Određivanje parametara putanje robotskog manipulatora metaheurističkim algoritmom

Grenko, Teodor

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij elektrotehnike

Diplomski rad

ODREĐIVANJE PARAMETARA PUTANJE ROBOTSKOG MANIPULATORA METAHEURISTIČKIM ALGORITMOM

Rijeka, rujan 2022.

Teodor Grenko

0069079266

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij elektrotehnike

Diplomski rad

ODREĐIVANJE PARAMETARA PUTANJE ROBOTSKOG MANIPULATORA METAHEURISTIČKIM ALGORITMOM

Mentor: prof. dr. sc. Zlatan Car

Rijeka, rujan 2022.

Teodor Grenko

0069079266

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA DIPLOMSKE ISPITE

Rijeka, 21. ožujka 2022.

Zavod: **Zavod za automatiku i elektroniku**
Predmet: **Osnove robotike**
Grana: **2.03.06 automatizacija i robotika**

ZADATAK ZA DIPLOMSKI RAD

Pristupnik: **Teodor Grenko (0069079266)**
Studij: Diplomski sveučilišni studij elektrotehnike
Modul: Automatika

Zadatak: **Određivanje parametara putanje robotskog manipulatora metaheurističkim algoritmom/Determining the trajectory characteristics of a robotic manipulator using metaheuristic algorithms**

Opis zadatka:

Izraditi pregled primjene umjetne inteligencije u planiranju putanje robotskih manipulatora. Dati opis planiranja putanje robotskog manipulatora. Izraditi metaheuristički algoritam za izračun parametara putanje. Usporediti performanse sa nasumičnim odabirom parametara. Komentirati primjenu algoritma.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 21. ožujka 2022.

Mentor:

Prof. dr. sc. Zlatan Car

Predsjednik povjerenstva za diplomski ispit:

Prof. dr. sc. Viktor Sučić

IZJAVA O SAMOSTALNOJ IZRADI RADA

Sukladno članku 9. Pravilnika o diplomskom radu, diplomskom ispitu i završetku diplomskih sveučilišnih studija Tehničkog fakulteta Sveučilišta u Rijeci izjavljujem da sam izradio ovaj diplomski rad samostalno, koristeći vlastito znanje i navedenu literaturu, u razdoblju od datuma zadavanja zadatka do datuma predaje.

Rijeka, rujan 2022.

Teodor Grenko

ZAHVALA

Zahvaljujem prof. dr. sc. Zlatanu Caru na prihvaćanju mentorstva za izradu rada i asist. Sandiju Baressi Šegoti, mag. ing. comp. na uputama i pomoći tijekom izrade.

SADRŽAJ

1. UVOD

Ovim diplomskim radom izradit će se pregled primjene umjetne inteligencije u planiranju putanje robotskog manipulatora, te dati opis putanje robotskog manipulatora. Za izračun parametara putanje potrebno je izraditi i primijeniti neki od metaheurističkih algoritma. Točnije, koristit će se genetski algoritam kako bi se optimiziralo gibanje robotskog manipulatora u svrhu smanjenja nepoželjnih trzaja robota.

Za ocjenjivanje kvalitete metaheurističkog algoritma, provest će se i metoda planiranja putanje koristeći tradicionalne metode. Tradicionalne metode određivanja putanje robotskog manipulatora ne uključuju primjenu umjetne inteligencije, već se temelje na nasumičnom generiranju putanja. Ako nasumično generirana putanja ne daje zadovoljavajuće rezultate, ista se odbacuje i nasumično se generira nova putanja sve dokle se ne dođe do željenog rezultata.

Problem je moguće riješiti i primjenom Ho-Cookovog algoritma, ali ovaj način može postati kompliciran za robotske manipulatore s više osi. Odnosno, više osi robotskog manipulatora uzrokuju veće i kompliciranje jednadžbe s mnogo nepoznanica. Ovaj problem će se više razraditi u nastavku rada.

Na samom kraju bit će odrađena simulacija. Dobivene putanje će se spremiti u Excel tablicu koju se zatim učitava u ABB-ovom RobotStudio programskom paketu te se primjenjuju na robotski manipulator ABB IRB 120.

2. ROBOTIKA

Robotika je interdisciplinarna grana informatike i inženjerstva koja podrazumijeva dizajn, konstrukciju, operacije i uporabu robota. Cilj robotike je izraditi strojeve koji mogu pomoći ljudima. Robotika je široka znanost koja uključuje područja strojarstva, elektrotehnike, mehatronike, informatike, matematike i ostalih [1].

Cilj je stvoriti strojeve koji mogu zamijeniti ljude u određenim poslovima te replicirati njihove radnje. Iako je primjena robota široka, danas se najčešće koristi za obavljanje opasnih poslova, npr. za analiziranje radioaktivnih materijala, za detekciju i uklanjanje eksplozivnih naprava ili u okolinama gdje čovjek ne bi mogao preživjeti, npr. u svemiru, u vodi ili na mjestima visokih temperatura [1].

Današnji roboti su izrađeni u mnogim oblicima za svakakve primjene. Usprkos njihovoj raznolikosti, postoje tri osnovna svojstva koje svi roboti dijele. Prvo svojstvo je kako svi roboti imaju nekakav tip mehaničke konstrukcije, oblika ili forme koji su dizajnirani za izvršavanje određenog zadatka, poput gusjenica za vožnju po teško prohodnim terenom [1].

Drugo svojstvo je posjedovanje električnih komponenti koje napajaju i kontroliraju robota. Najjednostavniji primjer je pogon robota. Pogon zahtjeva električni motor, bateriju i vodič koji će provoditi električnu energiju iz baterije u motore. Ovo predstavlja osnovni strujni krug no postoje i puno kompleksniji električni uređaji [1].

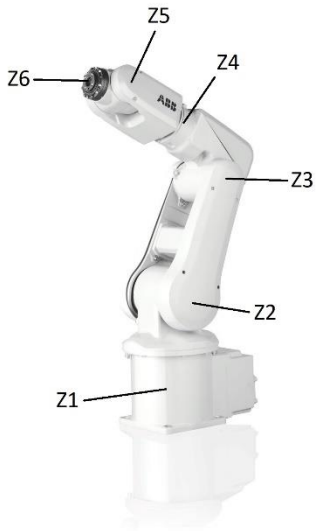
Zadnje svojstvo je postojanje programskog koda. Programski kod određuje kako će se robot ponašati ili kako će odlučiti što i kada će nešto učiniti. Ako se ponovo osvrnemo na primjer pogona gusjenica robota, programski kod će robotu reći koliko električne energije je potrebno primijeniti na pogon za uspješno savladavanje prepreke. U svojoj srži, programski kod je najbitnije svojstvo robota. Svi ostali aspekti robota mogu biti savršeni, ali ako je robot loše programiran njegovo djelovanje će biti drastično smanjeno [1].

Postoje tri vrste programa za robote, a to su: robot s daljinskim upravljanjem, robot s umjetnom inteligencijom i hibridni roboti. Roboti koji primjenjuju umjetnu inteligenciju su sposobni sami donositi upravljačke za optimalno izvršavanje zadatka [1].

2.1 Stručni pojmovi u robotici

Robot je definiran kao elektromehanička naprava koja ima više stupnjeva slobode te služi za izvršavanje zadanih radnih zadataka. Na primjer, industrijski robot je definiran kao više-funkcijski manipulator namijenjen za prijenos materijala, dijelova, alata ili posebnih naprava u svrhu izvođenja različitih zadaća [2].

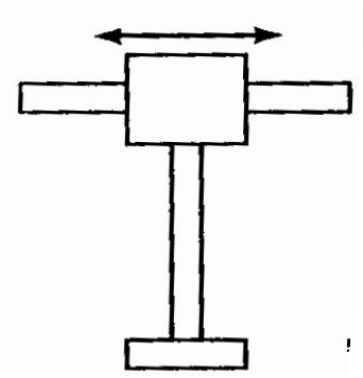
Nadalje, bitan pojam u robotici je stupanj slobode. Stupanj slobode označava broj zglobova robota koji imaju sposobnost gibanja. Kako bi se neki predmet u prostoru u potpunosti opisao, potrebno je najmanje šest stupnja slobode, a to su: Kartezijanske koordinate (x,y,z), orijentacija, kutni položaj i smjer predmeta. U robotici se iz tog razloga često spominju roboti sa šest stupnja slobode. Primjer takvog robota je prikazan na slici 2.1 [3].



Slika 2.1 Robot sa šest stupnja slobode
Izvor: ABB Robotics [4]

Robotski manipulator je reprogramabilna i multifunkcionalna mehanička naprava koja služi za izvršavanje raznih zadataka. Manipulator se sastoji od raznih zglobova koji skupa u seriji nalikuju ruci. Struktura manipulatora definiraju dohvat krajnjeg djelovatelja robota i njegov radni prostor [5].

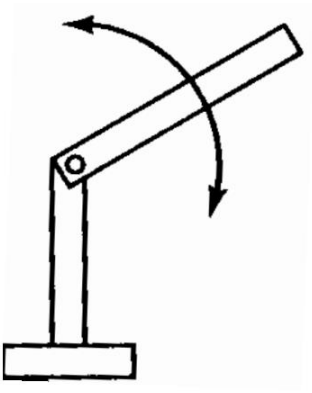
Zglobovi robota koji zajedno čine robotski manipulator se mogu podijeliti u dvije kategorije: prizmatične i revolutne. Prizmatični zglobovi (L), slika 2.2, dobivaju svoj naziv jer je njihov presjek sačinjen od osnovnih geometrijskih prizmi. Ovi zglobovi dozvoljavaju pravocrtno kretanje [1].



Slika 2.2 Prizmatičan zglob
Izvor: Uvod u robotiku [2]

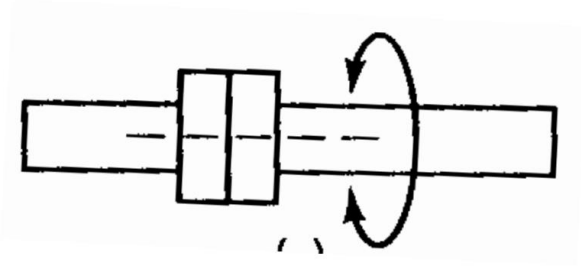
Revolutni zglobovi dozvoljavaju samo kutno okretanje između spona te se dijele na obrtajne (R), zavrtne (T) i okretne (V) zglobove te su njihove osnovne razlike prikazane slikama 2.3, 2.4 i 2.5.

Obrtajni zglob, slika 2.3, djeluje obrtom oko osi okomite susjedne spone tako se mijenja kut između dviju susjednih spona [2].



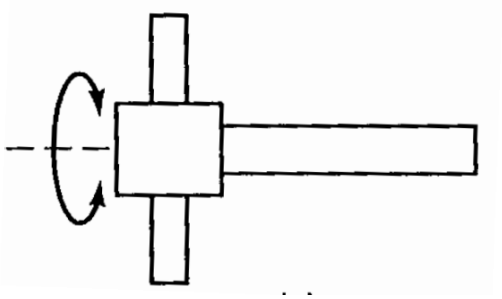
Slika 2.3 Revolutni (Obrtajni - R) zglob
Izvor: Uvod u robotiku [2]

Zavrtni zglob, slika 2.4, djeluje slično kao obrtajni zglob, no razlika je u tome što se okretanje odvija usporedno s dvije susjedne spone te su njihove osi uvijek paralelne [2].



Slika 2.4 Revolutni (Zavrtni - T) zglob
Izvor: Uvod u robotiku [2]

Okretanje kod okretnog zgloba, slika 2.5, se odvija oko osi usporedne sa susjednim sponama. Spone su uglavnom okomite i okretanje uključuje okret jedne spone u odnosu na drugu [2].



Slika 2.5 Revolutni (Okretni - V) zglob
Izvor: Uvod u robotiku [2]

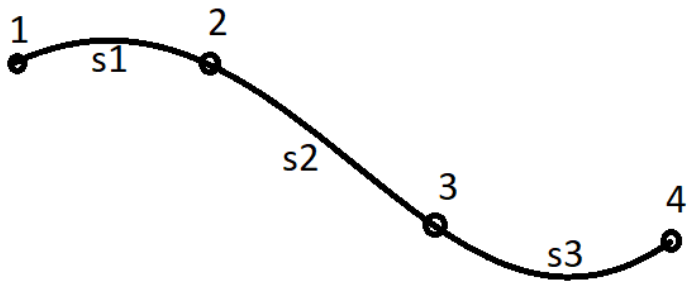
U analizi kretanja robota se susrećemo s dva problema: određivanje koordinata krajnje hvataljke za zadani skup koordinata zgloba i određivanje koordinata zgloba za dani položaj krajnje hvataljke. Pozicija krajnje hvataljke može biti određena pomoću dva sustava, a to su zglobovski prostor i prostor alata [2].

Kod zglobovskog prostora parametri zgloba (kut okretanja ili kut zavrtanja) i promjenjive dužine spona služe za predstavljanje položaja krajnje hvataljke. S druge strane, u prostoru alata se zajedničke koordinate i osnovni kartezijski sustav koriste za određivanje krajnje hvataljke robota. Pretvorba iz zglobovskog prostora u prostor alata se naziva kinematska pretvorba unaprijed, dokle se pretvorba iz prostora alata u zglobovski prostor naziva kinematska pretvorba unazad [2].

2.2 Planiranje putanje Ho-Cookovom metodom

Planiranje putanje označava niz točaka kroz koje vrh robota mora proći. Ideja je pronaći funkciju koja će opisati krivulju u prostoru konfiguracije alata ako je zadan niz točaka. Postoje dva načina za planiranje putanje: od točke do točke i kontinuirano po putanji. Ho-Cookova metoda spada u metode za izračun gibanja robota kontinuirano po putanji [6].

Kod Ho-Cookove metode prvo je potrebno odrediti točke kroz koje želimo da robot prođe tijekom gibanja. Skup tih točaka čine trajektoriju koja se sastoji od n točaka i n-1 segmenata. Obično su definirane prolazne točke te ograničenje brzine i ubrzanje robota. Putanja robota se dobije interpolacijom između zadanih točaka, gdje su prvi i zadnji segmenti polinomi 4. stupnja, a svi ostali segmenti su polinomi 3. stupnja. Ideja je dobiti glatku putanju te se zato odabiru veći stupnjevi polinoma [6].



Slika 2.6. Prikaz Ho-Cookove metode

Interpolacijske funkcije u zglobovskom prostoru su sljedećeg oblika.

Putanje po segmentima krivulje se definiraju kao:

$$q_k(t) = B_{0k} + B_{1k}t + B_{2k}t^2 + B_{3k}t^3, \qquad 2 \le k \le m-2, 0 \le t \le t_{k+1}, \tag{2.1}$$

$$q_k(t) = B_{0k} + B_{1k}t + B_{2k}t^2 + B_{3k}t^3 + B_{4k}t^4, \qquad k = 1, m-1, 0 \le t \le t_{k+1}. \tag{2.2}$$

Brzine po segmentima krivulje se definiraju kao:

$$\dot{q}_k(t) = B_{1k} + 2B_{2k}t + 3B_{3k}t^2, \qquad 2 \le k \le m-2, 0 \le t \le t_{k+1}, \tag{2.3}$$

$$\dot{q}_k(t) = B_{1k} + 2B_{2k}t + 3B_{3k}t^2 + 4B_{4k}t^3, \qquad k = 1, m-1, 0 \le t \le t_{k+1}. \tag{2.4}$$

Ubrzanja po segmentima krivulje se definiraju kao:

$$\ddot{q}_k(t) = 2B_{2k} + 6B_{1k}t, \qquad 2 \le k \le m-2, 0 \le t \le t_{k+1}, \tag{2.5}$$

$$\ddot{q}_k(t) = 2B_{2k} + 6B_{1k}t + 12B_{4k}t^2, \qquad k = 1, m-1, 0 \le t \le t_{k+1}. \tag{2.6}$$

Gdje je:

B – koeficijenti interpolacijskih polinoma,

t – vrijeme,

k – određena točka putanje i

m – ukupni broj točaka putanje.

Izrazi 2.2, 2.4 i 2.6 se odnose za prvi i zadnji segment te je vidljivo kako su jednog reda veći od izraza 2.1, 2.3 i 2.5 koji predstavljaju ostale segmente.

Matrice koeficijenta interpolacijskih polinoma za prvi i zadnji segment glase:

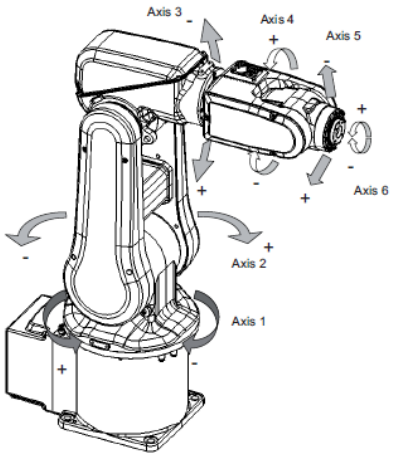
$$B = \begin{bmatrix} B_{00} & B_{01} & B_{02} & B_{03} & B_{04} \\ B_{10} & B_{11} & B_{12} & B_{13} & B_{14} \\ B_{20} & B_{21} & B_{22} & B_{23} & B_{24} \\ B_{30} & B_{31} & B_{32} & B_{33} & B_{34} \\ B_{40} & B_{41} & B_{42} & B_{43} & B_{44} \\ B_{50} & B_{51} & B_{52} & B_{53} & B_{54} \end{bmatrix}. \tag{2.7}$$

Matrice koeficijenta interpolacijskih polinoma za sve segmente između glase:

$$B = \begin{bmatrix} B_{00} & B_{01} & B_{02} & B_{03} \\ B_{10} & B_{11} & B_{12} & B_{13} \\ B_{20} & B_{21} & B_{22} & B_{23} \\ B_{30} & B_{31} & B_{32} & B_{33} \\ B_{40} & B_{41} & B_{42} & B_{43} \\ B_{50} & B_{51} & B_{52} & B_{53} \end{bmatrix}. \tag{2.8}$$

Broj redaka i stupaca koeficijenata interpolacijskih polinoma, izrazi 2.7 i 2.8, ovise o konstrukciji robotskog manipulatora i o redu polinoma. Kako je prethodno spomenuto, prvi i zadnji segment su polinomi 4. reda, tako će i prva i zadnja matrica B imati 5 stupaca (t^0, t^1, t^2, t^3, t^4). Svi ostali segmenti će imati 4 stupaca koji odgovaraju polinomu 3. reda (t^0, t^1, t^2, t^3).

Nadalje, broj redaka ovisi o broju zglobova robotskog manipulatora. Robotski manipulator s kojim se radi u ovom diplomskom radu je ABB IRB 120, slika 2.7, koji se sastoji od 6 zglobova od čega su 3 obrtajni zglobovi (R) i 3 zavrtni zglobovi (T) postavljeni u konfiguraciji TRRTRT.



Slika 2.7. ABB IRB 120, shematski prikaz, izometrijski pogled
Izvor: ABB Robotics [4]

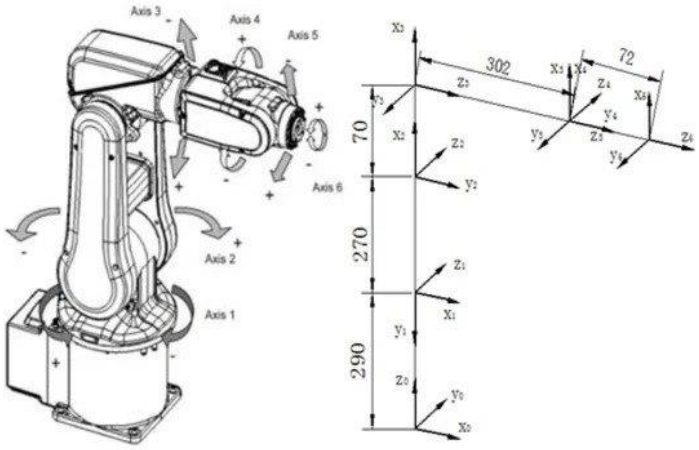
Za određivanje matrica koeficijenta interpolacijskih polinoma moguće je pratiti Ho-Cookov algoritam. Ali kako se povećava broj zglobova, tako izračuni postaju sve složeniji. Za robotski manipulator koji ima tri zgloba je moguće pratiti Ho-Cookov algoritam bez računalnog računanja, no ako problem primijenimo na ABB IRB 120 koji ima 6 zglobova, izračun se zakomplicira.

Kinematska pretvorba unaprijed se koristi za dobivanje trajektorije u operacijskom prostoru. Denavit-Hartenbergovom metodom moguće je izvršiti kinematsku pretvorbu unaprijed. U strojarstvu, Denavit-Hartenbergovi parametri (također zvani DH parametri) četiri su parametra povezana s određenom konvencijom za pričvršćivanje referentnih okvira na karike prostornog kinematičkog lanca ili robotskog manipulatora [5].

Četiri parametara koji se dobivaju Denavit-Hartenbergovom metodom su [5]:

- d – udaljenost zgloba,
- Θ – kut zgloba,
- a – duljina članka i
- α – zakret članka.

Slika 2.8 prikazuje rješenje kinematske pretvorbe unaprijed s pridruženim pripadajućim ortonormiranim koordinatnim sustavima i dimenzijama.



Slika 2.8: ABB IRB 120 s pridruženim ortonormiranim K.S.
Izvor: Research of Calibration Method for Industrial Robot Based on Error Model of Position [6]

Denavit-Hartenbergovom metodom dolazimo do slijedećih rezultata:

Tablica 1: Rezultati Denavit-Hartenbergove metode

Θ [rad]	d [mm]	a [mm]	α [rad]
$\Theta_1 = q_1$	$d_1 = 290$	$a_1 = 0$	$\alpha_1 = -\pi/2$
$\Theta_2 = q_2$	$d_2 = 0$	$a_2 = 270$	$\alpha_2 = 0$
$\Theta_3 = q_3$	$d_3 = 0$	$a_3 = 70$	$\alpha_3 = -\pi/2$
$\Theta_4 = q_4$	$d_4 = 302$	$a_4 = 0$	$\alpha_4 = \pi/2$
$\Theta_5 = q_5$	$d_5 = 0$	$a_5 = 0$	$\alpha_5 = -\pi/2$
$\Theta_6 = q_6$	$d_6 = 72$	$a_6 = 0$	$\alpha_6 = 0$

Nadalje mogu se dobiti transformacijske matrice za sve zglobove koristeći izraz 2.9:

$$T_{k-1}^{k} = \begin{bmatrix} \cos\Theta_k & -\cos\alpha_k \sin\Theta_k & \sin\alpha_k \sin\Theta_k & a_k \cos\Theta_k \\ \sin\Theta_k & \cos\alpha_k \cos\Theta_k & -\sin\alpha_k \cos\Theta_k & a_k \sin\Theta_k \\ 0 & \sin\alpha_k & \cos\alpha_k & d_k \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (2.9)$$

Matrica manipulatora se dobiva množenjem svih n transformacijskih matrica:

$$T_0^6 = T_0^1 * T_1^2 * T_2^3 * T_3^4 * T_4^5 * T_5^6, \tag{2.10}$$

gdje je:

$$T_0^1 = \begin{bmatrix} \cos q_1 & 0 & -\sin q_1 & 0 \\ \sin q_1 & 0 & \cos q_1 & 0 \\ 0 & -1 & 0 & 0.29 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$T_1^2 = \begin{bmatrix} \cos q_2 & -\sin q_2 & 0 & 0.29 \cos q_2 \\ \sin q_2 & \cos q_2 & 0 & 0.29 \sin q_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$T_2^3 = \begin{bmatrix} \cos q_3 & 0 & -\sin q_3 & 0.07 \cos q_3 \\ \sin q_3 & 0 & \cos q_3 & 0.07 \sin q_3 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$T_3^4 = \begin{bmatrix} \cos q_4 & 0 & \sin q_4 & 0 \\ \sin q_4 & 0 & -\cos q_4 & 0 \\ 0 & 1 & 0 & 0.302 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$T_4^5 = \begin{bmatrix} \cos q_5 & 0 & -\sin q_5 & 0 \\ \sin q_5 & 0 & \cos q_5 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$T_5^6 = \begin{bmatrix} \cos q_6 & -\sin q_6 & 0 & 0 \\ \sin q_6 & \cos q6 & 0 & 0 \\ 0 & 0 & 1 & 0.072 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

$$T_0^6 = [[1.0 * ((sin(q1) * sin(q4) + cos(q1) * cos(q4) * cos(q2 + q3)) * cos(q5) - sin(q5) * sin(q2 + q3) * cos(q1)) * cos(q6) + 1.0 * (sin(q1) * cos(q4) - 1.0 * sin(q4) * cos(q1) * cos(q2 + q3)) * sin(q6), 1.0 * (-(sin(q1) * sin(q4) + cos(q1) * cos(q4) * cos(q2 + q3)) * cos(q5) + sin(q5) * sin(q2 + q3) * cos(q1)) * sin(q6) + 1.0 * (sin(q1) * cos(q4) - 1.0 * sin(q4) * cos(q1) * cos(q2 + q3)) * cos(q6), -1.0 * (sin(q1) * sin(q4) + cos(q1) * cos(q4) * cos(q2 + q3)) * sin(q5) - 1.0 * sin(q2 + q3) * cos(q1) * cos(q5), -0.072 * sin(q1) * sin(q4) * sin(q5) - 0.072 * sin(q5) * cos(q1) * cos(q4) * cos(q2 + q3) - 0.072 * sin(q2 + q3) * cos(q1) * cos(q5) - 0.302 * sin(q2 + q3) * cos(q1) + 0.29 * cos(q1) * cos(q2) + 0.07 * cos(q1) * cos(q2 + q3)], [1.0 * ((sin(q1) * cos(q4) * cos(q2 + q3) - sin(q4) * cos(q1)) * cos(q5) - sin(q1) * sin(q5) * sin(q2 + q3)) * cos(q6) - 1.0 * (sin(q1) * sin(q4) * cos(q2 + q3) + cos(q1) * cos(q4)) * sin(q6), 1.0 * ((-sin(q1) * cos(q4) * cos(q2 + q3) + sin(q4) * cos(q1)) * cos(q5) + sin(q1) * sin(q5) * sin(q2 + q3)) * sin(q6) - 1.0 * (sin(q1) * sin(q4) * cos(q2 + q3) + cos(q1) * cos(q4)) * cos(q6), 1.0 * (-sin(q1) * cos(q4) * cos(q2 + q3) + sin(q4) * cos(q1)) * sin(q5) - 1.0 * sin(q1) * sin(q2 + q3) * cos(q5), -0.072 * sin(q1) * sin(q5) * cos(q4) * cos(q2 + q3) - 0.072 * sin(q1) * sin(q2 + q3) * cos(q5) - 0.302 * sin(q1) * sin(q2 + q3) + 0.29 * sin(q1) * cos(q2) + 0.07 * sin(q1) * cos(q2 + q3) + 0.072 * sin(q4) * sin(q5) * cos(q1)], [-1.0 * (sin(q5) * cos(q2 + q3) + sin(q2 + q3) * cos(q4) * cos(q5)) * cos(q6) + 1.0 * sin(q4) * sin(q6) * sin(q2 + q3), 1.0 * (sin(q5) * cos(q2 + q3) + sin(q2 + q3) * cos(q4) * cos(q5)) * sin(q6) + 1.0 * sin(q4) * sin(q2 + q3) * cos(q6), 1.0 * sin(q5) * sin(q2 + q3) * cos(q4) - 1.0 * cos(q5) * cos(q2 + q3), -0.29 * sin(q2) + 0.072 * sin(q5) * sin(q2 + q3) * cos(q4) - 0.07 * sin(q2 + q3) - 0.072 * cos(q5) * cos(q2 + q3) - 0.302 * cos(q2 + q3) + 0.29], [0, 0, 0, 1]]$$

Iz matrice manipulatora mogu se dobiti izrazi za trajektoriju u operacijskom prostoru:

$$w1 = -0.072 * sin(q1) * sin(q4) * sin(q5) - 0.072 * sin(q5) * cos(q1) * cos(q4) * cos(q2 + q3) - 0.072 * sin(q2 + q3) * cos(q1) * cos(q5) - 0.302 * sin(q2 + q3) * cos(q1) + 0.29 * cos(q1) * cos(q2) + 0.07 * cos(q1) * cos(q2 + q3), \quad (2.11)$$

$$w2 = -0.072 * sin(q1) * sin(q5) * cos(q4) * cos(q2 + q3) - 0.072 * sin(q1) * sin(q2 + q3) * cos(q5) - 0.302 * sin(q1) * sin(q2 + q3) + 0.29 * sin(q1) * cos(q2) + 0.07 * sin(q1) * cos(q2 + q3) + 0.072 * sin(q4) * sin(q5) * cos(q1), \quad (2.12)$$

$$w3 = -0.29 * sin(q2) + 0.072 * sin(q5) * sin(q2 + q3) * cos(q4) - 0.07 * sin(q2 + q3) - 0.072 * cos(q5) * cos(q2 + q3) - 0.302 * cos(q2 + q3) + 0.29, \quad (2.13)$$

$$w4 = -1.0 * \left(sin(q1) * sin(q4) + cos(q1) * cos(q4) * cos(q2 + q3)\right) * sin(q5) - 1.0 * sin(q2 + q3) * cos(q1) * cos(q5), \quad (2.14)$$

$$w5 = 1.0 * \left(-sin(q1) * cos(q4) * cos(q2 + q3) + sin(q4) * cos(q1)\right) * sin(q5) - 1.0 * sin(q1) * sin(q2 + q3) * cos(q5), \quad (2.15)$$

$$w6 = -0.072 * sin(q1) * sin(q5) * cos(q4) * cos(q2 + q3) - 0.072 * sin(q1) * sin(q2 + q3) * cos(q5) - 0.302 * sin(q1) * sin(q2 + q3) + 0.29 * sin(q1) * cos(q2) + 0.07 * sin(q1) * cos(q2 + q3) + 0.072 * sin(q4) * sin(q5) * cos(q1). \quad (2.16)$$

Ove izraze je nadalje potrebno preurediti tako da su q1, q2, q3, q4, q5 i q6 na lijevoj strani. Ovaj korak je bitan kako bi se mogli odrediti koeficijenti interpolacijskih polinoma. Kada bi se nastavio Ho-Cookov algoritam, trebalo bi provesti izraze 2.17, 2.18, 2.19 i 2.20 za dobivanje koeficijenta interpolacijskih polinoma.

$$\begin{bmatrix} \dot{q}_2 \\ \dot{q}_3 \\ \vdots \\ \dot{q}_{m-2} \\ \dot{q}_{m-1} \end{bmatrix} \begin{bmatrix} \frac{3}{t_2}+\frac{2}{t_3} & t_4 & 0 & \dots & 0 & 0 \\ \frac{1}{t_3} & 2(t_3+t_4) & t_5 & \dots & 0 & 0 \\ 0 & t_3 & 2(t_4+t_5) & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & t_{m-1} & 0 \\ 0 & 0 & 0 & \dots & 2(t_{m-2}+t_{m-1}) & \frac{t}{t_{m-1}} \\ 0 & 0 & 0 & \dots & t_{m-2} & \frac{2}{t_{m-1}}+\frac{3}{t_m} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{6}{t_2^2}(q_2-q_1)+\frac{3}{t_3^2}(q_3-q_2) \\ \frac{3}{t_3 t_4}[t_3^2(q_4-q_3)+t_4^2(q_3-q_2)] \\ \frac{3}{t_4 t_5}[t_4^2(q_5-q_4)+t_5^2(q_4-q_3)] \\ \frac{3}{t_{m-2}t_{m-1}}[t_{m-2}^2(q_{m-1}-q_{m-2})+t_{m-1}^2(q_{m-2}-q_{m-3})] \\ \frac{6}{t_m^2}(q_m-q_{m-1})+\frac{3}{t_{m-1}^2}(q_{m-1}-q_{m-2}) \end{bmatrix}. \quad (2.17)$$

Prvi segment:

$$[B_{01} \quad B_{11} \quad B_{21} \quad B_{31} \quad B_{41}] = [q_1 \quad q_2 \quad \dot{q}_1 \quad \dot{q}_2] \begin{bmatrix} 1 & 0 & 0 & -\frac{4}{t_2^3} & \frac{3}{t_2^4} \\ 0 & 0 & 0 & \frac{4}{t_2^3} & -\frac{3}{t_2^4} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{t_2^2} & \frac{1}{t_2^3} \end{bmatrix}. \quad (2.18)$$

Među-segmenti:

$$[B_{0k} \quad B_{1k} \quad B_{2k} \quad B_{3k}] = [q_k \quad q_{k+1} \quad \dot{q}_k \quad \dot{q}_{k+1}] \begin{bmatrix} 1 & 0 & -\frac{3}{t_{k+1}^2} & \frac{2}{t_{k+1}^3} \\ 0 & 0 & \frac{3}{t_{k+1}^2} & -\frac{2}{t_{k+1}^3} \\ 0 & 1 & -\frac{2}{t_{k+1}} & \frac{1}{t_{k+1}^2} \\ 0 & 0 & -\frac{1}{t_{k+1}} & \frac{1}{t_{k+1}^2} \end{bmatrix}. \tag{2.19}$$

Posljednji segment:

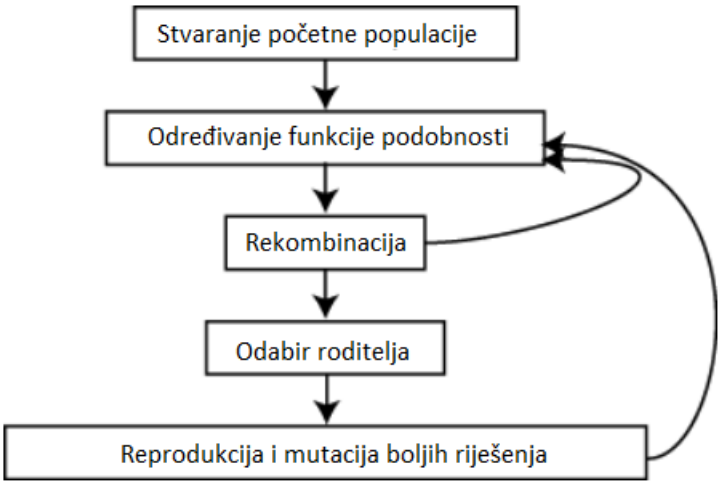
$$\begin{aligned} &[B_{0(m-1)} \quad B_{1(m-1)} \quad B_{2(m-1)} \quad B_{3(m-1)} \quad B_{4(m-1)}] \\ &= [q_{m-1} \quad q_m \quad \dot{q}_{m-1} \quad \dot{q}_m] \begin{bmatrix} 1 & 0 & -\frac{6}{t_m^2} & \frac{8}{t_m^3} & -\frac{3}{t_m^4} \\ 0 & 0 & \frac{6}{t_m^2} & -\frac{8}{t_m^3} & \frac{3}{t_m^4} \\ 0 & 1 & -\frac{3}{t_2} & \frac{3}{t_2^2} & -\frac{1}{t_2^3} \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \end{aligned} \tag{2.20}$$

No vidljivo je kako ovaj zadatak svakim sljedećim korakom postaje sve više kompleksan te se u ovom diplomskom radu nastoji pronaći neki drugi način određivanja istih. Točnije, primjenom umjetne inteligencije i genetskih algoritma.

3. GENETSKI ALGORITAM

Genetski algoritam je metoda za rješavanje problema ograničene i neograničene optimizacije koja se temelji na prirodnoj selekciji, procesu koji pokreće biološku evoluciju. Genetski algoritam više puta modificira populaciju pojedinačnih rješenja. U svakom koraku genetski algoritam odabire pojedince iz trenutne populacije da budu roditelji i koristi ih za stvaranje potomka za sljedeću generaciju. Tijekom uzastopnih generacija populacija "evoluira" prema optimalnom rješenju [9].

Genetski algoritam se može primijeniti za rješavanje raznih optimizacijskih problema koji nisu dobro prilagođeni standardnim optimizacijskim algoritmima, uključujući probleme u kojima je ciljna funkcija diskontinuirana, nediferencijabilna, stohastička ili izrazito nelinearna [9]. Slika 3.1 prikazuje osnovni tok izvođenja genetskog algoritma.



Slika 3.1 Shema izvođenja genetskog algoritma

3.1 Populacija i inicijalizacija genetskog algoritma

Populacija u genetskom algoritmu predstavlja skup svih mogućih rješenja (jedinki) u generaciji [10]. Svaka jedinka u populaciji ima svojstva koja se mogu mutirati ili promijeniti u svrhu dobivanja nove populacije [11]. Kvaliteta populacije je određena njenom raznolikosti i veličinom populacije. Raznolika populacija sprječava ranu konvergenciju. Prevelika populacija povećava vrijeme izvođenja algoritma i daje veliku raznolikost članova, dokle mala populacija ima smanjenu raznolikost [10].

Veličina populacije je ovisna o vrsti problema, ali često sadrži nekoliko stotina do nekoliko tisuća rješenja [11]. Inicijalizacija se može izvršiti na dva načina: slučajnom i heurističkom inicijalizacijom. Kod slučajne inicijalizacije početna populacija se stvara potpuno nasumično što utječe na veliku raznolikost populacije. Heuristička inicijalizacija podrazumijeva stvaranje populacije koristeći neku od heurističkih metoda. Korištenjem ove inicijalizacije dobiva se manja raznolikost između članova populacije [10].

Osim vrste inicijalizacije populacije potrebno je definirati populacijski model. Dva najčešća populacijska modela su stacionarni i generacijski model. Kod stacionarnog modela se u svakoj iteraciji jedan do dva stara člana zamijene s novonastalim potomcima. Kod generacijskog modela svi stari članovi se zamijene s populacijom novih članova [10].

3.2 Funkcija podobnosti

Genetski algoritam se bazira na odabiru boljih jedinki za vršenje genetskih operatora. Kako bi znali koje jedinke su bolje, potrebno ih je ocijeniti po nekom kriteriju. U tu svrhu se koristi funkcija podobnosti čije ulazne podatke predstavlja cijela populacija, a izlazni podaci su vrijednosti koji predstavljaju ocjenu rješenja [10].

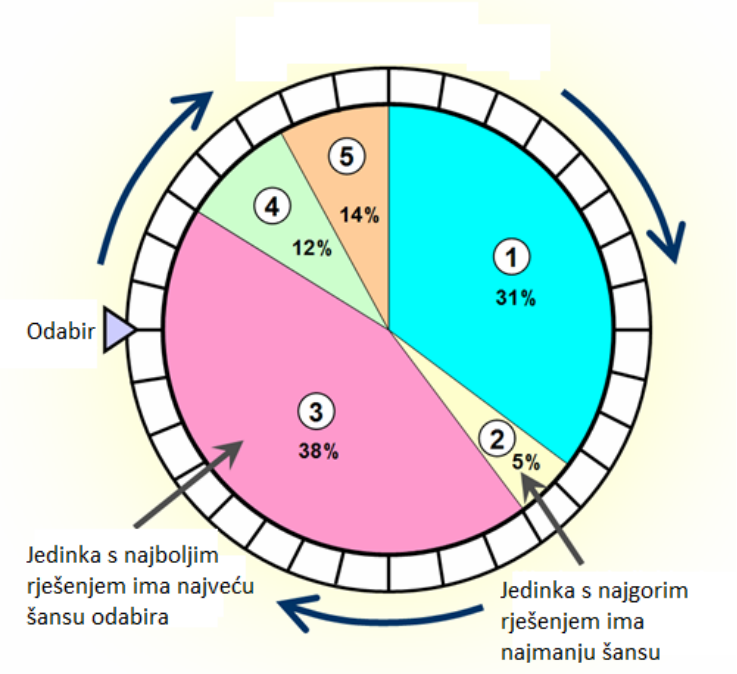
Funkcija podobnosti se mora izračunati za svaki član populacije u svakoj iteraciji izvođenja genetskog algoritma. Iz tog razloga bitan zahtjev je vrijeme izračuna funkcije podobnosti. Potrebno je što kraće određivanje funkcije podobnosti kako ne bi utjecali na vrijeme izvođenja programa [10].

Funkciju podobnosti u ovom radu predstavljaju razlike između segmenata putanje robotskog manipulatora. Kod nekih problema veća izlazna vrijednost funkcije podobnosti predstavljaju bolje rješenje. U ovom slučaju pošto želimo neprekinutu putanju, bolje jedinke su one s manjom izlaznom vrijednosti. Izlazna vrijednost će se računati kao srednja vrijednost razlika svih segmenata

3.3 Odabir roditelja

U svakoj generaciji, jedinke se odabiru u svrhu izvršavanja genetskih operatora i stvaranja nove populacije. Odabir roditelja je također jedan od važnih parametara genetskog algoritma zato što neadekvatnim odabirom roditelj može doći do rane konvergencije rješenja, odnosno želja je održati dobru raznolikost populacije. Odabirom samo najboljih rješenja također može dovesti do prerane konvergencije, te se iz tog razloga uvode različite metode odabira roditelja [10].

Najpopularnija metoda odabira roditelja i metoda koja je odabrana za ovaj diplomski rad je selekcija proporcionalna funkciji podobnosti. Odnosno, članovi s boljom funkcijom podobnosti imaju veću vjerojatnost postati roditelj. Primjer takve selekcije je selekcija temeljena na ruletu prikazana slikom 3.2 [10].



Slika 3.2 Selekcija temeljena na ruletu
Izvor: Roulette wheel selection in Python [12]

Kod selekcije temeljene na ruletu, kotač se sastoji od fiksne točke i n-dijelova gdje n predstavlja broj jedinki. Jedinke s boljom funkcijom podobnosti zauzimaju veću površinu kotača, dokle jedinke s lošijom funkcijom podobnosti zauzimaju manji dio. Kotač se zatim zavrti i jedinka na kojoj se kotač zaustavi se odabire kao roditelj [12].

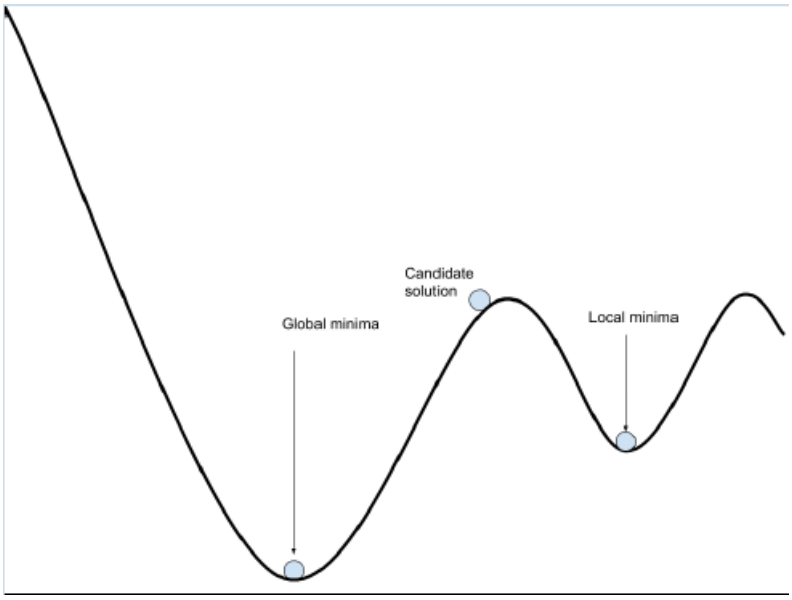
Nadalje, slično selekciji temeljenoj na ruletu postoji i stohastički univerzalno uzorkovanje. Ova metoda je identična selekciji temeljenoj na ruletu, ali umjesto jedne fiksne točke postoje dvije ili više te se odabir svih roditelja dogodi u samo jednom okretu kotača [10].

3.4 Križanje i mutacija

Križanje i mutacija su najvažniji koraci genetskog algoritma za stvaranje iduće generacije. Odabir roditelja se vrši na temelju funkcije podobnosti i vršenjem genetskih operatora dobiju se novi potomci koji čine iduću generaciju [11].

Križanje se temelji na reprodukciji i biološkom križanju. Primjenom križanja na dvije jedinke iz populacije dobiva se nova jedinka (potomak) koji sadrži genetski materijal svojih roditelja. Ovisno o tipu jedinke ovisit će i tip križanja koji je potrebno primijeniti [10].

Kod mutacije, novi potomak nastaje tako da se nasumično promijene vrijednosti kromosoma. Cilj mutacije je održati raznolikost u populaciji te spriječiti konvergenciju rješenja u lokalni ekstrem. Broj operacija mutacije ipak ne smije biti prevelik jer u suprotnom se algoritam svodi na slučajno pretraživanje [10].



Slika 3.3 Lokalni i globalni ekstremi
Izvor: Uvod u Genetske algoritme [10]

Osim mutacije i križanja u algoritam se može uvesti i rekombinacija. Rekombinacijom se određen broj jedinki s boljom funkcijom podobnosti proslijede u iduću generaciju bez izvođenja dodatnih genetskih operacija [11].

3.5 Uvjet zaustavljanja

Genetski algoritam je iterativan postupak, te ga je moguće izvršavati koliko god je potrebno. Ipak, česta pojava je kako algoritam brzo dolazi do boljih rješenja unutar svega nekoliko generacija. Daljnje izvođenje algoritma ne stvara bolja rješenja te je potrebno uvesti uvjet zaustavljanja tako da rješenje bude optimalno [10].

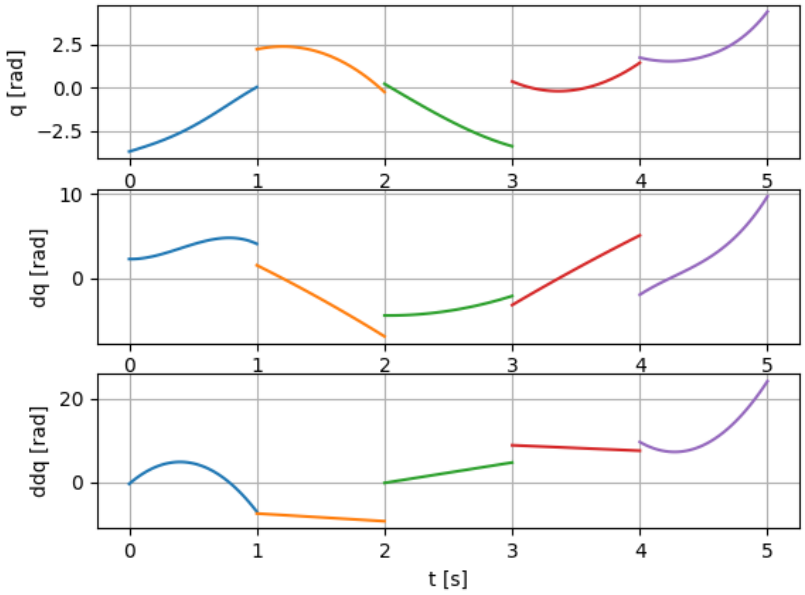
Algoritam se zaustavlja ako je ostvareno jedno od sljedećih uvjeta [10]:

- Stagnacija – nije došlo do značajne promjene rješenja u nekoliko generacija,
- Zadovoljenje graničnih vrijednosti – algoritam je pronašao rješenje unutar zadanih granica i
- Broj generacija – zadovoljen je broj generacija.

4. PROGRAMSKI KOD U PYTHONU

Prethodno su opisani osnovi pojmovi u robotici i genetski algoritam te se sada može krenuti u razradu problema. Problem određivanja parametara putanje robotskog manipulatora se svodi na određivanje koeficijenta interpolacijskih polinoma za koje se može dobiti putanja bez trzaja. U ovom diplomskom radu u svrhu usporedbe su izrađena dva koda za rješavanje navedenog problema, a to su genetski algoritam i tradicionalan pristup.

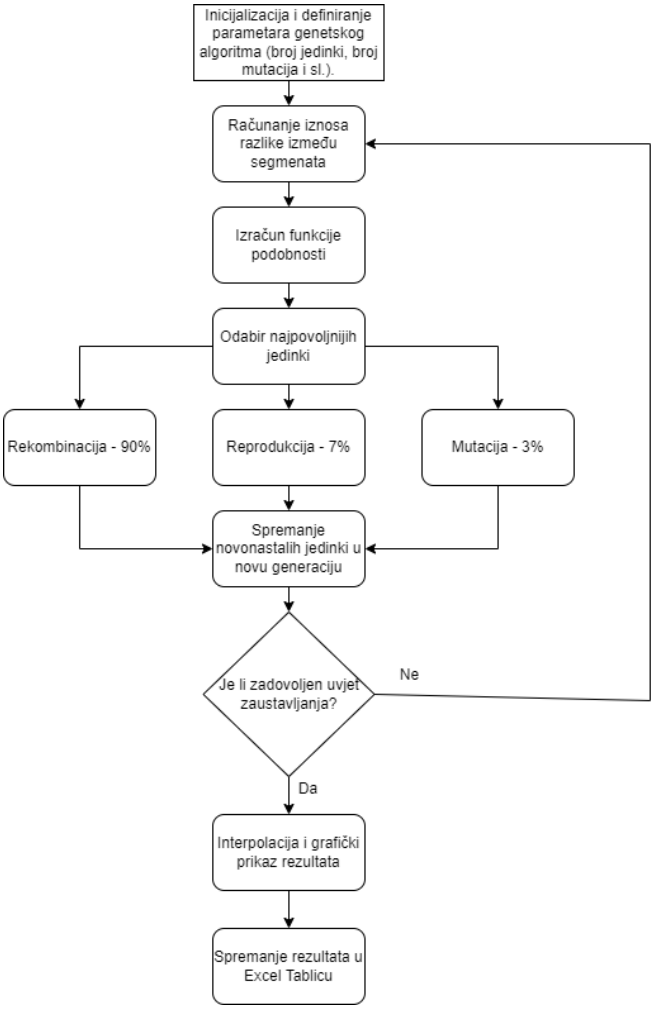
Nasumičnim odabirom koeficijenta interpolacijskih polinoma se dolazi do isprekidanosti segmenata kako je prikazano slikom 4.1. Svaki segment je označen zasebnom bojom, te je vidljiv jasan prekid između njih. Poželjno je da su svi segmenti putanja, brzina i ubrzanja neprekidni ili da su barem razlike između segmenata što manje. Neprekinutost grafa predstavlja štetne trzaje robota koje mogu dovesti do njegovog oštećenja.



Slika 4.1 Prikaz putanja, brzina i ubrzanja za nasumično generirane koeficijente interpolacijskih polinoma

4.1 Genetski algoritam

Kako bi se smanjile razlike između segmenata primijenit će se genetski algoritam koji je objašnjen u nastavku. Slikom 4.2 prikazan je dijagram toka koji prikazuje tok izvođena programa napisanog u Pythonu.



Slika 4.2 Dijagram toka genetskog algoritma

```
Broj_Jedinki = 10000

B = np.random.uniform(-5, 5, (Broj_Jedinki, 22))

Elementi = np.split(B, Broj_Jedinki)
Redni_Broj_Elementa = [i for i in range(Broj_Jedinki)]
```

Prvo je potrebno definirati broj jedinki u populaciji. Što je veći broj jedinki, potrebno je puno manje generacija kako bi se došlo do odgovarajućeg rezultata, ali se zato povećava vrijeme

izvođenja programa. Na temelju pokušaja i pogreške, zaključeno je kako najbolji rezultati proizlaze pri 10000 jedinki u populaciji.

Novonastale jedinke se spajaju u jednu cjelinu koja čini populaciju. Populaciju čine matrice koeficijenata interpolacijskih polinoma B. Pošto se radi o 6 točaka putanje, bit će 5 segmenata krivulje, odnosno 5 matrica B (B_1, B_2, B_3, B_4, B_5). Prva i zadnja matrica B imaju 5 stupaca, te ostale matrice imaju 4 stupaca. Radi jednostavnosti sve matrice su spojene u jedan redak, te dobivamo matricu dimenzija 1x22 za jedan zglob robota. Matrica B je popunjena uniformnom distribucijom od -5 do 5. Na kraju je još potrebno dodijeliti svakoj matrici redni broj kako bi se kasnije olakšalo traženje i uzimanje matrice za izvođenje genetskih operacija.

```
Broj_Rekombinacija = int(0.9*Broj_Jedinki)
Broj_Reprodukcija = int(0.07*Broj_Jedinki)
Broj_Mutacija = int(0.03*Broj_Jedinki)
```

Sljedeći korak je odabrati broj rekombinacija, reprodukcija i mutacija. U ovom slučaju je određeno kako će se reprodukcije izvoditi u 90% iteracija, rekombinacije u 7% iteracija i mutacije u 3%. Prvobitno je bilo odabrano 1% mutacija, ali 3% je pokazalo nešto bolje rezultate.

```
for Generacija in range(10):
    print("Generacija:", Generacija)
    for i in range(Broj_Jedinki):
        q1  =  B[i,0]  +  B[i,1]*tq1  +  B[i,2]*(tq1**2)  +  B[i,3]*(tq1**3)  +
B[i,4]*(tq1**4)
        q2 = B[i,5] + B[i,6]*tq2 + B[i,7]*(tq2**2) + B[i,8]*(tq2**3)
        q3 = B[i,9] + B[i,10]*tq3 + B[i,11]*(tq3**2) + B[i,12]*(tq3**3)
        q4 = B[i,13] + B[i,14]*tq4 + B[i,15]*(tq4**2) + B[i,16]*(tq4**3)
        q5  =  B[i,17]  +  B[i,18]*tq5  +  B[i,19]*(tq5**2)  +  B[i,20]*(tq5**3)  +
B[i,21]*(tq5**4)

        dq1 = B[i,1] + 2*B[i,2]*(tq1) + 3*B[i,3]*(tq1**2) + 4*B[i,4]*(tq1**3)
        dq2 = B[i,6] + 2*B[i,7]*(tq2) + 3*B[i,8]*(tq2**2)
        dq3 = B[i,10] + 2*B[i,11]*(tq3) + 3*B[i,12]*(tq3**2)
        dq4 = B[i,14] + 2*B[i,15]*(tq4) + 3*B[i,16]*(tq4**2)
        dq5 = B[i,18] + 2*B[i,19]*(tq5) + 3*B[i,20]*(tq5**2) + 4*B[i,21]*(tq5**3)

        ddq1 = 2*B[i,2] + 6*B[i,3]*(tq1) + 12*B[i,4]*(tq1**2)
        ddq2 = 2*B[i,7] + 6*B[i,8]*(tq2)
        ddq3 = 2*B[i,11] + 6*B[i,12]*(tq3)
        ddq4 = 2*B[i,15] + 6*B[i,16]*(tq4)
        ddq5 = 2*B[i,19] + 6*B[i,20]*(tq5) + 12*B[i,21]*(tq5**2)

        d1_q = abs(q1[-1] - q2[0])
        d2_q = abs(q2[-1] - q3[0])
        d3_q = abs(q3[-1] - q4[0])
        d4_q = abs(q4[-1] - q5[0])
        q = [d1_q, d2_q, d3_q, d4_q]
```

```
        dist_q.append(q)

        d1_dq = abs(dq1[-1] - dq2[0])
        d2_dq = abs(dq2[-1] - dq3[0])
        d3_dq = abs(dq3[-1] - dq4[0])
        d4_dq = abs(dq4[-1] - dq5[0])
        dq = [d1_dq, d2_dq, d3_dq, d4_dq]
        dist_dq.append(dq)

        d1_ddq = abs(ddq1[-1] - ddq2[0])
        d2_ddq = abs(ddq2[-1] - ddq3[0])
        d3_ddq = abs(ddq3[-1] - ddq4[0])
        d4_ddq = abs(ddq4[-1] - ddq5[0])
        ddq = [d1_ddq, d2_ddq, d3_ddq, d4_ddq]
        dist_ddq.append(ddq)
```

Nadalje, broj generacija je odabran kao uvjet zaustavljanja. Nakon X generacija rezultat počinje stagnirati. Broj generacija koji je potreban ovisi o broju jedinki te se određuje na principu pokušaja i pogreške. Kako bi se rješenje kasnije procijenilo potrebno je iscrtati grafove svih jedinki u generaciji i izračunati razliku između njihovih segmenata krivulja. Iscrtavanje grafova se vrši po izrazima (2.1 – 2.6) za putanje, brzine i ubrzanja.

Nakon što su dobiveni svi grafovi za svaku jedinku u generaciji, mjeri se razlika između segmenata i sprema se u zasebnu listu koja će se kasnije koristiti kao ulazni podaci u funkciju podobnosti.

```
    distances = np.hstack((dist_q, dist_dq, dist_ddq))

    Prosjek_Stupaca = distances.mean(1).tolist()

    fitness = Prosjek_Stupaca

    print('Minimum:', min(fitness))
    print('Maximum:', max(fitness))

    Fitness_Sum = sum(fitness)
    Vjerojatnost_Odabira = [x/Fitness_Sum for x in fitness]
    Vjerojatnost_Odabira = 1 - np.array(Vjerojatnost_Odabira)

    Suma_Vjerojatnosti = sum(Vjerojatnost_Odabira)
    Vjerojatnost_Odabira = Vjerojatnost_Odabira/Suma_Vjerojatnosti
```

Razlike između segmenata svake pojedine jedinke se sprema u matricu *distances*. Matrica *distances* je dimenzija 10000x12, gdje redci predstavljaju broj jedinki, a stupci predstavljaju razliku između segmenata. Prvih 4 stupaca predstavljaju razlike između segmenata grafa putanje, sljedećih 4 su razlike između segmenata grafa brzina i zadnjih 4 su razlike između segmenata grafa ubrzanja za prvu jedinku.

Nadalje, matrica *distances* se usrednjava po redcima kako bi se dobila srednja vrijednost razlika između segmenata svake jedinke u generaciji. Ovime je provedena funkcija podobnosti gdje smo svaku jedinku ocijenili po svom rješenju. Pošto želimo smanjiti razlike između segmenata, jedinke koje imaju manju srednju vrijednost razlike između segmenata se smatraju boljim jedinkama.

Što se tiče odabira roditelja, implementirana je selekcija temeljena na ruletu. Za njegovu implementaciju potrebno je svakoj jedinci dodijeliti postotnu vrijednost odabira, odnosno površinu koju će zauzimati na kotaču. Prvo se sumiraju sve vrijednosti funkcije podobnosti, te se zatim vrijednosti funkcije podobnosti svake pojedine jedinke podijele sa sumom, prema izrazu 4.1. Kao rezultat dobivamo veću vjerojatnost odabira jedinki s većom razlikom između segmenata.

$$p_i[\%] = \frac{f_i}{\sum_{k=1}^{P} f_k}, \tag{4.1}$$

gdje je:

f – funkcija podobnosti jedinke,

P – ukupni broj jedinki i

p[%] – postotna vrijednost dodijeljena jedinki.

Ipak, u ovom slučaju tražimo jedinke koje imaju što manju razliku između segmenata krivulje. Kako bi prednosti dali jedinkama s manjom udaljenosti potrebno je postotne vrijednosti svih jedinki oduzeti od 1. No ni tu nije kraj jer suma postotnih vrijednosti sada ne iznosi sveukupno 100%, tako da je potrebno još jednom podijeliti sve postotne vrijednosti sa sumom postotnih vrijednosti, prikazano izrazom 4.2. Tek sada su svakoj jedinci dodijeljene odgovarajuće postotne vrijednosti za postati roditelj.

$$p_{i,min}[\%] = \frac{1 - p_i[\%]}{\sum_{k=1}^{P} p_k}, \tag{4.2}$$

gdje je:

p_{min}[%] – inverzna postotna vrijednost.

```
    for i in range(Broj_Rekombinacija):

        Odabir1 = np.random.choice(Redni_Broj_Elementa, p=Vjerojatnost_Odabira)
        Odabir2 = np.random.choice(Redni_Broj_Elementa, p=Vjerojatnost_Odabira)

        while(Odabir1 == Odabir2):
            Odabir2            =            np.random.choice(Redni_Broj_Elementa,
p=Vjerojatnost_Odabira)

        Rekombinacija = (Elementi[Odabir1] + Elementi[Odabir2])/2

        while (np.abs(np.mean(Rekombinacija)) < 0.01):
            Odabir1            =            np.random.choice(Redni_Broj_Elementa,
p=Vjerojatnost_Odabira)
            Odabir2            =            np.random.choice(Redni_Broj_Elementa,
p=Vjerojatnost_Odabira)

            while(Odabir1 == Odabir2):
                Odabir1          =          np.random.choice(Redni_Broj_Elementa,
p=Vjerojatnost_Odabira)
                Odabir2          =          np.random.choice(Redni_Broj_Elementa,
p=Vjerojatnost_Odabira)

            Rekombinacija = (Elementi[Odabir1] + Elementi[Odabir2])/2


        Novi_Elementi.append(Rekombinacija)
```

Sljedeći korak je izvođenje genetskih operacija. Prvo se izvode rekombinacije, odnosno križanje jedinki. Na principu selekcije temeljene na ruletu odabiru se dvije bolje jedinke iz populacije, te se vrši provjera da nije odabrana jedna te ista jedinka. Zatim, izvodi se križanje kako bi se stvorio potomak. Potomak je rezultat srednje vrijednosti dvaju roditelja.

Kako bi se spriječilo nastajanje potomaka koji su jednaki nul-matricama potrebno je postaviti granični uvjet. Odnosno, postavljen je uvjet koji govori kako apsolutna vrijednost sume elemenata matrice ne smiju biti manji od broja X. Uvjet X je podesiv u ovisnosti kakva rješenja želimo, no eksperimentalno je pokazano kako je taj broj najbolji kada se nalazi u intervalu dan izrazom 4.3.

$$0.05 \leq X < 0.2. \tag{4.3}$$

Pri završetku križanja, potomci se spremaju u zasebnu listu koja predstavlja novu populaciju.

```
for i in range(Broj_Reprodukcija):

        Reprodukcija           =           np.random.choice(Redni_Broj_Elementa,
p=Vjerojatnost_Odabira)
        while (Reprodukcija in Provjera_Reprodukcije) == True:
            Reprodukcija         =         np.random.choice(Redni_Broj_Elementa,
p=Vjerojatnost_Odabira)

        Provjera_Reprodukcije.append(Reprodukcija)
        Novi_Elementi.append(Elementi[Reprodukcija])
```

Osim križanja i mutacija, u 7% slučajeva događa se reprodukcija, odnosno 7% boljih rješenja se ne podlažu genetskim operacijama već se samo prosljeđuju u sljedeću generaciju. Ove jedinke se također biraju na principu selekcije temeljene na ruletu. Uvedena je provjera koja provjerava da nije odabrana dva puta ista jedinka.

```
for i in range(Broj_Mutacija):
        Mutacija = np.random.choice(Redni_Broj_Elementa, p=Vjerojatnost_Odabira)

        Matrica_Promjene  = Elementi[Mutacija].copy()
        Matrica_Mutacija = np.random.uniform(-5, 5, (1, 22))

        for j in range(5):

            Poz_Stupac = np.random.randint(0, 22)
            Matrica_Promjene[0][Poz_Stupac]                                    =
Matrica_Mutacija[0][Poz_Stupac].copy()

        Novi_Elementi.append(Matrica_Promjene)
```

U ostalih 3% iteracije izvršava se mutacija. Kako je prethodno spomenuto, jedinka se sastoji od matrice dimenzije 1x22, te se mutacija izvršava tako da se nasumično promijene do 5 elemenata matrice nasumičnim brojem od -5 do 5.

```
    Elementi.clear()
    Elementi = Novi_Elementi.copy()
    B = Elementi.copy()
    B = np.array(B)
    B = np.reshape(B, (Broj_Jedinki,22))

    Novi_Elementi.clear()
    dist_q.clear()
    dist_dq.clear()
    dist_ddq.clear()

    minimalne_udaljenosti.append(min(fitness))
```

Na samom kraju, potrebno je popuniti populaciju s novim jedinkama koje će se koristiti u sljedećoj iteraciji genetskog algoritma. Također je potrebno očistiti sve liste kako ne bi ostali

tragove prijašnjih generacija koje bi narušavale kvalitetu izvođenja programa. Kako bi se mogla procijeniti kvaliteta izrađenog genetskog algoritma i kvaliteta svake generacije, najbolje vrijednosti funkcije podobnosti svake generacije se spremaju u zasebnu listu.

```
    q1_int, q2_int, q3_int, q4_int, q5_int = Plotanje(B, fitness, t)
    q_int = np.vstack((q1_int, q2_int, q3_int, q4_int, q5_int))

    interp = Interpolacija_6(q_int, 10, t)

    q_ovi.append(interp[::10])

q_ovi = np.array(q_ovi)
q_ovi = np.around(q_ovi.T, 9)
pd.DataFrame(q_ovi).to_csv('Qovi.csv', index=False)
```

Ovaj postupak se ponovi još pet puta za preostalih pet zglobova koje posjeduje robotski manipulator ABB IRB 120. Osim toga, rezultati se prikazuju grafički i spremaju u zasebnu Excel tablicu koja će kasnije biti iskorištena u svrhu izvođenja simulacije u programu RobotStudio.

4.2 Dodatne funkcije

Za što veću preglednost glavnog koda, funkcije crtanja i interpolacije su napisane kao funkcije u drugoj skripti.

4.2.1 Crtanje grafova

```
def Plotanje(Matrica, fitness_Matrice, t):

    t2 = t[1, -1]
    t3 = t[2, -1]
    t4 = t[3, -1]
    t5 = t[4, -1]

    Pronadi = fitness_Matrice.index(min(fitness_Matrice))

    q1 = Matrica[Pronadi,0] + Matrica[Pronadi,1]*t[0] +
Matrica[Pronadi,2]*(t[0]**2) + Matrica[Pronadi,3]*(t[0]**3) +
Matrica[Pronadi,4]*(t[0]**4)
    q2 = Matrica[Pronadi,5] + Matrica[Pronadi,6]*t[1] +
Matrica[Pronadi,7]*(t[1]**2) + Matrica[Pronadi,8]*(t[1]**3)
    q3 = Matrica[Pronadi,9] + Matrica[Pronadi,10]*t[2] +
Matrica[Pronadi,11]*(t[2]**2) + Matrica[Pronadi,12]*(t[2]**3)
    q4 = Matrica[Pronadi,13] + Matrica[Pronadi,14]*t[3] +
Matrica[Pronadi,15]*(t[3]**2) + Matrica[Pronadi,16]*(t[3]**3)
```

```
    q5 = Matrica[Pronadi,17] + Matrica[Pronadi,18]*t[4] +
Matrica[Pronadi,19]*(t[4]**2) + Matrica[Pronadi,20]*(t[4]**3) +
Matrica[Pronadi,21]*(t[4]**4)

    dq1 = Matrica[Pronadi,1] + 2*Matrica[Pronadi,2]*(t[0]) +
3*Matrica[Pronadi,3]*(t[0]**2) + 4*Matrica[Pronadi,4]*(t[0]**3)
    dq2 = Matrica[Pronadi,6] + 2*Matrica[Pronadi,7]*(t[1]) +
3*Matrica[Pronadi,8]*(t[1]**2)
    dq3 = Matrica[Pronadi,10] + 2*Matrica[Pronadi,11]*(t[2]) +
3*Matrica[Pronadi,12]*(t[2]**2)
    dq4 = Matrica[Pronadi,14] + 2*Matrica[Pronadi,15]*(t[3]) +
3*Matrica[Pronadi,16]*(t[3]**2)
    dq5 = Matrica[Pronadi,18] + 2*Matrica[Pronadi,19]*(t[4]) +
3*Matrica[Pronadi,20]*(t[4]**2) + 4*Matrica[Pronadi,21]*(t[4]**3)

    ddq1 = 2*Matrica[Pronadi,2] + 6*Matrica[Pronadi,3]*(t[0]) +
12*Matrica[Pronadi,4]*(t[0]**2)
    ddq2 = 2*Matrica[Pronadi,7] + 6*Matrica[Pronadi,8]*(t[1])
    ddq3 = 2*Matrica[Pronadi,11] + 6*Matrica[Pronadi,12]*(t[2])
    ddq4 = 2*Matrica[Pronadi,15] + 6*Matrica[Pronadi,16]*(t[3])
    ddq5 = 2*Matrica[Pronadi,19] + 6*Matrica[Pronadi,20]*(t[4]) +
12*Matrica[Pronadi,21]*(t[4]**2)

    plt.figure()
    plt.subplot(3,1,1)
    plt.plot(t[0], q1)
    plt.plot(t[1]+t2, q2)
    plt.plot(t[2]+t2+t3, q3)
    plt.plot(t[3]+t2+t3+t4, q4)
    plt.plot(t[4]+t2+t3+t4+t5, q5)
    plt.ylabel('q [rad]')
    plt.xlabel('t [s]')
    plt.grid()

    plt.subplot(3,1,2)
    plt.plot(t[0], dq1)
    plt.plot(t[1]+t2, dq2)
    plt.plot(t[2]+t2+t3, dq3)
    plt.plot(t[3]+t2+t3+t4, dq4)
    plt.plot(t[4]+t2+t3+t4+t5, dq5)
    plt.ylabel('dq [rad]')
    plt.xlabel('t [s]')
    plt.grid()

    plt.subplot(3,1,3)
    plt.plot(t[0], ddq1)
    plt.plot(t[1]+t2, ddq2)
    plt.plot(t[2]+t2+t3, ddq3)
    plt.plot(t[3]+t2+t3+t4, ddq4)
    plt.plot(t[4]+t2+t3+t4+t5, ddq5)
    plt.ylabel('ddq [rad]')
    plt.xlabel('t [s]')
    plt.grid()

    return q1, q2, q3, q4, q5
```

Dobivanje grafova vrši se uvrštavanjem u izraze 2.1 – 2.6. Od cijele generacije potrebno je pronaći jedinku koja ima najbolju funkciju podobnosti te se ta jedinka uzima za dobivanje grafa putanje, brzine i ubrzanja. Argumente koje ova funkcija prima su cijela populacija, funkcija

podobnosti svake jedinke i vrijeme trajanja svakog segmenta. Osim što iscrtava potrebne grafove, funkcija također vraća vrijednosti segmenata u određenim vremenskim trenutcima. Odabrani segmenti su bitni jer se kasnije spajaju u neprekinutu krivulju primjenom interpolacije.

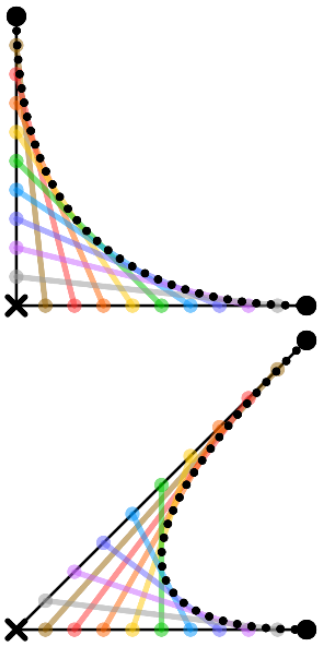
4.2.2 Interpolacija

```
def Interpolacija_6(q, Tocke, t):

    tq1 = t[0,:]
    tq2 = t[1,:] + 1
    tq3 = t[2,:] + 2
    tq4 = t[3,:] + 3
    tq5 = t[4,:] + 4

    tq1 = np.reshape(tq1[0:(50 - Tocke)], (50 - Tocke,1))
    tq2 = np.reshape(tq2[Tocke:(50 - Tocke)], (50 - 2*Tocke,1))
    tq3 = np.reshape(tq3[Tocke:(50 - Tocke)], (50 - 2*Tocke,1))
    tq4 = np.reshape(tq4[Tocke:(50 - Tocke)], (50 - 2*Tocke,1))
    tq5 = np.reshape(tq5[Tocke:50], (50 - Tocke,1))

    q1 = np.reshape(q[0, 0:(50 - Tocke)], (50 - Tocke,1))
    q2 = np.reshape(q[1, Tocke:(50 - Tocke)], (50 - 2*Tocke,1))
    q3 = np.reshape(q[2, Tocke:(50 - Tocke)], (50 - 2*Tocke,1))
    q4 = np.reshape(q[3, Tocke:(50 - Tocke)], (50 - 2*Tocke,1))
    q5 = np.reshape(q[4, Tocke:50], (50 - Tocke,1))

    q = np.vstack((q1, q2, q3, q4, q5))
    tq = np.vstack((tq1, tq2, tq3, tq4, tq5))
    y = q
    x = tq

    t, c, k = interpolate.splrep(x, y, s=0, k=2)

    N = 250
    xmin, xmax = x.min(), x.max()
    xx = np.linspace(xmin, xmax, N)
    spline = interpolate.BSpline(t, c, k, extrapolate=False)

    plt.subplot(2,1,2)
    plt.plot(x, y, 'bo')
    plt.plot(xx, spline(xx))
    plt.grid()
    plt.show()

    return spline(xx)
```

Uklanjanje razlika između segmenata se rješava interpolacijom, te je u tu svrhu napisana dana funkcija koja prima argumente svih segmenta, broj točaka koje se želi ukloniti i vrijeme trajanja segmenata.

Svaki segment se sastoji od 50 točaka, te kako bi se dobila glatka funkcija putanje robotskog manipulatora potrebno je ukloniti zadnjih nekoliko točaka svakog segmenta. Što je veći broj točaka koji će se ukloniti, funkcija će biti glađa, ali se gubi originalni segment. Premalen broj točaka daje strme prijelaze između segmenata koji su neželjeni. Eksperimentalno najbolje je uzeti 10-15 točaka koji će se zanemariti po segmentu.

Interpolacija je odrađena koristeći Bézierovih krivulja, slika 4.3. Bézierove krivulje su parametrične krivulje važne u području matematičke numeričke analize. Često se primjenjuju u računalnoj grafici. Bézierove krivulje se koriste za oblikovanje jasnih i glatkih krivulja. U praksi se primjenjuju u programima za grafičko crtanje poput Adobe Photoshopa ili u programima za animaciju poput Adobe Flasha [13].



Slika 4.3 Postupak dobivanja Bézierovih krivulja
Izvor: Bezier Curve [13]

Bézierove krivulje se matematički mogu opisati izrazom 4.1 [13]:

$$S(x) = \sum_{j=0}^{n-1} c_j B_{j,k,t}\,(x). \tag{4.1}$$

U jednadžbi 4.1., navedeni elementi predstavljaju:

t – broj čvorova,

c – koeficijent krivulje,

k – stupanj krivulje i

B – bazne funkcije stupnja k i čvorova t.

Bazne funkcije B su opisane izrazom 4.2 [14]:

$$B_{i,k}(x) = \frac{x - t_i}{t_{i+k} - t_i} B_{i,k-1}(x) + \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(x), \tag{4.2}$$

$$B_{i,0}(x) = \begin{cases} 1 \; za \; t_i \leq x < t_{i+1} \\ 0, inače \end{cases}. \tag{4.3}$$

Parametre t, c, k moguće je dobiti koristeći funkciju *interpolate.splrep*. Ova funkcija prima argumente segmenta i željen stupanj polinoma. Najbolji rezultati su dobiveni uz polinom drugog stupnja, te je isti i odabran.

4.3 Tradicionalan pristup

Tradicionalan pristup ovom problemu se podrazumijeva traženje rješenja bez primjene umjetne inteligencije. Dijagram toka prikazan je slikom 4.4.

Slika 4.4 Dijagram toka tradicionalnog pristupa

Na početku se definira vrijeme trajanja svakog segmenta i nasumično se odaberu koeficijenti interpolacijskih polinoma.

```
i = 0
while (np.around(abs(ddq1_1[-1] - ddq1_2[0]), preciznost) != 0):
    B2_1 = np.random.randint(-2000, 2000, (1, 4))/1000
    q1_2 = B2_1[0,0] + B2_1[0,1]*tq2 + B2_1[0,2]*(tq2**2) +
B2_1[0,3]*(tq2**3)

    if(np.around(abs(q1_1[-1] - q1_2[0]), preciznost) == 0):
        dq1_2 = B2_1[0,1] + 2*B2_1[0,2]*(tq2) + 3*B2_1[0,3]*(tq2**2)

        if(np.around(abs(dq1_1[-1] - dq1_2[0]), preciznost) == 0):
            ddq1_2 = 2*B2_1[0,2] + 6*B2_1[0,3]*(tq2)
            #file_out.write(str(np.around(abs(ddq1_1[-1] -
ddq1_2[0]))+str(preciznost)))

    i = i + 1

file_out.write('Prvi segment završen u '+str(i)+'iteracija.\n')
```

Nakon što su iscrtani svi segmenti, program se izvršava segment po segment. Odnosno, provjerava se ako je razlika između prvog i drugog segmenta manja od željene razlike. Ako je razlika između zadnje točke prvog segmenta i prve točke drugog segmenta veća od zadane

razlike, drugi segment se odbacuje i nasumično se generira sljedeći. Ovaj postupak se izvršava sve dokle se ne pronađe segment koji zadovoljava uvjet razlike.

Razlika između dva segmenta mora biti zadovoljavajuća za sva tri grafa putanje, brzine i ubrzanja. Također se broji koliko iteracije je potrebno za pronalazak rješenja. Ovaj postupak se ponovi za svaki segment. Krivulja putanje se sastoji od 5 segmenta te pošto su pronađena rješenja za prvi i drugi segment, potrebno je ponoviti postupak još 3 puta.

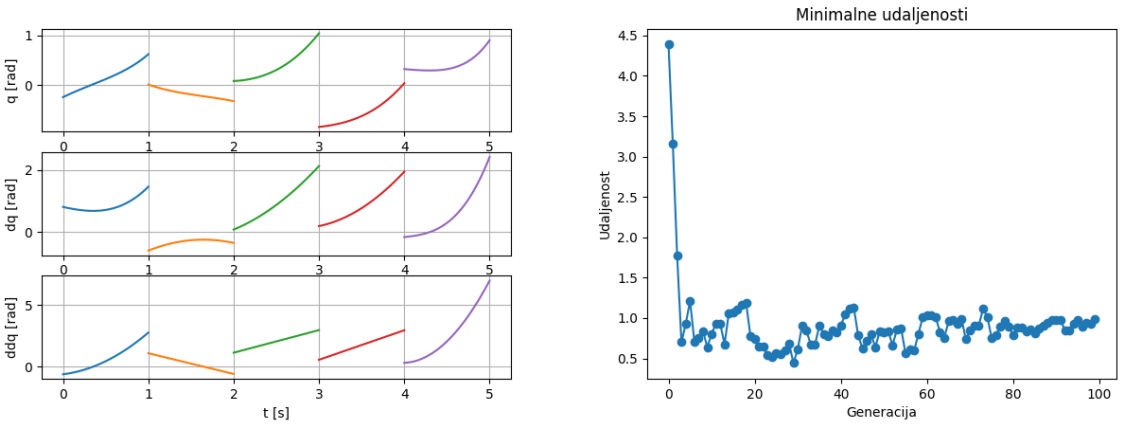
Bitan parametar u ovom postupku je željena razlika između dva segmenta. Ako je zadana razlika velika, potrebno je manje vremena da se pronađe odgovarajući segment, ali je razlika vidljivo velika. Manja željena razlika daje bolje rezultate, ali se vrijeme izvođenja programa znatno povećava.

5. REZULTATI

Prethodno je objašnjen postupak i programski kod, a sada će se rezultati razmatranja pokazati. Tijek izvođenja programa i rezultati ovise o parametrima koji se podese u genetskom algoritmu. Potrebno je pronaći što bolje vrijednosti broja jedinki, broja generacija i uvjete rezultata srednje vrijednosti kako bi sljedeća generacija bila što bolja.

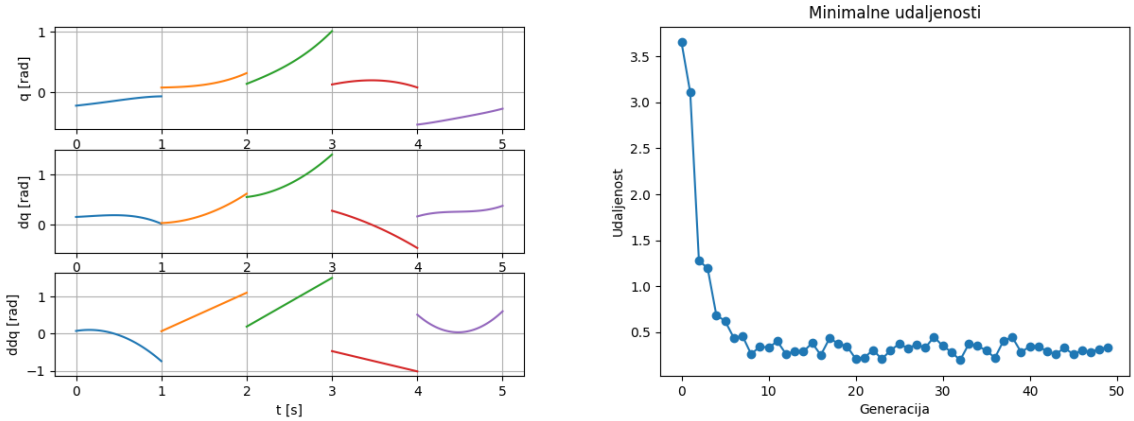
5.1 Broj jedinki u populaciji

Broj jedinki u populaciji znatno utječu na vrijeme izvođenja programa. Što je manji broj jedinki, program se brže izvodi, ali su zato rezultati poprilično loši. Slikom 5.1 prikazani su rezultati i funkcija podobnosti za slučaj kada se radi o 100 jedinki i 100 generacija. Prekidi između segmenata su veliki, a funkcija podobnosti stagnira oko 1, što je poprilično loše. Vrijeme izvođenja programa je nekoliko sekundi.



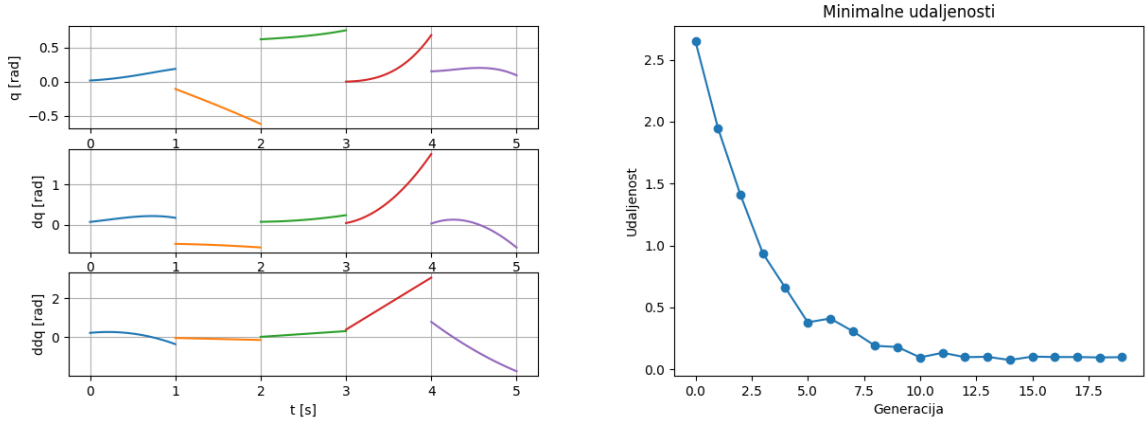
Slika 5.1 Rezultati za 100 jedinki i 100 generacija

Daljnjim povećanjem broja jedinki dobivaju se bolji rezultati, kako je prikazano na slici 5.2. U prethodnom slučaju, rješenje je stagniralo u prvih nekoliko iteracija, dok u ovom slučaju stagnacija se dogodila oko 10. generacije. Također je vidljivo kako je funkcija podobnosti manja (oko 0.33), te su ujedno i fluktuacije manje. To je direktno prikazano i na grafovima gdje su segmenti bliže jedan drugome nego u prethodnom slučaju. Vrijeme izvođenja programa je tek nekoliko sekundi duže od prethodnog.



Slika 5.2 Rezultati za 1000 jedinki i 50 generacija

Postavljanjem brojem jedinki u populaciji na 10000 najznačajnija je promjena u vremena izvođenja programa. Potrebno je otprilike 20-ak sekundi za jednu generaciju, što znači za 20 generacija je potrebno oko 7 minuta. Iz slike 5.3 je vidljivo kako su fluktuacije funkcije podobnosti minimalne, te funkcija podobnosti stagnira na 0.098 što je najmanja razlika za sva tri slučaja.

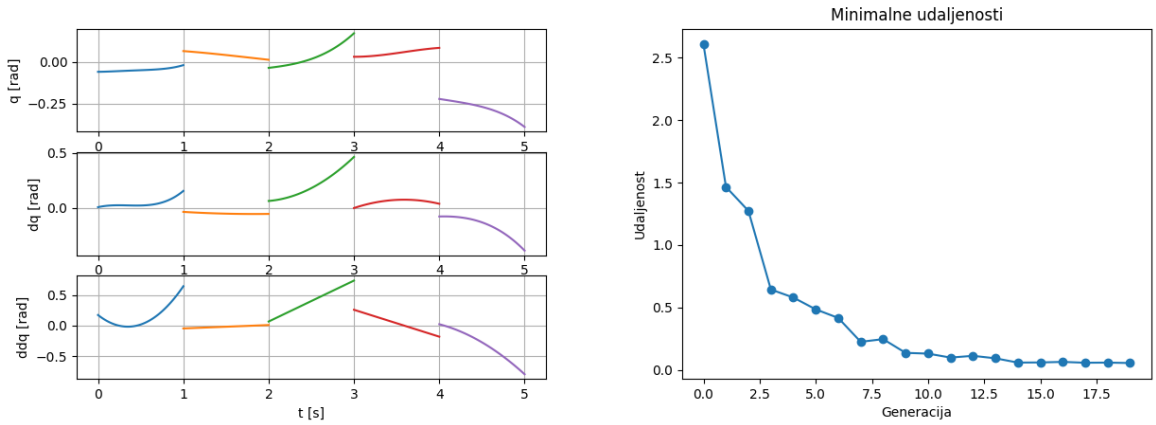


Slika 5.3 Rezultati za 10000 jedinki i 20 generacija

Za daljnje razmatranje rezultata, bit će odabrano 10000 jedinki i 10 generacija. Neće se razmatrati slučajevi za više od 10000 jedinki jer se program izvodi znatno sporije te nije praktično.

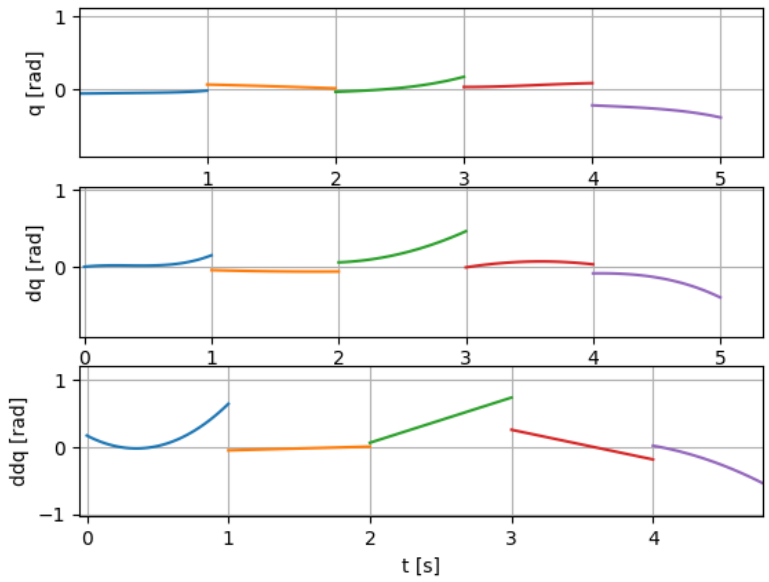
5.2 Granični uvjeti

Prethodno je spomenuto postavljanje graničnih uvjeta na rezultate križanja dvaju roditelja, odnosno sprječavanja da potomak postane nul-matrica. Proveden je algoritam bez graničnog uvjeta te je rezultat prikazan slikom 5.4.



Slika 5.4 Rezultati bez graničnih uvjeta

Na prvu izgleda kao da su rješenja dosta dobra, ali ako se podesi skala ordinate na interval -1 do 1, prikazano slikom 5.5, iz grafa za putanju je vidljivo kako je genetski algoritam pronašao najbolje rješenje u nuli. U realnosti bi to predstavljalo minimalne pomake robota.



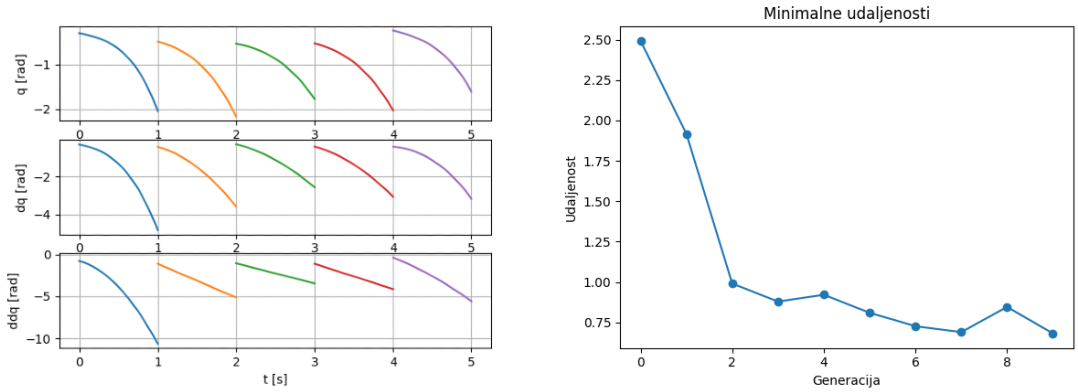
Slika 5.5 Rezultati bez graničnih uvjeta uz skaliranu ordinatu

Također se može ispisati dobivena matrica B i ona je oblika:

```
In[18]: B[fitness.index(min(fitness))]
Out[18]:
array([-0.06028181,  0.00502937,  0.08673064, -0.18223007,  0.13036242,
        0.06358135, -0.03804457, -0.02315551,  0.00928992, -0.03672226,
        0.0615794 ,  0.03343481,  0.11168696,  0.02962278, -0.00299406,
        0.12960198, -0.07341306, -0.22240143, -0.08008791,  0.01133155,
       -0.06105722, -0.03773234])
```

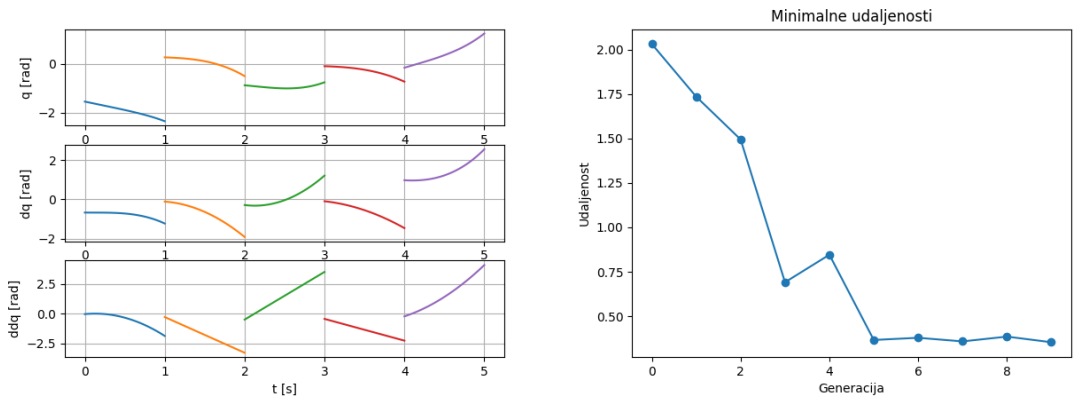
Bez postavljenih uvjeta, matrica B teži k nuli, te je potrebno implementirati uvjete.

Ako se postavi uvjet da je apsolutna vrijednost sume elemenata matrice B veća od 0.2 dobivaju se vrijednosti prikazane na slici 5.6. Odnosno, vidljivo je kako svi segmenti poprimaju isti oblik.



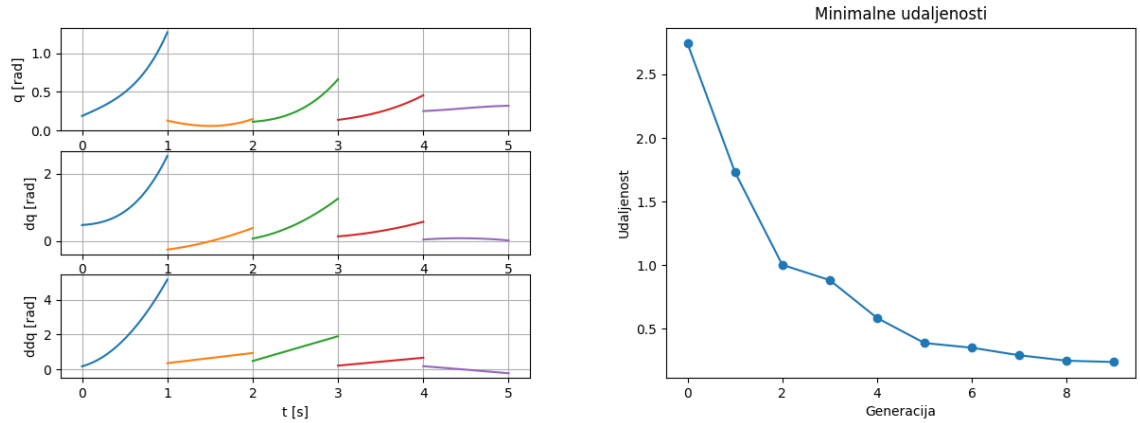
Slika 5.6 Rezultati s graničnim uvjetom postavljenim na 0.2

Daljnjim smanjivanjem uvjeta mogu se primijetiti raznolikosti segmentna, slika 5.7, no razlike segmenata su preveliki te se može dobiti i bolje.



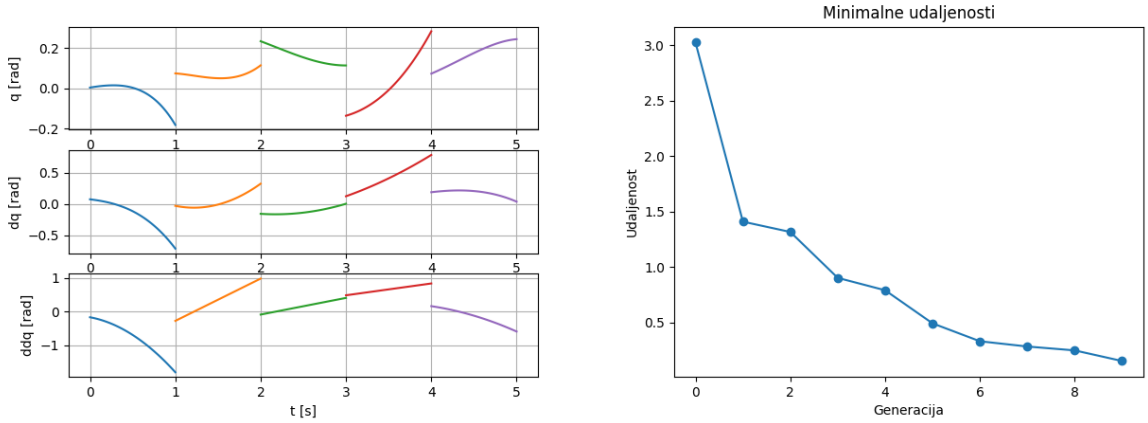
Slika 5.7 Rezultati s graničnim uvjetom postavljenim na 0.1

Kada bi se granični uvjet još smanjio na 0.05 dobili bi prikaz ilustriran na slici 5.8. Ovi rezultati su bolji, uz veće pomake zglobova i manje udaljenosti između segmenata.



Slika 5.8 Rezultati s graničnim uvjetom postavljenim na 0.05

Treba paziti da se granični uvjet ne postavi prenizak jer u suprotnom gubi svoj smisao. Granični uvjet postavljen na 0.005 prikazan je slikom 5.9, te je odmah vidljivo da se dolazi na istu situaciju kao i kada nema uvjeta, odnosno elementi B matrice teže ka nuli.

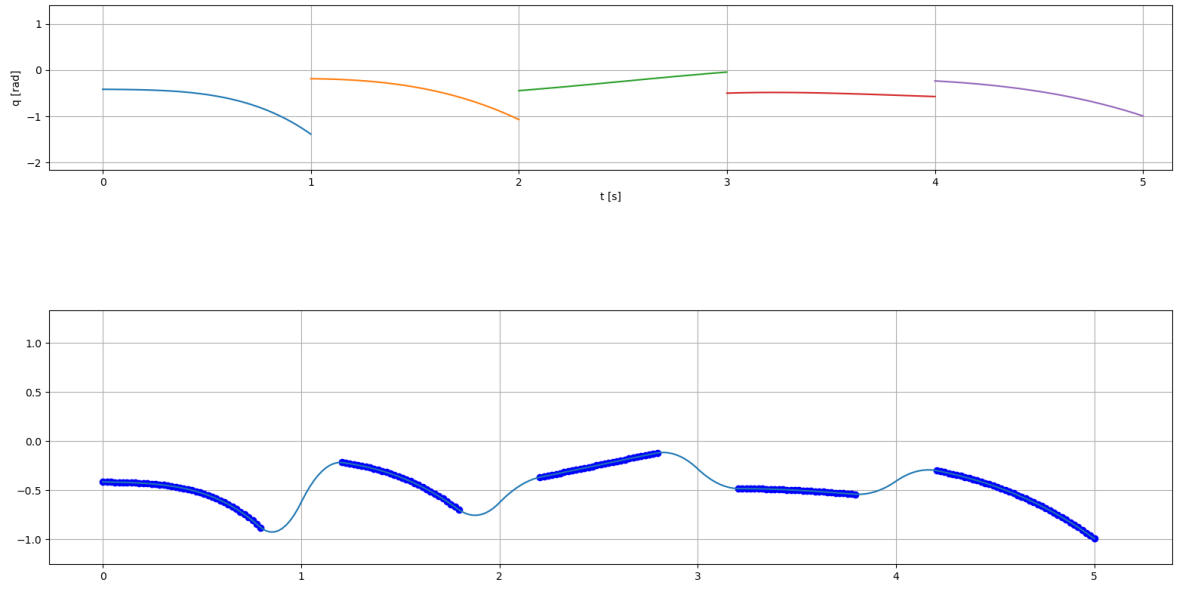


Slika 5.9 Rezultati s graničnim uvjetom postavljenim na 0.005

5.3 Interpolacija

Primjenom genetskog algoritma nije moguće dobiti savršeno poklapanje segmenata. Ipak, ako pogledamo prosječnu vrijednost razlike po generaciji vidimo kako se ta razlika može smanjiti. Kada je razlika dovoljno mala, može se primijeniti interpolacija kako bi se povezali svi

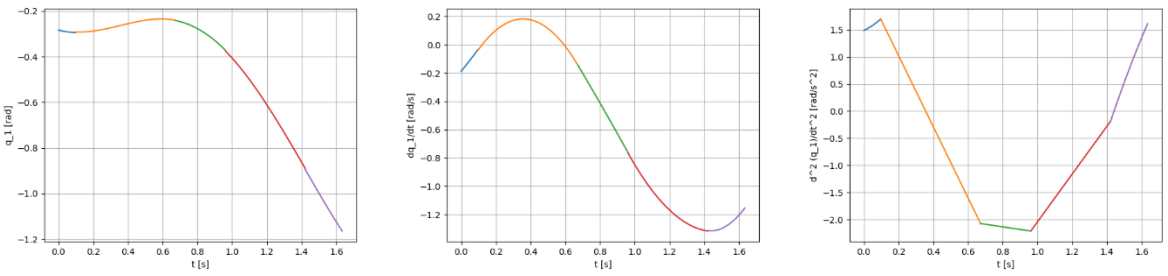
segmenti. Interpolacijom se zanemare prvih 10 i zadnjih 10 točke segmenta kako je prikazano slikom 5.10. Originalne točke segmenta koje nisu zanemarene su također iscrtane na donjem grafu. Ovakva rješenja sad imaju smisla i moguće ih je primijeniti na robotski manipulator.



Slika 5.10 Interpolacija segmenta

5.4 Rezultati tradicionalnog pristupa

Kako bi se rješenje genetskog algoritma moglo ocijeniti, napravljen je i algoritam za rješavanje ovog problema tradicionalnog pristupa. Tu se radi o iterativnom postupku nasumičnog generiranja matrica B sve dokle se ne pronađe dovoljno dobro rješenje. Rezultati tradicionalnog pristupa prikazan je slikom 5.11.



Slika 5.11 Rezultati tradicionalnog pristupa

Jasno je vidljivo kako su ovakvi rezultati znatno bolji od genetskog algoritma, ali nedostatak ovog pristupa je vrijeme izvršavanja. Za pronaći ovakve rezultate trebalo je 104.599.884 iteracija za prvi segment, 68.338.578 iteracija za drugi segment, 58.698.305 iteracija za treći segment i 17.617.284 iteracija za posljednji segment. Sveukupno je potrebno 249.254.051 iteracija za pronalazak matrica B koji odgovaraju prikazanim grafovima.

Prikazana rješenja su samo za jedan zglob. Za dobivanje rješenja za svih 6 zglobova cijeli postupak je potrebno ponoviti još pet puta. Ako je bilo potrebno 250 milijuna iteracija za pronaći jedno rješenje, za svih 6 će biti potrebno preko milijardu iteracija.

6. SIMULACIJA U ROBOTSTUDIO PROGRAMU

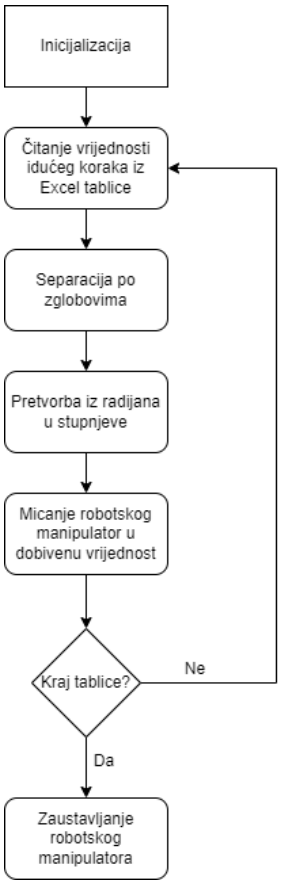
Robotski manipulator ABB IRB 120 je proizvod tvrtke ABB, te se koristeći njihov program RobotStudio može odraditi simulacija. RobotStudio dozvoljava programiranje robotskog manipulatora preko računala bez potrebe za gašenjem pogona. Ovo svojstvo dozvoljava programiranje, optimizaciju i treniranje robota bez narušavanja stanja pogona gdje se robot koristi [15].

Ovaj program podržava virtualno upravljanje, koje pruža veoma realistične simulacije upravljanja robotom u stvarnom postrojenju [15]. RobotStudio koristi RAPID programski jezik. RAPID je programski jezik koji se bazira na ARLA programskom jeziku, te je uveden 1994. godine od strane ABB tvrtke za svoje robote [16].

Kako bi se prebacili iz Python okruženja u RobotStudio potrebno je rezultate genetskog algoritma spremiti u Excel tablice koje će RobotStudio moći učitati. Imamo 5 segmenata i 50 točaka za svaki segment što daje 250 točaka po zglobu. Za simuliranje nije potrebno uzeti sve točke već je dovoljno svaku 10. točku, tako da je potrebno samo 25 točaka spremiti u tablicu za simuliranje. Slika 6.1 prikazuje rezultate spremljene u tablice koji će se simulirati. Jedan redak predstavlja jedan korak s vrijednostima svih 6 zglobova robota. Dijagram toka je dan slikom 6.2.

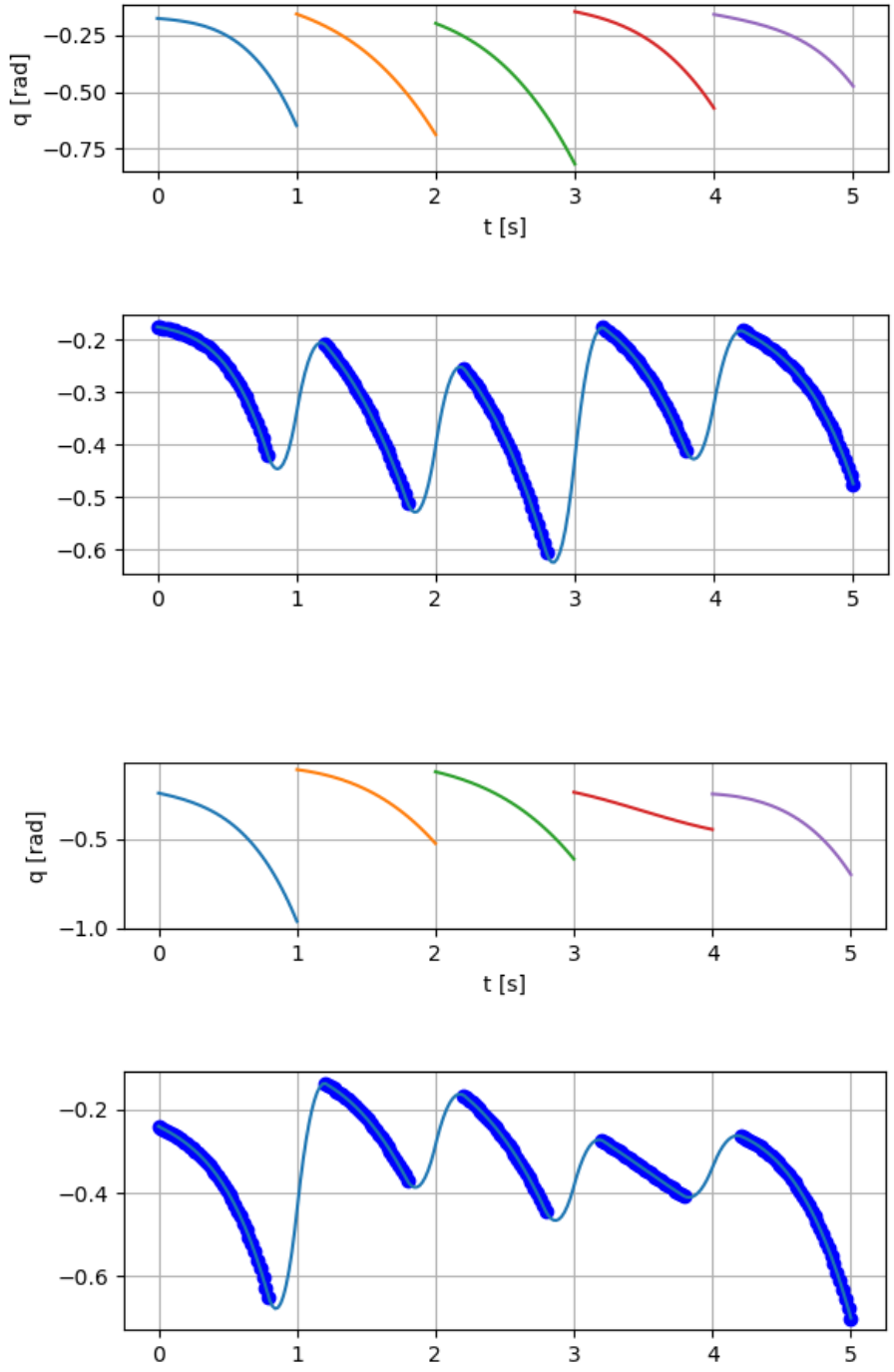
	A	B	C	D	E	F	G	H
1	0,1,2,3,4,5							
2	-0.414155438,-0.175375161,-0.240224005,-0.244463109,-0.24199634,-0.224722186							
3	-0.426985244,-0.189900269,-0.281130667,-0.296208142,-0.269125645,-0.267114019							
4	-0.476776733,-0.223179804,-0.34830695,-0.374184372,-0.287469995,-0.325097027							
5	-0.61136692,-0.294085214,-0.46421651,-0.501266335,-0.327866705,-0.388993163							
6	-0.890374551,-0.426420094,-0.65743136,-0.705512146,-0.430013857,-0.434646258							
7	-0.606693543,-0.334957628,-0.430548339,-0.464771838,-0.337991954,-0.220579675							
8	-0.215954899,-0.209278503,-0.137550289,-0.147171402,-0.172205054,-0.065322508							
9	-0.3010767,-0.282279692,-0.187659243,-0.187533587,-0.103472183,-0.193172669							
10	-0.460966296,-0.38294829,-0.265182229,-0.266187465,0.021701142,-0.357144775							
11	-0.713463022,-0.516797075,-0.375780404,-0.390441071,0.191556023,-0.568677424							
12	-0.609128369,-0.389429759,-0.274914344,-0.345266771,-0.00575597,-0.47546524							
13	-0.36365734,-0.257060814,-0.167045864,-0.287466867,-0.279076009,-0.381024565							
14	-0.280694637,-0.341054595,-0.233321342,-0.373556488,-0.34984599,-0.544636136							
15	-0.195360246,-0.459279922,-0.326718528,-0.492785774,-0.448590124,-0.742248234							
16	-0.115055707,-0.616514024,-0.452182707,-0.650324058,-0.576777935,-0.976920952							
17	-0.303780626,-0.396681582,-0.368997494,-0.469204928,-0.392865336,-0.644137441							
18	-0.482854456,-0.178672079,-0.275807213,-0.291697952,-0.231085295,-0.319929639							
19	-0.490534616,-0.229252147,-0.320949505,-0.360565445,-0.313848722,-0.392661804							
20	-0.512927926,-0.307616959,-0.367679028,-0.458457745,-0.429821736,-0.489668785							
21	-0.540392007,-0.420079837,-0.409926497,-0.588227128,-0.581911966,-0.612674344							
22	-0.385109803,-0.300090843,-0.325742919,-0.373478106,-0.338038984,-0.360775797							
23	-0.303497547,-0.184856366,-0.265458546,-0.196269429,-0.008687688,-0.121188973							
24	-0.405788765,-0.2187542,-0.308658334,-0.266407022,0.143332073,-0.087366119							
25	-0.552373291,-0.270828551,-0.38759642,-0.379059112,0.251825853,0.029956616							
26	-0.753852609,-0.35542289,-0.519039869,-0.562984554,0.290589393,0.318783102							

Slika 6.1 Tablica s točkama za simulaciju



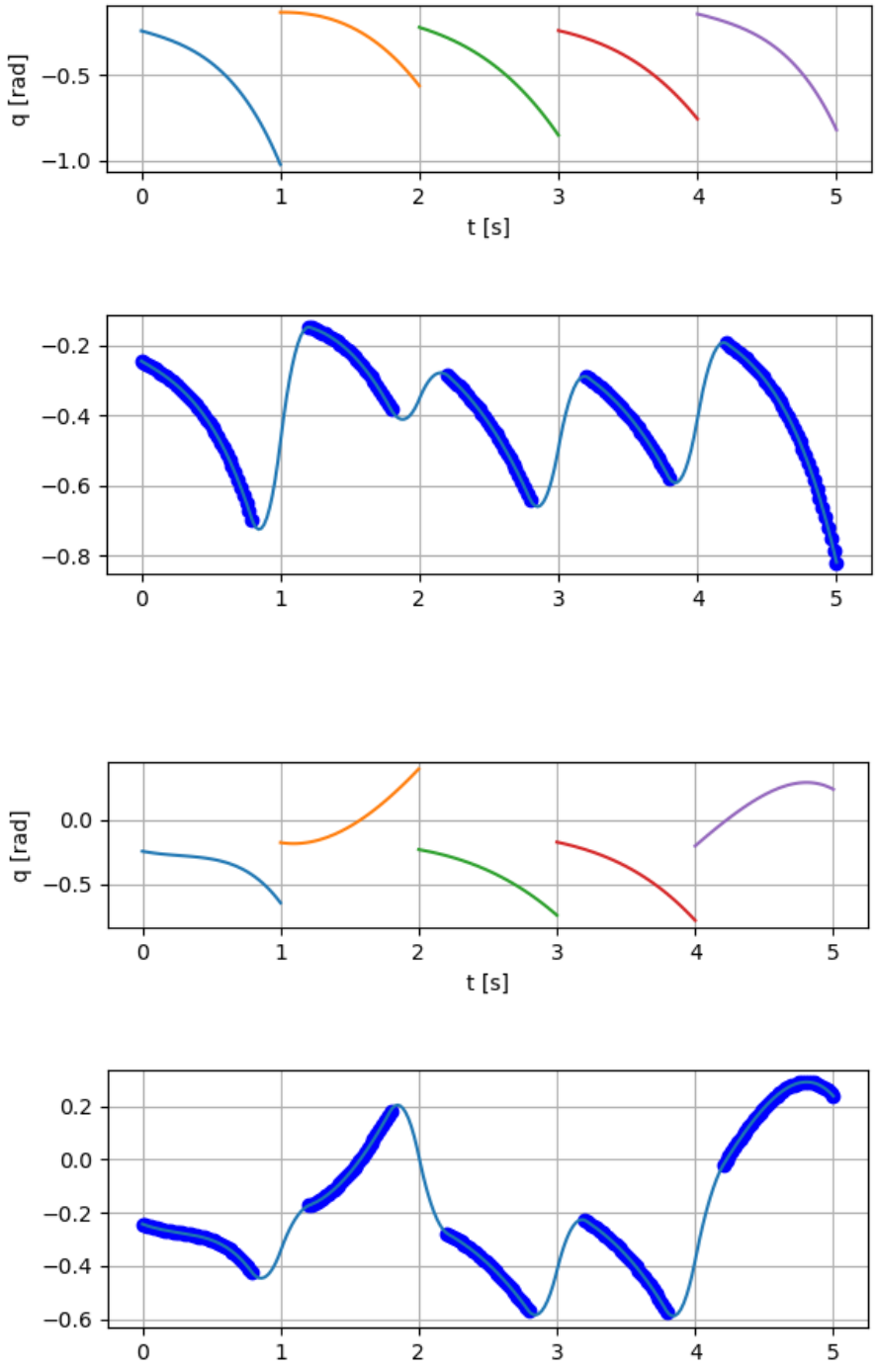
Slika 6.2 Dijagram toka simulacije u RobotStudiu

Slika 5.10 prikazuje putanju prvog zgloba, a slike 6.3 – 6.7 prikazuju putanje ostala pet zgloba koja će se simulirati.



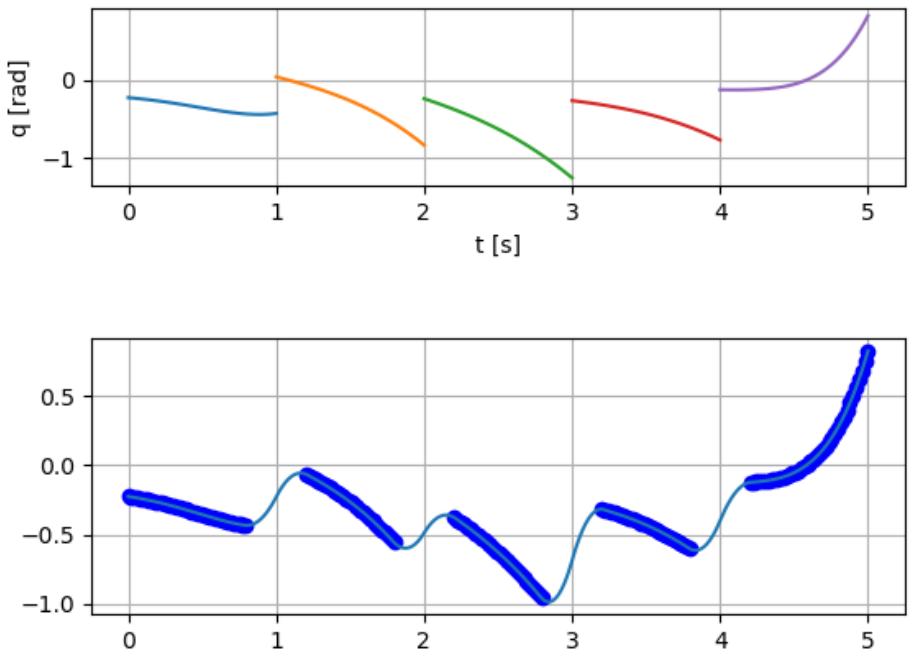
Slika 6.3 Putanja drugog i trećeg zgloba

Putanje između segmenata nekih zglobova vidljivo imaju strmije prijelaze. Prijelazi se mogu umanjiti tako da se smanji broj originalnih točaka između kojih će se odraditi interpolacija. Ipak, u ovom slučaju su unutar prihvatljivih granica te su ostavljeni kakvi jesu.



Slika 6.4 Putanja petog zgloba

Slika 6.5 prikazuje putanje za zadnji zglob ABB IRB 120 robotskog manipulatora. Interpolacija je uspješno izrađena za sve zglobove te se može krenuti u izradu simulacije.



Slika 6.5 Putanja zadnjeg zgloba

Nadalje će se objasniti programski kod napisan u RAPID okruženju.

```
Open "HOME:/Qovi.csv",infile\Read;

WHILE data<>EOF DO
    data:=ReadStr(infile);
    IF data=EOF THEN
        ClearIOBuff infile;
        Close infile;
        korak:=-1;
        GOTO next;
    ENDIF

    Incr korak;
    TPWrite "Korak - "+ValToStr(korak);
```

Tablicu je potrebno spremiti u „HOME“ folder RobotStudia. Tablica se zatim čita korak po korak, te se svaki korak ispisuje.

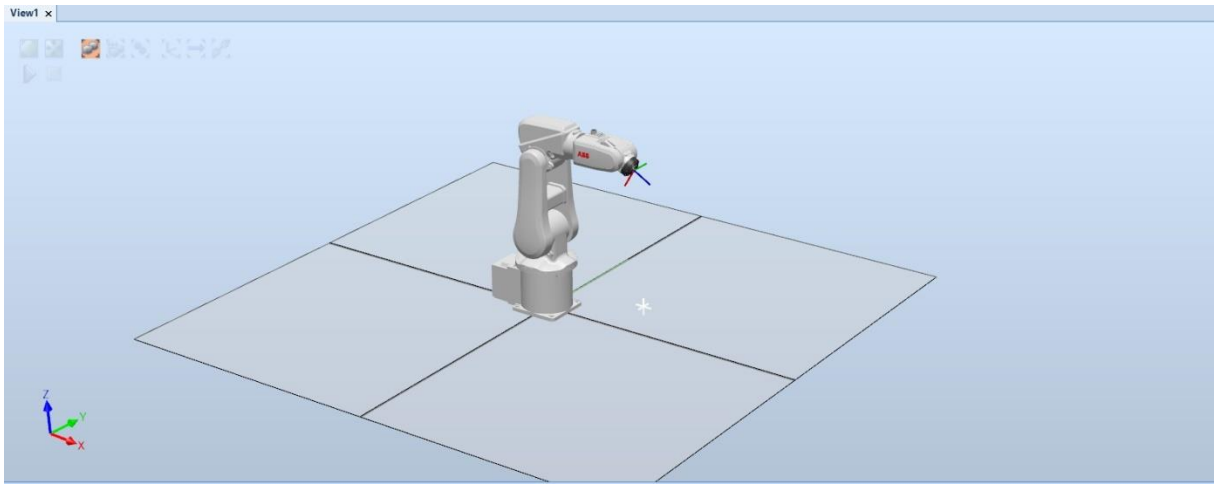
```
FUNC jointtarget StringToTarget(string value)
    VAR jointtarget tmpTarget;
    VAR jointtarget Target;
    VAR bool bResult;

    VAR num rax1;
    VAR num rax2;
    VAR num rax3;
    VAR num rax4;
```

```
        VAR num rax5;
        VAR num rax6;

        rax1:=StrFind(value,1,",");
        rax2:=StrFind(value,rax1+1,",");
        rax3:=StrFind(value,rax2+1,",");
        rax4:=StrFind(value,rax3+1,",");
        rax5:=StrFind(value,rax4+1,",");
        rax6:=StrFind(value,rax5+1,",");

        bResult:=StrToVal(StrPart(value,1,rax1-1),tmpTarget.robax.rax_1);
        bResult:=StrToVal(StrPart(value,rax1+1,rax2-rax1-
1),tmpTarget.robax.rax_2);
        bResult:=StrToVal(StrPart(value,rax2+1,rax3-rax2-
1),tmpTarget.robax.rax_3);
        bResult:=StrToVal(StrPart(value,rax3+1,rax4-rax3-
1),tmpTarget.robax.rax_4);
        bResult:=StrToVal(StrPart(value,rax4+1,rax5-rax4-
1),tmpTarget.robax.rax_5);
        bResult:=StrToVal(StrPart(value,rax5+1,rax6-rax5-
1),tmpTarget.robax.rax_6);

        TPWrite ValToStr(tmpTarget.robax);

        Target.robax.rax_1:=tmpTarget.robax.rax_1*57.2957795;
        Target.robax.rax_2:=tmpTarget.robax.rax_2*57.2957795;
        Target.robax.rax_3:=tmpTarget.robax.rax_3*57.2957795;
        Target.robax.rax_4:=tmpTarget.robax.rax_4*57.2957795;
        Target.robax.rax_5:=tmpTarget.robax.rax_5*57.2957795;
        Target.robax.rax_6:=tmpTarget.robax.rax_6*57.2957795;

        RETURN Target;

    ENDFUNC
```

Vrijednosti spremljene u tablicu su u radijanima, a RobotStudio radi s vrijednostima u stupnjevima, te je potrebno napraviti konverziju. Također u jedan redak u tablici su spremljene vrijednosti za svaki zglob, te ih je potrebno razdvojiti i primijeniti na odgovarajući zglob. Priložena funkcija radi konverziju iz radijana u stupnjeve i odvaja vrijednosti po zglobovima.

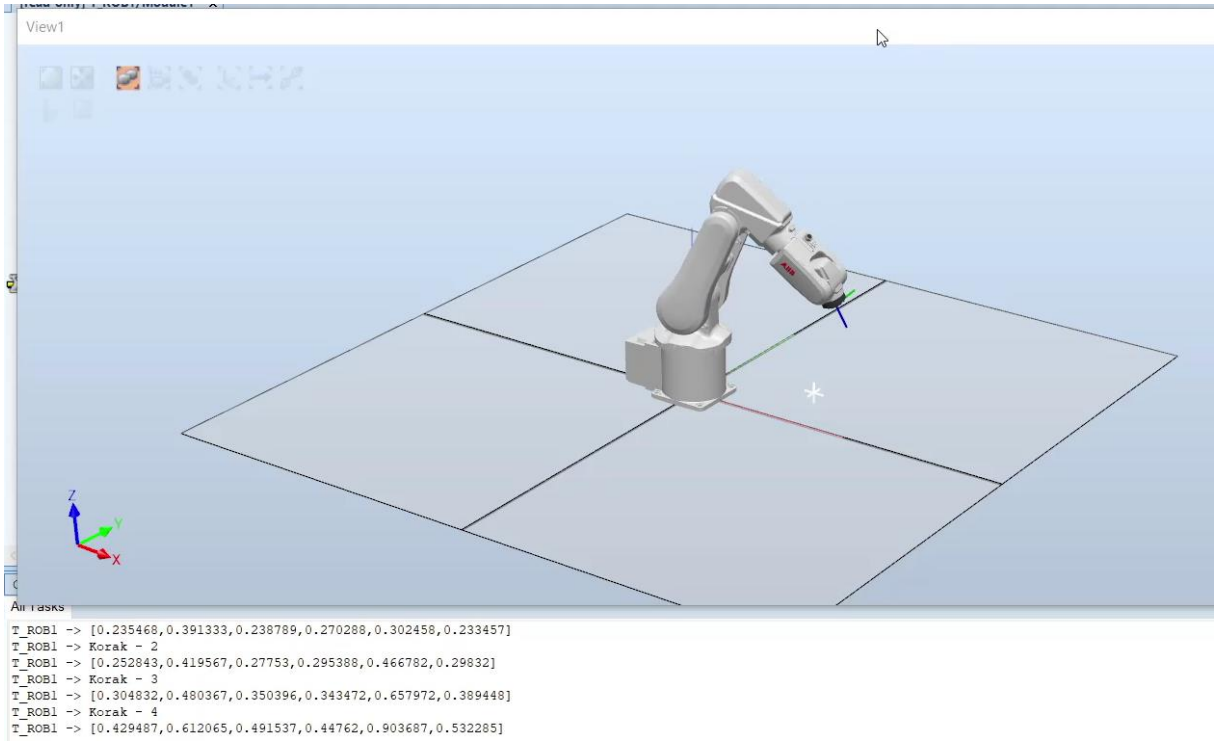
```
            pTarget:=StringToTarget(data);

            MoveAbsJ pTarget,v1000,z200,tool0\WObj:=wobj0;
!            IF korak>0 THEN
!                WaitRob\ZeroSpeed;
!            ENDIF
```

Na samom kraju potrebno je dobivene vrijednosti primijeniti na robot i izvršiti kretanje robota. Otvaranjem programa RobotStudio i odabir željenog robotskog manipulatora podešen je početni položaj prikazan slikom 6.6.



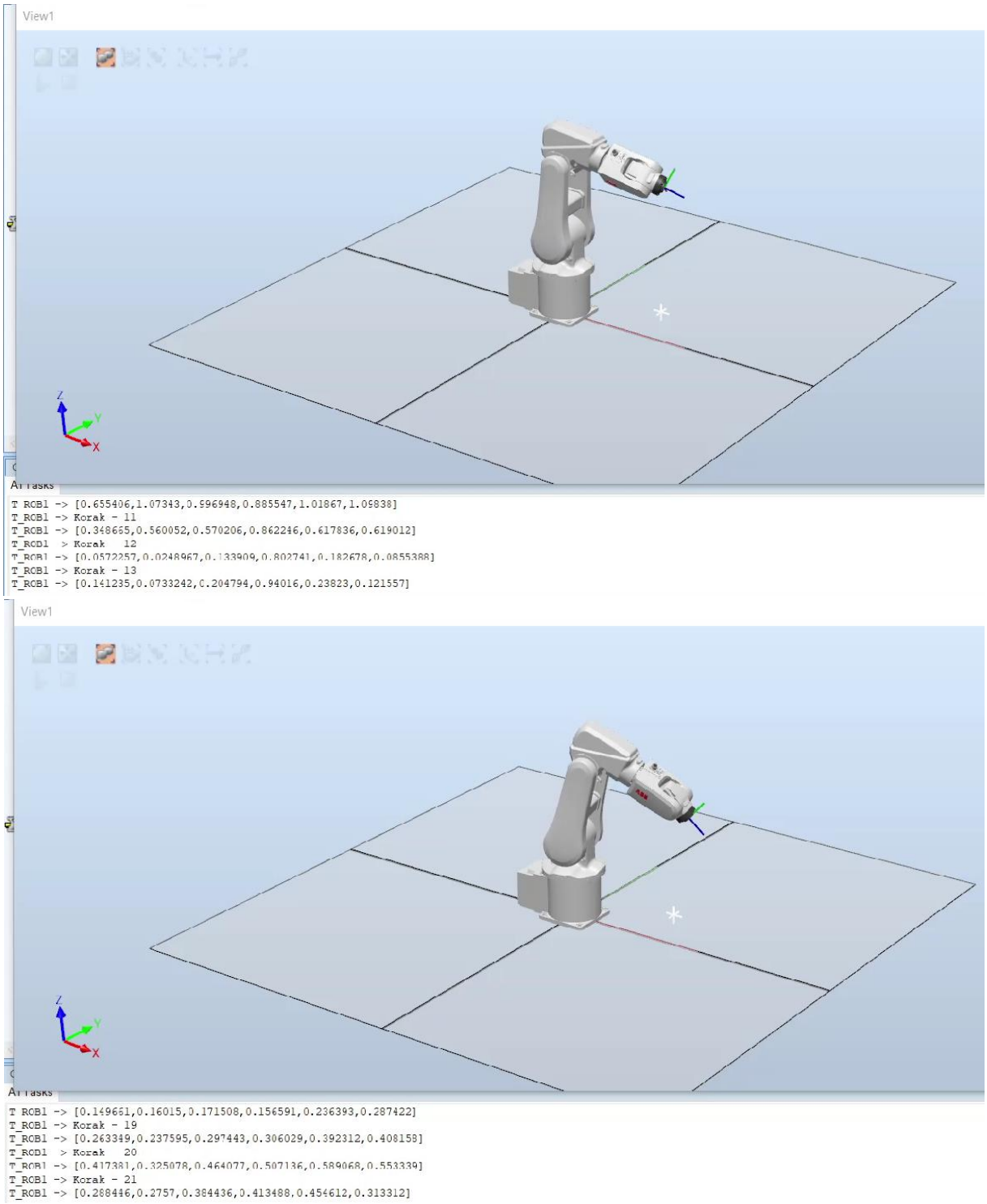
Slika 6.6 Prikaz RobotStudio sučelja i početni položaj robota

Učitavanjem vrijednosti iz tablice, odrađena je simulacije te su prikazani par koraka simulacije slikama 6.7 – 6.9. Na korisničkom sučelju se ispisuje u kojem koraku simulacije se robotski manipulator nalazi, te se također ispisuju vrijednosti svakog zgloba u tom koraku.



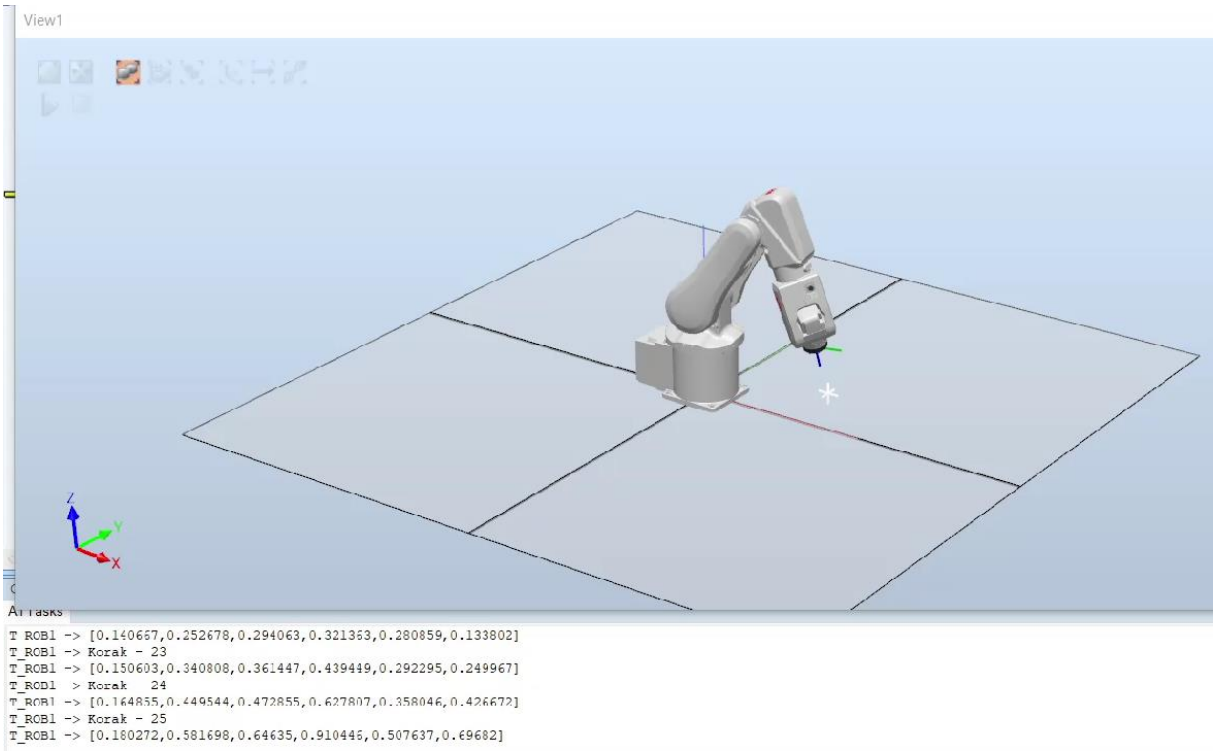
Slika 6.7 Četvrti korak simulacije

Nadalje, slika 6.8 prikazuje 13. i 21. korak izvođenja simulacije. Pokreti robota su jasni i precizni te zaključujemo kako se simulacija izvodi na očekivan način.



Slika 6.8 Trinaest i dvadesetprvi korak simulacije

Slika 6.9 prikazuje zadnji korak simulacije. Kada program očita zadnju vrijednost iz Excel tablica, robotski manipulator se zaustavlja u zadnjoj poziciji u kojoj se nalazio.



Slika 6.9 Zadnji korak simulacije

7. ZAKLJUČAK

Ovim diplomski radom provedeno je istraživanje na temu upravljanja robotskim manipulatorom. Upravljanje se vrši određivanjem parametara putanje koja je dobivena primjenom umjetne inteligencije i tradicionalnim pristupom.

Prvo je odrađen pregled primjene umjetne inteligencije na ovaj problem. Genetski algoritam nije uspio pronaći idealno rješenje, ali je uspio pronaći optimalno rješenje. Takvi rezultati su mogli i biti očekivani jer kada se govori o genetskim algoritmima, govori se o optimizaciji. Odnosno, nećemo dobiti idealno rješenje, ali će se dobiti dovoljno dobro – optimalno rješenje.

Bez primjene genetskog algoritma, razlike između segmenata su prevelike i vršenje interpolacije bi i dalje dovodilo do velikih trzaja robota. Smanjivanjem razlike između segmenata omogućava primjenu Bézierovih krivulja i povezivanje svih segmenata u jednu krivulju.

S druge strane, tradicionalni pristup je mnogo jednostavniji za razumjeti i za implementirati. Rješenja dobivena tradicionalnim pristup se mogu podesiti tako da se dobije potpuno povezani segmenti, odnosno idealno rješenje.

Prednost koju genetski algoritam ima iznad tradicionalnog pristupa je vrijeme izvođenja i računalni resursi. Rješenja tradicionalnog pristupa vrlo je teško dobiti na prosječnom stolnom računalu. Rješenja tradicionalnog pristupa priložena u ovom diplomskom radu su dobivena na superračunalu, te se algoritam izvodio nekoliko dana. S druge strane, rješenja genetskog algoritma moguće je dobiti na stolnom računalu, te ovisno o željenoj točnosti i podešenim parametrima, vrijeme izvođenja se može svesti unutar 30 minuta.

Ovaj faktor je velika razlika između dvije metode. Također, ovim pristupom se izbjegava računanje velikih i kompleksnih jednadžbi koje bi u suprotnom trebalo riješiti za određivanje parametara robotskih manipulatora s više zglobova.

8. LITERATURA

[1] „Robotics“, s Interneta, https://en.wikipedia.org/wiki/Robotics,19.07.2022.

[2] Car, Z.: „Uvod u robotiku“, s Interneta, https://moodle.srce.hr/2020-2021/pluginfile.php/4314256/mod_resource/content/0/1%20Uvod%20u%20robotiku.pdf, 19.07.2022.

[3] „Unraveling Degrees of Freedom and Robot Axis: What does it mean to have a multiple axis pick and place or multiple axis robot?“, s Interneta, https://motioncontrolsrobotics.com/unraveling-degrees-of-freedom-and-robot-axis-what-does-it-mean-to-have-a-multiple-axis-pick-and-place-or-multiple-axis-robot/, 19.07.2022.

[4] „IRB 120 - Industrial Robots Portfolio – ABB“, s interneta, https://new.abb.com/products/robotics/industrial-robots/irb-120, 30.08.2022.

[5] „Denavit-Hartenberg Parameters“, s Interneta, https://en.wikipedia.org/wiki/Denavit%E2%80%93Hartenberg_parameters, 30.08.2022.

[6] Chen, T., Lin, J., Wu, D., Wu, H.: „Research of Calibration Method for Industrial Robot Based on Error Model of Position.“, s interneta, https://doi.org/10.3390/app11031287, 30.08.2022.

[7] „What is a Robotic Manipulator“, s Interneta, https://robotsdoneright.com/Articles/what-is-a-robotic-manipulator.html, 19.07.2022.

[8] Car, Z.: „Planiranje trajektorije“, s Interneta, https://moodle.srce.hr/2020-2021/pluginfile.php/4298026/mod_resource/content/0/Vje%C5%BEba%203%20planiranje%20trajektorije.pdf, 19.07.2022.

[9] „What is the genetic algorithm“, s Interneta, https://www.mathworks.com/help/gads/what-is-the-genetic-algorithm.html, 20.07.2022.

[10] Z. Car, Uvod u Genetske algoritme

[11] „Genetic Algorithm“, s Interneta, https://en.wikipedia.org/wiki/Genetic_algorithm, 20.07.2022.

[12] Reguant, R.: „Roulette wheel selection in Python“, s Interneta, https://rocreguant.com/roulette-wheel-selection-python/2019/, 20.07.2022.

[13] „Bezier Curve“, s Interneta, https://en.wikipedia.org/wiki/B%C3%A9zier_curve, 27.07.2022.

[14] scipy.interpolate.BSpline, s Interneta , https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.BSpline.html, 27.07.2022.

[15] „RobotStudio“, s Interneta, https://new.abb.com/products/robotics/robotstudio, 26.07.2022.

[16] „RAPID“, s Interneta, https://en.wikipedia.org/wiki/RAPID, 26.07.2022.

9. POPIS SLIKA

10. SAŽETAK I KLJUČNE RIJEČI NA HRVATSKOM I ENGLESKOM JEZIKU

U ovom diplomskom radu izrađen je pregled primjene umjetne inteligencije u planiranju putanje robotskog manipulatora sa šest zglobova. Dan je opis planiranja putanje robotskog manipulatora. Izrađen je genetski algoritam za izračun parametara putanje. Uz genetski algoritam, problem je riješen i tradicionalnim pristupom bez primjene umjetne inteligencije. Uspoređena su rješenja i karakteristike dva algoritma te na samom kraju provedena je i simulacija.

Ključne riječi:

Umjetna inteligencija, robotika, upravljanje, robotski manipulator, robot, python, ABB IRB 120, RobotStudio, genetski algoritam, Ho-Cook algoritam, simulacija, interpolacija, Bezierove krivulje.

In this diploma thesis, an overview of the application of artificial intelligence in the planning of the path of a robot manipulator with six joints was made. A description of the path planning of the robotic manipulator is given. A genetic algorithm was developed for the calculation of path parameters. In addition to the genetic algorithm, the problem was also solved using a traditional approach without the use of artificial intelligence. The solutions and characteristics of the two algorithms were compared, and at the very end, a simulation was performed.

Key words:

Artificial intelligence, robotics, control, robot manipulator, robot, python, ABB IRB 120, RobotStudio, genetic algorithm, Ho-Cook algorithm, simulation, interpolation, Bezier curves.

11.DODATAK A – PROGRAMSKI KOD GENETSKOG ALGORITMA

```
import numpy as np
import matplotlib.pyplot as plt
from Funkcije import Plotanje
from Funkcije import Interpolacija_6
import pandas as pd
from datetime import datetime

def printf(*arg, **kwarg):
    timestamp = datetime.now().strftime("%H:%M:%S")
    print(timestamp, *arg, **kwarg)

try:
    from IPython import get_ipython
    get_ipython().magic('clear')
    get_ipython().magic('reset -f')
except:
    pass
plt.close('all')

Broj_Jedinki = 1000

B = np.random.uniform(-5, 5, (Broj_Jedinki, 22))

Elementi = np.split(B, Broj_Jedinki)
Redni_Broj_Elementa = [i for i in range(Broj_Jedinki)]
Novi_Elementi = []

t2 = 1
t3 = 1
t4 = 1
t5 = 1
t6 = 1

tq1 = np.linspace(0, 1)
tq2 = np.linspace(0, 1)
tq3 = np.linspace(0, 1)
tq4 = np.linspace(0, 1)
tq5 = np.linspace(0, 1)

t = np.array([tq1, tq2, tq3, tq4, tq5])

dist_q = []
dist_dq = []
dist_ddq = []

Provjera_Reprodukcije = []
q_ovi = []
minimalne_udaljenosti = []

Broj_Rekombinacija = int(0.9*Broj_Jedinki)
Broj_Reprodukcija = int(0.07*Broj_Jedinki)
Broj_Mutacija = int(0.03*Broj_Jedinki)

for Broj_Zglobova in range(6):
    printf('Zglob', Broj_Zglobova)
    for Generacija in range(15):
        printf("Generacija:", Generacija)
```

```
        for i in range(Broj_Jedinki):
            q1 = B[i,0] + B[i,1]*tq1 + B[i,2]*(tq1**2) + B[i,3]*(tq1**3) +
B[i,4]*(tq1**4)
            q2 = B[i,5] + B[i,6]*tq2 + B[i,7]*(tq2**2) + B[i,8]*(tq2**3)
            q3 = B[i,9] + B[i,10]*tq3 + B[i,11]*(tq3**2) + B[i,12]*(tq3**3)
            q4 = B[i,13] + B[i,14]*tq4 + B[i,15]*(tq4**2) +
B[i,16]*(tq4**3)
            q5 = B[i,17] + B[i,18]*tq5 + B[i,19]*(tq5**2) +
B[i,20]*(tq5**3) + B[i,21]*(tq5**4)

            dq1 = B[i,1] + 2*B[i,2]*(tq1) + 3*B[i,3]*(tq1**2) +
4*B[i,4]*(tq1**3)
            dq2 = B[i,6] + 2*B[i,7]*(tq2) + 3*B[i,8]*(tq2**2)
            dq3 = B[i,10] + 2*B[i,11]*(tq3) + 3*B[i,12]*(tq3**2)
            dq4 = B[i,14] + 2*B[i,15]*(tq4) + 3*B[i,16]*(tq4**2)
            dq5 = B[i,18] + 2*B[i,19]*(tq5) + 3*B[i,20]*(tq5**2) +
4*B[i,21]*(tq5**3)

            ddq1 = 2*B[i,2] + 6*B[i,3]*(tq1) + 12*B[i,4]*(tq1**2)
            ddq2 = 2*B[i,7] + 6*B[i,8]*(tq2)
            ddq3 = 2*B[i,11] + 6*B[i,12]*(tq3)
            ddq4 = 2*B[i,15] + 6*B[i,16]*(tq4)
            ddq5 = 2*B[i,19] + 6*B[i,20]*(tq5) + 12*B[i,21]*(tq5**2)

            d1_q = abs(q1[-1] - q2[0])
            d2_q = abs(q2[-1] - q3[0])
            d3_q = abs(q3[-1] - q4[0])
            d4_q = abs(q4[-1] - q5[0])
            q = [d1_q, d2_q, d3_q, d4_q]
            dist_q.append(q)

            d1_dq = abs(dq1[-1] - dq2[0])
            d2_dq = abs(dq2[-1] - dq3[0])
            d3_dq = abs(dq3[-1] - dq4[0])
            d4_dq = abs(dq4[-1] - dq5[0])
            dq = [d1_dq, d2_dq, d3_dq, d4_dq]
            dist_dq.append(dq)

            d1_ddq = abs(ddq1[-1] - ddq2[0])
            d2_ddq = abs(ddq2[-1] - ddq3[0])
            d3_ddq = abs(ddq3[-1] - ddq4[0])
            d4_ddq = abs(ddq4[-1] - ddq5[0])
            ddq = [d1_ddq, d2_ddq, d3_ddq, d4_ddq]
            dist_ddq.append(ddq)

        distances = np.hstack((dist_q, dist_dq, dist_ddq))

        Prosjek_Stupaca = distances.mean(1).tolist()

        fitness = Prosjek_Stupaca

        print('Minimum:', min(fitness))
        print('Maximum:', max(fitness))

        Fitness_Sum = sum(fitness)
        Vjerojatnost_Odabira = [x/Fitness_Sum for x in fitness]
        Vjerojatnost_Odabira = 1 - np.array(Vjerojatnost_Odabira)

        Suma_Vjerojatnosti = sum(Vjerojatnost_Odabira)
        Vjerojatnost_Odabira = Vjerojatnost_Odabira/Suma_Vjerojatnosti
```

```
        for i in range(Broj_Rekombinacija):

            Odabir1 = np.random.choice(Redni_Broj_Elementa,
p=Vjerojatnost_Odabira)
            Odabir2 = np.random.choice(Redni_Broj_Elementa,
p=Vjerojatnost_Odabira)

            while(Odabir1 == Odabir2):
                Odabir2 = np.random.choice(Redni_Broj_Elementa,
p=Vjerojatnost_Odabira)

            Rekombinacija = (Elementi[Odabir1] + Elementi[Odabir2])/2

            while (np.abs(np.mean(Rekombinacija)) < 0.05):
                Odabir1 = np.random.choice(Redni_Broj_Elementa,
p=Vjerojatnost_Odabira)
                Odabir2 = np.random.choice(Redni_Broj_Elementa,
p=Vjerojatnost_Odabira)

                while(Odabir1 == Odabir2):
                    Odabir1 = np.random.choice(Redni_Broj_Elementa,
p=Vjerojatnost_Odabira)
                    Odabir2 = np.random.choice(Redni_Broj_Elementa,
p=Vjerojatnost_Odabira)
                    ####
                Rekombinacija = (Elementi[Odabir1] + Elementi[Odabir2])/2

            Novi_Elementi.append(Rekombinacija)

        Provjera_Reprodukcije.clear()

        for i in range(Broj_Reprodukcija):

            Reprodukcija = np.random.choice(Redni_Broj_Elementa,
p=Vjerojatnost_Odabira)
            while (Reprodukcija in Provjera_Reprodukcije) == True:
                Reprodukcija = np.random.choice(Redni_Broj_Elementa,
p=Vjerojatnost_Odabira)

            Provjera_Reprodukcije.append(Reprodukcija)
            Novi_Elementi.append(Elementi[Reprodukcija])

        for i in range(Broj_Mutacija):
            Mutacija = np.random.choice(Redni_Broj_Elementa,
p=Vjerojatnost_Odabira)

            Matrica_Promjene  = Elementi[Mutacija].copy()
            Matrica_Mutacija = np.random.uniform(-5, 5, (1, 22))

            for j in range(5):

                Poz_Stupac = np.random.randint(0, 22)
                Matrica_Promjene[0][Poz_Stupac] =
Matrica_Mutacija[0][Poz_Stupac].copy()

            Novi_Elementi.append(Matrica_Promjene)

        Elementi.clear()
        Elementi = Novi_Elementi.copy()
        B = Elementi.copy()
        B = np.array(B)
```

```
            B = np.reshape(B, (Broj_Jedinki,22))

            Novi_Elementi.clear()
            dist_q.clear()
            dist_dq.clear()
            dist_ddq.clear()

            minimalne_udaljenosti.append(min(fitness))

        q1_int, q2_int, q3_int, q4_int, q5_int = Plotanje(B, fitness, t)
        q_int = np.vstack((q1_int, q2_int, q3_int, q4_int, q5_int))

        interp = Interpolacija_6(q_int, 10, t)

        q_ovi.append(interp[::10])

q_ovi = np.array(q_ovi)
q_ovi = np.around(q_ovi.T, 9)
pd.DataFrame(q_ovi).to_csv('Qovi.csv', index=False)
```

12. DODATAK C – FUNKCIJE

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.interpolate as interpolate

def Plotanje(Matrica, fitness_Matrice, t):

    t2 = t[1, -1]
    t3 = t[2, -1]
    t4 = t[3, -1]
    t5 = t[4, -1]

    Pronadi = fitness_Matrice.index(min(fitness_Matrice))

    q1 = Matrica[Pronadi,0] + Matrica[Pronadi,1]*t[0] +
Matrica[Pronadi,2]*(t[0]**2) + Matrica[Pronadi,3]*(t[0]**3) +
Matrica[Pronadi,4]*(t[0]**4)
    q2 = Matrica[Pronadi,5] + Matrica[Pronadi,6]*t[1] +
Matrica[Pronadi,7]*(t[1]**2) + Matrica[Pronadi,8]*(t[1]**3)
    q3 = Matrica[Pronadi,9] + Matrica[Pronadi,10]*t[2] +
Matrica[Pronadi,11]*(t[2]**2) + Matrica[Pronadi,12]*(t[2]**3)
    q4 = Matrica[Pronadi,13] + Matrica[Pronadi,14]*t[3] +
Matrica[Pronadi,15]*(t[3]**2) + Matrica[Pronadi,16]*(t[3]**3)
    q5 = Matrica[Pronadi,17] + Matrica[Pronadi,18]*t[4] +
Matrica[Pronadi,19]*(t[4]**2) + Matrica[Pronadi,20]*(t[4]**3) +
Matrica[Pronadi,21]*(t[4]**4)

    dq1 = Matrica[Pronadi,1] + 2*Matrica[Pronadi,2]*(t[0]) +
3*Matrica[Pronadi,3]*(t[0]**2) + 4*Matrica[Pronadi,4]*(t[0]**3)
    dq2 = Matrica[Pronadi,6] + 2*Matrica[Pronadi,7]*(t[1]) +
3*Matrica[Pronadi,8]*(t[1]**2)
    dq3 = Matrica[Pronadi,10] + 2*Matrica[Pronadi,11]*(t[2]) +
3*Matrica[Pronadi,12]*(t[2]**2)
    dq4 = Matrica[Pronadi,14] + 2*Matrica[Pronadi,15]*(t[3]) +
3*Matrica[Pronadi,16]*(t[3]**2)
    dq5 = Matrica[Pronadi,18] + 2*Matrica[Pronadi,19]*(t[4]) +
3*Matrica[Pronadi,20]*(t[4]**2) + 4*Matrica[Pronadi,21]*(t[4]**3)

    ddq1 = 2*Matrica[Pronadi,2] + 6*Matrica[Pronadi,3]*(t[0]) +
12*Matrica[Pronadi,4]*(t[0]**2)
    ddq2 = 2*Matrica[Pronadi,7] + 6*Matrica[Pronadi,8]*(t[1])
    ddq3 = 2*Matrica[Pronadi,11] + 6*Matrica[Pronadi,12]*(t[2])
    ddq4 = 2*Matrica[Pronadi,15] + 6*Matrica[Pronadi,16]*(t[3])
    ddq5 = 2*Matrica[Pronadi,19] + 6*Matrica[Pronadi,20]*(t[4]) +
12*Matrica[Pronadi,21]*(t[4]**2)

    plt.figure()
    plt.subplot(3,1,1)
    plt.plot(t[0], q1)
    plt.plot(t[1]+t2, q2)
    plt.plot(t[2]+t2+t3, q3)
    plt.plot(t[3]+t2+t3+t4, q4)
    plt.plot(t[4]+t2+t3+t4+t5, q5)
    plt.ylabel('q [rad]')
    plt.xlabel('t [s]')
    plt.grid()

    plt.subplot(3,1,2)
    plt.plot(t[0], dq1)
```

```
    plt.plot(t[1]+t2, dq2)
    plt.plot(t[2]+t2+t3, dq3)
    plt.plot(t[3]+t2+t3+t4, dq4)
    plt.plot(t[4]+t2+t3+t4+t5, dq5)
    plt.ylabel('dq [rad]')
    plt.xlabel('t [s]')
    plt.grid()

    plt.subplot(3,1,3)
    plt.plot(t[0], ddq1)
    plt.plot(t[1]+t2, ddq2)
    plt.plot(t[2]+t2+t3, ddq3)
    plt.plot(t[3]+t2+t3+t4, ddq4)
    plt.plot(t[4]+t2+t3+t4+t5, ddq5)
    plt.ylabel('ddq [rad]')
    plt.xlabel('t [s]')
    plt.grid()

    return q1, q2, q3, q4, q5

def Interpolacija(q, Tocke, t):

    t2 = t[1, -1]
    t3 = t[2, -1]
    t4 = t[3, -1]
    t5 = t[4, -1]

    #-------------------------------------------------------------------------
-------
    q1 = q[0,:]
    q2 = q[1,:]
    q3 = q[2,:]
    q4 = q[3,:]
    q5 = q[4,:]

    tq1 = t[0,:]
    tq2 = t[1,:]
    tq3 = t[2,:]
    tq4 = t[3,:]
    tq5 = t[4,:]

    plt.figure()
    plt.subplot(2,1,1)
    plt.plot(tq1, q1)
    plt.plot(tq2+t2, q2)
    plt.plot(tq3+t2+t3, q3)
    plt.plot(tq4+t2+t3+t4, q4)
    plt.plot(tq5+t2+t3+t4+t5, q5)
    plt.ylabel('q [rad]')
    plt.xlabel('t [s]')
    plt.grid()
    #-------------------------------------------------------------------------
-------


    tq1 = t[0,:]
    tq2 = t[1,:] + 1
    tq3 = t[2,:] + 2
    tq4 = t[3,:] + 3
    tq5 = t[4,:] + 4
```

```
    tq1 = np.reshape(tq1[0:(50 - Tocke)], (50 - Tocke,1))
    tq2 = np.reshape(tq2[Tocke:(50 - Tocke)], (50 - 2*Tocke,1))
    tq3 = np.reshape(tq3[Tocke:(50 - Tocke)], (50 - 2*Tocke,1))
    tq4 = np.reshape(tq4[Tocke:(50 - Tocke)], (50 - 2*Tocke,1))
    tq5 = np.reshape(tq5[Tocke:50], (50 - Tocke,1))

    q1 = np.reshape(q[0, 0:(50 - Tocke)], (50 - Tocke,1))
    q2 = np.reshape(q[1, Tocke:(50 - Tocke)], (50 - 2*Tocke,1))
    q3 = np.reshape(q[2, Tocke:(50 - Tocke)], (50 - 2*Tocke,1))
    q4 = np.reshape(q[3, Tocke:(50 - Tocke)], (50 - 2*Tocke,1))
    q5 = np.reshape(q[4, Tocke:50], (50 - Tocke,1))

    q = np.vstack((q1, q2, q3, q4, q5))
    tq = np.vstack((tq1, tq2, tq3, tq4, tq5))
    y = q
    x = tq

    t, c, k = interpolate.splrep(x, y, s=0, k=2)

    N = 250
    xmin, xmax = x.min(), x.max()
    xx = np.linspace(xmin, xmax, N)
    spline = interpolate.BSpline(t, c, k, extrapolate=False)

    plt.subplot(2,1,2)
    plt.plot(x, y, 'bo')
    plt.plot(xx, spline(xx))
    plt.grid()
    plt.show()

    plt.figure()
    plt.plot(xx, spline(xx))
    plt.grid()
    plt.show()

def Interpolacija_6(q, Tocke, t):

    tq1 = t[0,:]
    tq2 = t[1,:] + 1
    tq3 = t[2,:] + 2
    tq4 = t[3,:] + 3
    tq5 = t[4,:] + 4

    tq1 = np.reshape(tq1[0:(50 - Tocke)], (50 - Tocke,1))
    tq2 = np.reshape(tq2[Tocke:(50 - Tocke)], (50 - 2*Tocke,1))
    tq3 = np.reshape(tq3[Tocke:(50 - Tocke)], (50 - 2*Tocke,1))
    tq4 = np.reshape(tq4[Tocke:(50 - Tocke)], (50 - 2*Tocke,1))
    tq5 = np.reshape(tq5[Tocke:50], (50 - Tocke,1))

    q1 = np.reshape(q[0, 0:(50 - Tocke)], (50 - Tocke,1))
    q2 = np.reshape(q[1, Tocke:(50 - Tocke)], (50 - 2*Tocke,1))
    q3 = np.reshape(q[2, Tocke:(50 - Tocke)], (50 - 2*Tocke,1))
    q4 = np.reshape(q[3, Tocke:(50 - Tocke)], (50 - 2*Tocke,1))
    q5 = np.reshape(q[4, Tocke:50], (50 - Tocke,1))

    q = np.vstack((q1, q2, q3, q4, q5))
    tq = np.vstack((tq1, tq2, tq3, tq4, tq5))
    y = q
    x = tq

    t, c, k = interpolate.splrep(x, y, s=0, k=2)
```

```
N = 250
xmin, xmax = x.min(), x.max()
xx = np.linspace(xmin, xmax, N)
spline = interpolate.BSpline(t, c, k, extrapolate=False)

plt.subplot(2,1,2)
plt.plot(x, y, 'bo')
plt.plot(xx, spline(xx))
plt.grid()
plt.show()

return spline(xx)
```

13.DODATAK C – PROGRAMSKI KOD TRADICIONALNOG PRISTUPA

```
import sympy as sym
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from sympy import simplify
from sympy.solvers import solve
import uuid
uuid_=uuid.uuid4()
file_out = open(str(uuid_)+".log","w")
t = np.random.rand(1, 5)
t = np.around(t, 3)
file_out.write('t = '+str(t)+'\n')
t_uk = t.sum()
file_out.write('t_uk = '+str(t_uk)+'\n')

t2 = float(t[0,0])
t3 = float(t[0,1])
t4 = float(t[0,2])
t5 = float(t[0,3])
t6 = float(t[0,4])

tq1 = np.linspace(0, t2)
tq2 = np.linspace(0, t3)
tq3 = np.linspace(0, t4)
tq4 = np.linspace(0, t5)
tq5 = np.linspace(0, t6)

file_out.write(str(t2)+"\n")
file_out.write(str(t3)+"\n")
file_out.write(str(t4)+"\n")
file_out.write(str(t5)+"\n")
file_out.write(str(t6)+"\n")

preciznost = 2

B1_1 = np.random.randint(-1000, 1000, (1, 5))/1000
B2_1 = np.random.randint(-1000, 1000, (1, 4))/1000
B3_1 = np.random.randint(-1000, 1000, (1, 4))/1000
B4_1 = np.random.randint(-1000, 1000, (1, 4))/1000
B5_1 = np.random.randint(-1000, 1000, (1, 5))/1000

q1_1 = B1_1[0,0] + B1_1[0,1]*tq1 + B1_1[0,2]*(tq1**2) + B1_1[0,3]*(tq1**3)
+ B1_1[0,4]*(tq1**4)
q1_2 = B2_1[0,0] + B2_1[0,1]*tq2 + B2_1[0,2]*(tq2**2) + B2_1[0,3]*(tq2**3)
q1_3 = B3_1[0,0] + B3_1[0,1]*tq3 + B3_1[0,2]*(tq3**2) + B3_1[0,3]*(tq3**3)
q1_4 = B4_1[0,0] + B4_1[0,1]*tq4 + B4_1[0,2]*(tq4**2) + B4_1[0,3]*(tq4**3)
q1_5 = B5_1[0,0] + B5_1[0,1]*tq5 + B5_1[0,2]*(tq5**2) + B5_1[0,3]*(tq5**3)
+ B5_1[0,4]*(tq5**4)

dq1_1 = B1_1[0,1] + 2*B1_1[0,2]*(tq1) + 3*B1_1[0,3]*(tq1**2) +
4*B1_1[0,4]*(tq1**3)
dq1_2 = B2_1[0,1] + 2*B2_1[0,2]*(tq2) + 3*B2_1[0,3]*(tq2**2)
dq1_3 = B3_1[0,1] + 2*B3_1[0,2]*(tq3) + 3*B3_1[0,3]*(tq3**2)
dq1_4 = B4_1[0,1] + 2*B4_1[0,2]*(tq4) + 3*B4_1[0,3]*(tq4**2)
```

```
dq1_5 = B5_1[0,1] + 2*B5_1[0,2]*(tq5) + 3*B5_1[0,3]*(tq5**2) +
4*B5_1[0,4]*(tq5**3)

ddq1_1 = 2*B1_1[0,2] + 6*B1_1[0,3]*(tq1) + 12*B1_1[0,4]*(tq1**2)
ddq1_2 = 2*B2_1[0,2] + 6*B2_1[0,3]*(tq2)
ddq1_3 = 2*B3_1[0,2] + 6*B3_1[0,3]*(tq3)
ddq1_4 = 2*B4_1[0,2] + 6*B4_1[0,3]*(tq4)
ddq1_5 = 2*B5_1[0,2] + 6*B5_1[0,3]*(tq5) + 12*B5_1[0,4]*(tq5**2)

i = 0
while (np.around(abs(ddq1_1[-1] - ddq1_2[0]), preciznost) != 0):
    B2_1 = np.random.randint(-2000, 2000, (1, 4))/1000
    q1_2 = B2_1[0,0] + B2_1[0,1]*tq2 + B2_1[0,2]*(tq2**2) +
B2_1[0,3]*(tq2**3)

    if(np.around(abs(q1_1[-1] - q1_2[0]), preciznost) == 0):
        dq1_2 = B2_1[0,1] + 2*B2_1[0,2]*(tq2) + 3*B2_1[0,3]*(tq2**2)

        if(np.around(abs(dq1_1[-1] - dq1_2[0]), preciznost) == 0):
            ddq1_2 = 2*B2_1[0,2] + 6*B2_1[0,3]*(tq2)
            #file_out.write(str(np.around(abs(ddq1_1[-1] -
ddq1_2[0]))+str(preciznost)))

    i = i + 1
    #break

file_out.write('Prvi segment završen u '+str(i)+'iteracija.\n')

i = 0
while (np.around(abs(ddq1_2[-1] - ddq1_3[0]), preciznost) != 0):
    B3_1 = np.random.randint(-2000, 2000, (1, 4))/1000
    q1_3 = B3_1[0,0] + B3_1[0,1]*tq3 + B3_1[0,2]*(tq3**2) +
B3_1[0,3]*(tq3**3)

    if(np.around(abs(q1_2[-1] - q1_3[0]), preciznost) == 0):
        dq1_3 = B3_1[0,1] + 2*B3_1[0,2]*(tq3) + 3*B3_1[0,3]*(tq3**2)

        if(np.around(abs(dq1_2[-1] - dq1_3[0]), preciznost) == 0):
            ddq1_3 = 2*B3_1[0,2] + 6*B3_1[0,3]*(tq3)

    i = i + 1
    #break

file_out.write('Drugi segment završen u'+str(i)+'iteracija.\n')

i = 0
while (np.around(abs(ddq1_3[-1] - ddq1_4[0]), preciznost) != 0):
    B4_1 = np.random.randint(-2000, 2000, (1, 4))/1000
    q1_4 = B4_1[0,0] + B4_1[0,1]*tq4 + B4_1[0,2]*(tq4**2) +
B4_1[0,3]*(tq4**3)

    if(np.around(abs(q1_3[-1] - q1_4[0]), preciznost) == 0):
        dq1_4 = B4_1[0,1] + 2*B4_1[0,2]*(tq4) + 3*B4_1[0,3]*(tq4**2)

        if(np.around(abs(dq1_3[-1] - dq1_4[0]), preciznost) == 0):
            ddq1_4 = 2*B4_1[0,2] + 6*B4_1[0,3]*(tq4)

    i = i + 1
    #break

file_out.write('Treci segment završen u'+str(i)+'iteracija.\n')
```

```
i = 0
while (np.around(abs(ddq1_4[-1] - ddq1_5[0]), preciznost) != 0):
    B5_1 = np.random.randint(-2000, 2000, (1, 5))/1000
    q1_5 = B5_1[0,0] + B5_1[0,1]*tq5 + B5_1[0,2]*(tq5**2) +
B5_1[0,3]*(tq5**3) + B5_1[0,4]*(tq5**4)

    if(np.around(abs(q1_4[-1] - q1_5[0]), preciznost) == 0):
        dq1_5 = B5_1[0,1] + 2*B5_1[0,2]*(tq5) + 3*B5_1[0,3]*(tq5**2) +
4*B5_1[0,4]*(tq5**3)

        if(np.around(abs(dq1_4[-1] - dq1_5[0]), preciznost) == 0):
            ddq1_5 = 2*B5_1[0,2] + 6*B5_1[0,3]*(tq5) +
12*B5_1[0,4]*(tq5**2)

    i = i + 1
    #break

file_out.write('Zadnji segment završen u'+str(i)+'iteracija.\n')

plt.plot(tq1, q1_1)
plt.plot(tq2+t2, q1_2)
plt.plot(tq3+t2+t3, q1_3)
plt.plot(tq4+t2+t3+t4, q1_4)
plt.plot(tq5+t2+t3+t4+t5, q1_5)
plt.ylabel('q_1 [rad]')
plt.xlabel('t [s]')
plt.grid()
plt.savefig(str(uuid_)+"-q_1.png")

plt.figure()
plt.plot(tq1, dq1_1)
plt.plot(tq2+t2, dq1_2)
plt.plot(tq3+t2+t3, dq1_3)
plt.plot(tq4+t2+t3+t4, dq1_4)
plt.plot(tq5+t2+t3+t4+t5, dq1_5)
plt.ylabel('dq_1/dt [rad/s]')
plt.xlabel('t [s]')
plt.grid()
plt.savefig(str(uuid_)+"-dq_1.png")

plt.figure()
plt.plot(tq1, ddq1_1)
plt.plot(tq2+t2, ddq1_2)
plt.plot(tq3+t2+t3, ddq1_3)
plt.plot(tq4+t2+t3+t4, ddq1_4)
plt.plot(tq5+t2+t3+t4+t5, ddq1_5)
plt.ylabel('d^2 (q_1)/dt^2 [rad/s^2]')
plt.xlabel('t [s]')
plt.grid()
plt.savefig(str(uuid_)+"-ddq_1.png")
```

14.DODATAK D – PROGRAMSKI KOD ZA SIMULIRANJE U PROGRAMU ROBOTSTUDIO

```
MODULE Module1

    PROC main()

        VAR string data;
        VAR jointtarget pTarget;
        VAR string text;
        VAR iodev infile;
        VAR num korak;
        korak:=-1;

        Open "HOME:/Qovi.csv",infile\Read;

        WHILE data<>EOF DO
            data:=ReadStr(infile);
            IF data=EOF THEN
                ClearIOBuff infile;
                Close infile;
                korak:=-1;
                GOTO next;
            ENDIF

            Incr korak;
            TPWrite "Korak - "+ValToStr(korak);

            pTarget:=StringToTarget(data);

            MoveAbsJ pTarget,v1000,z200,tool0\WObj:=wobj0;
!            IF korak>0 THEN
!                WaitRob\ZeroSpeed;
!            ENDIF
        ENDWHILE
        next:

    ENDPROC

    FUNC jointtarget StringToTarget(string value)
        VAR jointtarget tmpTarget;
        VAR jointtarget Target;
        VAR bool bResult;

        VAR num rax1;
        VAR num rax2;
        VAR num rax3;
        VAR num rax4;
        VAR num rax5;
        VAR num rax6;

        rax1:=StrFind(value,1,",");
        rax2:=StrFind(value,rax1+1,",");
        rax3:=StrFind(value,rax2+1,",");
        rax4:=StrFind(value,rax3+1,",");
        rax5:=StrFind(value,rax4+1,",");
        rax6:=StrFind(value,rax5+1,",");
```

```
        bResult:=StrToVal(StrPart(value,1,rax1-1),tmpTarget.robax.rax_1);
        bResult:=StrToVal(StrPart(value,rax1+1,rax2-rax1-
1),tmpTarget.robax.rax_2);
        bResult:=StrToVal(StrPart(value,rax2+1,rax3-rax2-
1),tmpTarget.robax.rax_3);
        bResult:=StrToVal(StrPart(value,rax3+1,rax4-rax3-
1),tmpTarget.robax.rax_4);
        bResult:=StrToVal(StrPart(value,rax4+1,rax5-rax4-
1),tmpTarget.robax.rax_5);
        bResult:=StrToVal(StrPart(value,rax5+1,rax6-rax5-
1),tmpTarget.robax.rax_6);

        TPWrite ValToStr(tmpTarget.robax);

        Target.robax.rax_1:=tmpTarget.robax.rax_1*57.2957795;
        Target.robax.rax_2:=tmpTarget.robax.rax_2*57.2957795;
        Target.robax.rax_3:=tmpTarget.robax.rax_3*57.2957795;
        Target.robax.rax_4:=tmpTarget.robax.rax_4*57.2957795;
        Target.robax.rax_5:=tmpTarget.robax.rax_5*57.2957795;
        Target.robax.rax_6:=tmpTarget.robax.rax_6*57.2957795;

        RETURN Target;

    ENDFUNC

ENDMODULE
```