Financira Evropska unija
NextGenerationEU

DIGITOP

NAČRT ZA OKREVANJE IN ODPORNOST

Program

Digitalna transformacija robotiziranih tovarn prihodnosti

DIGITOP

Rezultat D2.1

Optimizacija in generiranje avtonomnega gibanja robota na nizki (trajektorije) in srednji ravni (posploševanje veščin)

Projekt/RRP		RRp 2: Industrijske raziskave: Rekonfiguracija, modularnost in avtonomna prilagoditev za napredne robotske celice
Vrsta dokumenta		Poročilo
Verzija dokumenta		Končna
Sodelujoči partnerji		IJS E1
Avtorji		Leon Žlajpah in Miha Deniša
Datum priprave	:	29.10.2024

Kazalo

1 Uvod **3**

2 Vodenje gibanja robotov **3**

3 Kinematika robota **4**

4 Načrtovanje poti **5**

4.1 Generacija poti z upoštevanjem ovir 6

5 Generacija trajektorij **7**

5.1 Interpolacija 7

5.2 Enostavna časovna parametrizacija 7

5.3 Časovno optimalna parametrizacija 7

6 Primer generacije poti in trajektorije **9**

Literatura **12**

1 Uvod

V projektu DIGITOP razvijamo rekonfigurabilno robotsko celico za avtomatizacijo novih in optimizacijo obstoječih proizvodnih procesov. Celica vključuje robotske roke, kamere in druge senzorje, prilagojene za naloge, kot so nadzor kakovosti in robotsko sestavljanje. Ključni poudarek projekta je na razvoju optimizacijskih algoritmov, ki omogočajo prilagajanje robotskih operacij s ciljem izboljšanja učinkovitosti in prilagodljivosti delovne celice.

Poročilo obravnava algoritme, ki omogočajo generiranje in optimizacijo robotskih trajektorij ter prilagajanje gibanja glede na spremembe v proizvodnem okolju. Trajektorije se načrtujejo tako, da upoštevajo omejitve sistema, kot so hitrosti in pospeški, ter omogočajo optimizirano in energetsko učinkovito gibanje. V tem kontekstu so podrobneje predstavljene metode interpolacije, časovne parametrizacije ter načrtovanja poti in izogibanja oviram.

Optimizacija na nižji ravni se osredotoča na natančno in prilagodljivo gibanje robotov, kar vključuje algoritme, ki omogočajo učenje iz demonstracij in iz CAD modelov za učinkovitejšo izvedbo nalog. S tem sistem omogoča tako lokalno kot globalno optimizacijo gibanja, kar zmanjšuje potrebo po prekinitvah in povečuje natančnost operacij. Na srednji ravni se trajektorije prilagajajo glede na nove zahteve celice. To lahko vključuje nov obdelovanec s pripadajočim CAD modelom, ali pa novo konfiguracijo same celice, npr. postavitev kamer za pregled kakovosti.

V nadaljevanju poročila so opisane metode programiranja robotov, s poudarkom na algoritmih za optimizacijo trajektorij in sprotno prilagajanje robotskih operacij za izboljšanje proizvodnega procesa.

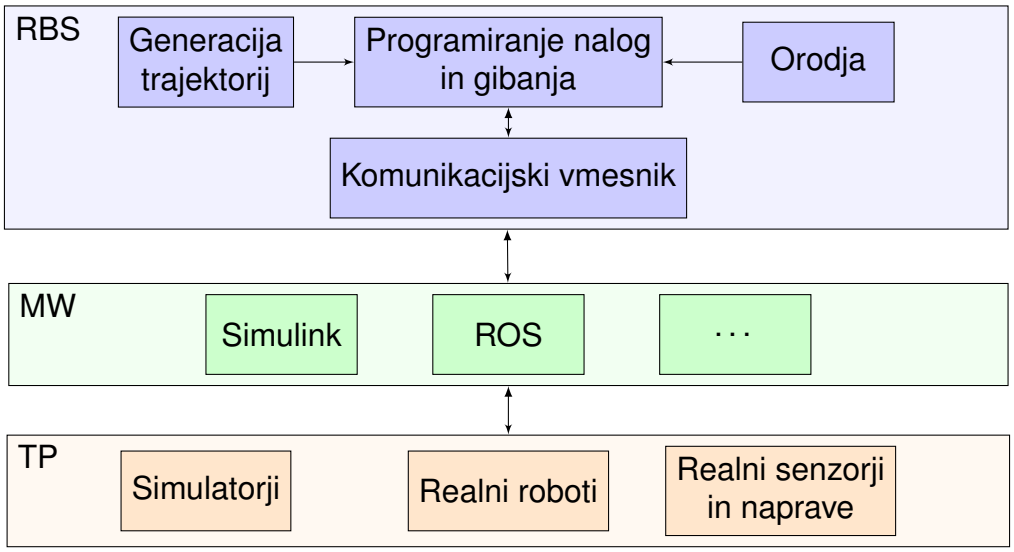
2 Vodenje gibanja robotov

Izvajanje gibanja robotov temelji na orodju RobotBlockset, ki omogoča enotno platformo za načrtovanje, testiranje in izvedbo robotskih aplikacij, ter premošča vrzel med simulacijami in realnimi sistemi. S svojimi vmesniki za simulatorje in robote ter orodji za modeliranje in načrtovanje gibanja poenostavlja razvojni proces, kar omogoča hitrejši prehod iz simulacij na realne sisteme ter krajše razvojne čase.

Celovito robotsko okolje mora biti prilagodljivo, odprto in enostavno za uporabo, da zadosti potrebam različnih uporabnikov. Ključne lastnosti so odprta arhitektura, enostavno preklapljanje konfiguracij ter integracija senzorjev in modularnih komponent, kot so vizualni in sila senzorji. RobotBlockset podpira tudi porazdeljeno arhitekturo za učinkovito izvajanje kompleksnih robotskih nalog in omogoča dostopno vizualizacijo okolja in nalog robota.

Struktura RBS okolja je razdeljena na tri dele (glej sliko 1):

- **RBS**: Glavni del je dejansko RBS orodje, ki nudi funkcije za programiranje robotov, generiranje trajektorij in gibanja ter različna orodja za analizo robotov. Vključuje tudi komunikacijski vmesnik za komunikacijo z robotskimi sistemi direktno ali preko prek vmesnega nivoja.



Slika 1: Struktura RBS

- **MW**: Nekateri roboti zahtevajo posebne komunikacijske protokole ali nizkonivojski nadzor robotov. V tem primeru uporabimo vmesni nivo med RBS in TP, kjer se ustrezna programska oprema za zagotavljanje želeno funkcionalnost. VTrenutno se na tem nivoju uporabljamo predvsem odprtokodno programsko okolje ROS ali Matlab/Simulink aplikacije za delov realnem času.
- **TP**: Ta del predstavlja ciljno platformo (TP), ki jo želimo nadzorovati. TP je lahko realni robot, vključno z različnimi senzorji in drugimi napravami, ali pa je TP simulator, kjer so simulirani robotski sistem in njegovo okolje.

Orodje je bilo prvotno razvito v MATLAB-u, nato pa je bilo preneseno v Python, s čimer je omogočen brezhiben prehod robotsko specifične funkcionalnosti v Python ekosistem. Za potrebe projekta DIGITOP smo v RBS vključili robote družine Universal Robots (UR10, UR10e in UR5e). Razvili smo kinematične modele in simulacijske modele za simulacijsko okolje MuJoCo. Za krmiljenje realnih UR robotov smo razvili vmesnike za ROS2 in za direktno vodenje robotov preko RTDE (Real-Time Data Excange) vmesnika, ki ga imajo UR roboti.

3 Kinematika robota

Položaj robota je določen s pozicijami v sklepih robota $\boldsymbol{q}$, ki so definirane kot vektor $\boldsymbol{q} \in \mathbb{R}^n$, kjer je n število sklepov robota. Zaloga vrednosti $\boldsymbol{q}$ določa t.i. *konfiguracijski prostor sklepov*. Prostor sklepov je omejen z minimalnimi in maksimalnimi vrednosti pozicij v posameznih sklopih.

Naloge so večinoma definirane v zunanjem (kartezijevem) prostoru kot gibanje točk na robotu. Te točke imenujemo *operativne točke* $\boldsymbol{x}$, $\boldsymbol{x} \in \mathbb{SE}(3)$. Operativne točke so lahko poljubne točke na robotski strukturi, vendar so običajno vezane lego vrha robota ali na prijemalo na robotu.

Povezavo med konfiguracijo robota in lego operativnih točk opisuje direktna kinematika robota

$$\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{q}) \tag{1}$$

kjer je $\boldsymbol{f}$ nelinearna funkcija, ki opisuje kinematično strukturo robota.

Hitrost vrha robota opišemo z vektorjem $\boldsymbol{v} \in \mathbb{R}^6$

$$\boldsymbol{v} = [\dot{\boldsymbol{p}}^T \boldsymbol{\omega}^T]^T \ , \tag{2}$$

kjer sta $\dot{\boldsymbol{p}}$ in $\boldsymbol{\omega}$ linearna and kotna hitrost vrha. Povezava med hitrostmi v sklepih $\dot{\boldsymbol{q}}$ in hitrostmi na vrhu robota $\boldsymbol{v}$ je določena z diferencialno direktno kinematiko v obliki

$$\boldsymbol{v} = \begin{bmatrix} \dot{\boldsymbol{p}} \\ \boldsymbol{\omega} \end{bmatrix} = \mathbf{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \ . \tag{3}$$

kjer je $\mathbf{J}(\boldsymbol{q}) \in \mathbb{R}^{6\times n}$ geometrična Jacobijeva matrika.

Z odvajanjem enačbe (3) dobimo povezavo med pospeški v kartezijevim prostorom in pospeški v sklepih

$$\dot{\boldsymbol{v}} = \mathbf{J}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\mathbf{J}} * \dot{\boldsymbol{q}} \ . \tag{4}$$

4 Načrtovanje poti

V robotiki je izvedba nalog povezana z gibanjem robota. Da bi dosegli želeno gibanje robota je potrebno določiti sekvenco položajev robota. Sekvenco položajev robota imenujemo *pot*. V nekaterih primerih je gibanje v celoti določena z nalogo (na primer, če mora robot slediti znanemu premikajočemu se predmetu). V drugih primerih, ko pa naloga zahteva samo, da robot preprosto pride iz enega položaja na drugega, je izbira gibanja med začetnim in končnim položajem poljubna. Ko pa določimo še čas, v katerem se mora robot premakniti iz začetnega položaja v končni oziroma hitrost gibanja po poti, govorimo o *trajektoriji*.

Določitev zahtevanih položajev robota je potrebno določiti glede na zahteve naloge in trenutne konfiguracije celice. V primeru vizualne kontrole kakovosti so te točke določene iz:

- CAD modelov obdelovancev,
- relativnih točk na obdelovancu, kjer je potreben pregled kakovosti,
- notranjih parametrov kamer,
- in položaja kamer.

Medtem ko večino teh podatkov pridobimo kot opis naloge, se položaj kamer lahko spreminja glede na samo konfiguracijo celice. Le-to pridobimo v postopku optimizacije celice. Ko so zahtevane točke poti oz. konfiguracije robota določene, je potrebno generirati celotno pot.

4.1 Generacija poti z upoštevanjem ovir

Če želimo načrtovati gibanje robota v okolju z ovirami, je potrebno najti pot iz začetne točke v končno točko, kjer ne pride do nobenega kontakta med robotom in ovirami.

Predpogoj, da lahko načrtujemo poti, je poznavanje okolja. Zato je potrebno poznati geometrijo objektov v delovnem prostoru (robota in ovir) in jih ustrezno modelirati. V večini primerov to pomeni, da objekte predstavimo z 3D mrežami ali pa jih aproksimiramo z enostavnimi telesi kot so na primer kocke, krogle ali cilindri.

Ko je poznano celotno delovno okolje in ostale omejitve (omejitve v sklepih robota, orientacije orodja, itd.), definiramo začetno in želeno končno lego robota kot pozicije sklepov ali pozi vrha robota in nato lahko začnemo načrtovati pot.

Za načrtovanje poti obstaja več algoritmov. Med pogosto uporabljenimi so:

- **OMPL** (Open Motion Planning Library) ki vključuje različna orodja in algoritme za načrtovanje poti v kompleksnih delovnih prostorih, kot jih imamo pri robotskih sistemih.
- **PRM** (Probabilistic Roadmap), ki gradi graf (mrežo) z naključnim vzorčenjem točk v delovnem prostoru in jih povezuje s kolizijsko neodvisnimi potmi. Algoritem najde pot brez trkov s povezovanjem začetne in ciljne pozicije skozi mrežo.
- **RRT** (Rapidly-exploring Random Trees), kjer se širi drevo iz začetne pozicije proti naključnim točkam v konfiguracijskem prostoru in išče pot do cilja, medtem ko se izogiba oviram.
- Na optimizaciji osnovani planerji, ki optimizirajo trajektorijo ob upoštevanju ovir in omejitev sklepov. Primeri takšnih planerjev sta **CHOMP** (Covariant Hamiltonian Optimization for Motion Planning) ali **STOMP** (Stochastic Trajectory Optimization for Motion Planning).
- **MoveIt**, ki je del robotskega operacijskega sistema ROS/ROS2 in vključuje različne planerje poti, kot je npr. OMPL.

Na vsakem koraku kateregakoli od omenjenih algoritma je potrebno preveriti, ali pride do trka med deli robota in ovirami. V ta namen so na razpolago različna orodja, kot so:

- **FCL** (Flexible Collision Library), se uporablja se v ROS sistemu za učinkovito preverjanje trkov.
- Razni simulatorji kot sta **Bullet**, **MuJoCo** ali **ODE**.
- Funkcije v raznih paketih, kot npr. v **MATLAB Robotics Toolbox** ali **Drake**

Končni rezultat načrtovanja poti je trkov prosta pot, t.j. sekvenca točk, ki povezuje začetno točko s končno. Opišemo jo lahko z enačbo (5).

5 Generacija trajektorij

Čeprav je gibanje lahko poljubno, mora biti pri generaciji trajektorij gladko in brez sunkov, z upoštevanjem omejitev hitrosti in pospeškov sistema. Za pretvorbo poti v gladko trajektorijo, ki ji robot lahko sledi, uporabljamo interpolacijo in časovno parametrizacijo. Pri načrtovanju poti upoštevamo omejitve glede lokacije, medtem ko generacija trajektorij doda časovne omejitve, kot so maksimalne hitrosti in pospeški v sklepih robota. Če naloga ne zahteva hitrega gibanja, lahko uporabimo enostavno časovno parametrizacijo brez dodatnih omejitev. Kadar pa želimo optimirati čas izvedbe ali porabo energije, postane proces bolj zahteven, saj je treba upoštevati vse omejitve sistema, kar pomeni, da optimalna trajektorija vedno doseže meje sistema.

5.1 Interpolacija

Prvi korak pri generiranju trajektorije je interpolacija med diskretnimi točkami poti. Interpolacija zagotavlja, da je gibanje med točkami gladko in neprekinjeno. Torej v primeru, da dobljena sekvenca točk ni dovolj gladka, uporabimo katerokoli interpolacijsko metodo in generiramo vmesne točke.

5.2 Enostavna časovna parametrizacija

Naslednji korak je določitev časa za vsako točko na poti, kar imenujemo *časovna parametrizacija*. Časovna parametrizacija zagotavlja, da robot napreduje po poti z ustrezno hitrostjo, hkrati pa upošteva omejitve robota in naloge.

Za zagotovitev gladkega gibanja moramo pri generaciji trajektorije vključiti boljše hitrostne in pospeševalne profile. Če uporabimo *trapezoidni hitrostni profil*, potem je zagotovljeno, da robot gladko pospešuje in zavira, brez nenadnih sprememb hitrosti. Če želimo gibanje dodatno pogladiti uporabimo polinomske profile, kjer se tudi pospešek postopoma povečuje in zmanjšuje, kar ustvarja bolj gladko gibanje z zmanjšanimi sunki.

Pri določitvi časovnega skaliranja pa je potrebno upoštevati še robne pogoje. Če upoštevamo robno pogoje glede hitrosti in pospeškov, je potrebno uporabiti polinom petega reda. Takšen polinomski profil je še posebej uporaben pri hitrem gibanju ali tam kjer potrebujemo večjo natančnost ali pa zmanjšanje mehanskih obremenitev.

5.3 Časovno optimalna parametrizacija

Pri *časovno optimalni parametrizaciji* se izračuna trajektorija, ki minimizira čas, potreben za izvedbo poti, hkrati pa trajektorija upošteva omejitve hitrosti in pospeškov. Pravzaprav je potrebno namesto omejitev pospeškov upoštevati omejitve navorov v sklepih, vendar je

potrebno za to uporabiti dinamičen model robota. V primeru, ko dinamičnega modela ne poznamo, je smiselno uporabiti ocenjene maksimalne pospeške.

Izhodišče postopka je geometrijska pot, definirana z enačbo (5)

$$\boldsymbol{x} = \boldsymbol{f}(s) \tag{5}$$
$$\boldsymbol{v} = \mathbf{J}_s \dot{s} \tag{6}$$
$$\dot{\boldsymbol{v}} = \mathbf{J}_s \ddot{s} + \mathbf{J}'_s \dot{s}^2 \,, \tag{7}$$

Omejitve, ki jih moramo upoštevati pri generacije trajektorije in se nanašajo na omejene hitrosti in pospeške v prostoru sklepov in kartezijevem prostoru, pa definiramo kot

$$0 < \dot{s} \leq \dot{s}_{min} \tag{8}$$
$$|\ddot{s}| \leq \ddot{s}_{max} \tag{9}$$
$$|\boldsymbol{v}_i] \leq \boldsymbol{v}_{max,i}, \quad i = 1, ..., 6 \tag{10}$$
$$|\dot{\boldsymbol{v}}_i| \leq \dot{\boldsymbol{v}}_{max,i}, \quad i = 1, ..., 6 \tag{11}$$
$$|\dot{\boldsymbol{q}}_j| \leq \dot{\boldsymbol{q}}_{max,j}, \quad j = 1, ..., n \tag{12}$$
$$|\ddot{\boldsymbol{q}}_j| \leq \ddot{\boldsymbol{q}}_{max,j}, \quad j = 1, ..., n \tag{13}$$

Cilj pa je najti izvedljivo trajektorijo bodisi v delovnem prostoru ($\boldsymbol{x}(t)$, $\boldsymbol{v}(t)$ in $\dot{\boldsymbol{v}}(t)$) bodisi v sklepnem prostoru ($\boldsymbol{q}(t)$, $\dot{\boldsymbol{q}}(t)$ in $\ddot{\boldsymbol{q}}(t)$).

Ker je gibanje robota omejeno na pot s je smiselno vse omejitve pretvoriti na omejitve $\dot{s}$ in $\ddot{s}$. Če vstavimo enačbi (6) in (7) v enačbi (3) in (4) dobimo

$$\dot{\boldsymbol{q}} = \mathbf{J}^{\#} \mathbf{J}_s \dot{s} \tag{14}$$
$$\ddot{\boldsymbol{q}} = \mathbf{J}^{\#} (\mathbf{J}_s \ddot{s} + \mathbf{J}'_s \dot{s}^2 - \dot{\mathbf{J}} \dot{\boldsymbol{q}}) \,, \tag{15}$$

kjer je $\mathbf{J}^{\#}$ generaliziran inverse Jacobijeve matrike $\mathbf{J}$. Iz pogojev (8) – (13) dobimo z upostevanjem enačb (6), (7), (14) in (15) omejitve oziroma pogoje za $\dot{s}$ in $\ddot{s}$.

Da bi našli časovno optimalno trajektorijo, je potrebno uporabiti nek optimizacijski pristop. Med različnimi možnimi kriterijskimi funkcijami, ki jih optimiramo, smo izbrali minimalni čas izvedbe poti t_e, ki ga definiramo s sledečo enačbo

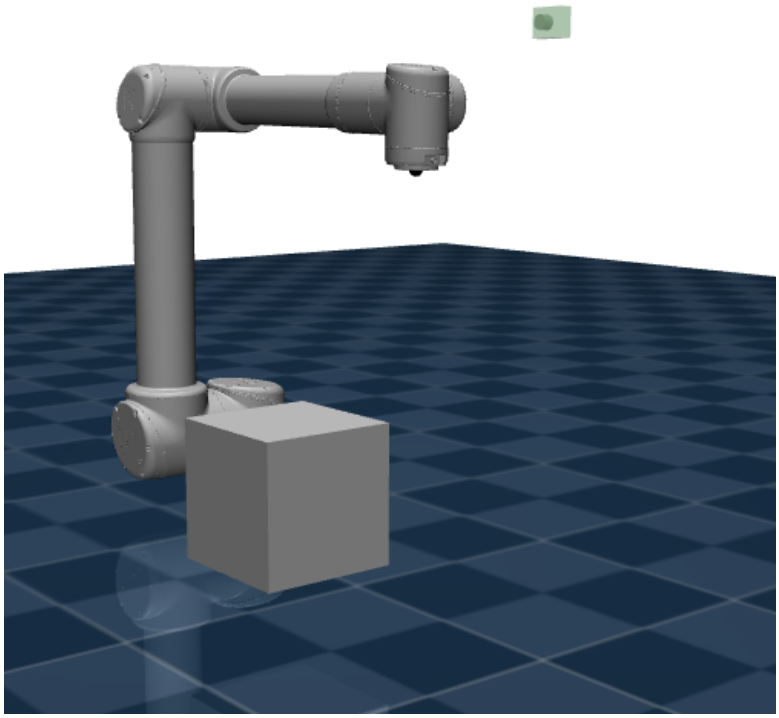
$$t_e = \int_{s_0}^{s_{end}} \frac{1}{\dot{s}(s)} \, \mathrm{d}\, s \tag{16}$$

V teoriji obstaja kar nekaj metod za reševanje optimizacijskih problemov, vendar imajo vse te metode pomanjkljivost, da so precej kompleksen in časovno zahtevne. Zato smo razvili metode, ki izkorišča specifične lastnosti kriterijske funkcije in robotskega sistema.

Če pogledamo kriterijsko funkcijo (16), je očitno, da mora biti hitrost poti čim večja v vsakem trenutku, pri tem pa se ne smejo biti navedene omejitve nikoli kršene. Poleg tega je potrebno uporabiti maksimalno možno pospeševanje za povečanje hitrosti in maksimalno zaviranje za njeno zmanjšanje. Dobljena trajektorija je sestavljena iz segmentov z maksimalnim pospeševanjem in zaviranjem, ki jih dobimo s ponavljajočim iskanjem izvedljivih trajektorij z maksimalnim pospeševanjem naprej in zaviranjem nazaj. Vmes pa so lahko segmenti, kjer doseže hitrost $\dot{s}$ maksimlano dovoljeno vrednost, ki je določena z omejitvami hitrostmi sklepov in zahtevami naloge, kot tudi z omejitvami pospeškov. V teh delih trajektorije, je pospešek $\ddot{s}$ med minimalno in maksimalno dopustno vrednostjo.

6 Primer generacije poti in trajektorije

Kot primer generacije poti in trajektorije smo izbrali nalogo vizualnega pregleda kakovosti. Robot UR10 vzame iz podstavka predmet in ga prenese pred kamero (slika 2). Začetna in končna točka sta bili določeni preko modela predmeta, notranjih parametrov kamere in njene poze.



Slika 2: Testno okolje robota UR10 v simulacijskem paketu MuJoCo

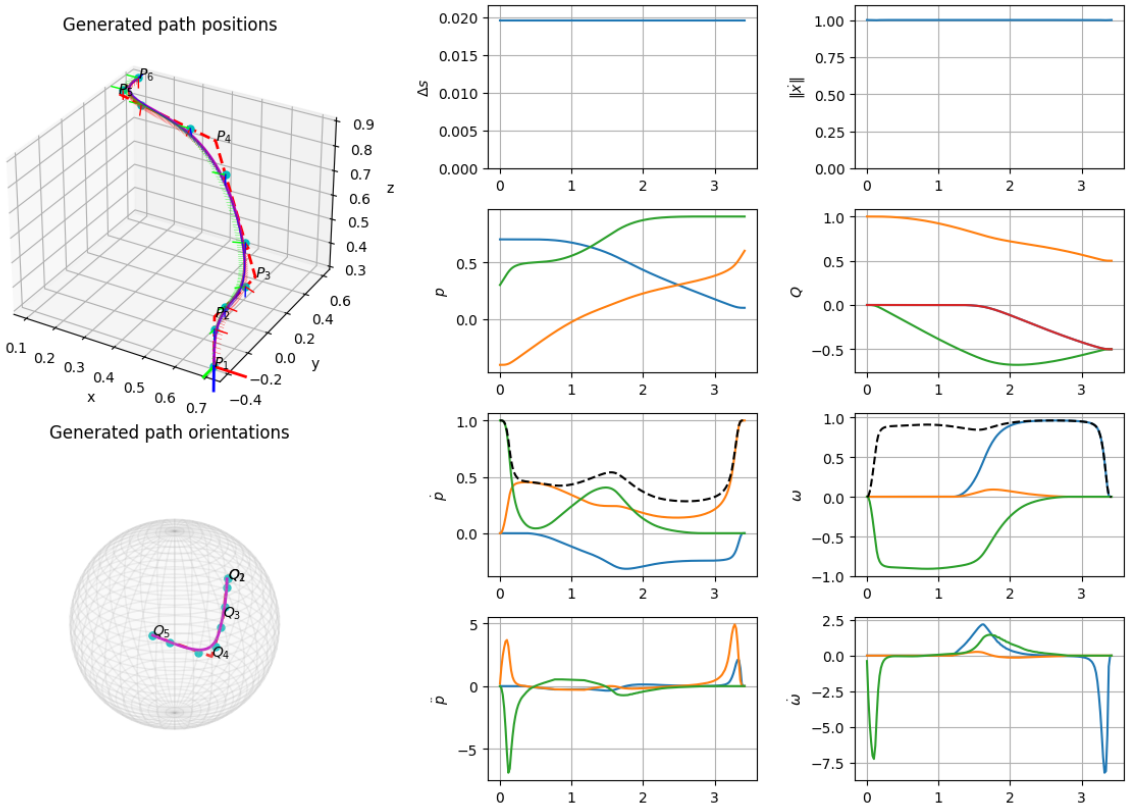
Da robot izvede gibanje varno smo uporabili pri generaciji poti vmesne točke, mimo katerih se naj robot giblje.

```
r = robot_ur10()
points6d = np.array(
  [
    [0, -0.2, -0.4, 0, 0, np.pi],
    [0.0, -0.2, -0.2, 0, 0, np.pi],
    [0.0, 0.2, -0.2, -np.pi / 4, 0, np.pi],
    [-0.2, 0.4, 0.2, -np.pi / 2, 0, np.pi],
    [-0.6, 0.6, 0.2, -np.pi / 2, np.pi/2, np.pi],
    [-0.6, 0.8, 0.2, -np.pi / 2, np.pi/2, np.pi],
  ]
)
points6d_init=[0.7, -0.2, 0.7, 0, 0, 0]  # Offset
for i in range(points6d.shape[0]):
```

```
  points6d[i,:] = points6d[i,:] + points6d_init
points = prpy2x(points6d) # transform Euler angles to quaternions
```

Pot smo generirali z uporabo zlepkov (spline) tako, da se je robot gibal mimo vmesnih točk. Slika 3 prikazuje generirano pot.

```
path_x, path_s = pathoverpoints(
  points,
  interp="inner",
  auxpoints="relative",
  auxdistance=0.25,
  viapoints=False,
  n_points=100,
  natural=True,
  normscale=1,
  plot=True,
)
```



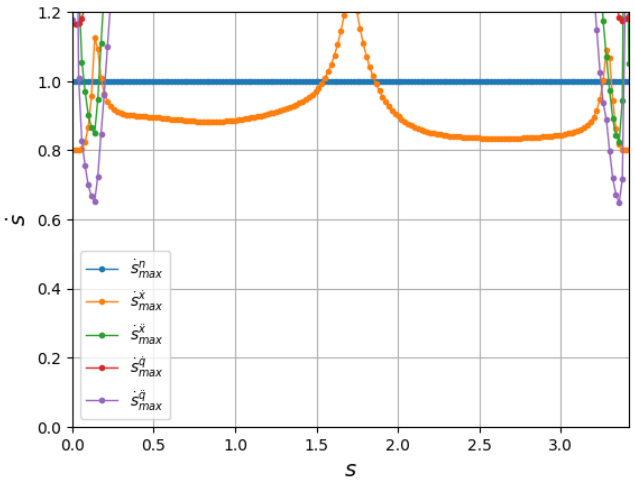
Slika 3: Generirana pot

Ker je pot generirana v kartezijevem prostoru, izračunamo pot v prostoru sklepov z uporabo inverzne kinematike

```
path_q, _err = r.IKinPath(path_x, r.q_init, null_space_task='None')
```

Nato določimo omejitve hitrosti in pospeškov. Slika 4 prikazuje maksimalne dopustne hitrosti $\dot{s}$ za posamezne omejitve po celotni poti. Generirana trajektorija ne sme nikoli preseči minimalno vrednost maksimalno dopustnih hitrosti $\dot{s}$.

```
sdn = 1
sddn = 5
xdmax = np.ones(6) * 0.8
xddmax = np.ones(6) * 5
qdmax = np.ones(r.nj) * 2
qddmax = np.ones(r.nj) * 5
dkin = r.Kinmodel
plot_path_bounds(path_x, path_q, dkin, sdn, xdmax, xddmax, qdmax, qddmax)
```



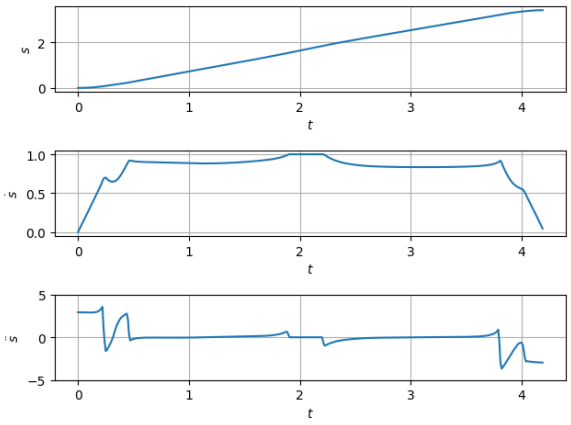
Slika 4: Maksimalne dopustne hitrosti $\dot{s}$ za posamezne omejitve po celotni poti (fazna ravnina $s - \dot{s}$)

Na osnovi poti in z upoštevanjem omejitev generiramo časovno optimalno trajektorijo v kartezijevem protoru in prostoru sklepov, ki so prikazane na slikah 5a, 5b in 6a. Vidimo lahko, da so vse hitrosti in pospeški v predpisanih mejah in da je v vsakem trenutku vsaj ena hitrost oziroma pospešek na omejitvi.

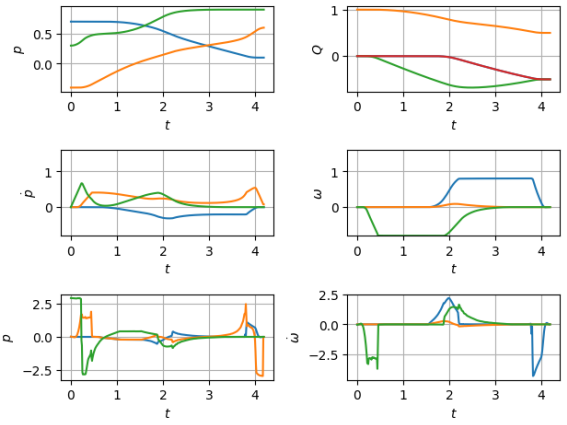
```
T, sp, sv, sa = timeopttraj(path_x, path_q, dkin, s0=0, send=None, sd0=0,
    sdend=0, sdn=sdn, sddn=sddn, xdmax=xdmax, xddmax=xddmax, qdmax=qdmax,
    qddmax=qddmax, tsamp=0.01, plot=False)
path_rx, path_rxd, path_rxdd = s2x(sp, sv, sa, path_s, path_x)
path_rq, path_rqd, path_rqdd = s2q(sp, sv, sa, path_s, path_x, path_q,
   dkin)
```

Dobljeno trajektorijo lahko nato izvedemo na robotu. Na sliki 6b so prikazani časovni poteki referenc in dejanskih vrednosti pozicij in hitrosti vsklepih ter pozicij, orientacij in hitrosti vrha robota.

```
r.JMove(path_rq[0,:], 2) # Move to initial point
r.StartCapture()
r.JPath(path_rq, t_path) # Move along optimal trajectory
```



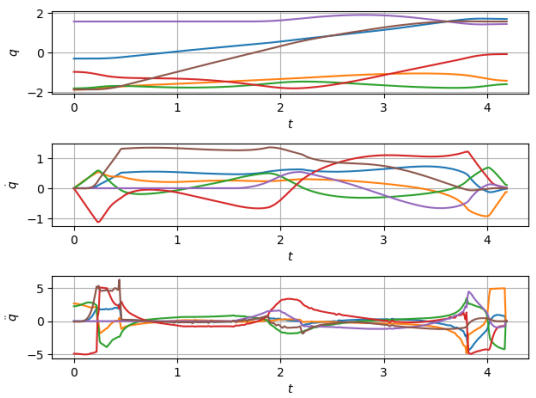
(a) Časovno optimalna trajektorija - pozicija, hitrost in pospešek po poti



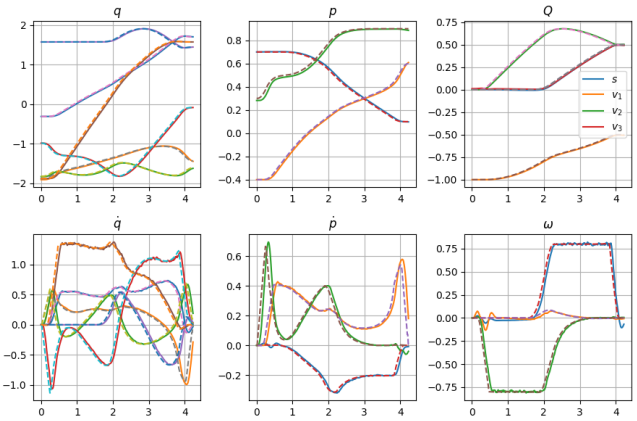
(b) Časovno optimalna trajektorija - pozicija, hitrost in pospešek vrha robota

Slika 5: Prikaz časovno optimalnih trajektorij po poti in na vrhu robota

```
r.StopCapture()
r.Wait(3)
r.JMove(r.q_init, 2)
```



(a) Časovno optimalna trajektorija - pozicije, hitrosti in pospeški v sklepih



(b) Izvedba optimalne trajektorije na robotu UR10

Slika 6: Prikaz časovno optimalne trajektorije in izvedbe na robotu UR10

Literatura

[1] C. Atkeson, A. Moore, and S. Schaal. Locally weighted learning for control. *Artificial Intelligence Review*, 11(1-5):75–113, 1997.

[2] X. Broquere, D. Sidobre, and K. Nguyen. From motion planning to trajectory control with bounded jerk for service manipulator robots. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 4505 – 4510, 06 2010.

[3] K. Kyriakopoulos and G. Saridis. Minimum jerk path generation. In *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, pages 364–369. IEEE Comput. Soc. Press, 1988.

[4] R. M. Murray, Z. Li, and S. S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[5] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. *Robotics - Modelling, Planning and Control*. Advanced Textbooks in Control and Signal Processing. Springer-Verlag London, 2009.

[6] A. Visioli. Trajectory planning of robot manipulators by using algebraic and trigonometric splines. *Robotica*, 18(6):611–631, 2000.

[7] R. Volpe. Task space velocity blending for real-time trajectory generation. In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, number 2, pages 680–687. IEEE Comput. Soc. Press, 1993.

[8] L. Žlajpah. On time optimal path control of manipulators with bounded joint velocities and torques. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 2, pages 1572–1577. IEEE, 1996.

[9] L. Žlajpah and T. Petrič. Generation of Smooth Cartesian Paths Using Radial Basis Functions. In S. Zeghloul, M. Amine Labiri, and J. S. Sandoval Arevalo, editors, *Advances in Service and Industrial Robotics. RAAD 2020*, pages 171–180, Cham, 2020. Springer International Publishing.

[10] L. Žlajpah and T. Petrič. RobotBlockSet (RBS) - A comprehensive robotics framework. In D. Pisla, G. Carbone, D. Condurache, and C. Vaida, editors, *Advances in Service and Industrial Robotics. RAAD 2024, 33nd International Conference on Robotics in Alpe-Adria-Danube Region, 12-14 June 2024, Cluj-Napoca, Romania*, pages 439–450, Cham, 2024. Springer.

[11] L. Žlajpah and T. Petriè. Obstacle Avoidance for Redundant Manipulators as Control Problem. In Serdar Küçük, editor, *Serial and parallel robot manipulators - kinematics, dynamics, control and optimization*, pages 203–230. InTech, 2012.