldsymbol{A}_p$ dodati vektor $\boldsymbol{a}_j$ koji korespondira $\min\left\{\lambda_j\right\}$ u koraku 4., onda izračunati:

$$\boldsymbol{H}_{p+1}^{i+1} = \boldsymbol{H}_p^i - \frac{\boldsymbol{H}_p^i \boldsymbol{a}_j \boldsymbol{a}_j^T \boldsymbol{H}_p^i}{\boldsymbol{a}_j^T \boldsymbol{H}_p^i \boldsymbol{a}_j} \qquad \text{(b-10)}$$

Evaluirati $p \leftarrow p+1$ i $i \leftarrow i+1$ i preći u Korak 2.

Korak 6. Inače, staviti $\boldsymbol{s}_i = \boldsymbol{q}^{i+1} - \boldsymbol{q}^i = \alpha_i \boldsymbol{p}_i$, $\boldsymbol{y}_i = \boldsymbol{g}_{i+1}\text{-}\boldsymbol{g}_i$ i ažurirati matricu $\boldsymbol{H}_p^i$:

Ako je $\boldsymbol{s}_i^T \boldsymbol{y}_i \geq \boldsymbol{y}_i^T \boldsymbol{H}_p^i \boldsymbol{y}_i$ koristiti formulu (2.56).

inače za ažurirnje $\boldsymbol{H}_p^i$ koristiti formulu (2.66). U (2.56) i (2.66) formalno uvrstimo da je $\boldsymbol{H}_i = \boldsymbol{H}_i^p$ i $\boldsymbol{H}_{i+1} = \boldsymbol{H}_{i+1}^p$.

U Koraku 4. mogu se uzeti $\delta_1 = 0.0001$ i $\delta_2) = 0.5$.

Zaključak

U navedenoj simulaciji je uzet sample time T_S=0.1 s, a sample time S-funkcije je uzet *inherentan* ostatku simulacije, što je u izvornom kodu fajla BFGS_IK_KUKA_IIWA_14R820.cpp zadano u linji 72. U simulaciji elementi dijagonalnih matrica $\boldsymbol{M}_P$ i $\boldsymbol{M}_O$ iz (2.13) su jednaki 2, odnosno 0.02, respektivno, što je zadano u linijama 85. i 86. biblioteke bfgslibrary. Takođe, parametri kriterija zaustavljanja algoritma pretraživanja po pravcu, odnosno BFGS algoritma su zadani vrijednostima 0.0001 za oba kriterija, što je zadano u liniji 42. biblioteke linesearch i liniji 116. biblioteke bfgsalgorithm. Rezultati simulacije su predstavljeni za ove vrijednosti navedenih parametara. Iz vremenskih dijagrama greški pozicije i greške orijentcije koji su predstavljeni na slikama 1.9 i 1.10 vidimo da su vrijednosti greške orijentacije izraženije, što je bilo i za očekivati jer smo uzeli da su težinski kojeficijenti u matrici $\boldsymbol{M}_O$ za dva reda veličine manji od težinskih kojeficijenata matrice $\boldsymbol{M}_P$. Zbog toga se orijenacija računa sa većom tolerancijom u odnosu na računanje pozicije pa je vrijednost greške orijentacije veća. Na smanjenje greške pozicije i orijentacije moguće je uticati smanjenjem parametra kriterija zaustavljanjanja BFGS algoritma, tj. izborom manjeg ε u (2.59), što bi zahtijevalo veći broj iteracija algoritma. Ove matrične i parametre kriterija zaustavljanja uz određene dodatke moguće bi bilo mijenjati kao parametre bloka navedene S-funkcije, što ovdje nije urađeno zbog obima rada.

Na slici 1.11 vidimo da iako je vrijednost fukcije kriterija reda veličine 10^{-6}, nije postignuto idealno rješenje dato sa (2.14). Ovo se može pripisati kriteriju zaustavljanja algoritma (2.59), jer mjera stagnacije gradijenta nije ujedno i mjera stagnacije promjene vektora zglobnih varijabli u problemskom prostoru. Drugu pomenutu mjeru smo mogli uzeti za kriterij zaustavljanja algoritma , što bi opet zahtijevalo veći broj iteracija da bi bili bliži idealnom riješenju (2.14), i time smanjili pomenute greške.

Umjesto egzaktnog metoda jednodimenzionalnog pretraživanja datog sa (2.24), koji je primijenjen u ovom radu, može se primijeniti neegzaktni metod jednodimenzionalnog pretraživanja zasnovan na uslovima (2.25). Ovaj metod zasnovan na (2.25) približno određuje veličinu optimalnog koraka, što se pokazuje da je dovoljno. Ovim se može umanjiti ukupna numerička zahtjevnost algoritma.

Treba imati u vidu i to da algoritam implementiran u ovom radu pronalazi stacionarne tačke funkcije kriterija, i da bi imao punu funkcionalnost u realnim primjenama, uz tretiranje ograničenja potrebno je uključiti i algoritam koji u slučaju pronalaska prevojne (sedlo) tačke resetuje pretraživanje za tu iteraciju polazeći iz nove početne aproksimacije izabrane ovim algoritmom.

U slučaju kad se ne zahtijevaju veliki iznosi brzine i ubrzanja vrha manipulatora, tj. kada ne dolazi do izražaja dimanika mehaničke strukture robotskog manipulatora, ova struktura algoritma inverzne kinematike može biti konceptualno adaptirana za upravljanje manipulatorom. Ova struktura je poznata pod imenom *kinematičko upravljanje*.

Literatura

[1] J. Nocedal, S. J. Wright. Numerical Optimization. Springer, 2006.

[2] B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo. Robotics: Modeling, Planning and Control. Advanced Textbooks in Control and Signal Processing. Springer, 2009.

[3] The Matworks. Matlab Simulink: Writing S-Functions. The Mathworks, Inc. 2007.

[4] J. Zhao, N. I. Badler. "Inverse Kinematics Positioning Using Nonlinear Programming for Highly Articulated Figures". ACM Transactions on Graphics Vol. 13, No. 4 (1994).

[5] T. Sugihara. "Solvability-Uncocerned Inverse Kinematics by the Levemberg-Marquardt Method". IEEE Transactions on Robotics Vol. 27 , No. 5 (2011).

[6] C. Juan ."Closed-Loop Manipulator Control Using Quaternion Feedback". IEEE Journal of Robotics and Automation Vol. 4, No. 4. (1988).

Sadržaj