Razvoj petosnog robota paralelne kinematike

Božić, Matej

Master's thesis / Diplomski rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:235:236966

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-08**

Repository / Repozitorij:

Repository of Faculty of Mechanical Engineering and Naval Architecture University of Zagreb

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Matej Božić

Zagreb, 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentori:

Prof. dr. sc. Bojan Jerbić, dipl. ing.
Dr. sc. Marko Švaco, dipl. ing.

Student:

Matej Božić

Zagreb, 2020.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se svim profesorima i asistentima na prenesenom znanju koje sam mogao primijeniti prilikom izrade diplomskog rada. Posebno bih se zahvalio mentorima prof. dr. sc. Bojanu Jerbiću i dr. sc. Marku Švaci na stručnim savjetima i dobronamjernim sugestijama kako bih napravio što bolji diplomski rad. Također bih se zahvalio svim zaposlenicima Katedre za projektiranje izradbenih i montažnih sustava koji su mi svojim stručnim sugestijama i savjetima pomogli da riješim probleme s kojima sam se susreo prilikom izrade diplomskog rada. Nakraju bih se zahvalio svojoj obitelji i djevojci koji su mi bili velika moralna podrška tijekom cijelog studija i izrade ovog rada.

Matej Božić

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske radove studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala te mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum:	Prilog:
Klasa:	
Ur. broj:	

DIPLOMSKI ZADATAK

Student: **MATEJ BOŽIĆ** Mat. br.: 0035200811

Naslov rada na hrvatskom jeziku: **Razvoj petosnog robota paralelne kinematike**

Naslov rada na engleskom jeziku: **Design of five-bar parallel robot**

Opis zadatka:

U diplomskom radu potrebno je izraditi koncept, konstrukcijski razraditi te eksperimentalno validirati robota paralelne kinematike. Ograničenja zadatka su sljedeća:
1. Radni prostor robota je minimalno veličine 200 × 250 mm.
2. Kinematska struktura je paralelna s dvije upravljačke osi i tri pasivna rotacijska zgloba (eng. Five-Bar Parallel Robot).
3. Izvršni članak robota treba imati vakuumsku hvataljku s aktivnim vertikalnim hodom minimalno 30 mm.
4. Potrebno je minimirati ukupnu masu robota radi ostvarivanja boljih dinamičkih svojstava uz zadržavanje dovoljne krutosti cjelokupne konstrukcije.
5. Minimalna nosivost robota na izvršnom članku (hvataljci) treba biti 50 g.
6. Linearna brzina vrha robotskog alata treba biti najmanje 1 m/s, a akceleracija najmanje 9,81 m/s2.

Za upravljanje robotom potrebno je izraditi grafičko sučelje. U okviru validacije koncepta i pojedinih konstrukcijskih elemenata robota potrebno je:
1. Vrednovati pogonske koncepte: izravan prijenos u odnosu na prijenos s redukcijom broja okretaja motora.
2. Izvršiti odabir odgovarajućih standardnih konstrukcijskih, pogonskih i upravljačkih komponenata.
3. Oblikovati specifične elemente robota.
4. Odabrati materijal i tehnologiju izrade/obrade specifičnih konstrukcijskih elemenata robota.
5. Napraviti dinamičku i statičku FEM analizu za oblikovane pokretne dijelove robota.
6. Izmjeriti ponovljivost vrha robotskog alata.

U svrhu laboratorijske validacije rada robota paralelne kinematike potrebno je izraditi eksperimentalan postav u Laboratoriju za projektiranje izradbenih i montažnih sustava. Robotski sustav primijeniti na primjeru izuzimanja i odlaganja predmeta rada. Predmet rada potrebno je lokalizirati koristeći vizijski sustav na temelju boje ili oblika objekta.
U radu navesti korištenu literaturu i ostale izvore.

Zadatak zadan: 16. siječnja 2020.

Rok predaje rada: 19. ožujka 2020.

Predviđeni datum obrane: 23. ožujka do 27. ožujka 2020.

Zadatak zadao: prof. dr. sc. Bojan Jerbić

Predsjednica Povjerenstva: prof. dr. sc. Biserka Runje

SADRŽAJ

SADRŽAJ I
POPIS SLIKA III
POPIS TABLICA VII
POPIS TEHNIČKE DOKUMENTACIJE VIII
POPIS OZNAKA IX
SAŽETAK XI
SUMMARY XII
1. UVOD 1
1.1. Paralelni roboti 1
1.2. Izvedbe paralelnih robota 2
2. PETOSNI ROBOT PARALELNE KINEMATIKE 4
2.1. Parametri robota 4
2.2. Princip rada robota 4
2.3. Kinematika robota 5
2.3.1. Direktna kinematika 6
2.3.2. Inverzna kinematika 8
2.4. Analiza singularnosti 9
2.5. Radni prostor 12
2.6. Odabir konfiguracije robota 13
3. KONSTRUKCIJA I MONTAŽA ROBOTA 16
3.1. Prethodne verzije robota 16
3.2. Odabrana verzija robota 18
3.2.1. Članak 1 robota 21
3.2.2. Članak 2 robota 23
3.2.3. Članak 3 robota 25
3.2.4. Revolutni zglobovi 28
3.3. Vakuumska hvataljka s aktivnim vertikalnim hodom 29
3.4. Analiza robota 31
3.4.1. FEM analiza 1 32
3.4.2. Dinamička analiza 39
3.4.3. FEM analiza 2 45
3.5. Odabir aktuatora 48
3.6. Kućište robota 53
3.6.1. Kućište motora 55
3.6.2. Kućište radnog prostora robota 61
3.6.3. Kućište mikroračunala i kamere 64
3.7. Montaža robota 67
4. UPRAVLJANJE 81
4.1. Shema spajanja elektronike robota 81
4.2. Struktura koda 83

4.3. Inicijalizacija/kalibracija robota 84
4.3.1. Inicijalizacija robota 86
4.3.2. Kalibracija robota 87
4.3.3. Preskakivanje inicijalizacije robota 89
4.4. Upravljanje robotom kroz grafičko sučelje 90
4.4.1. Vizijski sustavi 101
4.4.1.1. Kalibracija kamere 102
4.4.1.2. Kalibracija koordinatnog sustava 106
4.4.1.3. Lokalizacija predmeta rada na temelju boje i oblika 109
4.4.1.4. Upravljanje robotom vizijskim sustavima kroz grafičko sučelje 114
5. MJERENJE 116
5.1. Mjerenje brzine i ubrzanja robota 116
5.2. Mjerenje ponovljivosti robota 120
5.3. Mjerenje nosivosti robota 124
6. ZAKLJUČAK 126
LITERATURA 127
PRILOZI 130

POPIS SLIKA

Slika 1. Kinematska struktura delta robota[2] ... 1
Slika 2. Heksapodni robot za simulaciju leta[3] ... 2
Slika 3. Delta robot, operacija izuzmi-odloži[4] ... 3
Slika 4. Mitsubishi RP5-AH[5] ... 3
Slika 5. Prikaz petosnog robota paralelne kinematike[6] ... 5
Slika 6. Petosni robot paralelne kinematike, za izvod[6] ... 6
Slika 7. Izvod direktne kinematike[6] ... 7
Slika 8. Rješenje direktne kinematike: a) – način rada, b) + način rada[6] ... 8
Slika 9. Rješenje inverzne kinematike: a) ++ način rada, b) +- način rada, c) +- način rada, d) – način rada[6] ... 9
Slika 10. Singularnosti prve vrste, a) desni lanac ispružen, b) lijevi lanac ispružen[6] ... 11
Slika 11. Singularnosti druge vrste[6] ... 12
Slika 12. Singularnosti treće vrste[6] ... 12
Slika 13. Radno područje robota[6] ... 13
Slika 14. Odabrano radno područje petosnog robota paralelne kinematike[6] ... 15
Slika 15. Verzija 1 ... 17
Slika 16. Verzija 2 ... 17
Slika 17. Verzija 3 ... 18
Slika 18. Konačna verzija robota, pogled 1 ... 18
Slika 19. Konačna verzija robota, pogled 2 ... 19
Slika 20. Konačna verzija robota, pogled 3 ... 19
Slika 21. Prikaz zvjezdastog oblika ... 20
Slika 22. Montiranje ležajeva[11] ... 21
Slika 23. Članak 1 robota, pogled ... 22
Slika 24. Članak 1 robota, presjek ... 22
Slika 25. Članak 1 robota, spoj sa vratilom ... 23
Slika 26. Članak 2 robota, pogled ... 24
Slika 27. Članak 2 robota, presjek ... 24
Slika 28. Članak 2 robota, samokočna matica ... 25
Slika 29. Članak 3 robota, pogled ... 26
Slika 30. Članak 3 robota, presjek ... 27
Slika 31. Članak 3 robota, lijepljenje matice ... 27
Slika 32. Revolutni zglob desni članak 1 i članak 2 robota ... 28
Slika 33. Revolutni zglob lijevi članak 1 i članak 3 robota ... 28
Slika 34. Revolutni zglob članak 2 i članak 3 robota ... 29
Slika 35. Vakuumska hvataljka: a)3D CAD model, b)sklop ... 30
Slika 36. *SMC ZU07SA* ... 31
Slika 37. *Festo MZH-3-M3-L-LED* ... 31
Slika 38. Položaj za FEM analizu 1 ... 33
Slika 39. Diskretizirani „linkageA0_1“ ... 33
Slika 40. Von Mises naprezanja za „linkageA0_1“, MPa ... 34
Slika 41. Deformacije za „linkageA0_1“, mm ... 34
Slika 42. Diskretizirani „linkageA0_2“ ... 35
Slika 43. Von Mises naprezanja za „linkageA0_2“, MPa ... 35
Slika 44. Deformacije za „linkageA0_2“, mm ... 36
Slika 45. Diskretizirani „linkageA1_1“ ... 36
Slika 46. Von Mises naprezanja za „linkageA1_1“, MPa ... 37
Slika 47. Deformacije za „linkageA1_1“, mm ... 37

Slika 48. Diskretizirani „linkageB1_1“ ... 38
Slika 49. Von Mises naprezanja za „linkageB1_1“, MPa ... 38
Slika 50. Deformacije za „linkageB1_1“, mm ... 39
Slika 51. Početni položaj robota u dinamičkoj simulaciji ... 40
Slika 52. Završni položaj robota u dinamičkoj simulaciji ... 41
Slika 53. Maksimalna postignuta brzina robota za dinamičku simulaciju ... 41
Slika 54. Položaj robota za maksimalnu postignutu brzinu ... 42
Slika 55. Maksimalna postignuta akceleracija robota za dinamičku simulaciju ... 42
Slika 56. Položaj robota za maksimalnu postignutu akceleraciju ... 43
Slika 57. Maksimalni potrebni moment za „linkageA0_1“ i „linkageB0_1“ ... 43
Slika 58. Položaj robota kod maksimalnog potrebnog momenta za „linkageA0_1“ ... 44
Slika 59. Položaj robota kod maksimalnog potrebnog momenta za „linkageB0_1“ ... 44
Slika 60. „linkageA0_1“: a) Von Mises naprezanja, MPa; b) deformacije, mm ... 45
Slika 61. „linkageA0_2“: a) Von Mises naprezanja, MPa; b) deformacije, mm ... 45
Slika 62. „linkageA1_1“: a) Von Mises naprezanja, MPa; b) deformacije, mm ... 46
Slika 63. „linkageB1_1“: a) Von Mises naprezanja, MPa ; b) deformacije, mm ... 46
Slika 64. Tlocrt robota za FEM analizu 2 ... 47
Slika 65. „linkageB0_2“: a) Von Mises naprezanja, MPa; b) deformacije, mm ... 47
Slika 66. Planetarni prijenos[15] ... 48
Slika 67. Remenski prijenos[16] ... 49
Slika 68. Harmonijski prijenos[17] ... 49
Slika 69. *ODrive robotics* driver ... 50
Slika 70. *APS5065 90 K_v 1800W* ... 51
Slika 71. *CUI AMT102-V* ... 51
Slika 72. Graf za odabir napajanja[19] ... 52
Slika 73. *MeanWell ERPF-40-24* ... 52
Slika 74. *Arcol HS50-R5-J* ... 53
Slika 75. Petosni robot paralelne kinematike ... 54
Slika 76. Radno područje robota s konstrukcijskim ograničenjima ... 55
Slika 77. Kućište motora, pogled 1 ... 55
Slika 78. Kućište motora, pogled 2 ... 56
Slika 79. „motorJoint_1“ ... 56
Slika 80. „motorJoint_2“ ... 57
Slika 81. Presjek kućišta motora ... 58
Slika 82. „motorJoint_3“ ... 59
Slika 83. „motorJoint_8“ ... 59
Slika 84. Povezivanje vratila s ostatkom robota ... 60
Slika 85. Kućište radnog prostora robota, pogled 1 ... 61
Slika 86. Kućište radnog prostora robota, pogled 2 ... 61
Slika 87. Kućište radnog prostora robota, pogled 3 ... 62
Slika 88. Montirano kućište 1 ... 63
Slika 89. Montirano kućište 2 ... 63
Slika 90. Kućište za mikroračunalo i kameru ... 64
Slika 91. „cameraHousing_1“ ... 65
Slika 92. „cameraHousing_2“ ... 65
Slika 93. „cameraHousing_3“ ... 66
Slika 94. „cameraHousing_4“ ... 66
Slika 95. „cameraHousing_5“ ... 66
Slika 96. Montaža robota 1 ... 67
Slika 97. Montaža robota 2 ... 67

Slika 98. Montaža robota 3 ... 68
Slika 99. Montaža robota 4 ... 68
Slika 100. Montaža robota 5 ... 69
Slika 101. Montaža robota 6 ... 69
Slika 102. Montaža robota 7 ... 70
Slika 103. Montaža robota 8 ... 70
Slika 104. Montaža robota 9 ... 71
Slika 105. Montaža robota 10 ... 71
Slika 106. Montaža robota 11 ... 72
Slika 107. Montaža robota 12 ... 72
Slika 108. Montaža robota 13 ... 73
Slika 109. Montaža robota 14 ... 73
Slika 110. Montaža robota 15 ... 74
Slika 111. Montaža robota 16 ... 74
Slika 112. Montaža robota 17 ... 75
Slika 113. Montaža robota 18 ... 75
Slika 114. Montaža robota 19 ... 76
Slika 115. Montaža robota 20 ... 76
Slika 116. Montaža robota 21 ... 77
Slika 117. Montaža robota 22 ... 77
Slika 118. Montaža robota 23 ... 78
Slika 119. Montaža robota 24 ... 78
Slika 120. Montaža robota 25 ... 79
Slika 121. Montaža robota 26 ... 79
Slika 122. Montaža robota 27 ... 80
Slika 123. Shema spajanja elektronike[24] ... 82
Slika 124. Kutija za elektroniku robota ... 82
Slika 125. Struktura koda[24] ... 83
Slika 126. GUI, početni prozor ... 85
Slika 127. GUI, prekidanje kalibracije ... 85
Slika 128. GUI, inicijalizacija robota ... 86
Slika 129. GUI, kalibracija robota ... 88
Slika 130. GUI, upravljanje robotom ... 90
Slika 131. Kalibracija kamere, slikanje kalibracijske mreže ... 103
Slika 132. Kalibracija kamere, pronalazak točaka ... 104
Slika 133. *Pinhole* model kamere[27] ... 106
Slika 134. Kalibracija koordinatnog sustava, detekcija markera ... 108
Slika 135. Osnovni geometrijski elementi za prepoznavanje ... 109
Slika 136. Slika u *HSV* prostoru boja ... 110
Slika 137. Maska za plavu boju ... 111
Slika 138. Prepoznavanje plavih objekata ... 113
Slika 139. Prepoznavanje svih objekata ... 113
Slika 140. Proces prepoznavanja boje i oblika objekta ... 114
Slika 141. Mjerenje brzine i ubrzanja, pozicija 1 u simulaciji ... 116
Slika 142. Mjerenje brzine i ubrzanja, pozicija 1, eksperimentalni postav ... 116
Slika 143. Mjerenje brzine i ubrzanja, pozicija 2 u simulaciji ... 117
Slika 144. Mjerenje brzine i ubrzanja, pozicija 2, eksperimentalni postav ... 117
Slika 145. Pomaci enkodera motora0 i motora1 ... 118
Slika 146. Maksimalna brzina vrha robota ... 119
Slika 147. Maksimalno ubrzanje vrha robota ... 119

Slika 148. Mahr MarCator 1086R 120
Slika 149. Mjerenje ponovljivosti, pozicija 1 120
Slika 150. Mjerenje ponovljivosti, pozicija 1, robot odmaknut 121
Slika 151. Mjerenje ponovljivosti, pozicija 1, robot primaknut 121
Slika 152. Mjerenje ponovljivosti, pozicija 2 122
Slika 153. Mjerenje ponovljivosti, pozicija 3 122
Slika 154. Mjerenje ponovljivosti, pozicija 4 123
Slika 155. Masa predmeta za testiranje nosivosti 124
Slika 156. Mjerenje nosivosti robota 1 124
Slika 157. Mjerenje nosivosti robota 2 125
Slika 158. Mjerenje nosivosti robota 3 125

POPIS TABLICA

Tablica 1. Rezultati mjerenja ponovljivosti. 123

POPIS TEHNIČKE DOKUMENTACIJE

MB_Diplomski_00	linkageA1_3/B1_3
MB_Diplomski_01	CarbonTube_122.5
MB_Diplomski_02	CarbonTube_182
MB_Diplomski_03	CarbonTube_184
MB_Diplomski_04	motorJoint_1
MB_Diplomski_05	motorJoint_2
MB_Diplomski_06	motorJoint_8
MB_Diplomski_07	motorJoint_9
MB_Diplomski_08	motorJoint_10
MB_Diplomski_09	motorJoint_6
MB_Diplomski_10	motorJoint_7
MB_Diplomski_11	housing_1
MB_Diplomski_12	housing_3

POPIS OZNAKA

Oznaka	Jedinica	Opis
A_0B_0	mm	Duljina desnog kraćeg članka robota
A_1B_1	mm	Duljina lijevog kraćeg članka robota
l_1	mm	Duljina kraćeg članka robota
B_0P	mm	Duljina lijevog duljeg članka robota
B_1P	mm	Duljina desnog duljeg članka robota
l_2	mm	Duljina duljeg članka robota
A_0A_1	mm	Udaljenost između dva aktivna zgloba robota
l_0	mm	Pola udaljenosti između dva aktivna zgloba robota
α	°	Ulazni kut aktivnog zgloba 0
β	°	Ulazni kut aktivnog zgloba 1
γ	°	Kut pasivnog zgloba 0
δ	°	Kut pasivnog zgloba 1
x_{B0}	mm	Središte pasivnog zgloba B_0 *x* os
y_{B0}	mm	Središte pasivnog zgloba B_0 *y* os
x_{B1}	mm	Središte pasivnog zgloba B_1 *x* os
y_{B1}	mm	Središte pasivnog zgloba B_1 *y* os
d	mm	Udaljenost između središta kružnice B_0 i B_1
a	mm	Duljina stranice lijevog trokuta
b	mm	Duljina stranice desnog trokuta
h	mm	Visina trokuta
x_{P0}	mm	Središte P_0 *x* os
y_{P0}	mm	Središte P_0 *y* os
x_{P1}	mm	Središte *P x* os
y_{P1}	mm	Središte *P y* os
F	-	Matrica odnosa ulaznih i izlaznih veličina robota
A	-	Jakobijan matrica izlaznih koordinata robota
B	-	Jakobijan matrica ulaznih kutova robota
$\boldsymbol{x}$	mm	Vektor izlaznih koordinata robota
$\boldsymbol{\theta}$	°	Vektor ulaznih kutova robota
$x_{rad\_ispravljeno\_r}$	mm	*x* koordinata radijalno ispravljenog piksela
$y_{rad\_ispravljeno\_r}$	mm	*y* koordinata radijalno ispravljenog piksela
$x_{rad\_ispravljeno\_t}$	mm	*x* koordinata tangencijalno ispravljenog piksela
$y_{rad\_ispravljeno\_r}$	mm	*x* koordinata tangencijalno ispravljenog piksela
k_1	-	*Radijalni distorzijski koeficijent 1*
k_2	-	*Radijalni distorzijski koeficijent 2*
k_3	-	*Radijalni distorzijski koeficijent 3*
p_1	-	*Tangencijalni distorzijski koeficijent 1*

p_2	-	*Tangencijalni distorzijski koeficijent 2*
dst	-	Matrica koeficijenta distorzije
f_x	piksel	Fokalna duljina *x* os
f_y	piksel	Fokalna duljina *y* os
c_x	piksel	Optički centar *x* os
c_y	piksel	Optički centar *y* os
mtx	piksel	Matrica kamere
tvec	mm	Vektor translacije koordinatnog sustava
rvec	rad	Vektor rotacije koordinatnog sustava
rotM	rad	Matrica rotacije koordinatnog sustava
s	mm	Faktor skaliranja
u	piksel	koordinata točke na slici *x* os
v	piksel	koordinata točke na slici *y* os
X_W	mm	koordinata točke u prostoru *x* os
Y_W	mm	koordinata točke u prostoru *y* os
Z_W	mm	koordinata točke u prostoru *z* os

SAŽETAK

Tema ovog rada je razvoj petosnog robota paralelne kinematike. Cilj je napraviti robota od početnog koncepta do eksperimentalnog postava koji može izvršavati dane zadatke. Razvoj je krenuo od odabira ulaznih parametra robota u ovisnosti o zadanim ograničenjima zadataka. Napravljena je kinematika robota i odabrana veličina radnog područja. Kroz nekoliko iteracija koncepata robota odabran je jedan koji je išao u daljnju razradu i za koji su provedene simulacije. Iz simulacija su odabrani aktuatori te je iz odabranih aktuatora napravljen ostatak robota. Nakon razrađene konstrukcije, dijelovi robota su išli na izradu te je nakraju robot montiran. Za robot je potom napravljeno upravljanje preko grafičkog sučelja i vizijskih sustava u Python programskom jeziku.

Ključne riječi: petosni robot paralelne kinematike, razvoj, konstruiranje, upravljanje, vizijski sustav

SUMMARY

The topic of this master thesis was the development of a 5-bar parallel kinematics robot. The goal of this thesis was to make a robot from the first concept to a working prototype that is capable of executing given tasks. Development started from picking appropriate robot parameters from constraints given in the task. The kinematics and working area of the robot were made. Through several iterations of robot concepts, one was chosen that went into the detailed development and simulations. Actuators were selected based on these simulations and from the rest of the robot, development was done based on actuators. After robot construction was done, its parts were sent to production and in the end, the robot was assembled. Then, the program for controlling a robot through a graphical user interface was done in Python programming language.

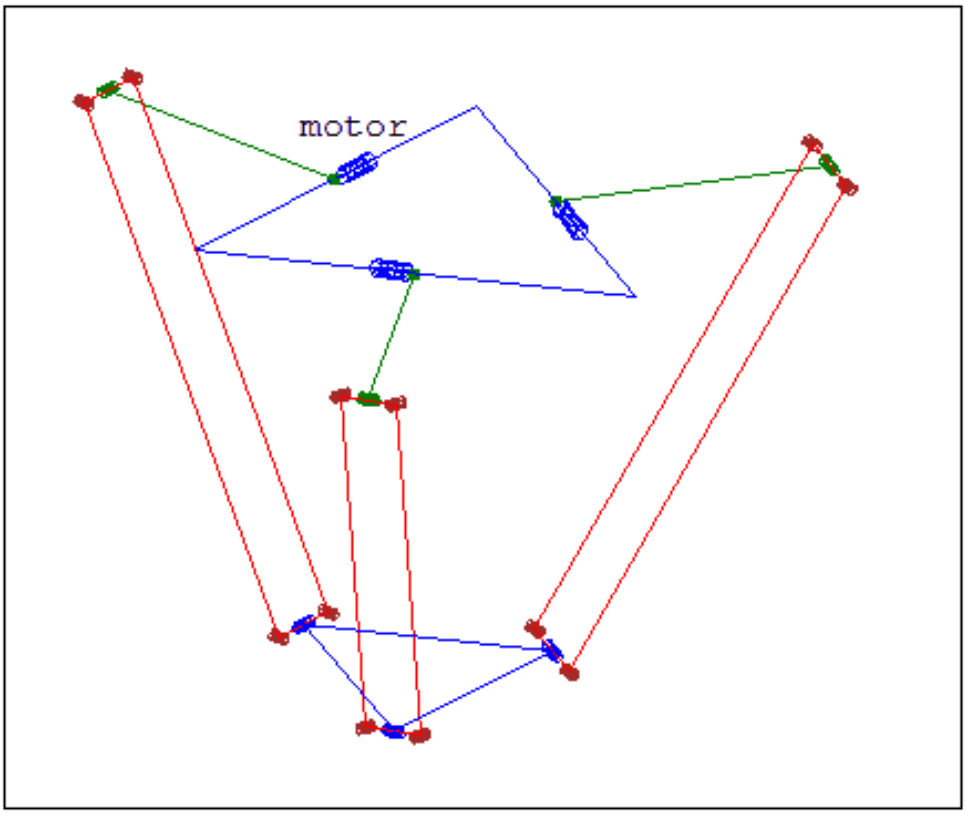
Key words: 5-bar parallel robot, development, design, control, vision system

1. UVOD

Unazad nekoliko godina, može se svjedočiti masovnoj automatizaciji od industrije do svakodnevnog života. Jedan od glavnih pokretača takozvane Industrije 4.0 su roboti. Njihova primjena je različita, od klasične industrijske primjene kao npr. roboti za zavarivanje na pokretnoj traci do kolaborativnih robota koji pomažu pri kupovini namirnica. Iz različite primjene robota nastale su i različite izvedbe robota. Gledajući više industrijske robote, podjela može biti na robote s otvorenim, serijskim, kinematskim lancem te zatvorenim, paralelnim, kinematskim lancem. Kako je tema ovog diplomskog rada robot zatvorene kinematske strukture, u uvodnom dijelu fokus je na generalni opis paralelnih robota, njihove podjele, prednosti i mane.

1.1. Paralelni roboti

Paralelni robot može se definirati kao sistem koji koristi nekoliko (2 ili više) otvorenih kinematskih lanaca koji su aktuirani i spojeni na istu bazu i isti vrh robota. [1] Slika 1 prikazuje kinematsku strukturu robota s paralelnom kinematikom i 3 stupnja slobode gibanja, tzv. delta robot.



Slika 1. Kinematska struktura delta robota[2]

Potrebno je napomenuti kako paralelna kinematska struktura ne implicira na geometrijski paralelizam, nego je to ustaljeni naziv za robote sa zatvorenim kinematskim lancima.

Paralelni roboti uvijek su dizajnirani tako da aktuatori, koji imaju velike mase, budu što bliže bazi robota. Takav dizajn rezultira robotom koji ima pokretne dijelove malih masa i inercija što mu omogućuje velike brzine rada. Nadalje, kako se svaki pokretani kinematski lanac spaja na zajednički radni vrh robota, dobiva se velika krutost vrha robota, koja raste s povećanjem broja kinematskih lanaca. Male mase i inercije zajedno s velikom krutosti omogućuju rukovanje predmetima velikih masa s velikim brzinama. Još jedna prednost takve izvedbe što se greška pozicioniranja ne zbraja nego se dobiva prosječna vrijednost.

S druge strane, kod serijskih robota, aktuator je smješten na kraj prethodnog kinematskog lanca. To dovodi do suprotnog efekta. Povećanjem broja aktuiranih kinematskih lanaca brzina robota pada, inercija raste, krutost pada i greška pozicioniranja raste.

Premda se čini kako je paralelni robot puno bolja opcija od serijskog robota, to nije uvijek istina. Njihova najveća mana je što paralelni robot za iste dimenzije kao serijski robot ima manji radni prostor. Razlog smanjenog radnog prostora je što kod serijskog robota doseg svakog zgloba nije ovisan o ostalim zglobovima, dok kod paralelnog je, stoga pozicija nekog zgloba paralelnog robota u jednom slučaju može biti moguća, a u drugom ne.

1.2. Izvedbe paralelnih robota

Postoje razne izvedbe paralelnih robota, no najpopularnije su heksapodni, delta i *5-bar* robot.

Njihova primjena je različita, npr. heksapodni robot ili drugi naziv *Stewartova* platforma najčešće se koristi za simulaciju leta avionom ili vožnje automobila. Slika 2 prikazuje primjenu heksapodnog robota za treniranje pilota aviokompanije *Lufthansa*.

Slika 2. Heksapodni robot za simulaciju leta[3]

Delta roboti se zbog svoje brzine najčešće koriste za operacije izuzmi-odloži (*eng. pick and place*), gdje robot velikom brzom razvrstava predmete koji dolaze na pokretnoj traci, kao što Slika 3 pokazuje.

Slika 3. Delta robot, operacija izuzmi-odloži[4]

Iako je petosni robot paralelne kinematike pogodan za operacije izuzmi-odloži, njegova popularnost i primjena nije ni približno velika kao u slučaju delta robota. Stoga je samo jedan od većih proizvođača napravio komercijalnu izvedbu takvog robota koji je dostupan za prodaju, a to je *Mitsubishi RP-1AH/2AH/3AH* kojeg Slika 4 prikazuje.

Slika 4. Mitsubishi RP5-AH[5]

2. PETOSNI ROBOT PARALELNE KINEMATIKE

Kako je tema ovog diplomskog rada razvoj petosnog robot paralelne kinematike, kroz ovo poglavlje postavit će se osnovni parametri i ograničenja diplomskog zadatka te će se detaljnije razraditi kinematika navedenog robota i problemi singularnosti.

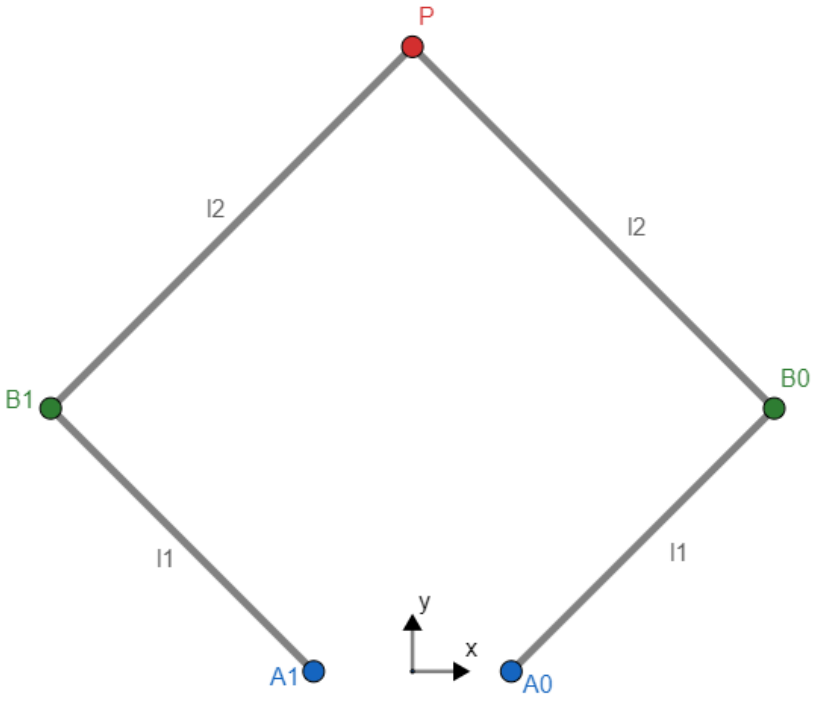
2.1. Parametri robota

Ograničenja i parametri robota zadani zadatkom su:

- Kinematska struktura s dvije upravljačke osi i tri pasivna zgloba
- Radni prostor minimalno 200 mm visine i 250 mm širine
- Izvršni članak robota treba imati vakuumsku hvataljku s minimalno 30 mm hoda
- Nosivost robota na izvršnom članku treba biti 50 g
- Konstrukcija pokretnih dijelova robota treba biti minimizirana radi boljih dinamičkih svojstava, ali uz zadržavanje dovoljne krutosti robota
- Linearna brzina izvršnog članka mora biti minimalno 1 m/s
- Akceleracija izvršnog članka mora biti minimalno 9,81 m/s^2

2.2. Princip rada robota

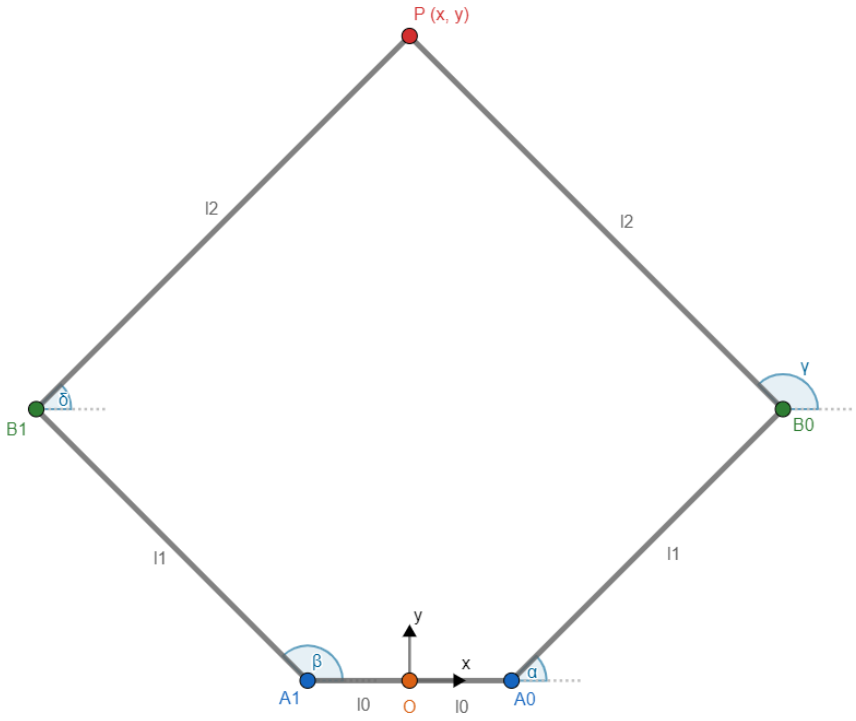
Slika 5 prikazuje pojednostavljenu strukturu petosnog robota paralelne kinematike. Slike koje imaju u svojem opisu referencu [6] izrađene su korištenjem alata s *GeoGebra Geometry*[6]. Slika 5 prikazuje robota koji ima 5 zglobova, od toga su 2 aktivna, A_0 i A_1 te 3 pasivna, B_0, B_1 i P (koji označava vrh robota, tj. izvršni članak). Vidljivo je kako se robot sastoji od 4 pokretna članka, redom A_0B_0, B_0P, A_1B_1, B_1P te jednog nepokretnog članka A_0A_1 koji se smatra nepokretnom bazom robota. Iz toga je jasno otkud naziv *5-bar parallel robot*.



Slika 5. Prikaz petosnog robota paralelne kinematike[6]

2.3. Kinematika robota

Kinematska analiza robota bitna je kako bi se dobio matematički model robota. Svrha matematičkog modela je dobiti poveznicu između ulaznih kutova aktivnih zglobova i vrha robota. U slučaju direktne kinematike to bi značilo da je za zadane ulazne kutove u zglobovima A_0 i A_1 potrebno dobiti položaj vrha robota tj. zgloba P u odnosu na koordinatni sustav robota. Za inverznu kinematiku je obrnuti slučaj, za zadani položaj vrha robota u koordinatnom sustavu robota potrebno je dobiti ulazne kutove u zglobovima A_0 i A_1. Vodeći se modelom kako Slika 5 prikazuje izvest će se matematički model robota. Zglobovi A_0 i A_1 leže na x osi, dok je y os simetrala između zglobova A_0 i A_1. Iako postoje mogućnosti različitih konfiguracija duljina članaka robota, u ovoj konfiguraciji koristit će se iste duljine članaka, tj. l_1 za bliže članke i l_2 za dalje članke čime se dobiva simetričnost robota. **Pogreška! Izvor reference nije pronađen.** n astavak je na prethodnu sliku te će ona poslužiti za izvod direktne i inverzne kinematike.



Slika 6. Petosni robot paralelne kinematike, za izvod[6]

Iz prethodne slike može se postaviti $A_0B_0 = A_1B_1 = l_1$, $B_0P = B_1P = l_2$ i $A_0A_1 = 2l_0$. Kutovi α i β pripadaju aktivnim zglobovima, dok γ i δ pripadaju pasivnim zglobovima. Koordinate vrha robota P imaju vrijednosti x i y.

2.3.1. Direktna kinematika

Rješenje inverzne kinematike može se dobiti sjecištem dvije kružnice, kako Slika 7 prikazuje. Sjecište dvije kružnice rješava se pomoću formula preuzetih sa stranice *http://paulbourke.net*[7]. Dvije kružnice imaju središta u točkama B_0 i B_1 te im je polumjer l_2. Središte kružnice B_0 dobiva se preko jednadžbi (1) i (4), dok se središte kružnice B_1 dobiva preko jednadžbi (3) i (4)

$$x_{B_0} = l_0 + l_1 cos\alpha \quad \textbf{(1)}$$

$$y_{B_0} = l_1 sin\alpha \quad \textbf{(2)}$$

$$x_{B_1} = -l_0 + l_1 cos\beta \quad \textbf{(3)}$$

$$x_{B_1} = l_1 sin\beta \quad \textbf{(4)}$$

Udaljenost između kružnica može se označiti slovom d i izračunava se pomoću jednadžbe (5).

$$d = \sqrt{(x_{b_0} - x_{b_1})^2 - (y_{b_1} - x_{b_1})^2} \quad \textbf{(5)}$$

Udaljenost d se može dobiti zbrajanjem udaljenosti a i b.

$$d = a + b \tag{6}$$

Udaljenosti a i b zajedno s polumjerom kružnice l_2 i visinom h čine 2 pravokutna trokuta stoga vrijede jednadžbe (7) i (8):

$$a^2 + h^2 = l_2^2 \tag{7}$$

$$b^2 + h^2 = l_2^2 \tag{8}$$

Ako se jednadžba (6) uvrsti u jednadžbu (8) te se dobivena jednadžba pokrati s jednadžbom (7) dobiva se jednadžba (9).

$$a = \frac{d}{2} \tag{9}$$

Koordinate točke P_0 dobivaju se pomoću jednadžbi (10) i (11).

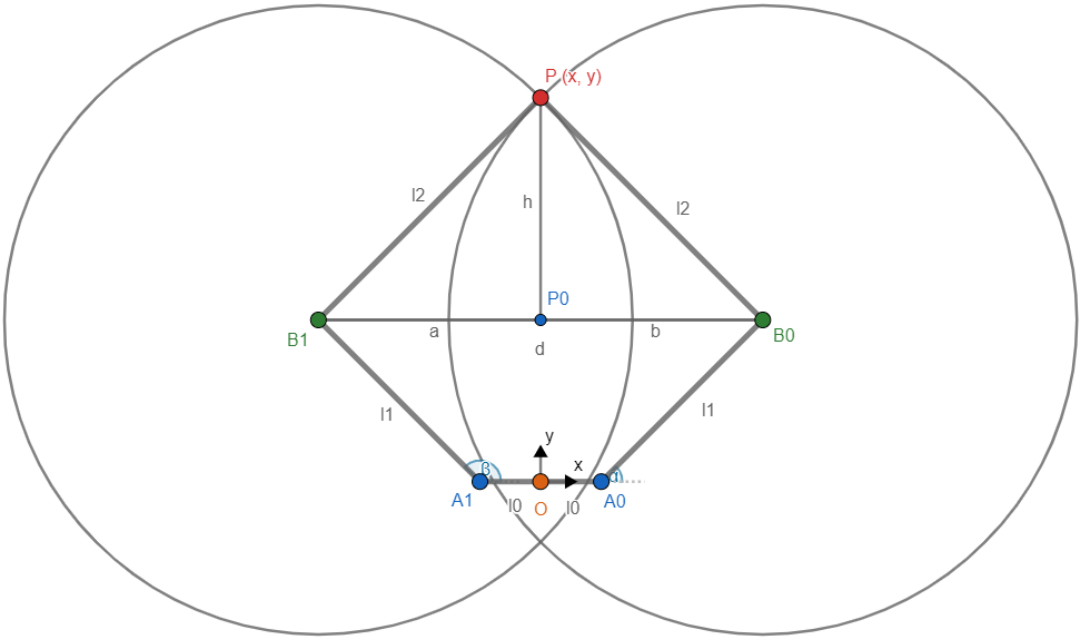
$$x_{P_0} = x_{B_1} + \frac{a(x_{B_1} - x_{B_0})}{d} \tag{10}$$

$$y_{P_0} = y_{B_1} + \frac{a(y_{B_1} - y_{B_0})}{d} \tag{11}$$

Nakraju, koordinate točke P dobivaju se pomoću jednadžbi (12) i (13).

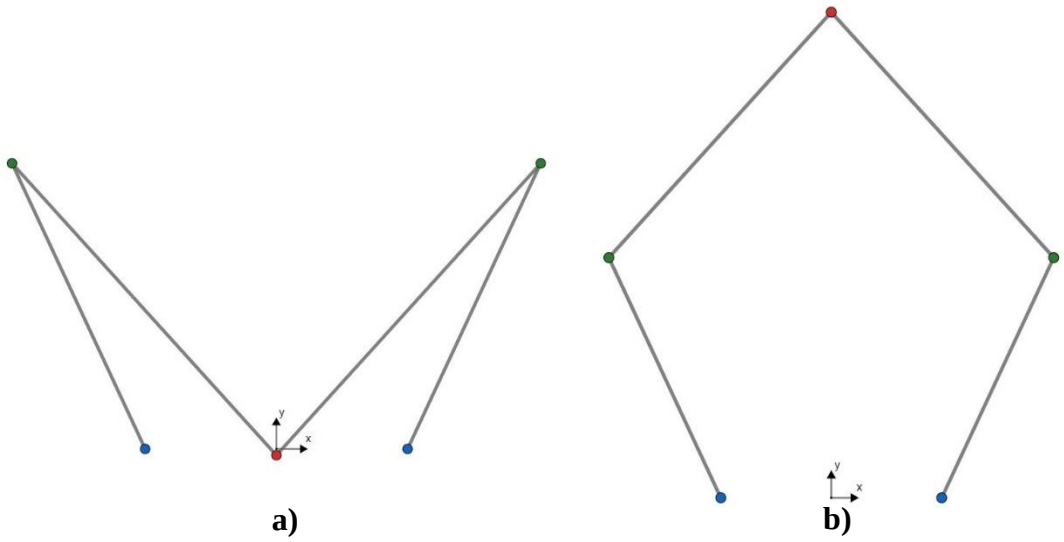
$$x_P = x_{P_0} \mp \frac{h(y_{B_1} - y_{B_0})}{d} \tag{12}$$

$$y_P = y_{P_0} \mp \frac{h(x_{B_1} - x_{B_0})}{d} \tag{13}$$



Slika 7. Izvod direktne kinematike[6]

Kako Slika 7 prikazuje, za udaljenost *d* između dvije kružnice postoje različita moguća rješenja. Kada je ta udaljenost manja od $2l_2$ i veća od 0, postoje 2 moguća rješenja. Zbog tog razloga jednadžbe (12) i (13) imaju dva moguća rješenja. Slika 8 a) i b) prikazuje dva moguća rješenja direktne kinematike, tj. – i + način rada.



Slika 8. Rješenje direktne kinematike: a) – način rada, b) + način rada[6]

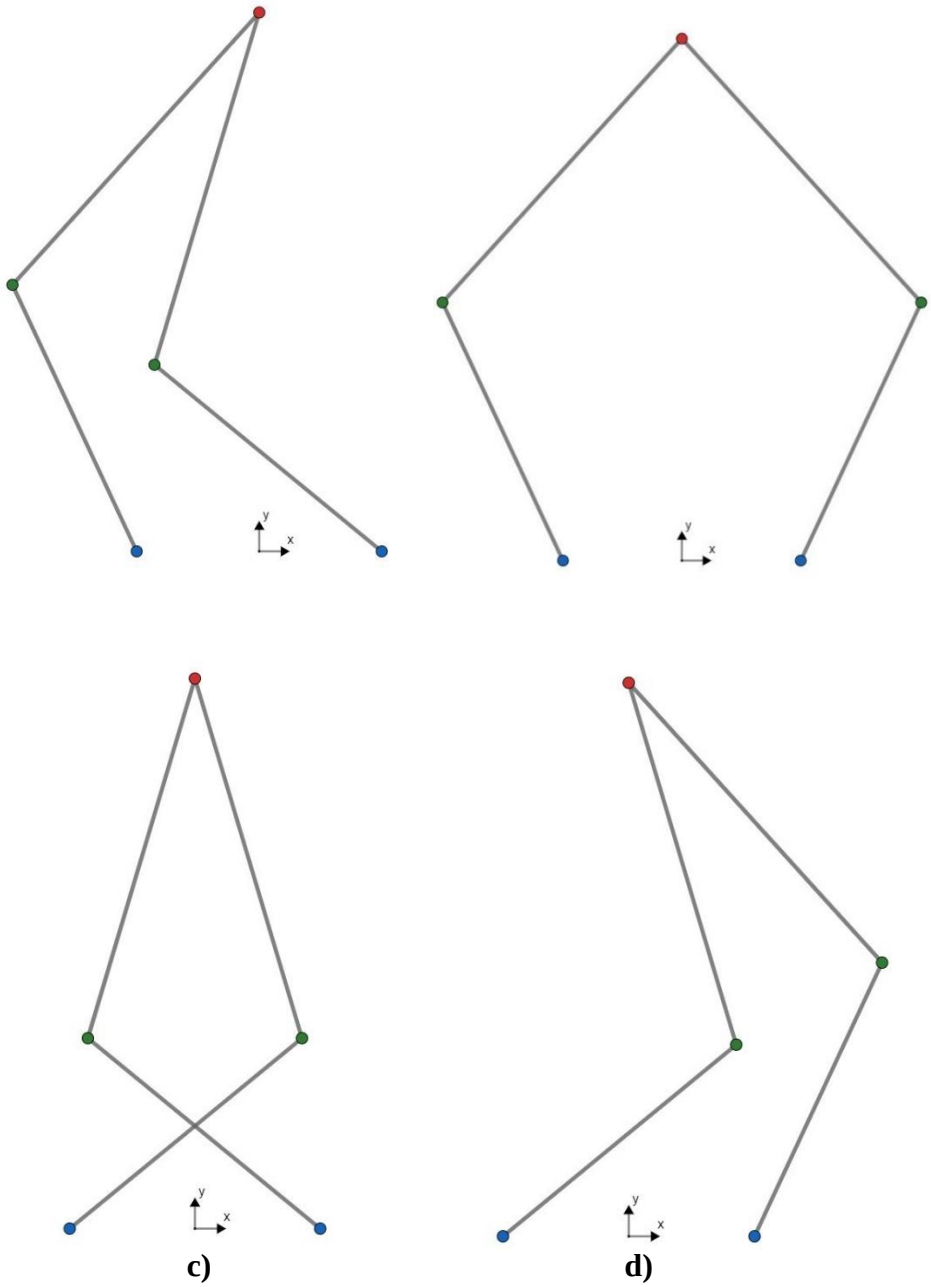
2.3.2. Inverzna kinematika

Za inverznu kinematiku potrebno je dobiti položaj kutova α i β iz zadanog položaja točke *P*. Formule za inverznu kinematiku preuzet su iz *A method for optimal kinematic design of five-bar planar parallel manipulators*[8] te su oznake prilagođene oznakama kako **Pogreška! Izvor r eference nije pronađen.** prikazuje. Jednadžbe (14) i (15) prikazuju rješenja inverzne kinematike te je moguće primijetiti kako postoje 4 moguće kombinacije rješenja inverzne kinematike.

$$\alpha = \operatorname{atan}\left(\frac{y}{l_0 + x}\right) \mp \operatorname{acos}\left(\frac{l_1^2 + ((l_0 + x)^2 + y^2) - l_2^2}{2l_1\sqrt{(l_0 + x)^2 + y^2}}\right) \quad \textbf{(14)}$$

$$\beta = \pi - \operatorname{atan}\left(\frac{y}{l_0 - x}\right) \mp \operatorname{acos}\left(\frac{l_1^2 + ((l_0 - x)^2 + y^2) - l_2^2}{2l_1\sqrt{(l_0 + x)^2 + y^2}}\right) \quad \textbf{(15)}$$

Slika 9 pod a), b), c) i d) prikazuje moguća rješenja inverzne kinematike koji redom predstavljaju ++, +–, – + i — načine rada.



Slika 9. Rješenje inverzne kinematike: a) ++ način rada, b) +- način rada, c) +- način rada, d) – način rada[6]

2.4. Analiza singularnosti

Kako je prethodno spomenuto, jedan od najvećih problema robota sa zatvorenim kinematskim lancem njihov je znatno manji radni prostor u odnosu na robota s otvorenim kinematskim lancem te se taj radni prostor često dodatno smanji zbog problema singularnosti. Stoga je važno

provesti analizu singularnosti kako bi se dobio najbolji mogući radni prostor. Problem singularnosti mogao bi se ukratko opisati kao područje gdje robot gubi jedan ili više stupnjeva slobode gibanja.

Ako se zatvoreni kinematski lanac opiše kao skup ulaznih veličina u formatu vektora **θ** dimenzije *n*, gdje vektor predstavlja *n* aktuiranih zglobova robota, a skup izlaznih veličina u formatu vektora **x** dimenzije *m*, gdje vektor predstavlja *m* izlaznih koordinata vrha robota u kartezijskom koordinatnom sustavu, njihov odnos može se opisati jednadžbom (16)[9]:

$$\mathbf{F}(\boldsymbol{\theta}, \boldsymbol{x}) = \mathbf{0}. \tag{16}$$

Pri tome se pretpostavlja da je broj ulaznih i izlaznih veličina iste dimenzije, inače se radi o redundantnom modelu. Kada se jednadžba (16) parcijalno derivira po vremenu *t* dobije se odnos ulaznih i izlaznih koordinata u obliku:

$$\mathbf{A}\dot{\boldsymbol{x}} + \mathbf{B}\dot{\boldsymbol{\theta}} = \mathbf{0}. \tag{17}$$

gdje **A** i **B** predstavljaju:

$$\mathbf{A} = \frac{\partial \mathbf{F}}{\partial x}, \tag{18}$$

$$\mathbf{B} = \frac{\partial \mathbf{F}}{\partial \theta}. \tag{19}$$

Matrice **A** i **B** predstavljaju *n* x *n* Jakobijan matrice. Kako je navedeno, do singularnosti dolazi kada neka od matrica postane singularna, tj. determinanta joj iznosi 0 što znači da matrice nisu invertibilne i za te slučajeve nema rješenja inverzne kinematike. Zaključuje se da postoje 3 vrste singularnosti:

- det(**A**) = 0
- det(**B**) = 0
- det(**A**) i det(**B**) = 0

Za petosni robot paralelne kinematike matrice **A** i **B** su dimenzija 2x2 te se dobivaju iz sljedećih jednadžbi:

$$l_1^2 = (x - l_0 - l_1 cos\alpha)^2 + (y - l_1 sin\alpha)^2, \tag{20}$$

$$l_1^2 = (x + l_0 - l_1 cos\beta)^2 + (y - l_1 sin\beta)^2. \tag{21}$$

Jednadžbe (20) i (21) parcijalno se deriviraju po vremenu *t* i dobivaju se jednadžbe (22) i (23).

$$(x - l_0 - l_1 cos\alpha)\dot{x} + (y - l_1 sin\alpha)\dot{y} = (l_1 y\, cos\alpha - (x - l_0) l_1 sin\alpha)\dot{\alpha} \tag{22}$$

$$(x + l_0 - l_1 cos\beta)\dot{x} + (y - l_1 sin\beta)\dot{y} = (l_1 y\, cos\beta - (x + l_0) l_1 sin\beta)\dot{\beta} \quad \textbf{(23)}$$

I nakraju se dobije:

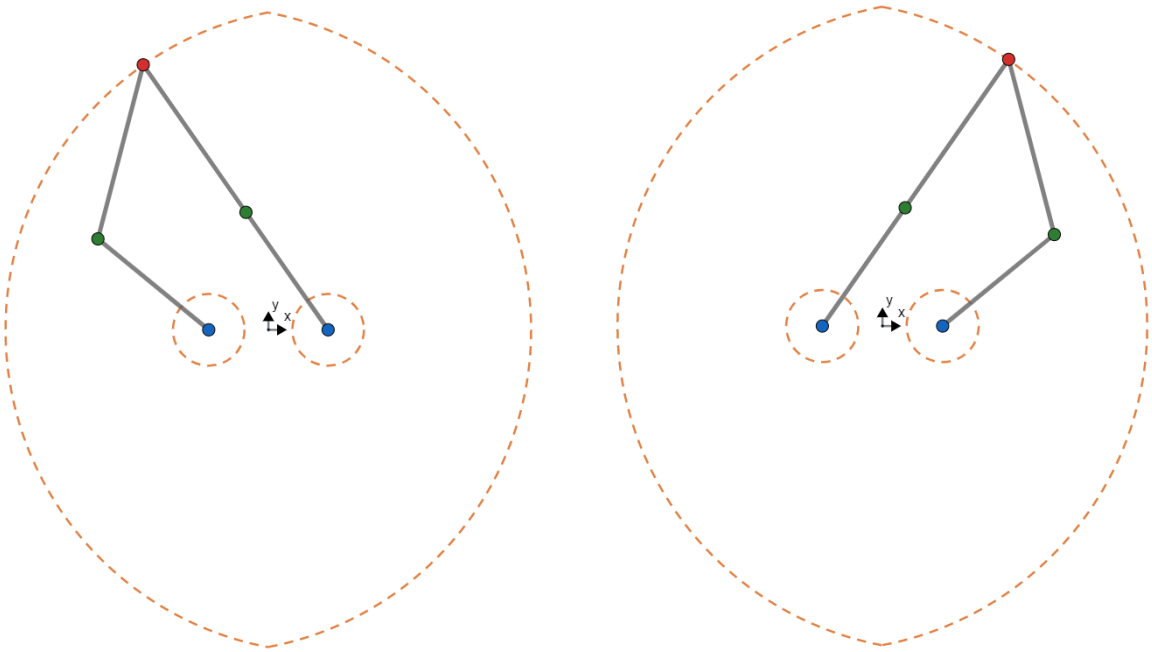
$$\boldsymbol{A} = \begin{bmatrix} x - l_0 - l_1 cos\alpha & y - l_1 sin\alpha \\ x + l_0 - l_1 cos\beta & y - l_1 sin\beta \end{bmatrix}, \quad \textbf{(24)}$$

$$\boldsymbol{B} = \begin{bmatrix} l_1 y\, cos\alpha - (x - l_0) l_1 sin\alpha & 0 \\ 0 & l_1 y\, cos\beta - (x + l_0) l_1 sin\beta \end{bmatrix}, \quad \textbf{(25)}$$

$$\dot{\boldsymbol{x}} = [\dot{x}, \dot{y}], \quad \textbf{(26)}$$

$$\dot{\boldsymbol{\theta}} = [\dot{\alpha}, \dot{\beta}]. \quad \textbf{(27)}$$

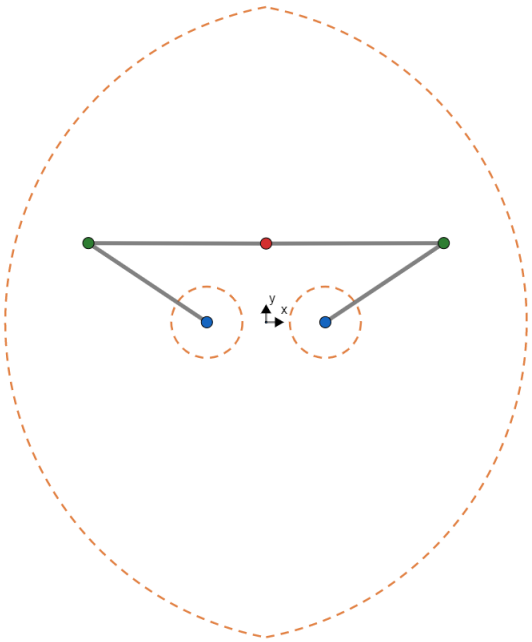
Singularnost prve vrste pojavljuje se kada je det(**B**) = 0. Do toga dolazi kad je jedan od krakova robota potpuno ispružen te vrh robota gubi stupanj slobode gibanja. Singularnosti prve vrste nalaze se na granici radnog područja robota te ju prikazuje Slika 10 a) za desni lanac potpuno ispružen i b) za lijevi lanac potpuno ispružen.



a) **b)**

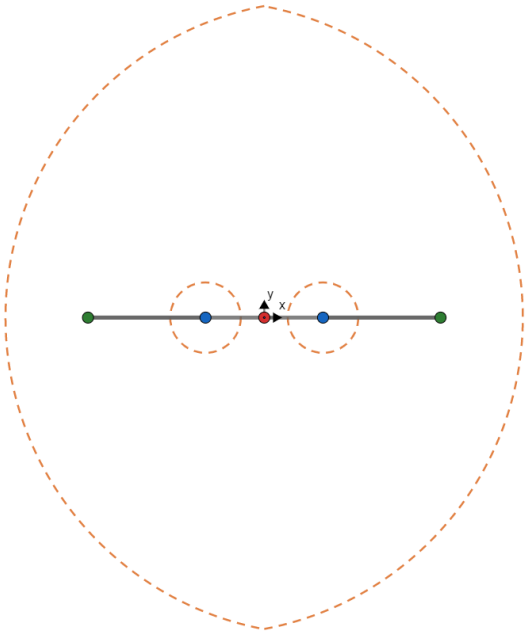
Slika 10. Singularnosti prve vrste, a) desni lanac ispružen, b) lijevi lanac ispružen[6]

Singularnosti druge vrste pojavljuju se kada je det(**A**) = 0. To se događa kada su pasivni članci robota u poziciji kako Slika 11 prikazuje. Dok se singularnosti prve vrste pojavljuju samo na granicama radnog područja robota, singularnosti druge vrste puno su kompleksnije te se mogu nalaziti na raznim pozicijama unutar radnog područja robota.



Slika 11. Singularnosti druge vrste[6]

Treća i zadnja vrsta singularnosti manifestira se kada su det(**A**) = 0 i det(**B**) = 0. To je poseban slučaj kada su svih 5 zglobova (2 aktivna i 3 pasivna) kolinearni kako Slika 12 prikazuje.

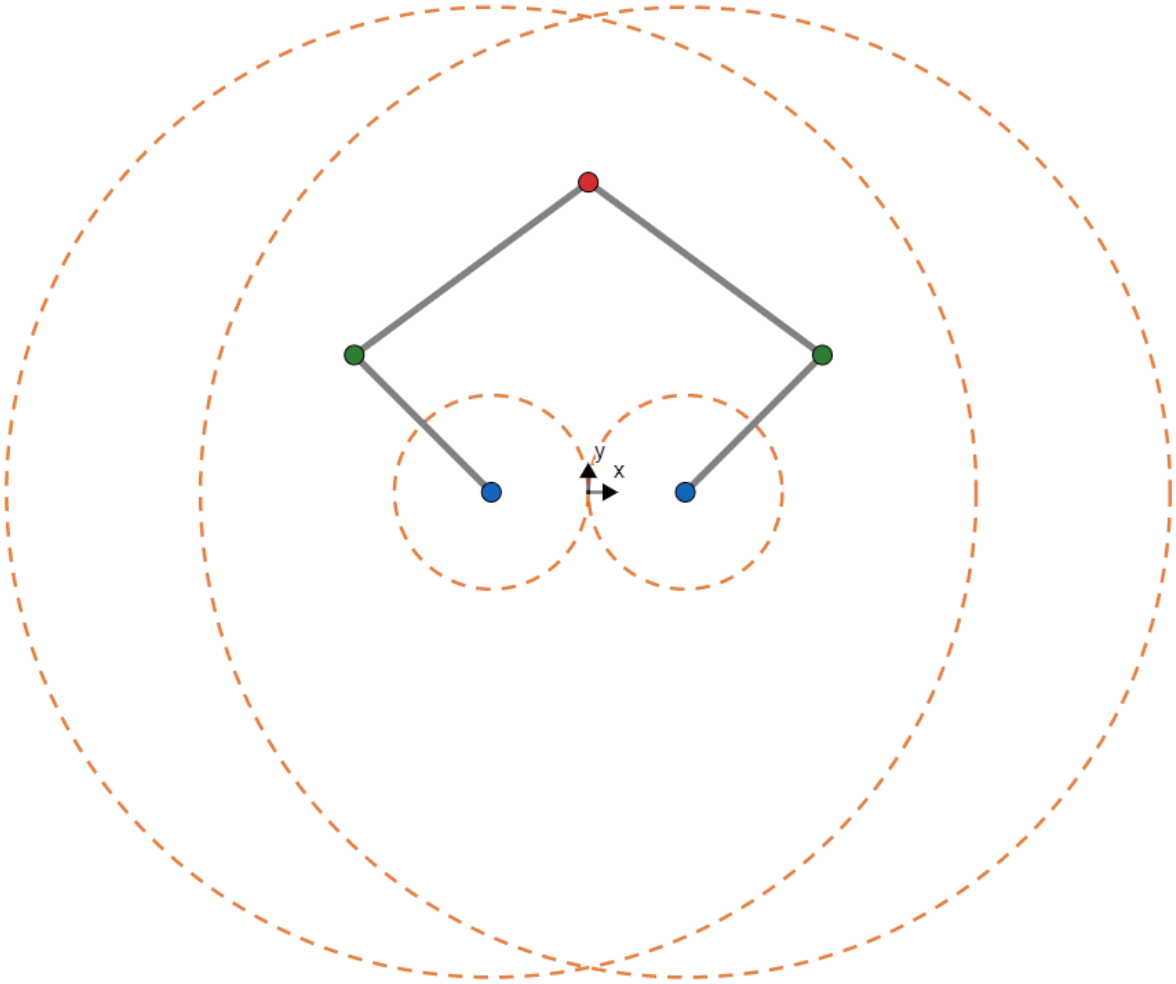


Slika 12. Singularnosti treće vrste[6]

2.5. Radni prostor

Radnim prostorom robota definira se svaka točka u prostoru koju vrh robota može dosegnuti. Kako petosni robot s paralelnom kinematikom ima samo 2 stupnja slobode gibanja, njegovo

radno područje je *xy* ravnina. Svaki kinematski lanac robota sastavljen je od 2 članka te ima radno područje u obliku kružnice s rupom. Kada se 2 kinematska lanca spoje, radno područje petosnog robota sjecište je radnih područja navedenih kružnica s rupom. Ovisno o duljini članaka robota l_1 i l_2 te razmaku između aktivnih zglobova l_0 mogu se dobiti različite konfiguracije robota. Slika 13 prikazuje jednu od mogućih konfiguracija robota. Promjer vanjske kružnice je $2(l_1 + l_2)$, dok je promjer unutarnje $2|l_1 - l_2|$.



Slika 13. Radno područje robota[6]

Presjek 2 velike i 2 male kružnice čini mogući radni prostor robota, ali ne uzimajući u obzir singularnosti i mehanička ograničenja.

2.6. Odabir konfiguracije robota

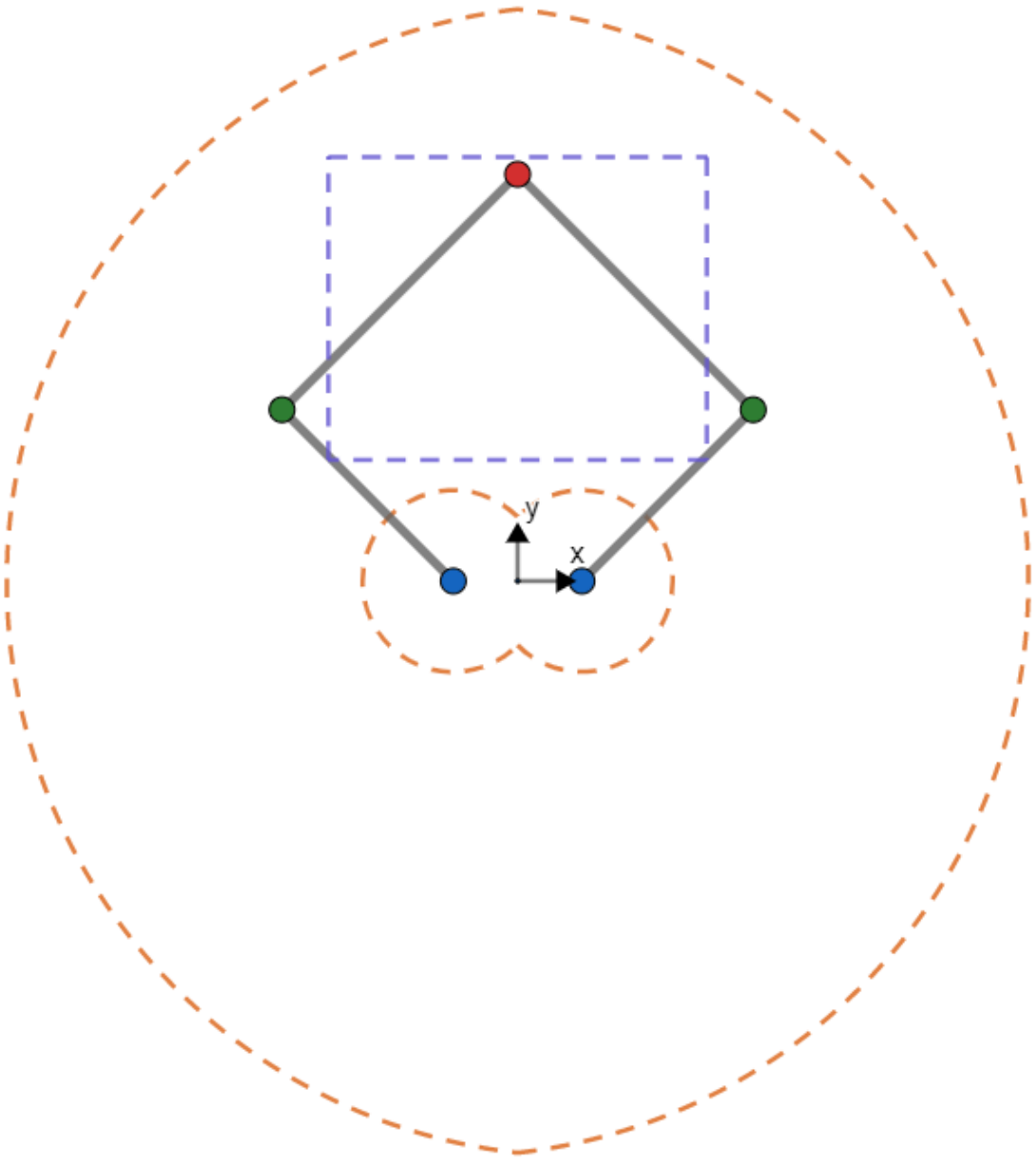
Kako bi se izvršio uspješan odabir parametara robota, u obzir treba uzeti zadatkom zadani minimalni radni prostor od 250x200 mm i ranije navedene singularnosti koje uvelike ovise o duljini članaka robota.

Singularnosti prve vrste rješavaju se ograničavanjem odlaska robota u granice radnog područja. Singularnost treće vrste je ovisno o duljinama članaka robota često i nemoguća te će robot prije imati mehanička ograničenja nego doći u područje singularnosti treće vrste. Najveći problem stvaraju singularnosti druge vrste, koje se mogu javiti na više mjesta kroz radno područje robota. Postoje različita rješenja problema singularnosti druge vrste za petosni robot paralelne kinematike, no većina rješenja ili nije dovoljno zadovoljavajuća ili je previše kompleksna. Jedno zanimljivo rješenje koje je i implementirano u praksi na robotu pod nazivom *DexTAR* radi na principu izmjene načina rada.[10] Kako je i pokazano u poglavlju 2.3.2 postoje različita rješenja inverzne kinematike, tj. ++, +–, –+, — način rada. Svaki način rada ima većinom i svoje singularnosti druge vrste, što znači da robot u nekoj poziciji u jednom načinu rada ima problem singularnosti, ali ako se prebaci u drugi način rada nema. No kako je takvo upravljanje robotom kompleksno izvesti, odlučeno je odabrati duljine članaka robota kako robot ne bi geometrijski mogao doći u singularnosti druge vrste. Ranije spomenuti komercijalni petosni robot paralelne kinematike, *Mitsubishi RP-5AH* ima optimizirano radno područje u kojem su izbjegnute singularnosti druge vrste. Proizvođač navodi pravokutno radno područje A4 papira. Iako bi radno područje za tolike duljine članaka (l_0 = 42.5 mm, l_1 = 200 mm, l_2 = 260 mm) trebale biti znatno veće, ono je ograničeno samom konstrukcijom robota, gdje bi u nekim područjima došlo do kolizije sa samim sobom.[10]

Odabir duljine članaka vođen je *Mitsubishijevim* optimiziranim radnim područjem. Odabrani su kraći krakovi jer je i zadano minimalno radno područje robota manje (200x250 mm naspram 210x297 mm od *Mitsubishija*) te robot ima drugačiju izvedbu članaka koja omogućuje veće radno područje. Odabrane duljine članaka robota za radno područje bez singularnosti druge vrste unutar svog radnog područja su:

- l_0 = 42.5 mm
- l_1 = 160 mm
- l_2 = 220 mm

Slika 14 prikazuje moguće radno područje robota s pozicioniranim pravokutnim radnim područjem od minimalno 200x250 mm. Potrebno je uzeti u obzir da će radno područje biti manje kada se uzmu u obzir konstrukcijska ograničenja, kako je u poglavlju 3.6 pokazano.



Slika 14. Odabrano radno područje petosnog robota paralelne kinematike[6]

3. KONSTRUKCIJA I MONTAŽA ROBOTA

Prije izrade prvih koncepata robota, odlučeno je da će zglobovi biti izrađeni postupkom aditivne proizvodnje, točnije Selektivnim laserskim sinteriranjem (*eng. Selective laser sintering , SLS*), dok će međusobni spoj zglobova biti ostvaren cijevima izrađenim od ugljičnih vlakana. Upotrebom laganih, a krutih materijala poput ugljičnih vlakana moguće je dobiti robot malih masa i inercija te zadovoljavajuće krutosti koji ima mogućnosti postizanja velikih brzina, što je kod ovakvog robota i cilj. Upotrebom aditivne proizvodnje omogućena je brza izrada prototipova složenijih oblika, koji za pravilno odabrani materijal, u ovom slučaju poliamid, imaju zadovoljavajuća konstrukcijska svojstva.

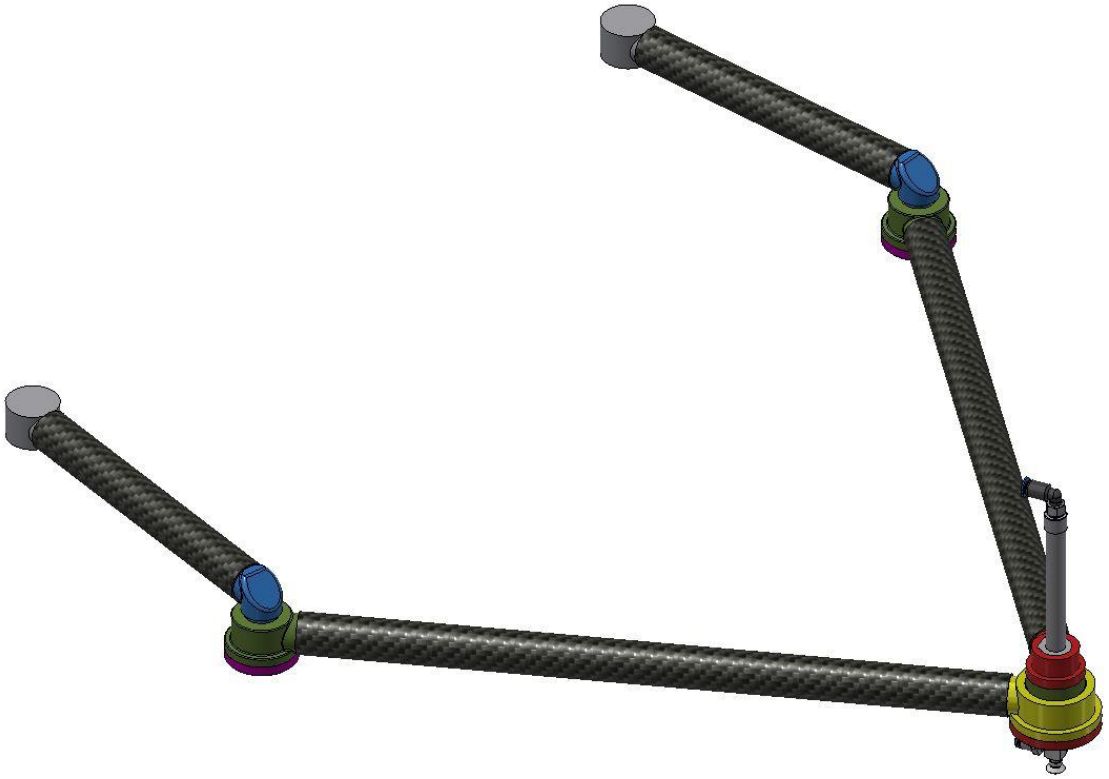
Prvo je napravljeno nekoliko početnih koncepata. Potom se odabrani koncept detaljnije razradio prije izrade robota kao eksperimentalnog postava. Za odabranu verziju robota napravljen je odabir standardnih dijelova te njegove hvataljke. Za dijelove koji se izrađuju aditivnom proizvodnjom, a predstavljaju kritična područja robota napravljena je analiza metodom konačnih elemenata (*eng. Finite element method, FEM)*. Nakon toga, za zadatkom zadanu brzinu i ubrzanje pronađen je potrebni moment za pogon robota. Iz izračunatog momenta odabrani su aktuatori oko kojih se razradio ostatak robota. Nakraju je pokazana montaža samog robota koja će se kasnije koristiti kao eksperimentalni postav.

Konstrukcija i simulacija robota napravljena je u programskom paketu *Autodesk Inventor Professional 2019.*

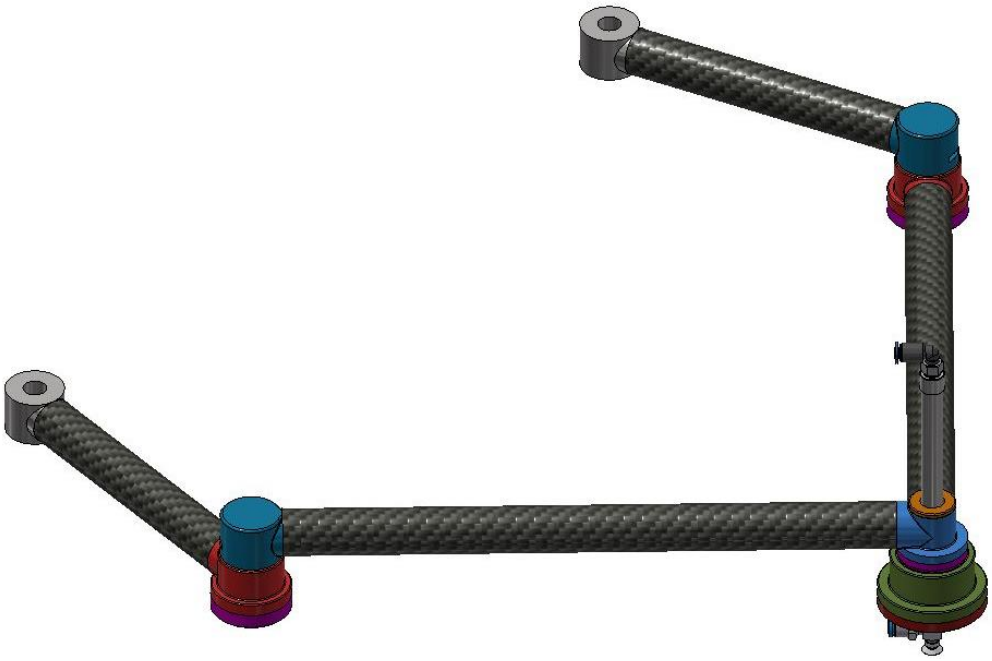
3.1. Prethodne verzije robota

Prije odabira finalne verzije robota, napravljeno je nekoliko koncepata. Kako tijekom izrade koncepata još nisu bile odabrane točne duljine članaka navedene u poglavlju 2.6 tako i postoje razlike u konfiguracijama robota koje prikazuje Slika 15, Slika 16 i Slika 17. Za sve navedene verzije robota postoje *CAD* modeli u *.step* formatu te se nalaze na priloženom CD-u.

Slika 15. Verzija 1



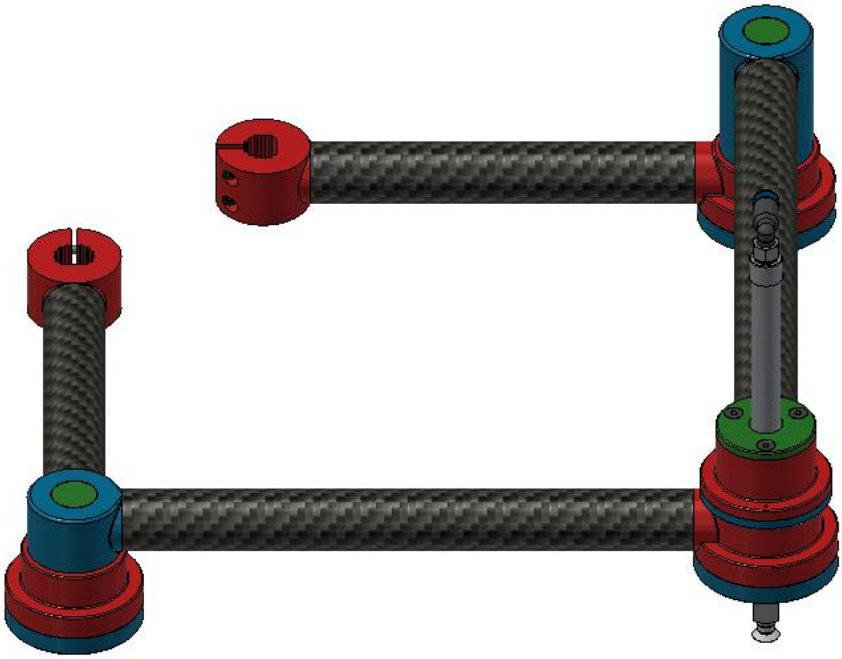
Slika 16. Verzija 2



Slika 17. Verzija 3

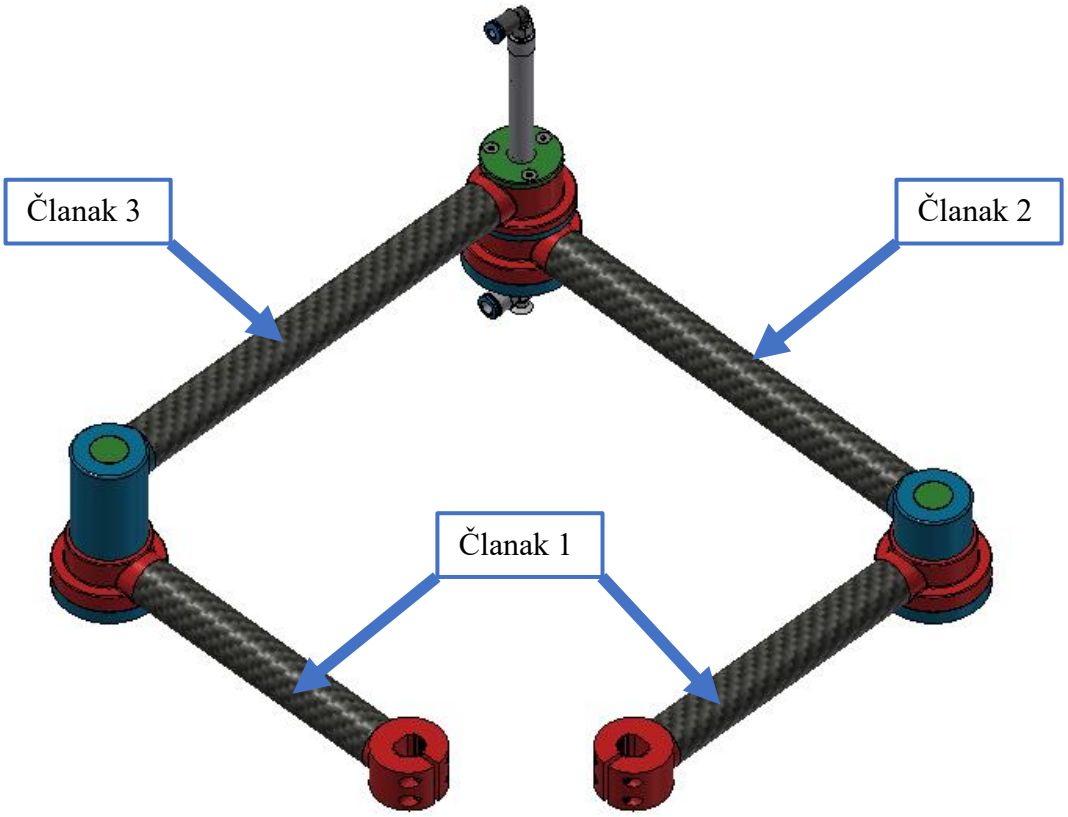
3.2. Odabrana verzija robota

Kroz iteracije koncepata prikazanih u poglavlju 3.1 i za duljine članaka odabrane u poglavlju 2.6 napravljena je konstrukcija robota kojeg prikazuje Slika 18, Slika 19 i Slika 20.



Slika 18. Konačna verzija robota, pogled 1

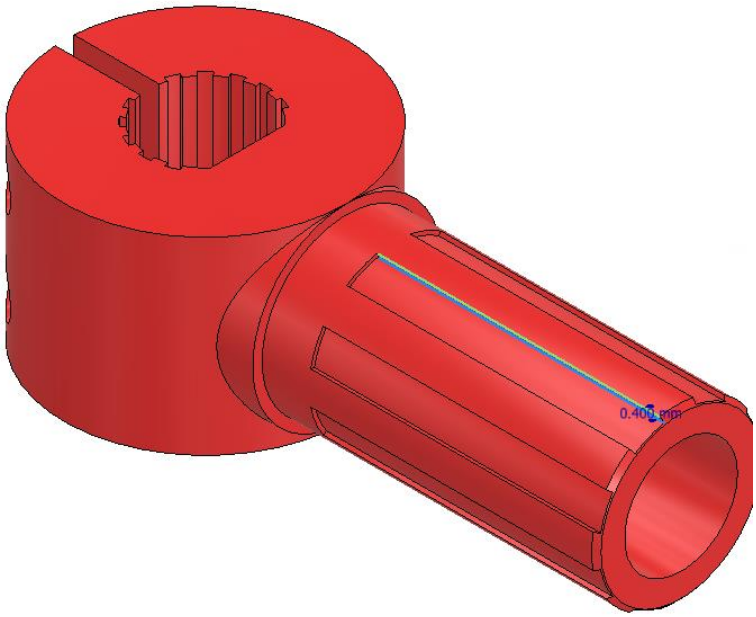
Slika 19. Konačna verzija robota, pogled 2



Slika 20. Konačna verzija robota, pogled 3

3D *CAD* model konačne verzije cijelog robota u *.step* i *.ipt*/*.iam* formatu također se nalazi na priloženom CD-u.

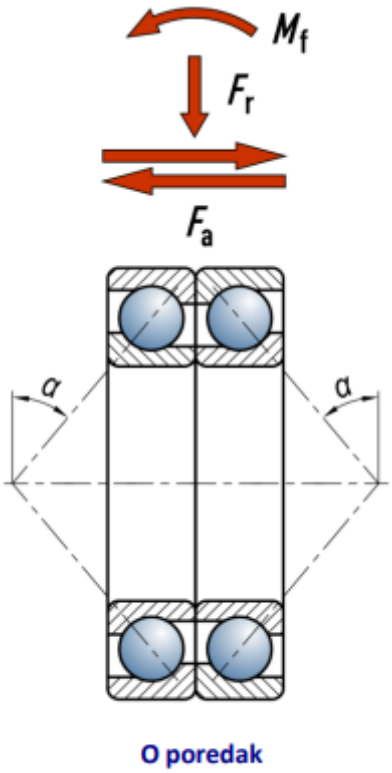
Svi spojevi između pozicija izrađenih aditivnim postupkom i cijevi od ugljičnih vlakana dimenzija 18x20 mm ostvareni su upotrebom dvokomponentong epoksidnog ljepila *Loctite EA 9492*. Iz tog razloga sve pozicije na površinama gdje se ostvaruje spoj lijepljenjem imaju zvjezdasti oblik gdje je dubina dola 0,4 mm te je predviđeno da u taj dol ulazi višak ljepila koji stvara tanki sloj koji poboljšava svojstva lijepljenog spoja. Razlog korištenja *SLS* postupka za izradu zglobova robota naspram klasičnog *FDM* postupka je što *SLS* ima bolju preciznost i laganu hrapavost površine koja pridonosi boljim svojstvima lijepljenog spoja. Slika 21 prikazuje jednu od pozicija sa zvjezdastim oblikom.



Slika 21. Prikaz zvjezdastog oblika

Kako Slika 20 prikazuje, članci robota mogu se podijeliti na 3 dijela. Prvi su dva kraća članka koji se nalaze bliže aktuatorima te su identični. Drugi dio je lijevi dalji članak i treći je desni dalji članak. Članak 2 robota se zajedno sa člankom 3 robota spaja u vrhu na kojem se nalazi vakuumska hvataljka s pneumatskim cilindrom o kojoj se govori u poglavlju 3.3.

Sva 3 pasivna zgloba su revolutna, a kako bi se omogućile glatke kretnje korišteni su jednoredni kuglični ležajevi s kosim dodirom. Između članaka 1 i 2/3 korišten je po jedan par 7902 ležajeva dimenzija 15x28 mm te debljine 7 mm, dok je između članaka 2 i 3 korišten par 7903 ležajeva dimenzija 17x30 mm isto debljine 7 mm. Svaki par ležaja montiran je u „O poredak“ kako Slika 22 prikazuje. Na taj način jedan par ležaja pokriva sve sile i momente koji se javljaju u zglobovima robota.

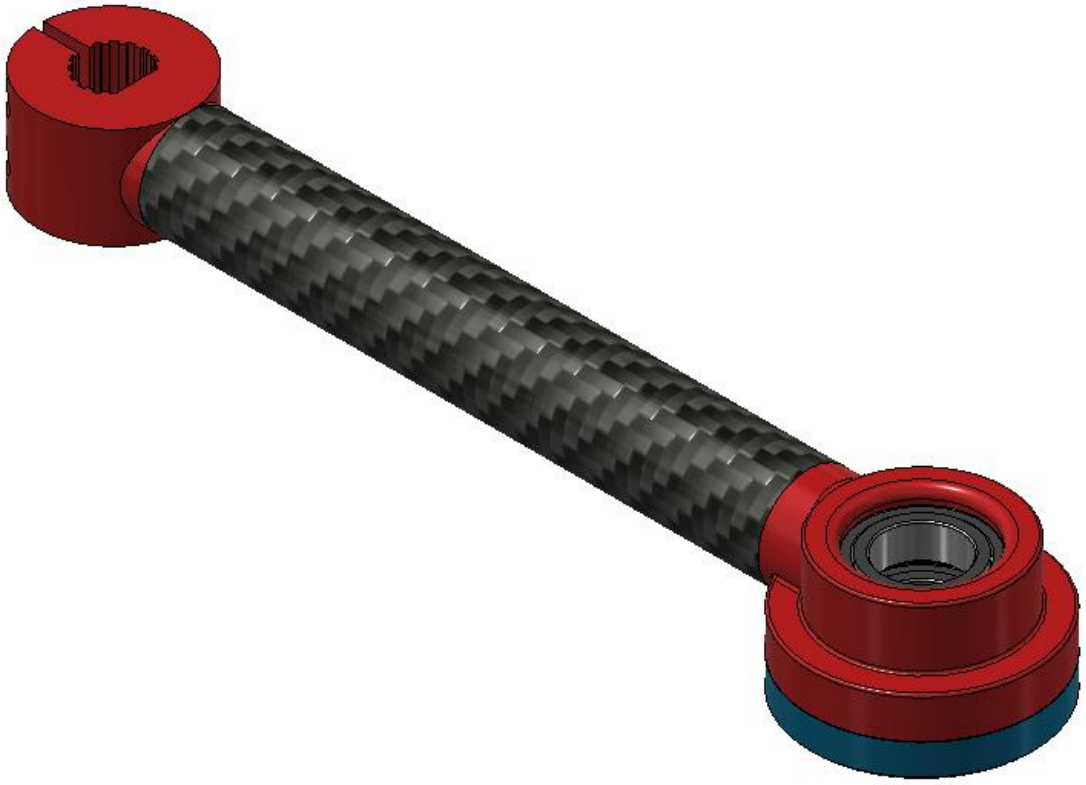


Slika 22. Montiranje ležajeva[11]

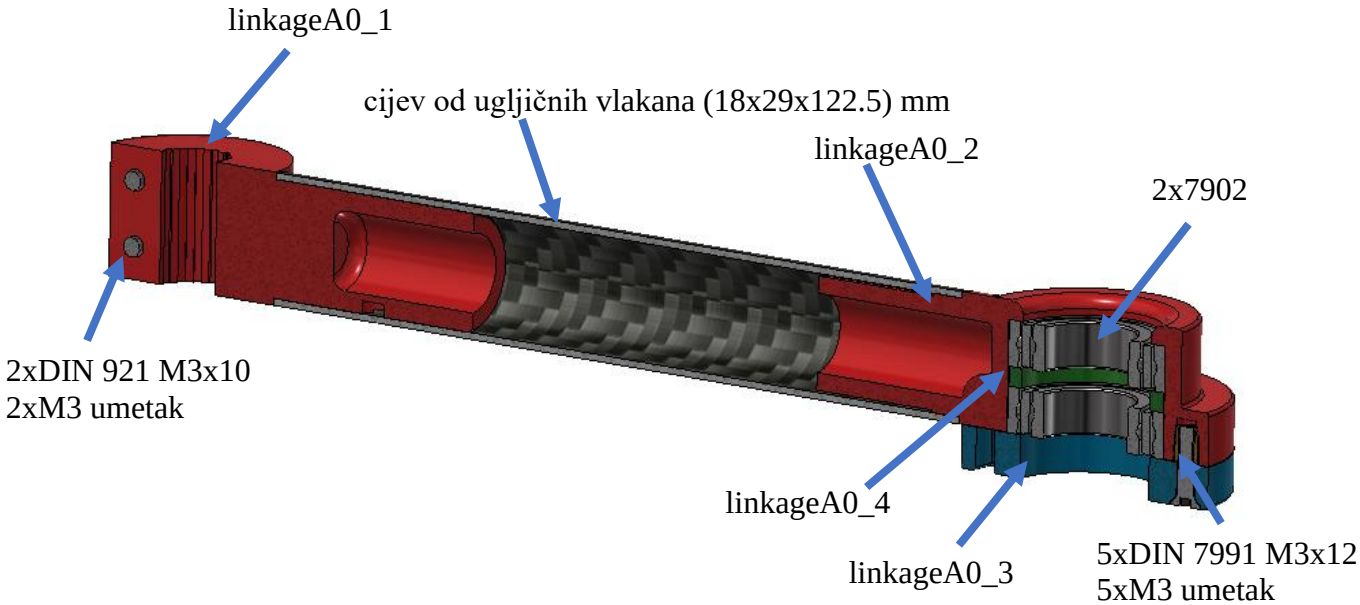
Osim lijepljenja, korišteni su i vijčani spojevi. Kako je navedeno, pozicije za izradu aditivnim postupkom izrađene su od poliamida, koji za razliku od aluminija ili čelika nije pogodan za urezivanje navoja. Iz tog razloga korišteni su toplo uprešani navojni umetci tvrtke *Amtec* veličine M2.5(0932 125 0005) i M3(0932 103 0055). Od ostale spojne tehnike korišteni su vijci DIN 912 i DIN 7991 veličine M2.5, M3 i M5 te samokočna matica DIN 985.

3.2.1. Članak 1 robota

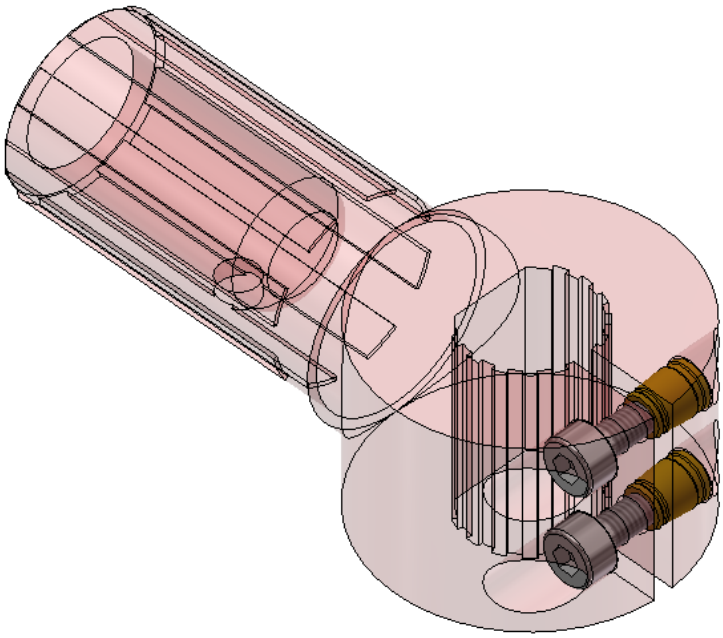
Kako je već spomenuto, robot ima 2 identična članka. Slika 23 prikazuje izgled takvog desnog članka, dok Slika 24 prikazuje presjek članka sa navedenim komponentama. Ovdje se mogu uočiti dva 7902 ležaja koji su uprešani u „linkageA0_2“ i između kojih se nalazi odstojni prsten „linkageA0_4. Vanjski prsten ležaja dodatno se pričvršćuje sa „linkageA0_3“ te je taj spoj ostvaren sa DIN 7991 M3x12 vijcima i M3 umetcima. Slika 25 prikazuje „linkageA0_1“ koji se spaja na vratilo. Osim spajanja vijcima i umetcima za toplo uprešavanje korišten je prijenos momenta oblikom. Kako je korišteni poliamid *PA2200* fleksibilniji materijal, napravljen je zvjezdasti oblik kako bi prilikom stezanja vijcima bio dodir u što više točaka te se tako ostvario bolji prijenos momenta.



Slika 23. Članak 1 robota, pogled



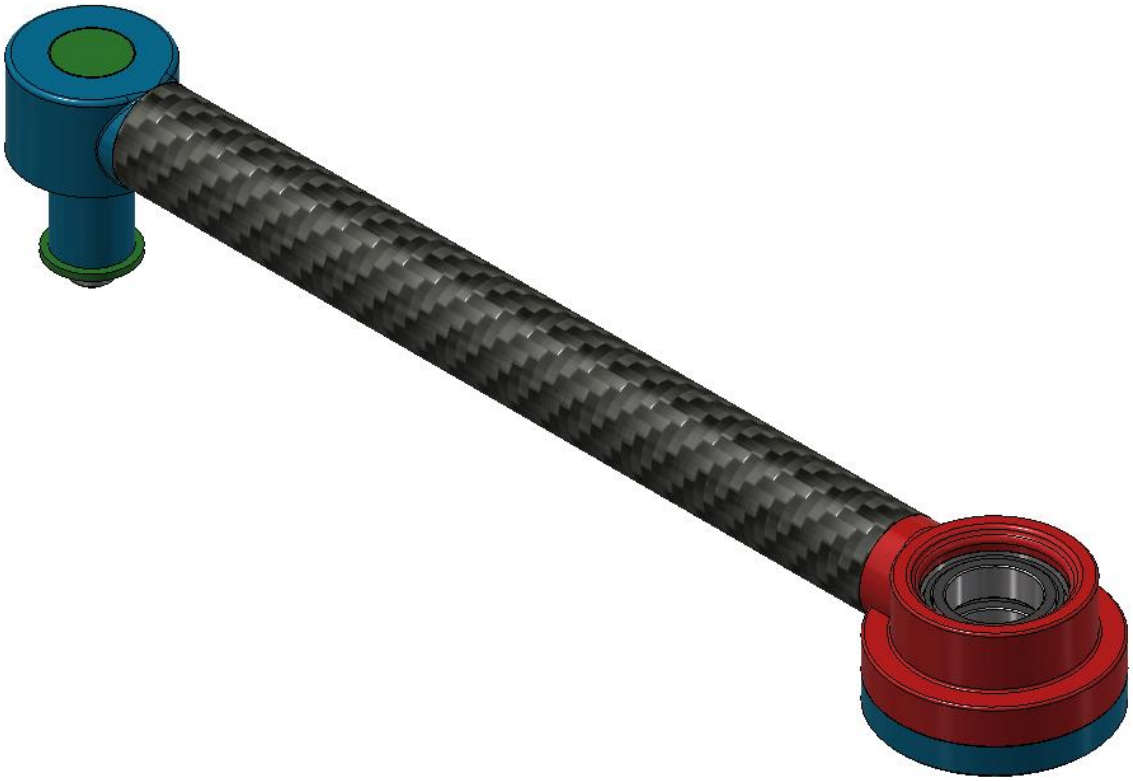
Slika 24. Članak 1 robota, presjek



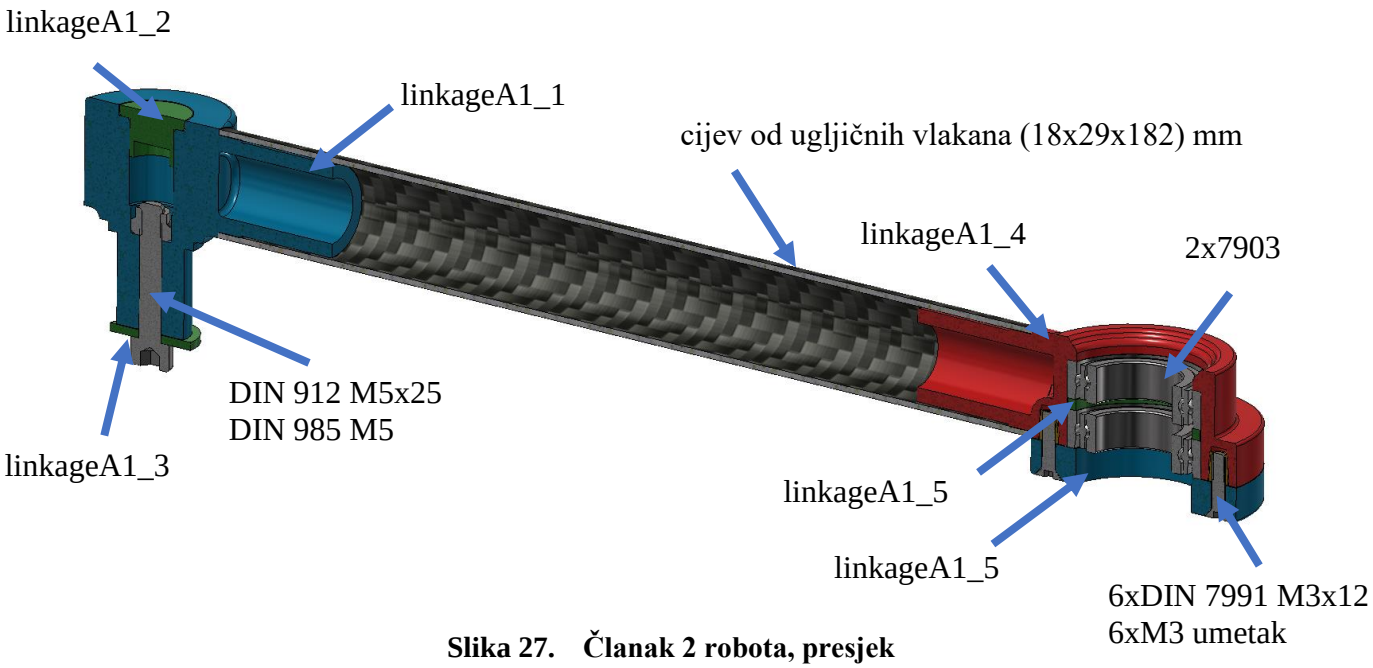
Slika 25. Članak 1 robota, spoj sa vratilom

3.2.2. Članak 2 robota

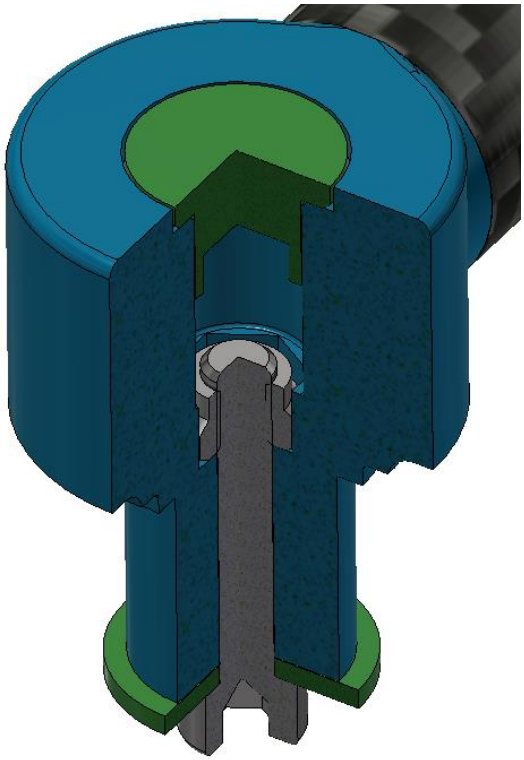
Članak 2 robota je desni dalji i prikazuje ga Slika 26 i Slika 27. Sastoji se od „linkageA1_1“ koji sa člankom „linageA0_1“ čini revolutni zglob preko para 7902 ležajeva. „linkageA1_1“ ostvaruje stezni spoj sa unutarnjim prstenom ležaja 7902 preko DIN 912 M5x25 vijka, samokočne matica DIN 985 i podložne pločice pod nazivom „linkageA1_3“. „linkageA1_3“ je dimenzija 5x18 mm, debljine 1,5 mm te je izrađena iz aluminija laserskim rezanjem. „linkageA1_2“ služi kao poklopac za rupu gdje se nalazi samokočna matica. „linkageA1_1“ svojim oblikom omogućuje međusobno zatezanje samokočne matice i vijka kako Slika 28 prikazuje. Članak 2 robota također ima na svom drugom kraju „linkageA1_4“ koji sa člankom 3 čini revolutni zglob i radni vrh robota na kojem se nalazi vakuumska hvataljka pogonjena pneumatskim cilindrom. Revolutni zglob ostvaren je parom 7903 ležajeva koji su uprešani u „linkageA1_4“ i između kojih se nalazi odstojni prsten „linkageA1_5“. Vanjski ležaj dodatno je osiguran sa „linkageA1_6“ koji je ostvaren vijčanim spojem korištenjem DIN 7991 M3x12 i M3 umetaka.



Slika 26. Članak 2 robota, pogled



Slika 27. Članak 2 robota, presjek

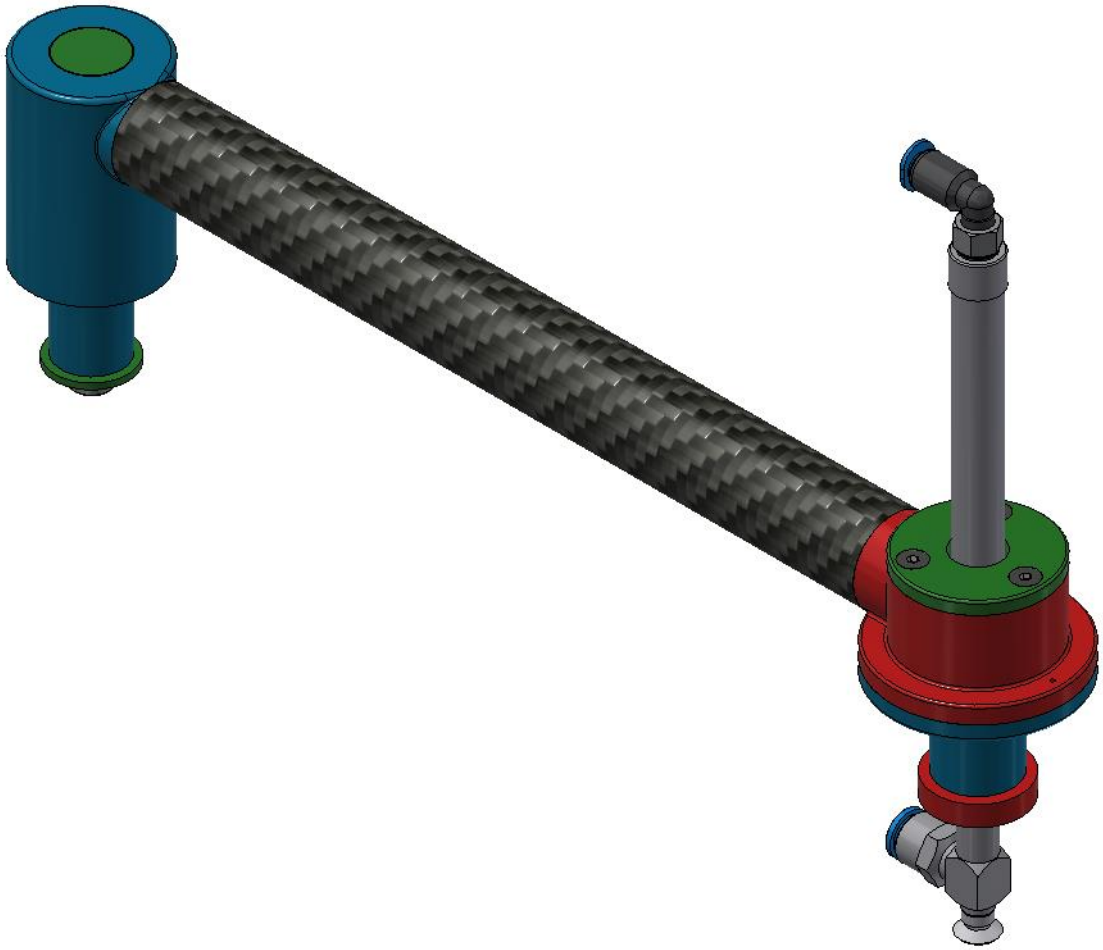


Slika 28. Članak 2 robota, samokočna matica

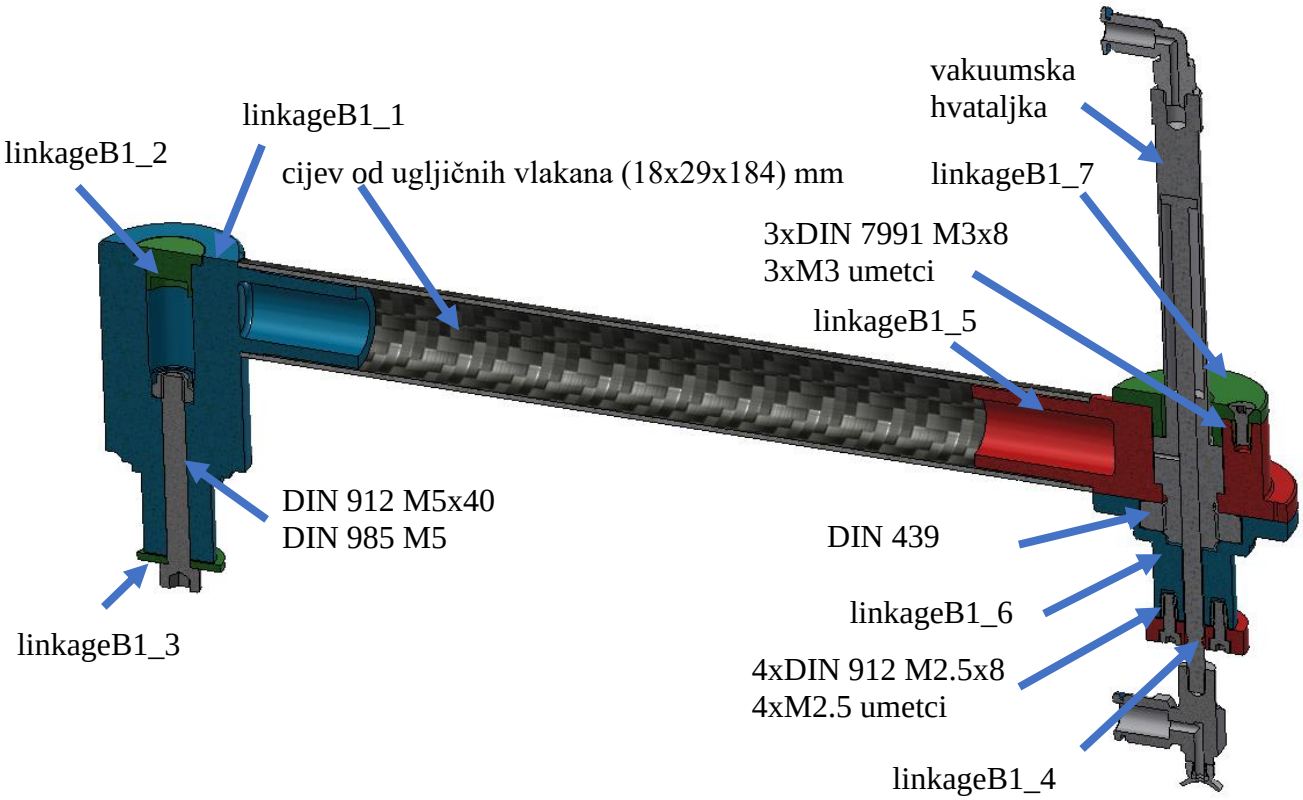
3.2.3. Članak 3 robota

Članak 3 robota je lijevi dalji te ga prikazuje Slika 29 i Slika 30. Može se uočiti kako je „linkageB1_1“ skoro pa isti kao „linkageA1_1“ kod članka 2 robota, osim što je viši za 23,9 mm. Slika 18 prikazuje da je to napravljeno kako bi „linakageA0_1“ i „linkageB0_1“ bili na istoj visini. Zbog toga je korišten i dulji vijak DIN 912 M5x40, dok su matica DIN 985 i podložna pločica od aluminija pod nazivom „linkageB1_4“ ostali isti. Na drugom kraju članka 3 robota se nalazi „linkageB1_5“ koji zajedno sa „linkageB1_6“ čini revolutni zglob sa člankom 2 robota. Može se i uočiti pneumatski cilindar s vakuumskom hvataljkom čiji je odabir detaljnije razrađen u poglavlju 3.3. Pneumatski cilindar na sebi ima M12 navoj na koji ide DIN 439 matica. Ta matica zalijepljena je na „linkageB1_6“ upotrebom istog ljepila kao i za spoj između karbonskih cijevi i poliamida, tj. *Loctite EA 9492*. Pneumatski cilindar prvo se montira na „linkageB1_5“ i stegne sa „linkageB1_7“ korištenjem DIN 7991 M3x8 vijaka i M3 umetaka što prikazuje Slika 30. Zatim se na „linkageB1_6“ koji u sebi ima DIN 439 M12 maticu stegne na pneumatski cilindar i tako čini dio revolutnog zgloba sa „linkageB1_5“. Slika 31 prikazuje

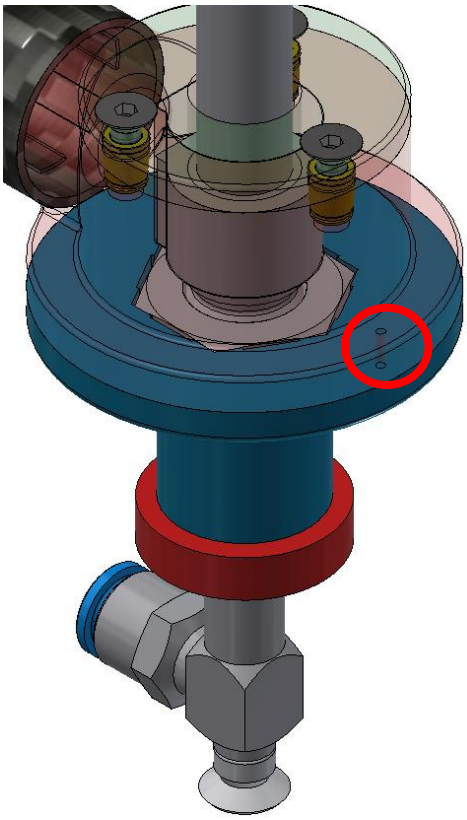
kako su i ovdje napravljeni zvjezdasti utori za stvaranje tankog sloja ljepila. Osim toga vidi se i zaokružena predrupa promjera 1 mm u koju se nakon finalne montaže „linkageB1_5“ i „linkageB1_6“ stavlja pin koji će onemogućiti odvrtanje matice. „linkageB1_6“ ima vratilo promjera 15 mm te na njega nasjeda unutarnji prsten ležaja 7903. Unutarnji prsten ležaja steže se na vratilo pomoću „linkageB1_4“ koje je sa vratilom povezano vijčanim spojem s DIN 912 M2.5x8 i M2.5 umetcima.



Slika 29. Članak 3 robota, pogled



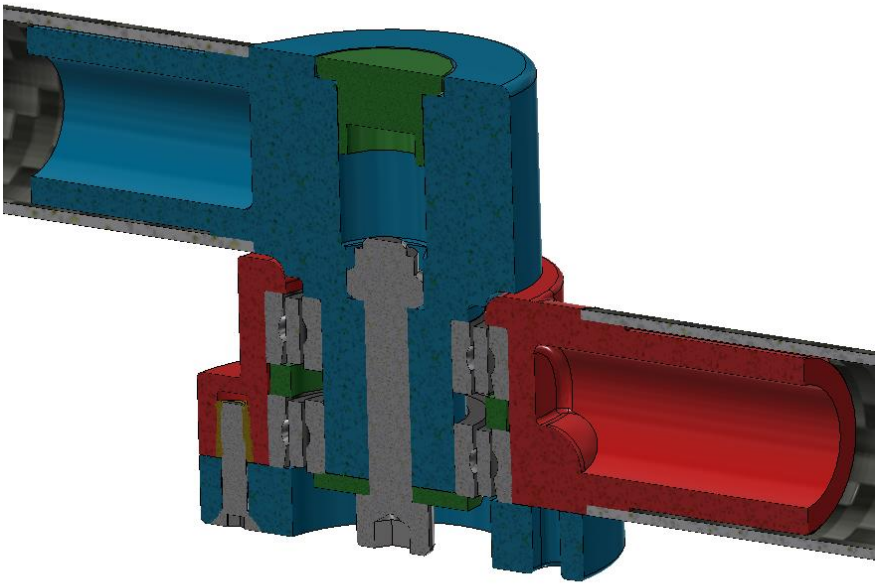
Slika 30. Članak 3 robota, presjek



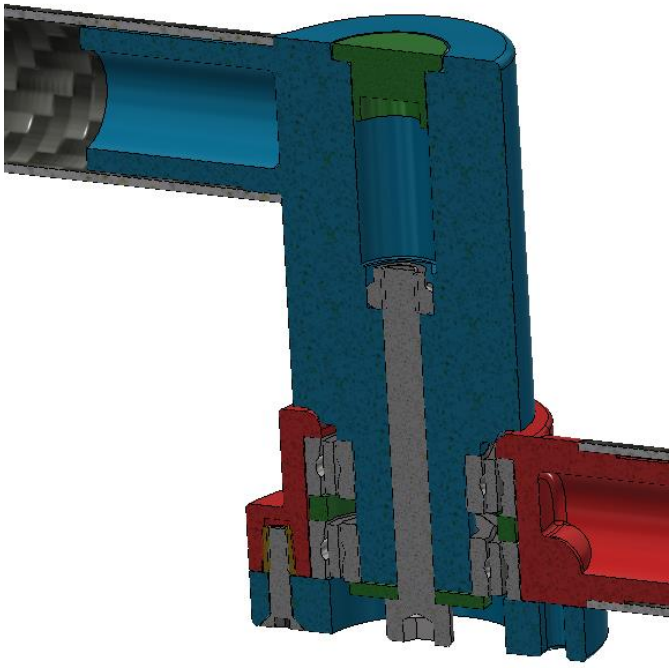
Slika 31. Članak 3 robota, lijepljenje matice

3.2.4. Revolutni zglobovi

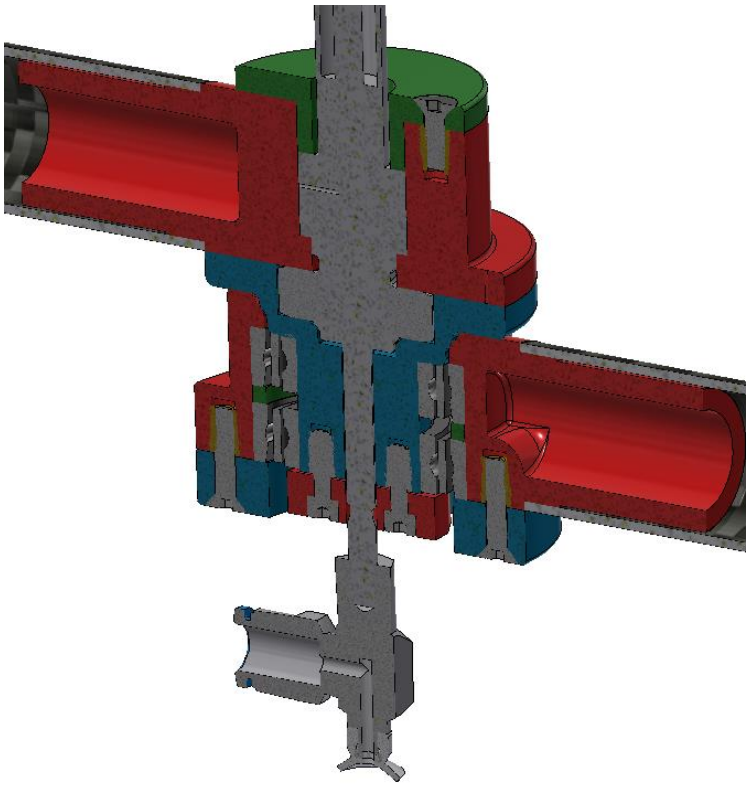
Ovdje je kroz slike pokazano kako izgledaju revolutni zglobovi. Slika 32 prikazuje revolutni zglob između desnog članka 1 robota i članka 2, Slika 33 između lijevog članka 1 robota i članka 3 robota i Slika 34 između članka 2 i članka 3 robota.



Slika 32. Revolutni zglob desni članak 1 i članak 2 robota



Slika 33. Revolutni zglob lijevi članak 1 i članak 3 robota



Slika 34. Revolutni zglob članak 2 i članak 3 robota

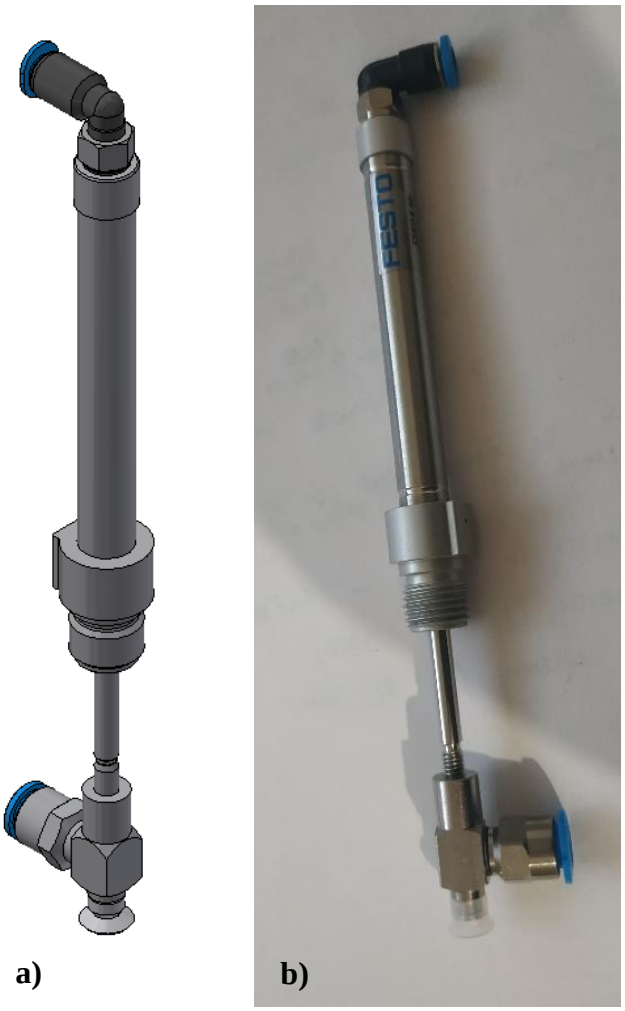
Kako je ranije spomenuto, „linkageA1_3“ i „linkageB1_3“ izrađeni su laserskim rezanjem te je za iste napravljena tehnička dokumentacija koja se nalazi u prilogu [II] pod brojem MB_Diplomski_00. Cijevi od ugljičnih vlakana rezane su korištenjem CNC glodalice te se tehnička dokumentacija isto nalazi u prilogu [II] pod brojem MB_Diplomski_01, MB_Diplomski_02 i MB_Diplomski_03. Ostali elementi su ili standardni ili su napravljeni postupkom aditivne proizvodnje za čiju izradu nije potrebna tehnička dokumentacija. Zadnje što je potrebno napomenuti kako su svi standardni dijelovi do sada prikazani preuzeti iz *Inventor Standard Library* knjižice standardnih konstrukcijskih elemenata.

3.3. Vakuumska hvataljka s aktivnim vertikalnim hodom

Petosni robot paralelne kinematike ima samo dva stupnja slobode gibanja stoga je potrebno dodati i treći stupanj slobode gibanja na koji ide vakuumska hvataljka. Kako je robot namijenjen za operacije izuzmi-odloži na predmetima jednake visine te je poželjno da aktuator koji će pokretati vakuumsku hvataljku bude što lakši kako bi se zadržala mala inercija robota, odabrani aktuator za treću os je jednoradni pneumatski cilindar *Festo ESNU-8-40-P-A-MA-15K8*. Cilindar ima radni hod od 40 mm te s vanjskim promjerom klipa od 8 mm ima male dimenzije. Masa mu je oko 40 g što ga čini dobrim izborom za aktuator vakuumske hvataljke. Zadatkom

je zadano da nosivost robota mora biti minimalno 50 g. Kako je ovo jednoradni cilindar, bitna informacija je koliku povratnu silu opruga cilindra ima pri 40 mm hoda. Iz tehničkih specifikacija cilindra *DSNU_EN*(strana 55)[12] može se očitati da je povratna sila opruge pri 50 mm 2,8 N što znači da sigurno zadovoljava postavljeni uvjet nosivosti robota.

Za vakuumsku hvataljku odabran je *Festo ESG-8-SS-HB-QS*. Vakuumska hvataljka ima na sebi ženski M4 navoj te se montira na pneumatski cilindar koji ima muški M4 navoj. S masom od 13 g lagan je što pridonosi maloj masi i inerciji robota. Iz tehničkih specifikacija vakuumske hvataljke *ESG_EN*(strana 10)[13] može se očitati da odabrana vakuumska kapica promjera 8 mm ima silu držanja od 2,3 N što znači da zadovoljava postavljeni uvjet nosivosti od minimalno 50 g. Za pneumatski cilindar odabran je još *Festo QSML-M5-4* priključak u obliku slova L koji s jedne strane ima muški M5 navoj koji ulazi u ženski M5 navoj cilindra, a s druge sustav za brzu izmjenu pneumatskih cijevi. Slika 35 pod a) i b) prikazuje cijeli sklop sa cilindrom, vakuumskom hvataljkom i L priključkom.



Slika 35. Vakuumska hvataljka: a)3D CAD model, b)sklop

Svi standardni dijelovi koje prikazuje Slika 35 a) preuzeti su sa *https://www.festo.com/cms*[14].

Važno je napomenuti kako vakuumska hvataljka na sebi nema vakuum generator nego je samo nosač vakuum kapice te se kao vakuum generator koristi *SMC ZU07SA*, kojeg prikazuje Slika 36, a spaja se na kompresor s jedne strane i vakuumsku hvataljku s druge.

Slika 36. ***SMC ZU07SA***

Za upravljanje pneumatskim cilindrom i vakuum generatorom koriste se 2 *Festo MZH-3-M3-L-LED* elektromagnetska ventila koje prikazuje Slika 37.

Slika 37. ***Festo MZH-3-M3-L-LED***

3.4. Analiza robota

U ovom poglavlju će se napraviti validacija najkritičnijih pokretnih dijelova koji su izrađeni postupkom aditivne proizvodnje. FEM analiza i Dinamička analiza također su rađene u programskom paketu *Autodesk Inventor Professional 2019* u modulima *Inventor Dynamic Simulation* i *Inventor Stress Analysis*. Kako je računalo na kojem je simulacija napravljena ograničenih performansi, korišten je pojednostavljeni model u kojem su uklonjeni svi standardni dijelovi poput vijaka, matica i ležajeva te odstojnih prstenova i podložnih pločica od aluminija. Umjesto njih, u programu su naknadno dodani utezi jednakih masa kao navedeni

standardni dijelovi. Tako su na „linkageA0_1“ i „linkageB0_1“ dodani utezi od 4 g, dok je na revolutni zglob između desnog članka 1 i članka 2 robota dodan uteg mase 49,4 g, između lijevog članka 1 i članka 3 uteg mase 51,4 g i između članka 2 i članka 3 robota dodani je uteg mase 63 g. Kako je ranije spomenuto dijelovi robota koji će se testirati su izrađeni *SLS* postupkom od materijala poliamid trgovačkog naziva *PA2020* te se njegova mehanička svojstva mogu iščitati iz dokumenta koji se nalazi u prilogu pod [III].

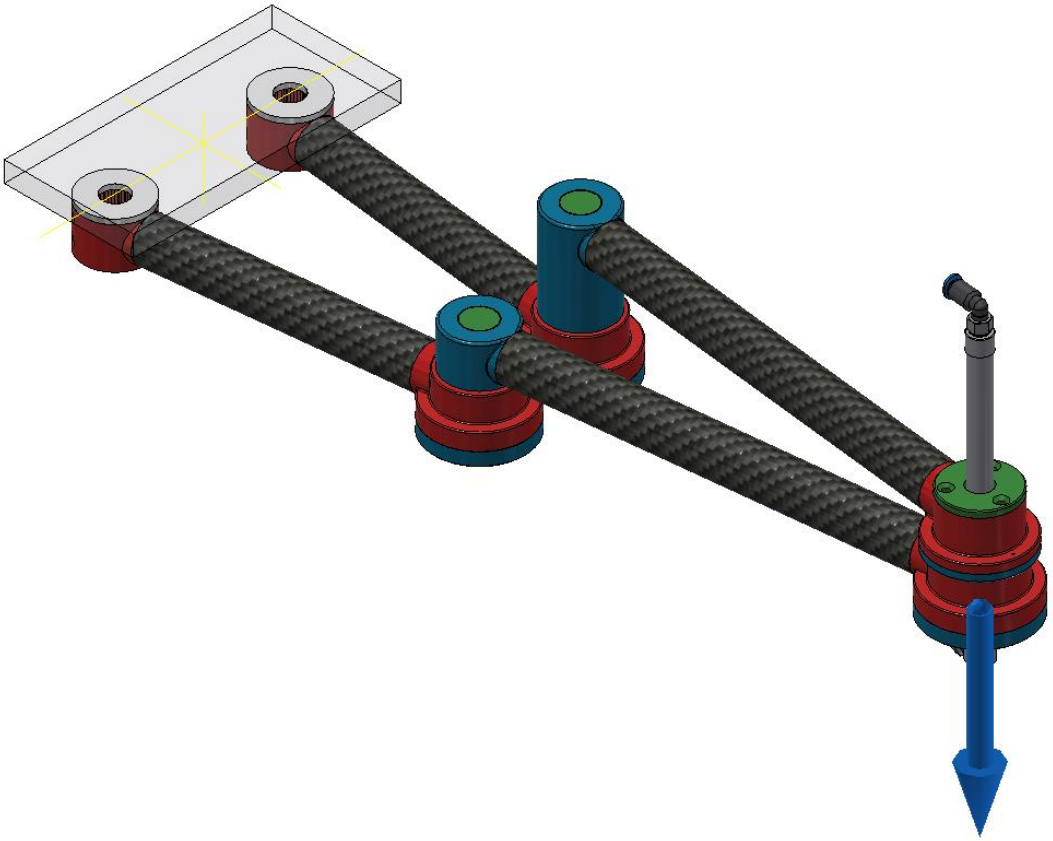
CAD model verzije robota za simulaciju u *.step* i *.ipt*/*.iam* formatu se također nalazi na priloženom CD-u.

3.4.1. FEM analiza 1

Prva *FEM* analiza napravljena je za slučaj kada se robot nalazi u najgorem mogućem položaju, tj. potpuno ispružen. *FEM* analiza napravljena je tako da je na pojednostavljeni model dodana masa u obliku utega kao kompenzacija za uklonjene standardne dijelove. Zatim je u okruženju *Inventor Dynamic Simulation* dodana masa na vrh robota. Zadatkom je zadana minimalna nosivost vrha robota od 50 g, no ovdje je radi sigurnosti korištena dodana masa od 100 g. Nakon postavljanja masa i dodavanja utjecaja gravitacije, u *Inventor Dynamic Simulationu* pokrene se simulacija u kojoj robot miruje u navedenom položaju. Zatim se svaki odabrani dio simulira u *Inventor Stress Analysis* okruženju s opterećenjima koje su dobivene u *Inventor Dynamic Simulationu.* U simulaciji će se za odabrane dijelove gledati Von Mises naprezanje koje će se usporediti s Youngovim modulom elastičnosti, koji se može iščitat iz priloga [III] te on iznosi za *PA2200* 1500 MPa. Osim Von Mises naprezanja gledaju se i deformacije. Pokazat će se rezultati simulacija za najkritičnije dijelove, a to su:

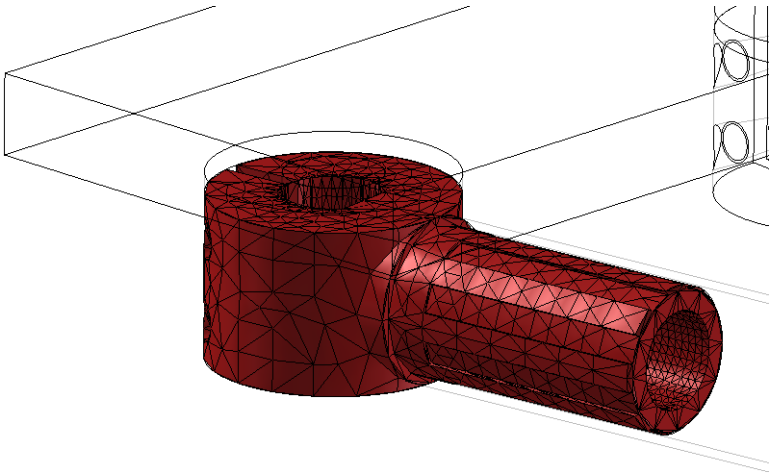
- „linkageA0_1“
- „linkageA0_2“
- „linkageA1_1“
- „linkageB1_1“

Slika 38 prikazuje položaj robota u poziciji u kojoj će se izvoditi simulacija.

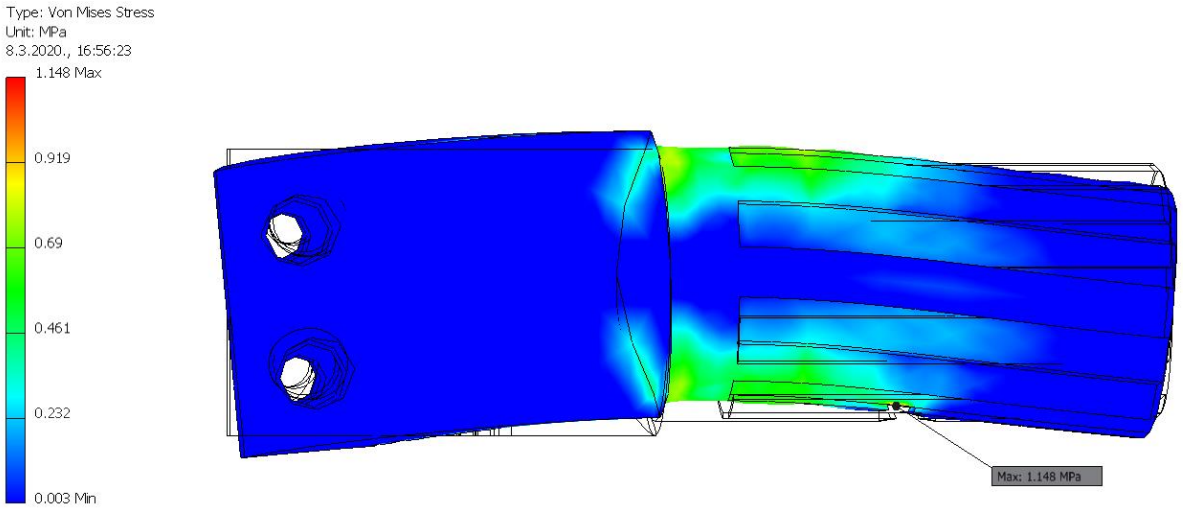


Slika 38. Položaj za FEM analizu 1

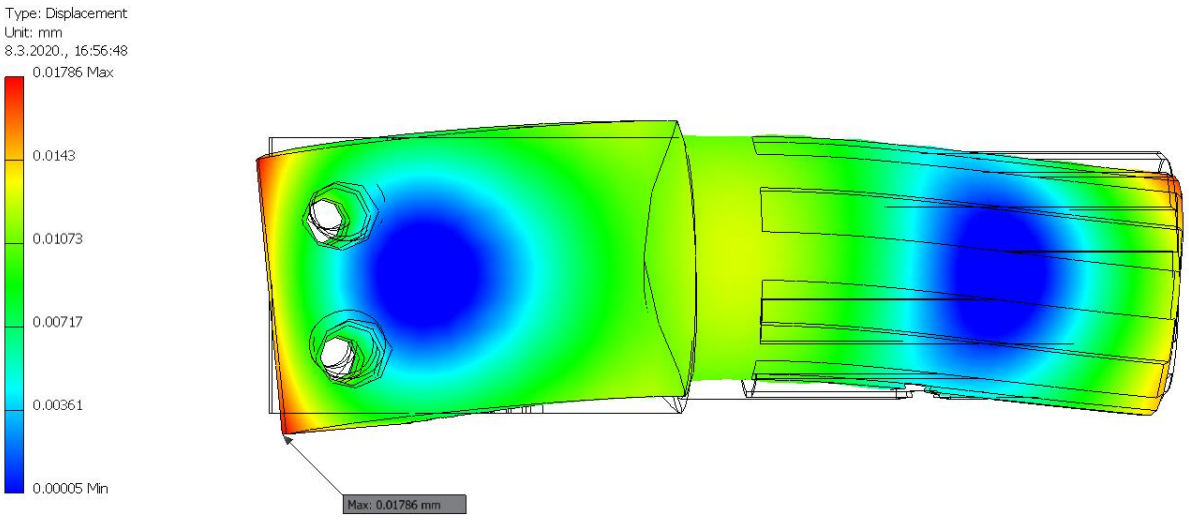
Slika 39 prikazuje izgled „linkageA0_1“ kada je diskretiziran na konačne elemente. Slika 40 prikazuje Von Mises naprezanja i može se očitati maksimalni iznos od 1,148 MPa. Slika 41 prikazuje deformacije i može se očitati maksimalni iznos od 0,01786 mm.



Slika 39. Diskretizirani „linkageA0_1“

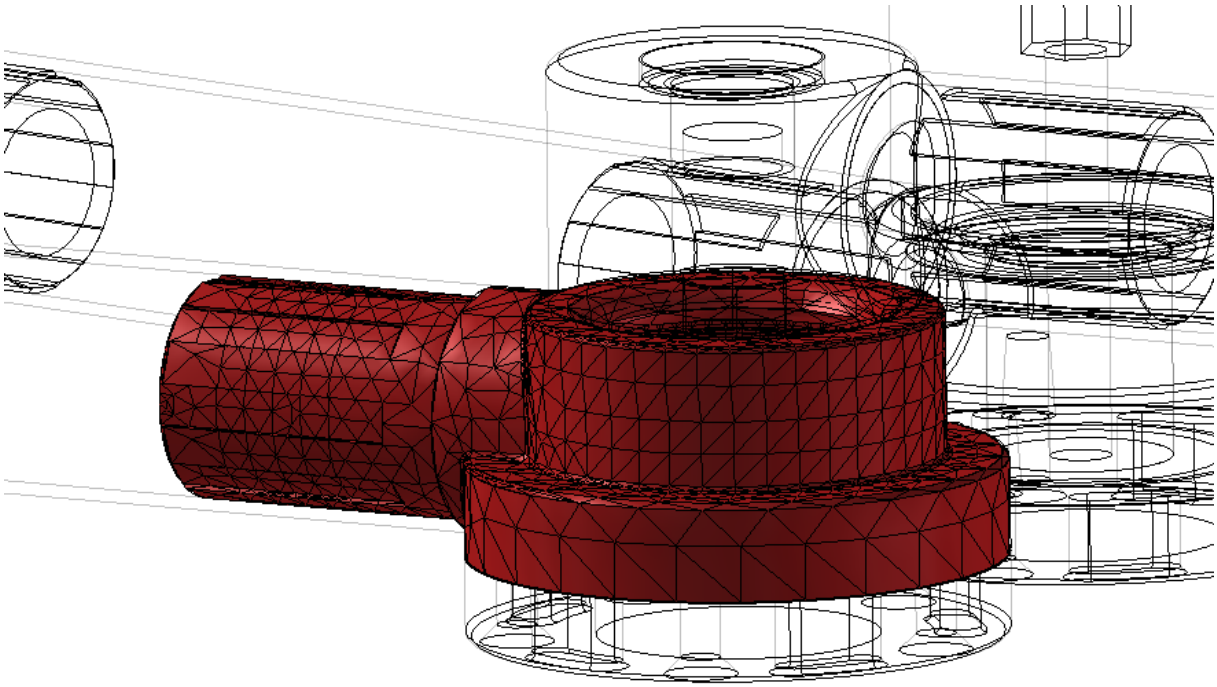


Slika 40. Von Mises naprezanja za „linkageA0_1“, MPa

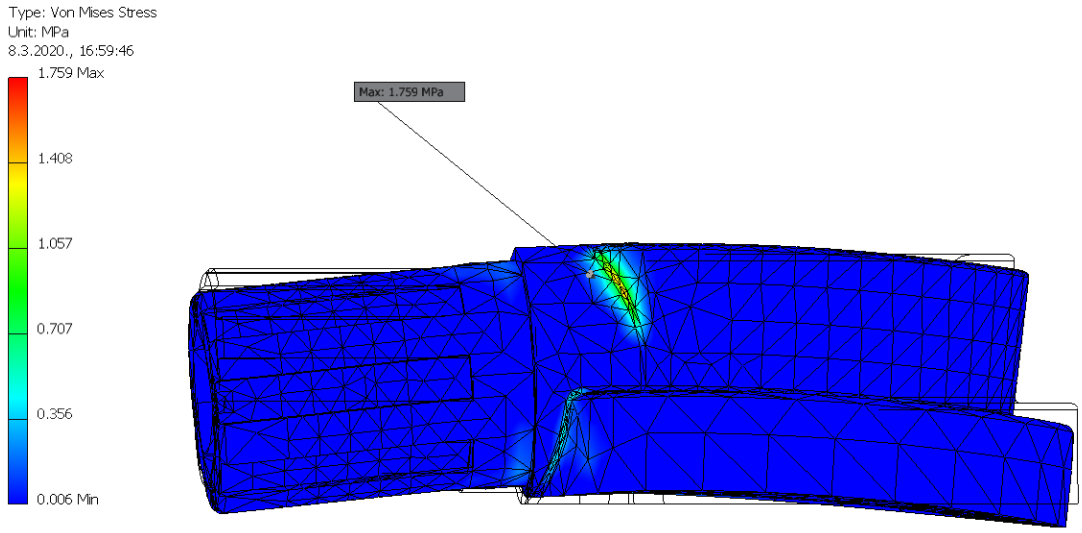


Slika 41. Deformacije za „linkageA0_1“, mm

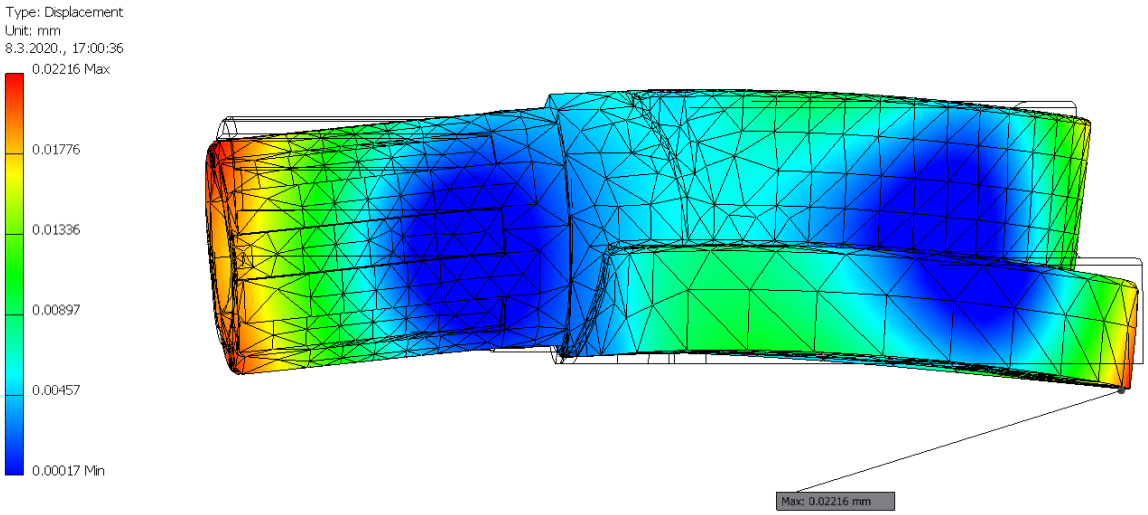
Slika 42 prikazuje izgled „linkageA0_2“ kada je diskretiziran na konačne elemente. Slika 43 prikazuje Von Mises naprezanja i može se očitati maksimalni iznos od 1,759 MPa. Slika 44 prikazuje deformacije i može se očitati maksimalni iznos od 0,02216 mm.



Slika 42. Diskretizirani „linkageA0_2“

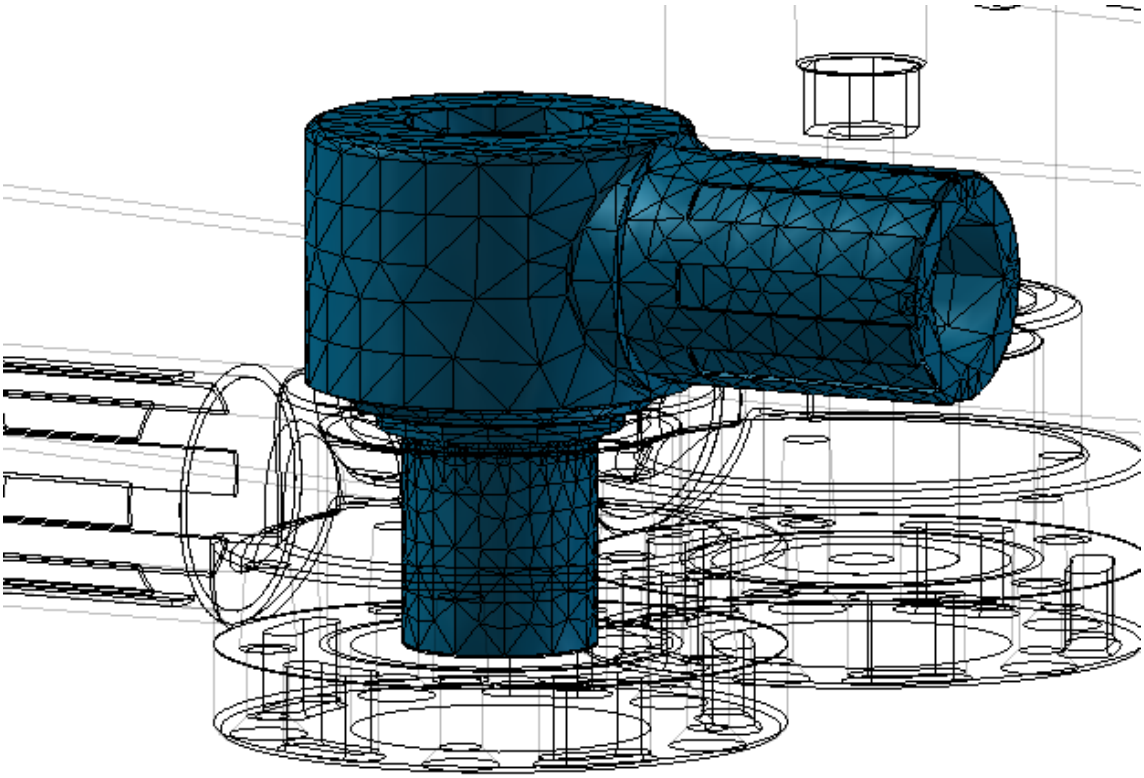


Slika 43. Von Mises naprezanja za „linkageA0_2“, MPa

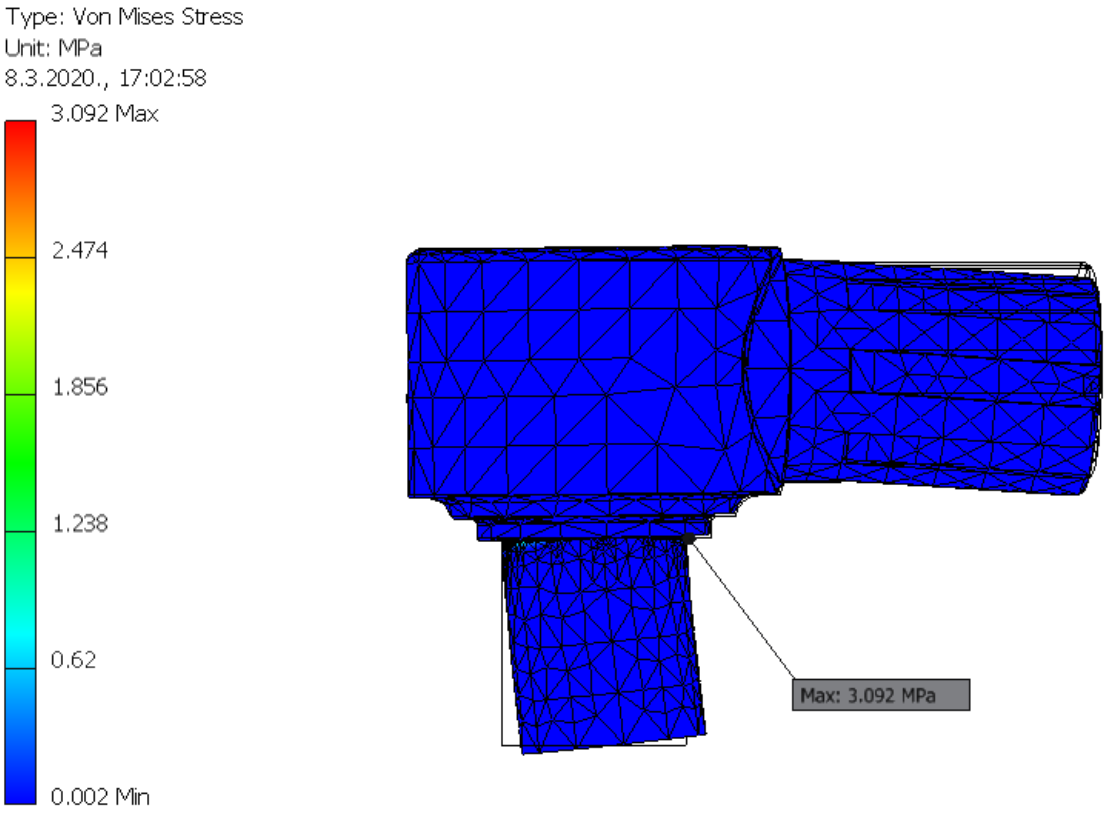


Slika 44. Deformacije za „linkageA0_2“, mm

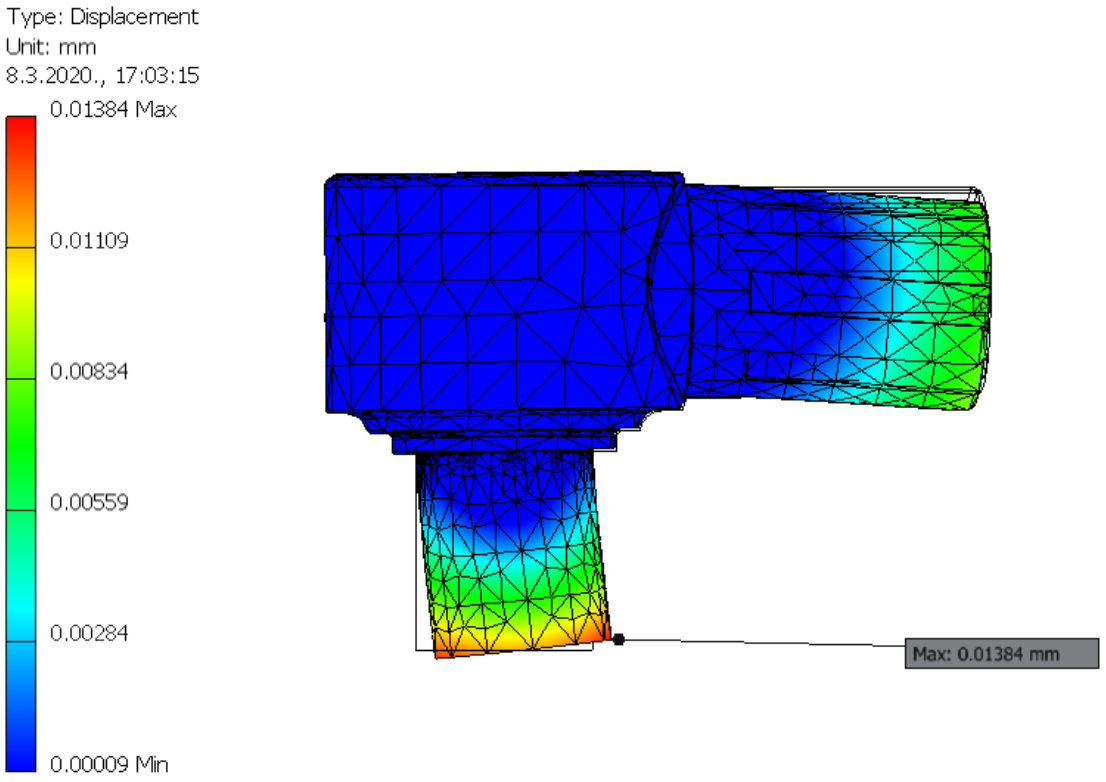
Slika 45 prikazuje izgled „linkageA1_1“ kada je diskretiziran na konačne elemente. Slika 46 prikazuje Von Mises naprezanja i može se može se očitati maksimalni iznos od 3,092 MPa. Slika 47 prikazuje deformacije i može se očitati maksimalni iznos od 0,01384 mm.



Slika 45. Diskretizirani „linkageA1_1“

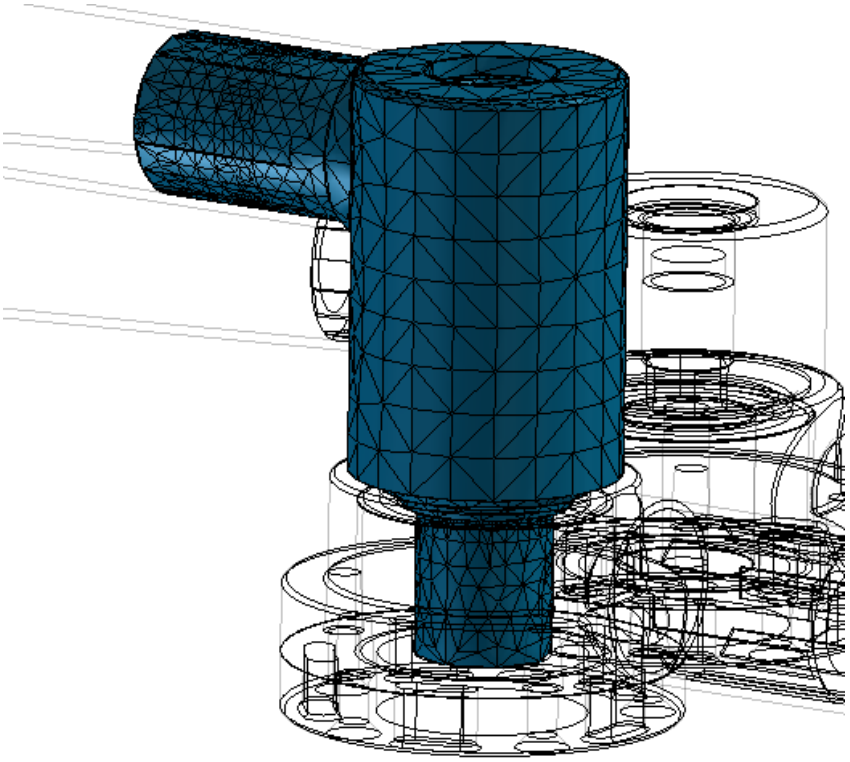


Slika 46. Von Mises naprezanja za „linkageA1_1“, MPa

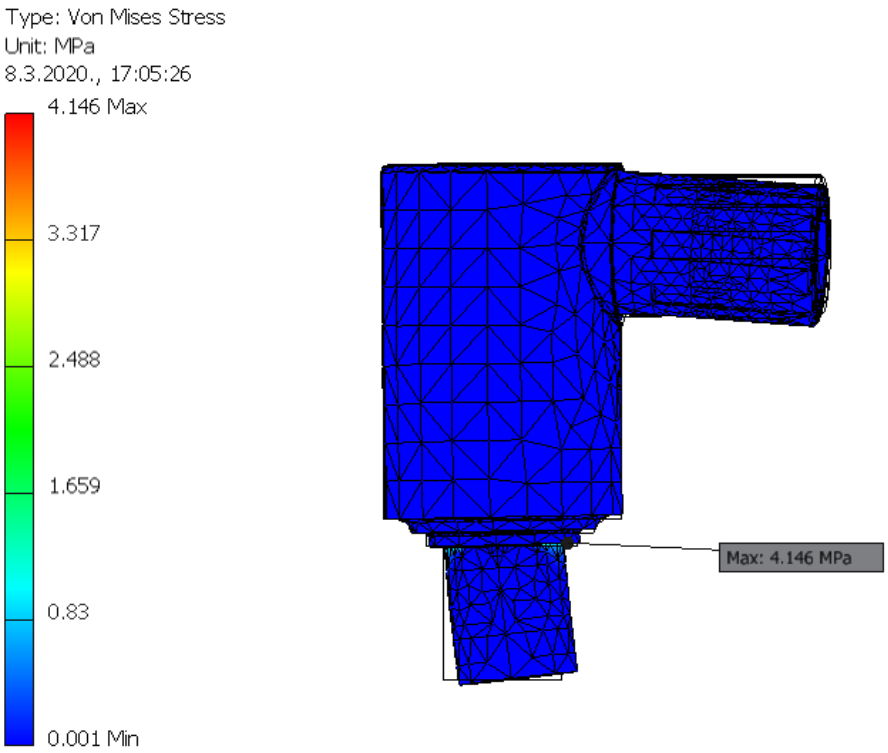


Slika 47. Deformacije za „linkageA1_1“, mm

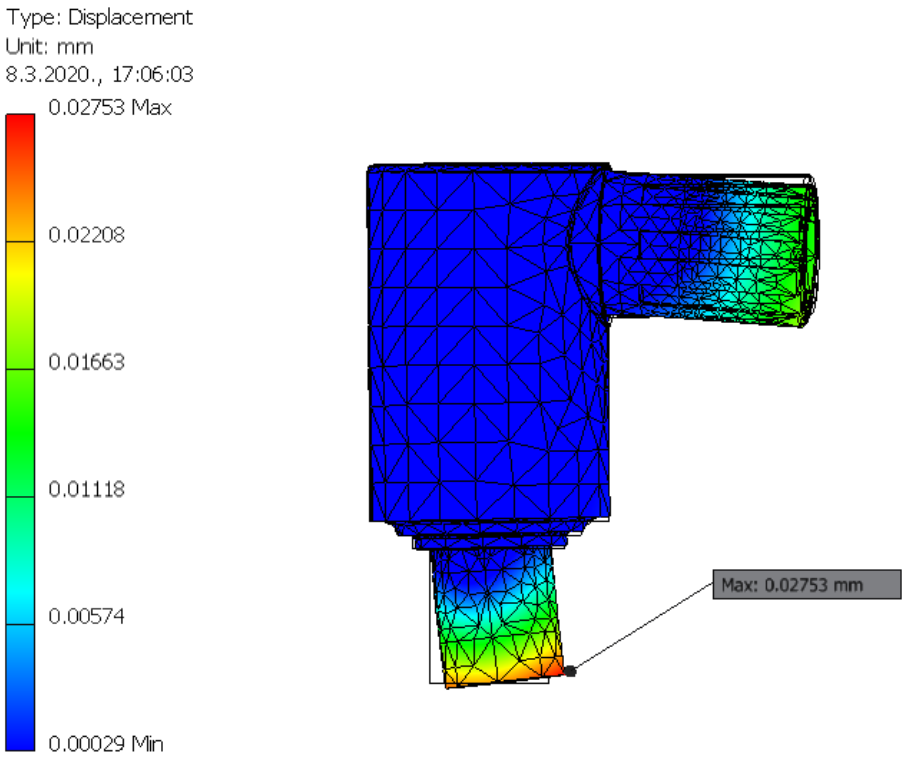
Slika 48 prikazuje izgled „linkageB1_1“ kada je diskretiziran na konačne elemente. Slika 49 prikazuje Von Mises naprezanja i može se očitati maksimalni iznos od 4,146 MPa. Slika 50 prikazuje deformacije i može se očitati maksimalni iznos od 0,02753 mm.



Slika 48. Diskretizirani „linkageB1_1“



Slika 49. Von Mises naprezanja za „linkageB1_1“, MPa



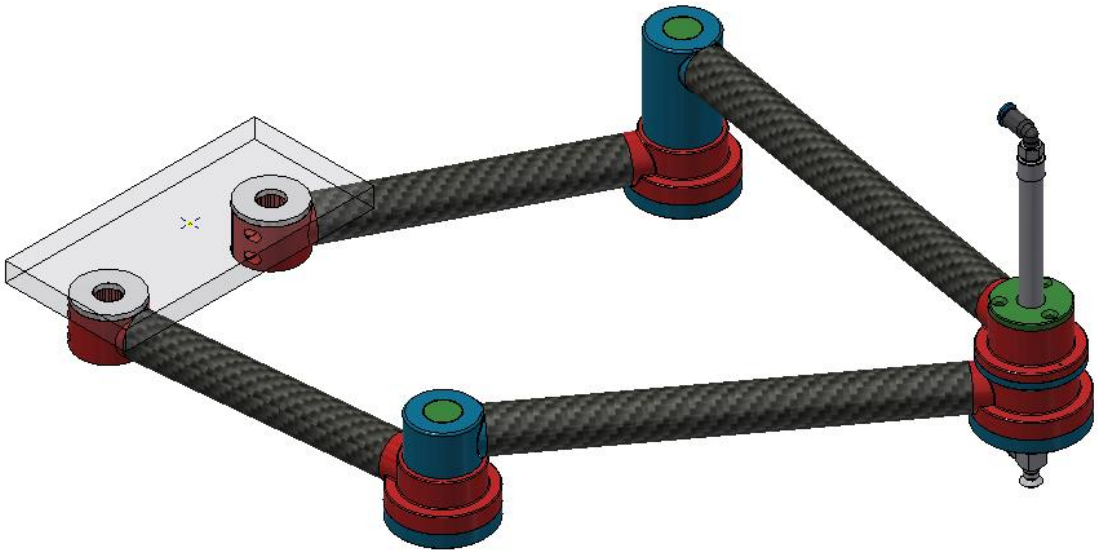
Slika 50. Deformacije za „linkageB1_1“, mm

Kao što je vidljivo Von Misesova naprezanja su niska, što daje jako veliki faktor sigurnosti i može se zaključiti da su simulirani dijelovi predimenzionirani. *FEM* analiza napravljena je pod pretpostavkom da je to puni materijal, ali kako su se dijelovi izrađivali *SLS* postupkom, u praksi dijelovi nisu izrađeni kao puni nego imaju takozvana olakšanja. To znači da materijal nije sinteriran po cijelom svojem presjeku nego je na nekim područjima ostao u praškastom obliku kao popuna. To je napravljeno kako ne bi došlo do vitoperenja materijala, a s tim su dobivene i bolje tolerancije. Ta olakšanja smanjuju predimenzioniranost materijala, ali nažalost nisu ranije poznata nego ih tvrtka koja izrađuje navedene dijelove sama određuje. Isto tako radili su se predimenzionirani dijelovi kako se izrada dijelova ne bi trebala 2 puta ponavljati zbog cijene izrade i vremenskih rokova izrade zadatka. No kako *PA2200* ima malu gustoću, i uz predimenzionirane dijelove, oni su ostali male mase. Ako se pogledaju i deformacije, može se vidjeti kako najveća deformacija iznosi 0,02753 mm, što se smatra prihvatljivim.

3.4.2. Dinamička analiza

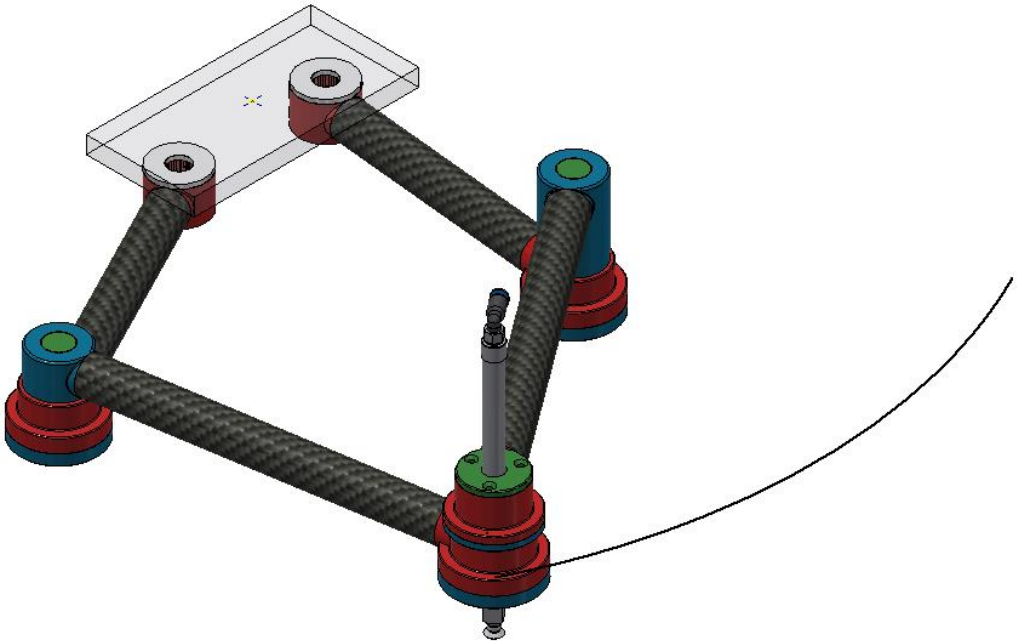
Nakon FEM analize, napravljena je dinamička analiza na istom pojednostavljenom modelu kojeg prikazuje Slika 38. Dinamička analiza napravljena je kako bi se pronašao moment potreban za poganjenje robota, a da njegov radni vrh postigne zadatkom zadanu minimalnu linearnu brzinu od 1 m/s i akceleraciju od 9,81 m/s^2. Dinamička analiza napravljena je također

u *Inventor Dynamic Simulationu* na način da su dijelovi robota „linkageA0_1“ i „linkageB0_1“ određeni kao pogonski te su im zadani početni i završni ulazni kutovi koje moraju preći za neko vrijeme t. Zatim se na vrh alata robota stavila tzv. *Trace* točka koja prati brzinu i akceleraciju. Slika 51 prikazuje početni položaj robota za t = 0 s gdje su ulazni kutovi za „linkageA0_1“ 95,59°, a za „linkageB0_1“ 152,94°. Ulazni kutovi gledaju se u odnosu na koordinatni sustav kojeg prikazuje Slika 14. Slika 52 prikazuje završni položaj robota za t = 0,25 s gdje su kutovi za „linkageA0_1“ 27,06 ° i za „linkageB0_1“ 84.41°. Pri toj simulaciji prijeđena je udaljenost od 300 mm u smjeru osi *x*.



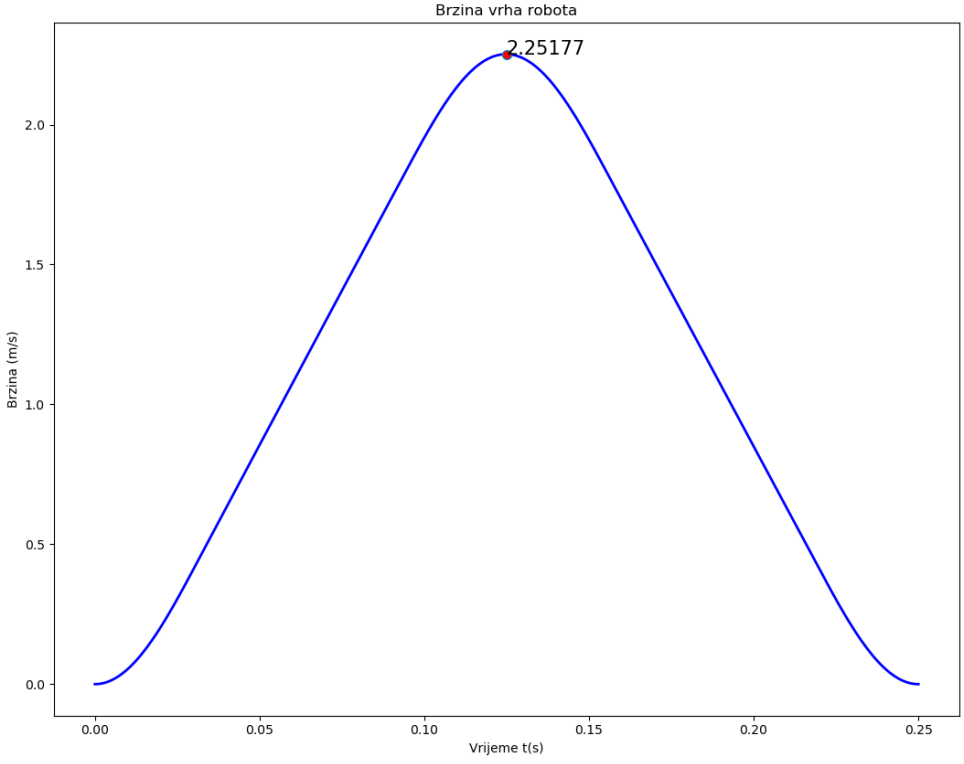
Slika 51. Početni položaj robota u dinamičkoj simulaciji

Grafovi koje prikazuje Slika 53, Slika 55 i Slika 57 napravljeni su u programskom jeziku *Python* korištenjem *Matplotlib* paketa s podacima dobivenim pomoću *Inventor Dynamic Simulation* modula.

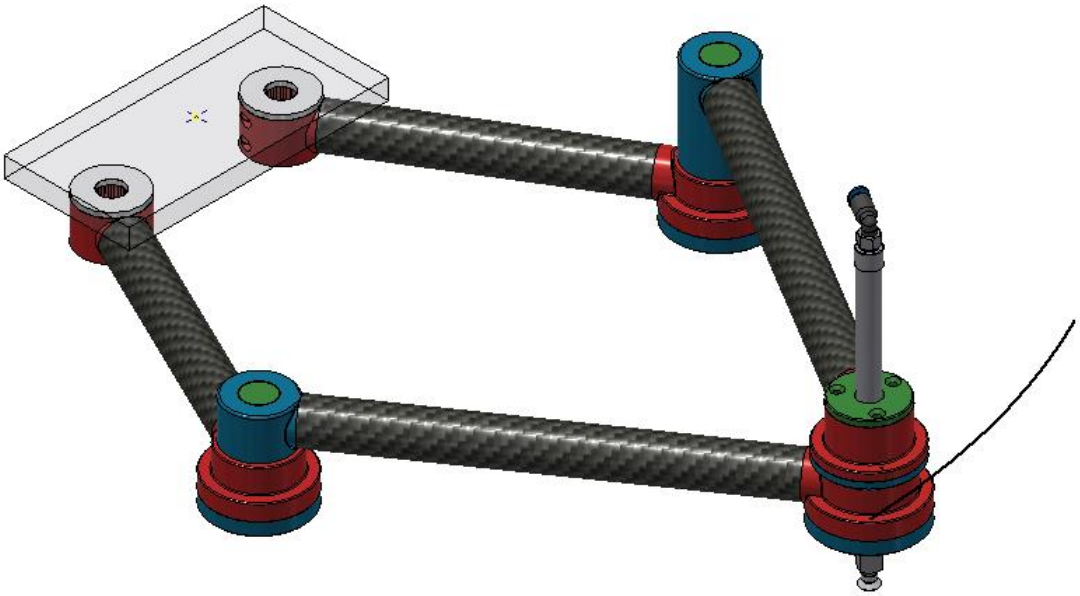


Slika 52. Završni položaj robota u dinamičkoj simulaciji

Pri simulaciji postignuta je najveća brzina radnog vrha robota od 2,25177 m/s kako Slika 53 prikazuje. Robot se tada nalazio u poziciji kako Slika 54 prikazuje.

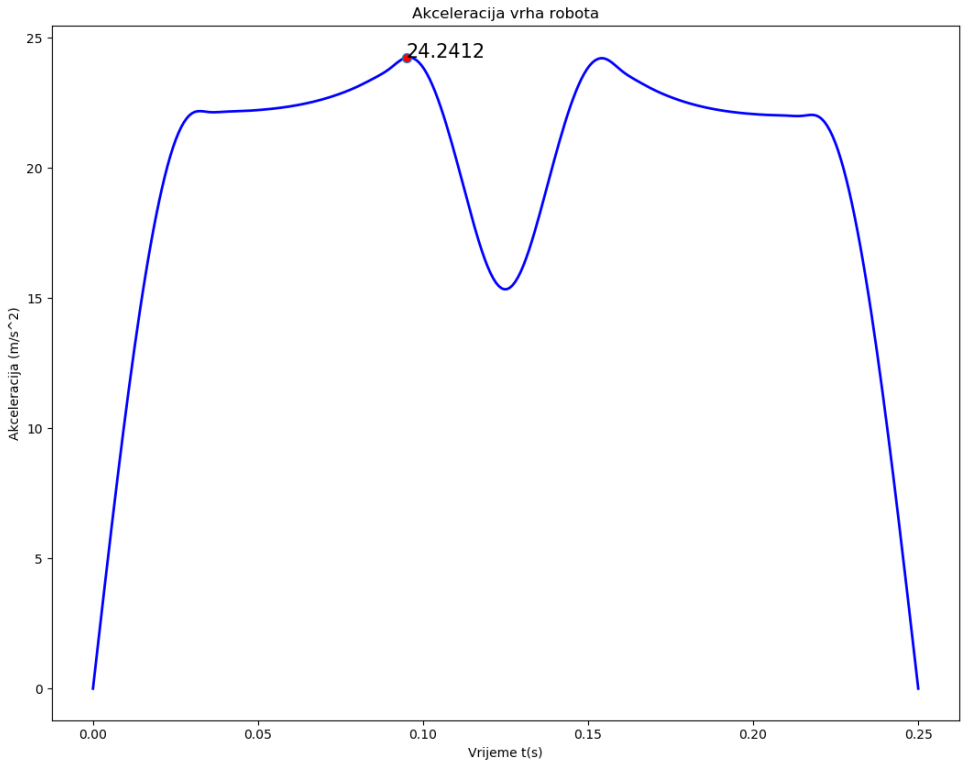


Slika 53. Maksimalna postignuta brzina robota za dinamičku simulaciju

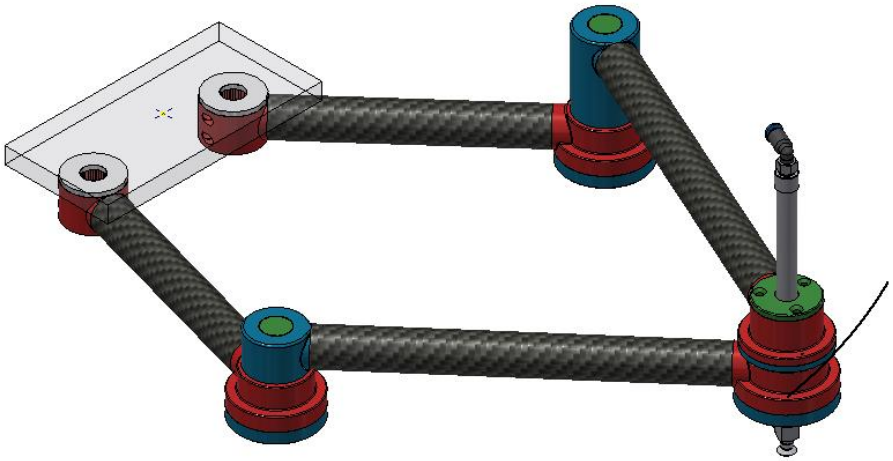


Slika 54. Položaj robota za maksimalnu postignutu brzinu

Najveća postignuta akceleracija je 24,2412 m/s^2 kako Slika 55 prikazuje, a robot se tada nalazio u pozicija kako Slika 56 prikazuje.

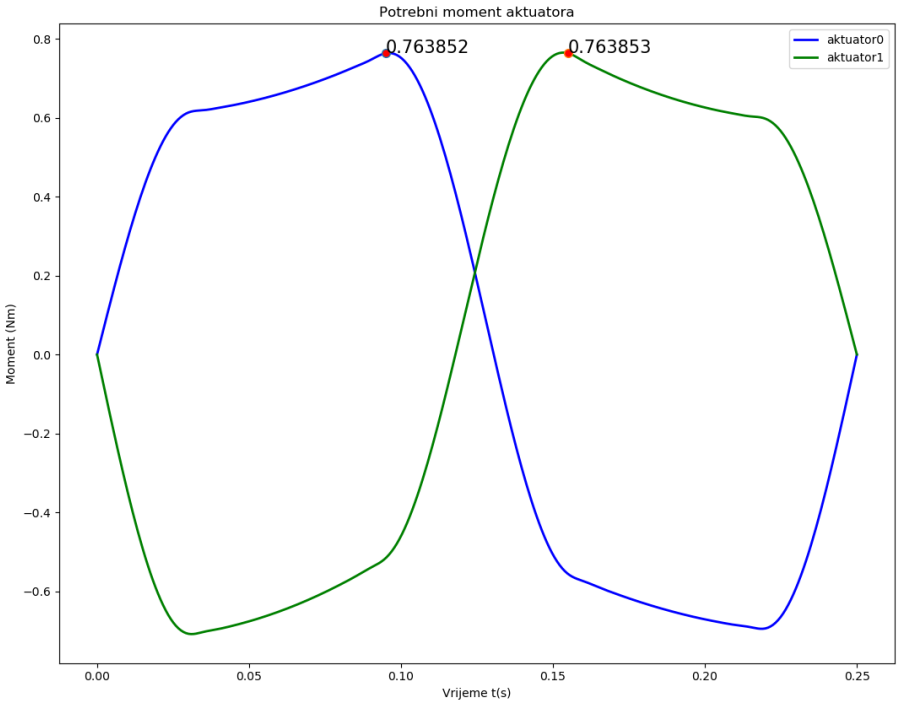


Slika 55. Maksimalna postignuta akceleracija robota za dinamičku simulaciju

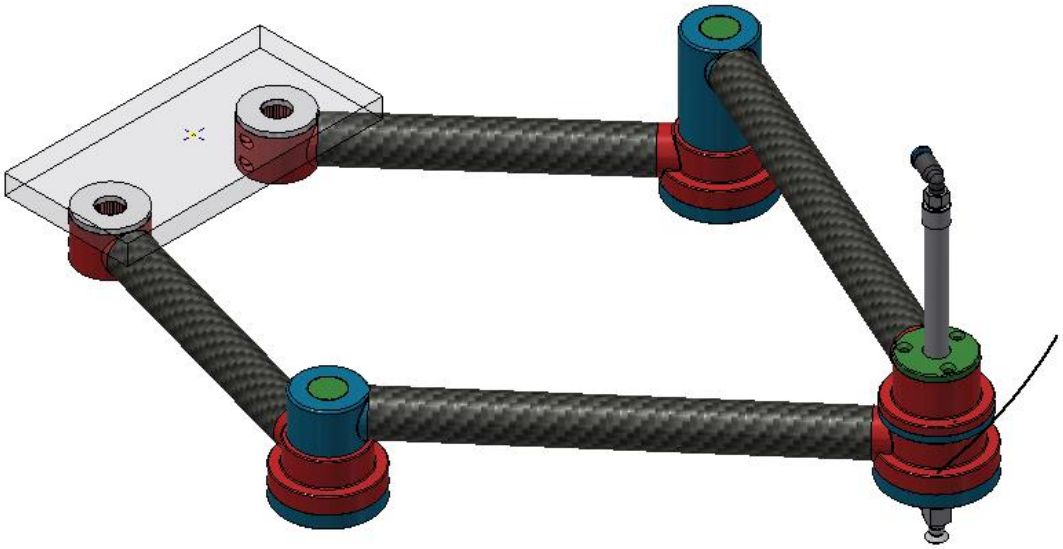


Slika 56. Položaj robota za maksimalnu postignutu akceleraciju

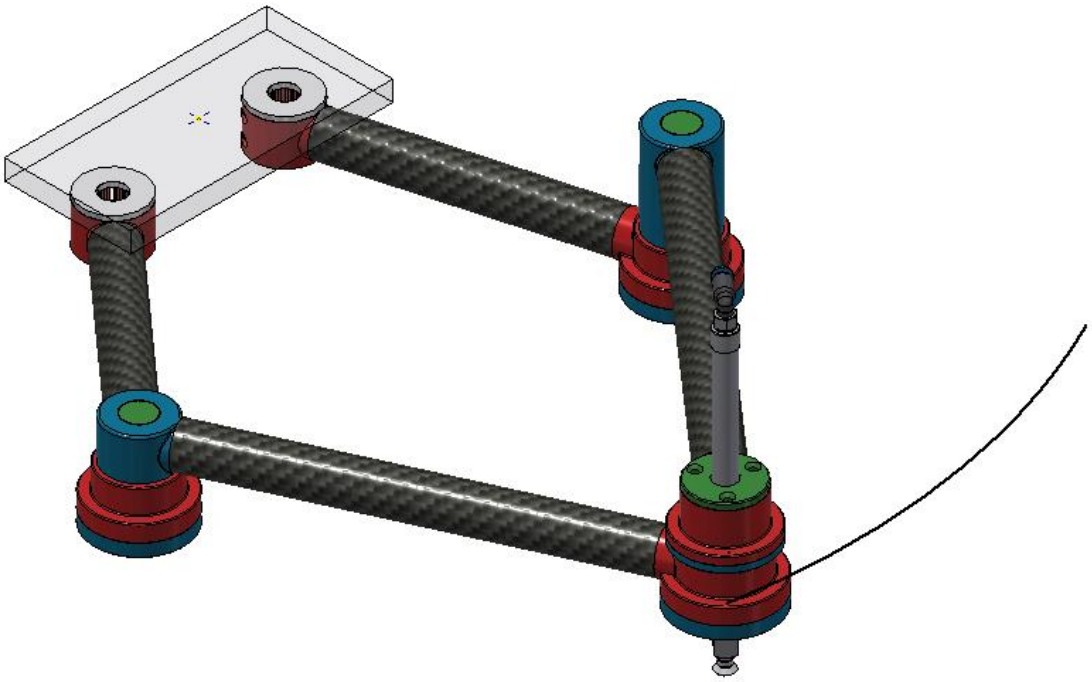
Slika 57 prikazuje potrebni pogonski moment za „linkageA0_1“ i za „linkageB0_1“. Najveći potrebni pogonski momenti iznose 0,763852 N/m za „linkageA0_1“(aktuator0) i 0,763853 N/m za „linkageB0_1“ (aktuator1). Robot se za maksimalni moment „linkageA0_1“ nalazio u poziciji kako Slika 58 prikazuje, dok se za maksimalni moment „linkageB0_1“ nalazio u poziciji kako Slika 59 prikazuje.



Slika 57. Maksimalni potrebni moment za „linkageA0_1“ i „linkageB0_1“



Slika 58. Položaj robota kod maksimalnog potrebnog momenta za „linkageA0_1“



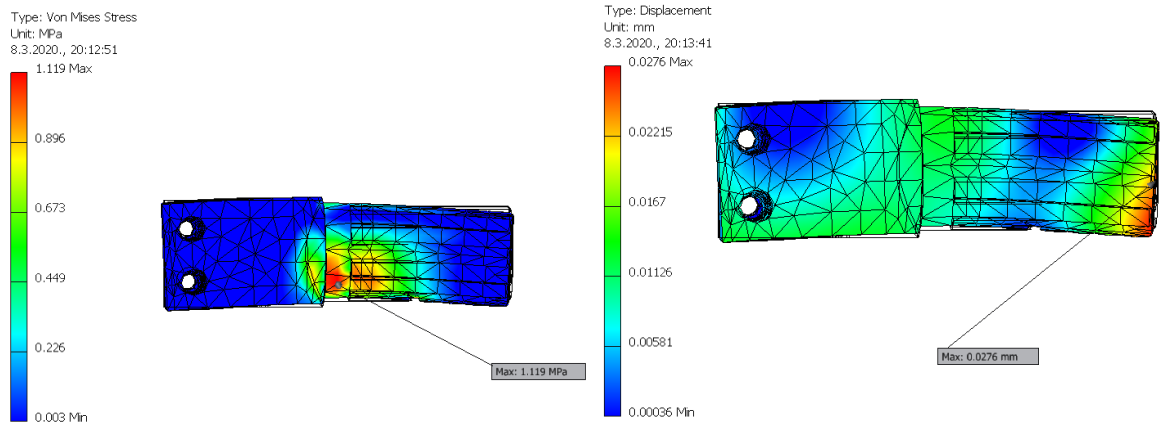
Slika 59. Položaj robota kod maksimalnog potrebnog momenta za „linkageB0_1“

Potrebno je napomenuti kako kod dinamičke analize nije uzet u obzir utjecaj trenja, otpor zraka te otpori pneumatskih cijevi koje se priključuju na pneumatski cilindar i vakuumsku hvataljku, stoga se zbog faktora sigurnosti uzima minimalni potrebni moment 1,5 N/m.

3.4.3. FEM analiza 2

U ovom poglavlju napravljena je FEM analiza na isti način kao i u poglavlju 3.4.1, ali uz razliku što se ne gledaju opterećenja dok robot miruje, nego se izabrala jedna od nepovoljnijih točaka u simulaciji, kad je robot pod povećanim opterećenjem. Odabrana je točka kada robot postiže maksimalnu akceleraciju od 24,2412 m/s^2.

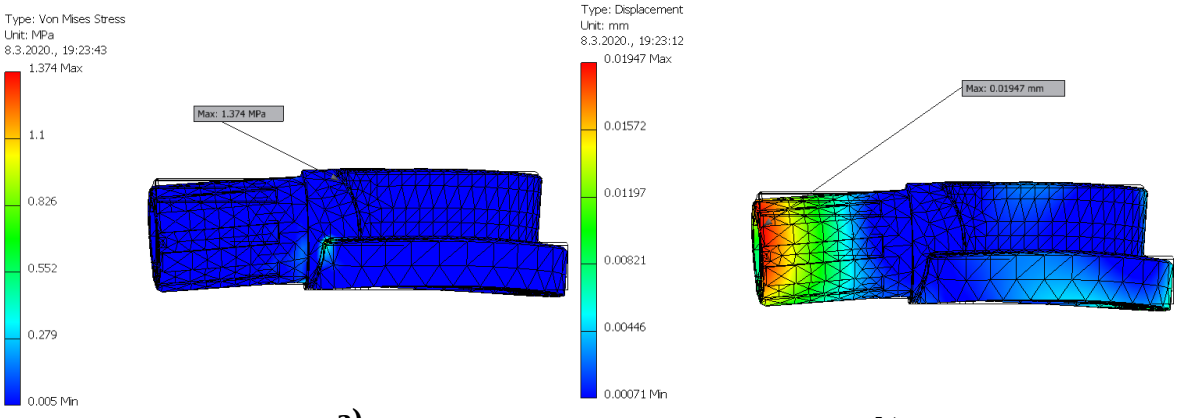
Slika 60 pod a) prikazuje Von Mises naprezanja, dok pod b) prikazuje deformacije za „linkageA0_1“. Maksimalno Von Mises naprezanje je 1,119 MPa, dok je maksimalna deformacija 0,0276 mm.



a) **b)**

Slika 60. „linkageA0_1“: a) Von Mises naprezanja, MPa; b) deformacije, mm

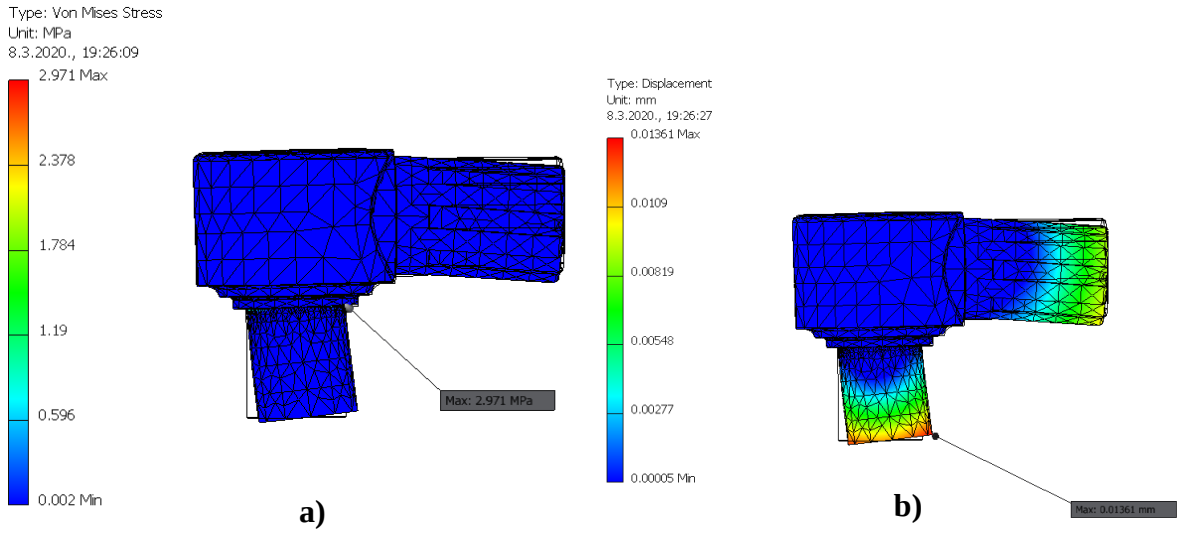
Slika 61 pod a) prikazuje Von Mises naprezanja, dok pod b) prikazuje deformacije za „linkageA0_2“. Maksimalno Von Mises naprezanje je 1,374 MPa, dok je maksimalna deformacija 0,01947 mm.



a) **b)**

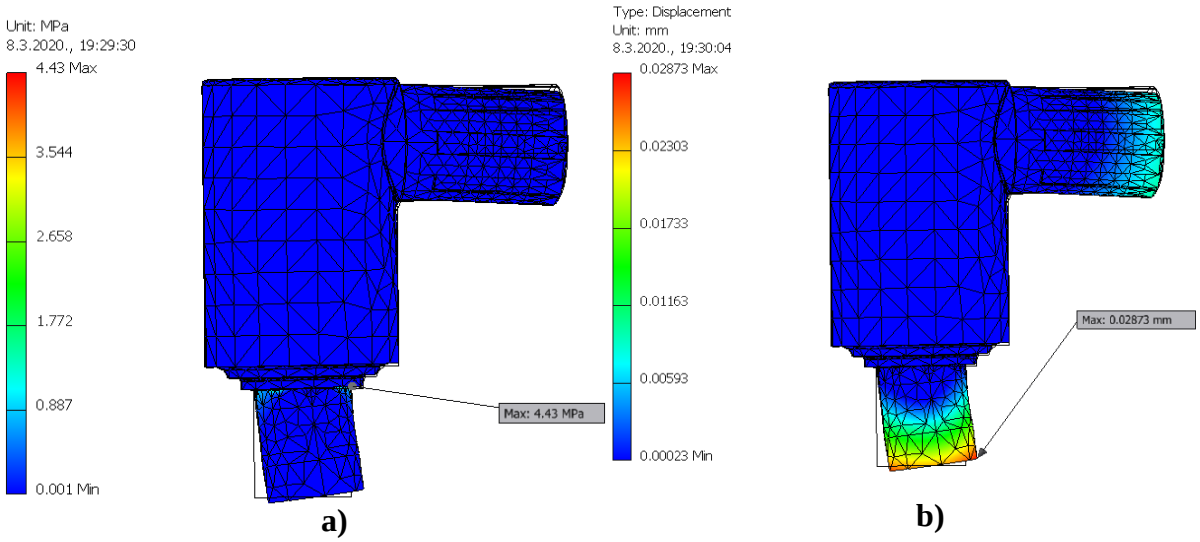
Slika 61. „linkageA0_2“: a) Von Mises naprezanja, MPa; b) deformacije, mm

Slika 62 pod a) prikazuje Von Mises naprezanja, dok pod b) prikazuje deformacije za „linkageA1_1“. Maksimalno Von Mises naprezanje je 2,971 MPa, dok je maksimalna deformacija 0,01361 mm.



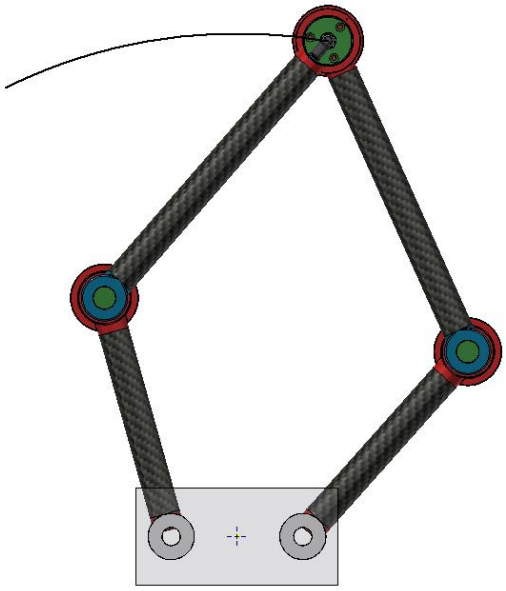
Slika 62. „linkageA1_1“: a) Von Mises naprezanja, MPa; b) deformacije, mm

Slika 63 pod a) prikazuje Von Mises naprezanja, dok pod b) prikazuje deformacije za „linkageA0_2“. Maksimalno Von Mises naprezanje je 4,43 MPa, dok je maksimalna deformacija 0,02873 mm.



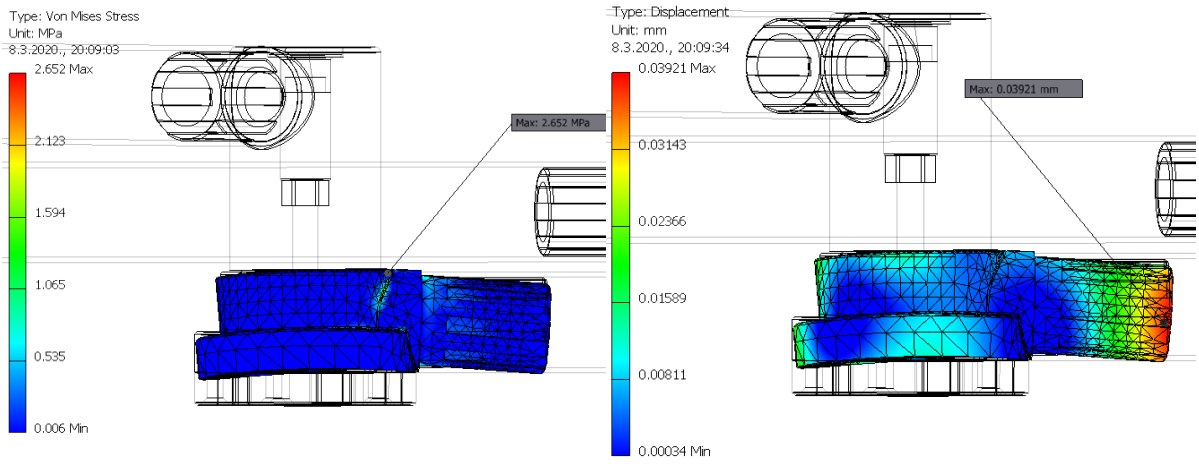
Slika 63. „linkageB1_1“: a) Von Mises naprezanja, MPa ; b) deformacije, mm

Ako se usporede deformacije i naprezanja iz poglavlja 3.4.1 i ovog poglavlja, može se uočiti da su za sve elemente osim „linkageB1_1“ naprezanja i deformacije manje. No ako se pogleda Slika 64 može se uočiti da je lijeva strana robota malo više ispružena od desne, te su zbog toga i opterećenja veća, ali robot nije potpuno ispružen kao što Slika 38 prikazuje.



Slika 64. Tlocrt robota za FEM analizu 2

Kao dodatna potvrda, Slika 65 prikazuje naprezanja identičnog dijela kao „linkageA0_1“ za istu poziciju robota pri maksimalnoj akceleraciji te se može vidjeti kako su naprezanja i deformacije veće. Maksimalno Von Mises naprezanje je 2,652 MPa, dok je maksimalna deformacija 0,03291 mm.



Slika 65. „linkageB0_2“: a) Von Mises naprezanja, MPa; b) deformacije, mm

Može se zaključiti kako analizirani dijelovi zadovoljavaju konstrukcijske uvjete.

3.5. Odabir aktuatora

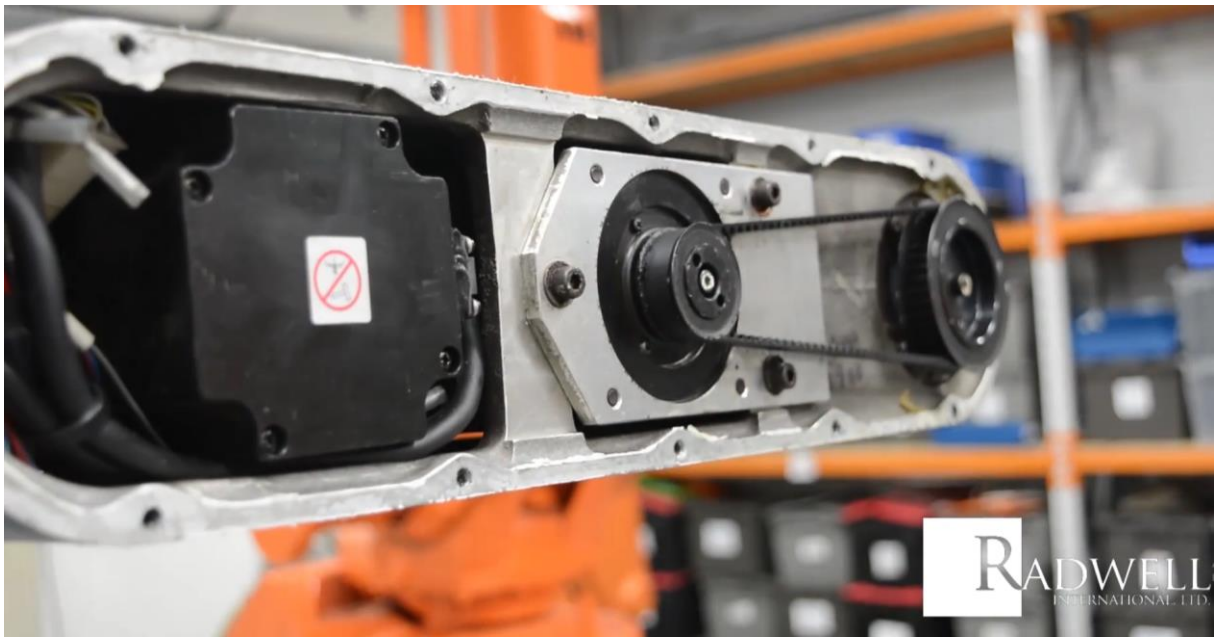
U poglavlju 3.4.2 preko dinamičke analize dobiven je maksimalni potrebni moment od 0,76358 N/m, no zbog faktora sigurnosti pretpostavlja se da aktuator mora isporučiti minimalno 1,5 N/m.

Dogovoreno je da aktuator mora biti elektromotor koji radi u zatvorenom regulacijskom krugu, tj. servomotor. Postoje izvedbe aktuatora sa servomotorima gdje se koristi redukcija broja okretaja te izvedbe gdje se elektromotor spaja preko spojke direktno na robota bez redukcije broja okretaja. U robotici je česta praksa korištenje motora s malim okretnim momentom, ali velikim brzinama vrtnje skupa sa reduktorom. Kroz reduktor se dobije mali broj okretaja i veliki moment, što je često poželjno za robote gdje su potrebni veliki okretni momenti i male brzine kretanja.

Koriste se različiti reduktori, no neki od najčešće zastupljenih su:

- Planetarni prijenos kojeg prikazuje Slika 66
- Remenski prijenos kojeg prikazuje Slika 67
- Harmonijski prijenos kojeg prikazuje Slika 68

Slika 66. Planetarni prijenos[15]



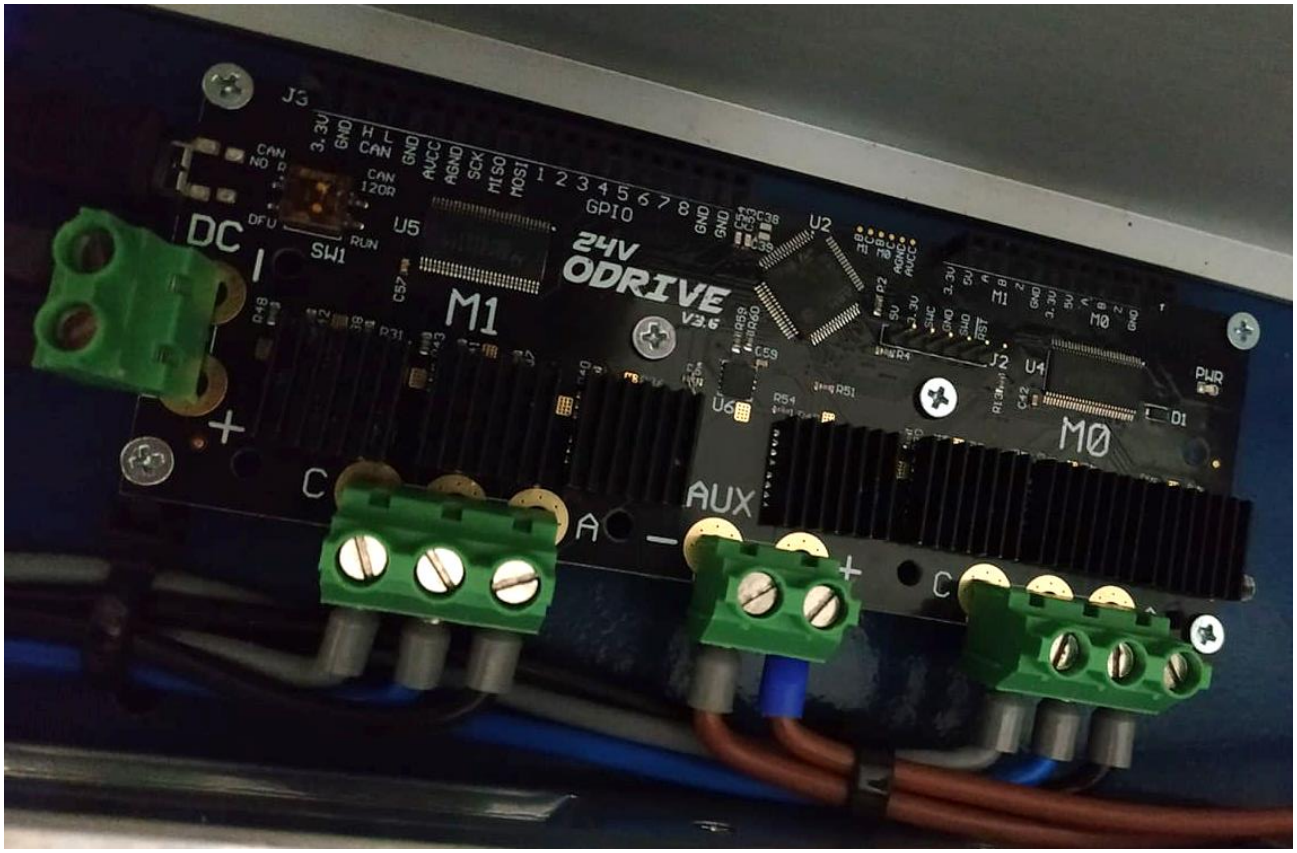
Slika 67. Remenski prijenos[16]

Slika 68. Harmonijski prijenos[17]

Za simulaciju provedenu u poglavlju 3.4.2 dobivena je maksimalna brzina 100 okr/min te bi reduktor s malim servomotorom bio idealna kombinacija. No kako bi upotreba reduktora s prihvatljivim tolerancijama i malim mrtvim hodom podignula cijenu izrade robota, odlučeno je koristiti servomotor sa spojkom bez redukcije brzine. Korištenjem direktnog prijenosa eliminira se utjecaj mrtvog hoda na robotu koji je često problem reduktora.

Kod odabira servomotora cilj je korištenjem ne industrijskih komponenti dobiti industrijske performanse. Pri tome se misli na odabir hobi motora i enkodera te kompatibilnog drivera koji bi davali dovoljno dobre performanse za nisku cijenu.

Stoga je odabran driver motora pod nazivom *ODrive Robotics* kojeg prikazuje Slika 69. Njegova velika prednost je što je softver otvorenog koda (*eng. open-source*) što znači da svatko može pridonijeti razvitku drivera te ga i samostalno izraditi. Postoje 56 V i 24 V verzija, za robota je odabrana 24V verzija.



Slika 69. *ODrive robotics* driver

Driver može istovremeno upravljati s dva elektromotora bez četkica (*eng. BLCD motor*), što ga čini idealnim za robota. Osim motora potrebni su i enkoderi kako bi motor mogao raditi u zatvorenoj regulacijskog petlji. Driver komunicira s računalom ili mikroračunalom preko USB protokola u *Shell* okruženju ili izvršavajući kod napisan u *Python* programskom jeziku.

Kako će motori raditi pri malom broju okretaja, bitno je da imaju dovoljno momenta pri malim brzinama. Većina hobi elektromotora bez četkica radi najbolje pri velikim režimima vrtnje dok se pri malima zagrijavaju i ne uspijevaju proizvesti dovoljno momenta. Kako bi se dobio što je veći mogući moment pri malim brzinama i što je moguće manje zagrijavanje, uzimaju se predimenzionirani motori sa što manjom tzv. K_V konstantom. Odabran je *ASP5065 90* K_V

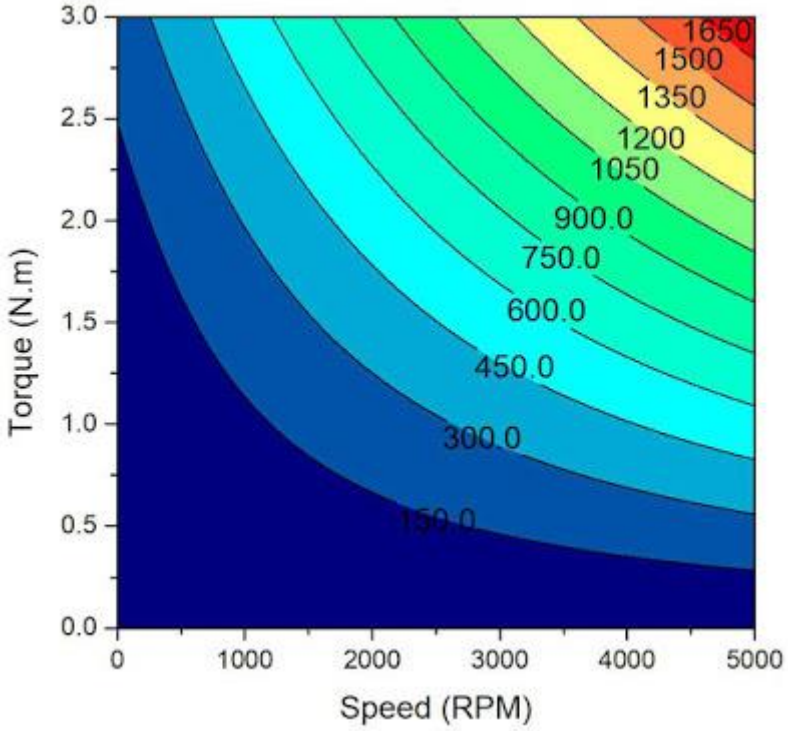
1800W motor. Na stranici *https://alienpowersystem.com*[18] se može očitati da motor ima maksimalni radni moment od 5,7 Nm, a to je 3,8 puta više od potrebnog momenta. Iako je motor previše predimenzioniran, izabran je taj motor zbog općenite dostupnosti motora s malom K_v konstantom u prihvatljivom cjenovnom rangu. Slika 70 prikazuje izgled motora.

Slika 70. ***APS5065 90 K_v 1800W***

Tip odabranog enkodera je *CUI AMT102-V,* koji prikazuje Slika 71. Enkoder ima rezoluciju od 8192 impulsa po okretaju, što ga čini prihvatljivim za upotrebu pri malim brzinama. Enkoder ima jedan Z puls koji služi driveru za kalibraciju enkodera, koja je potrebna prilikom svakog paljenja robota jer se radi o inkrementalnom, a ne apsolutnom enkoderu.

Slika 71. ***CUI AMT102-V***

Napajanje motora napravljeno je prema grafu koji prikazuje Slika 72. Prema grafu, dovoljna snaga napajanja za zadani moment i brzine manja je od 300 W, no ako se uzmu u obzir gubitci i faktor sigurnosti odabire se 24V napajanje snage do 400 W.



Slika 72. Graf za odabir napajanja[19]

Odabrano je *MeanWell ERPF-400-24* kojeg prikazuje Slika 73 te radi na 24V i ima maksimalnu snagu od 400W.

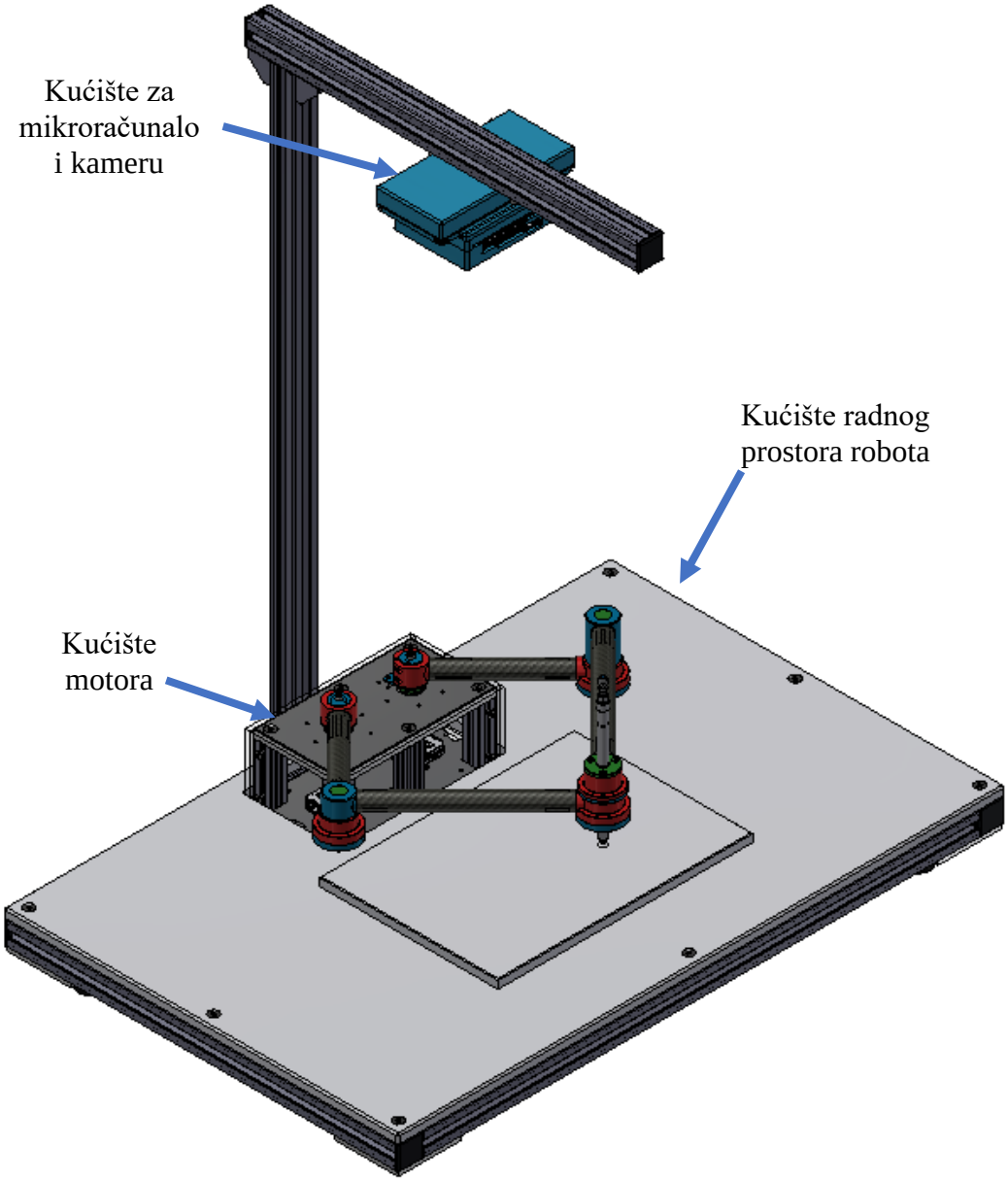
Slika 73. *MeanWell ERPF-40-24*

Driver prilikom zaustavljanja motora ima mogućnost regenerativnog kočenja. No kako se ovdje koristi klasično napajanje, potrebno je koristit otpornik koji će preuzeti energiju regenerativnog kočenja, inače bi moglo doći do oštećenja drivera ili napajanja. Stoga se koristi *Arcol HS50-R5-J* otpornik snage 50 W kojeg prikazuje Slika 74.

Slika 74. ***Arcol HS50-R5-J***

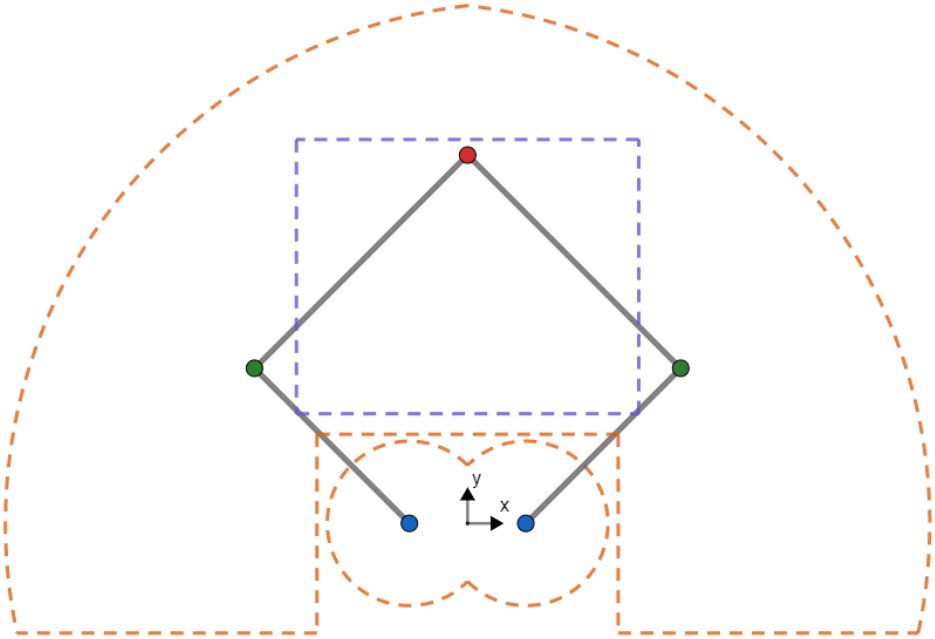
3.6. Kućište robota

Nakon što je definiran sustav za pogon robota, napravljena je konstrukcija za kućište istog. Osim toga napravljeno je kućište za radni prostor robota i kućište za mikroračunalo s kamerom koje upravlja robotom. Kućišta robota mogu se podijeliti na kućište motora, kućište radnog prostora robota i kućište mikroračunala i kamere kako Slika 75 prikazuje. *CAD* model konačne verzije cijelog robota u *.step* i *.ipt*/*.iam* formatu također se nalazi na priloženom CD-u.



Slika 75. Petosni robot paralelne kinematike

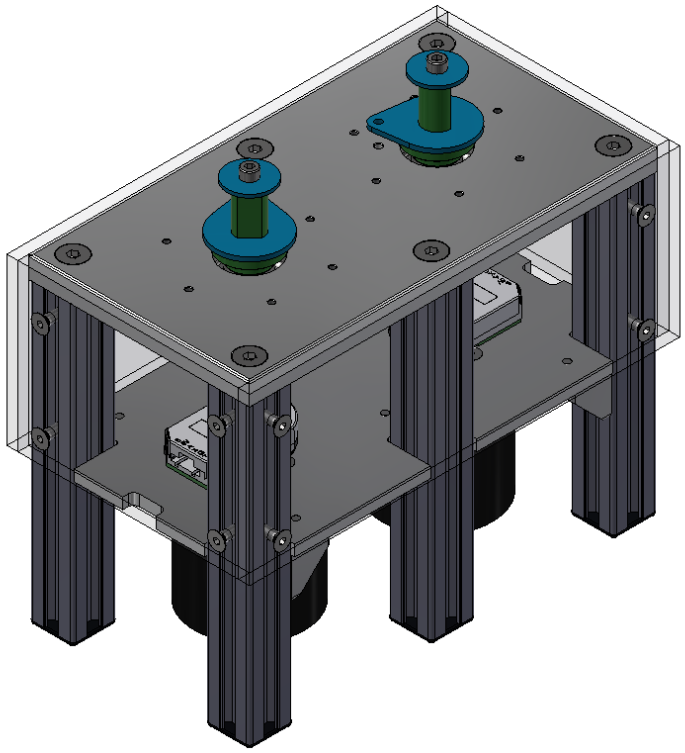
Ako usporede Slika 14 i Slika 75 može se uočiti da je radno područje robota upola manje nego prvotno prikazano. Slika 14 prikazuje radno područje koje ne uzima u obzir konstrukcijska ograničenja robota. Stoga novo radno područje robota sa konstrukcijskim ograničenjima prikazuje Slika 76.



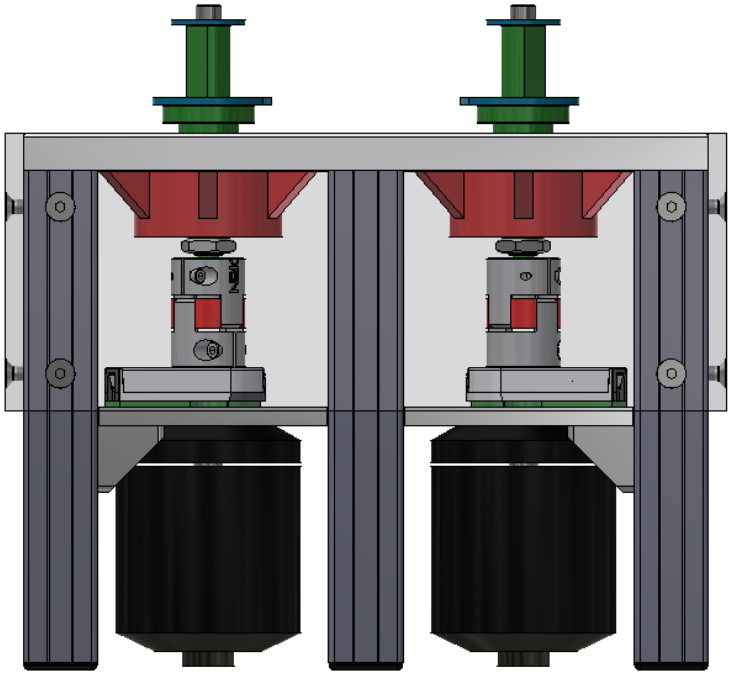
Slika 76. Radno područje robota s konstrukcijskim ograničenjima

3.6.1. Kućište motora

Za odabrane motore i enkodere iz poglavlja 3.5 napravljeno je kućište. Slika 77 i Slika 78 prikazuju izgled kućišta motora.

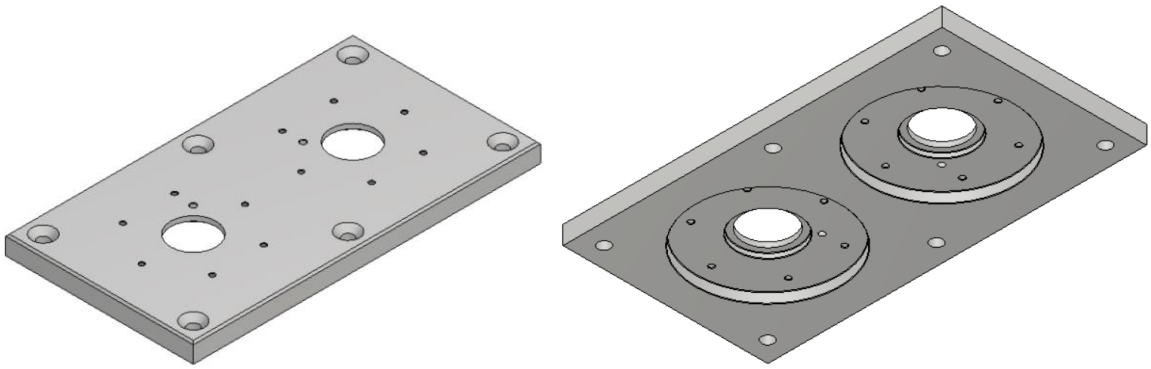


Slika 77. Kućište motora, pogled 1

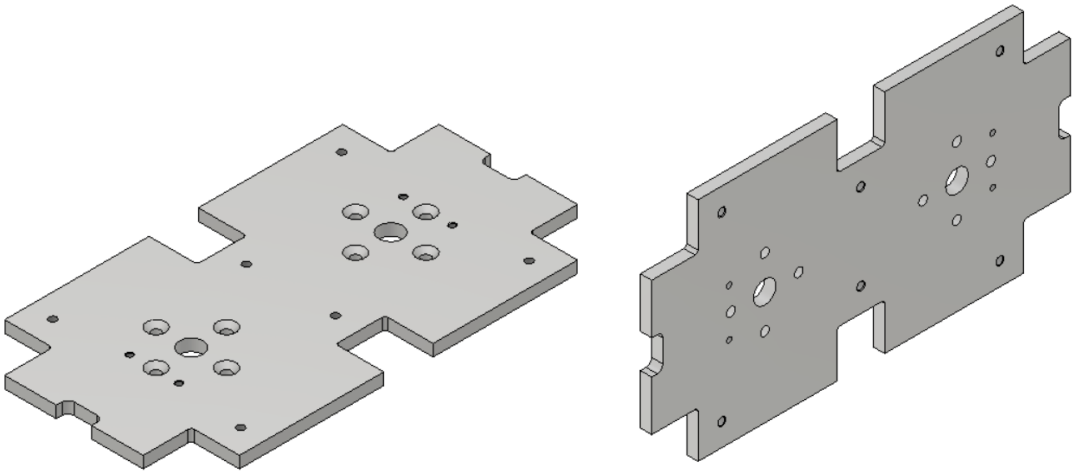


Slika 78. Kućište motora, pogled 2

Osnovni strukturni elementi kućišta motora čine aluminijska ploča dimenzija 185x100 mm i debljine 10 mm, aluminijska ploča dimenzija 185x100 mm i debljine 5 mm te 6 komada *Bosch Rexroth* ekstrudiranog aluminija 2020 duljine 133 mm. Slika 79 i Slika 80 prikazuju spomenute ploče. Kako su te ploče napravljene na CNC glodalici, u prilogu [II] nalazi se njihova tehnička dokumentacija pod brojem MB_Diplomski_04 i MB_Diplomski_05.

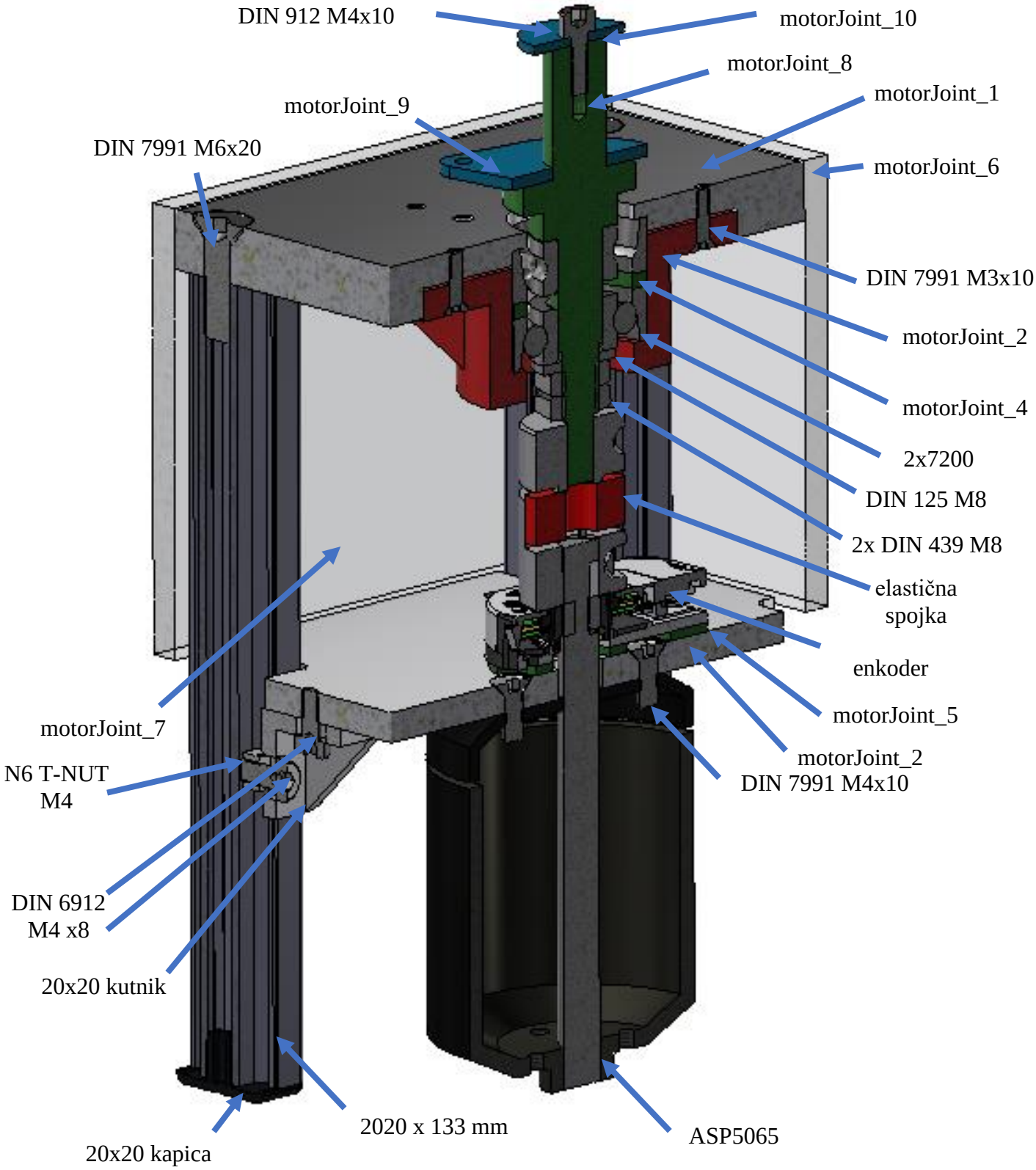


Slika 79. „motorJoint_1“



Slika 80. „motorJoint_2“

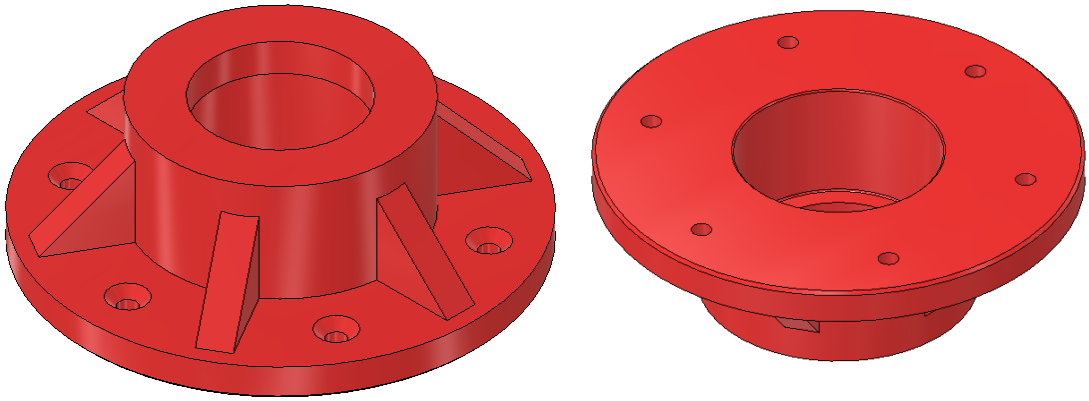
Ploče su s ekstrudiranim aluminijem povezane DIN 7991 M6x20 vijcima za koje je urezan M6 navoj u ekstrudirani aluminij te korištenjem kutnika za povezivanje ekstrudiranih aluminija. Postupak povezivanja prikazuje Slika 81. Motori su montirani na „motorJoint_2“ ploču korištenjem DIN 7991 M4x10 vijaka, dok su enkoderi montirani korištenjem DIN 912 M3x6 vijaka. Slika 81 prikazuje povezivanje motora i enkodera na ostatak kućišta motora. Kako enkoder ima prolaznu rupu, on se montira tako da vratilo motora prolazi kroz enkoder. Između vratila motora i vratila koje se spaja na robota stavljena je elastična spojka. Spojka ima vanjski promjer 20 mm, duljinu 30 mm te radi redukciju s 8 mm vratila motora na 6 mm vratila koje se spaja na robota. Model spojke preuzet je sa *https://webassistants.partcommunity.com*[20] i dodatno korigiran kako bi zadovoljavao zadane dimenzije. Kako je kućište motora simetrično oko osi *y*, na slici je prikazano samo pola presjeka.



Slika 81. Presjek kućišta motora

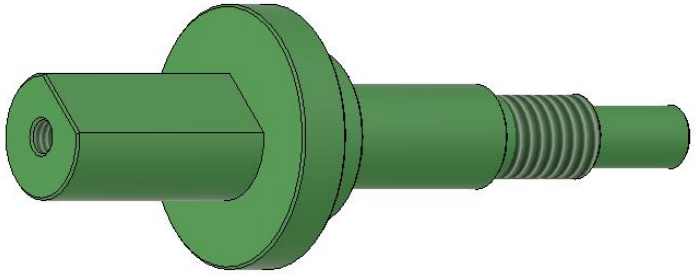
Vratilo robota („motorJoint_8“) povezuje motore s ostatkom robota te služi kao uležištenje cijelog robota. Vratilo je uležišteno pomoću dva 7200 ležaja s kosim dodirom dimenzija 10x15

mm i debljine 9 mm. Između ležajeva je odstojni prsten „motorJoint_4“. Ležajevi se nalaze u konfiguraciji kako prikazuje Slika 22 što ih čini pogodnim za sve vrste opterećenja koje se mogu pojaviti. Ležajevi su uprešani u kućište „motorJoint_3“ koje prikazuje Slika 82. Kućište za ležajeve i odstojni prsten izrađeni su aditivnom proizvodnjom, ali ne *SLS* nego *FDM* postupkom od *ABS* materijala. Razlog tome je što Laboratorij za projektiranje izradbenih i montažnih sustava ima *FDM* printer te nije bilo vremena za slanje na izradu *SLS* postupkom. No kako je postupak izrade odrađen na visokokvalitetnom *FDM* printeru, rezultati su više nego zadovoljavajući. „motorJoint_3“ se montira na aluminijsku ploču debljine 10 mm sa DIN 7991 M3x10 vijcima.

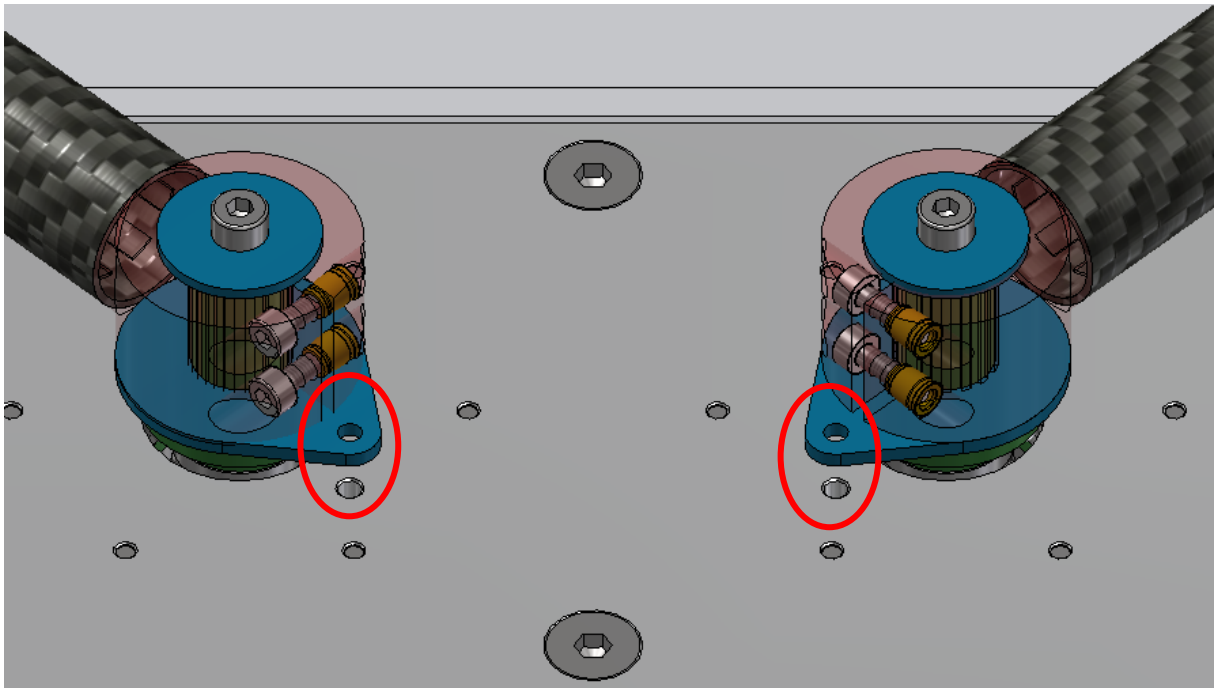


Slika 82. „motorJoint_3“

Vratilo se steže na unutarnji prsten 7200 ležaja sa DIN 439 M8 maticama, kako Slika 81 prikazuje. Između unutarnjeg prstena ležaja i matice stavljena je DIN 125 M8 podložna pločica. Slika 83 prikazuje izgled vratila. Vratilo je izrađeno od aluminija postupkom CNC tokarenja te se za isti tehnička dokumentacija može pronaći u prilogu [II] pod brojem MB_Diplomski_06.



Slika 83. „motorJoint_8“



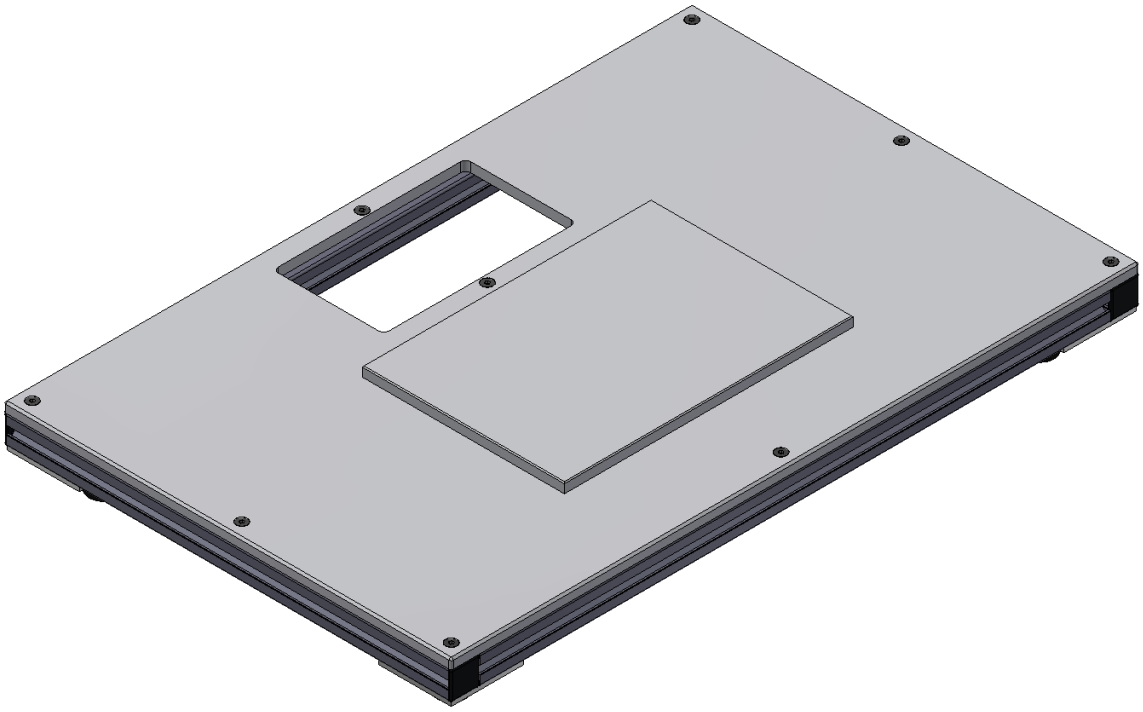
Slika 84. Povezivanje vratila s ostatkom robota

Osim kombinacije DIN 912 M3x10 vijaka i M3 umetaka za pričvršćivanje robota na vratilo, koriste se i DIN 912 M4x10 vijak s podložnom pločicom („motorJoint_10“) debljine 1,5 mm koja je izrađena iz aluminija laserskim rezanjem. Između spoja vratila robota i članka robota nalazi se aluminijska pločica („motorJoint_9“) debljine 2 mm koja je također izrađena laserskim rezanjem. „motorJoint_9“ služi za kalibraciju robota, tako da se kroz nju stavi pin koji se poklopi s označenim rupama na ploči debljine 10mm („motorJoint_1“). Slika 84 prikazuje spomenute aluminijske pločice i rupe za pinove prilikom kalibracije robota. Kako su navedene aluminijske pločice izrađene laserskim rezanjem, za iste se može pronaći tehnička dokumentacija u prilogu [II] pod brojem MB_Diplomski_07 i MB_Diplomski_08.

Na dio kućišta motora stavljene su ploče od prozirnog pleksiglasa debljine 5 mm kako bi postojala zaštita između motora i korisnika robota. Pleksiglas je pričvršćen na 2020 aluminijske profile kombinacijom DIN 7991 M4x10 vijaka i N6 T-NUT M4 matica. Slika 77 prikazuje kućište motora sa pleksiglasom. Ploče od pleksiglasa izrađene su laserskim rezanjem te se za njih u prilogu [II] može pronaći tehnička dokumentacija pod brojem MB_Diplomski_09 i MB_Diplomski_10

3.6.2. Kućište radnog prostora robota

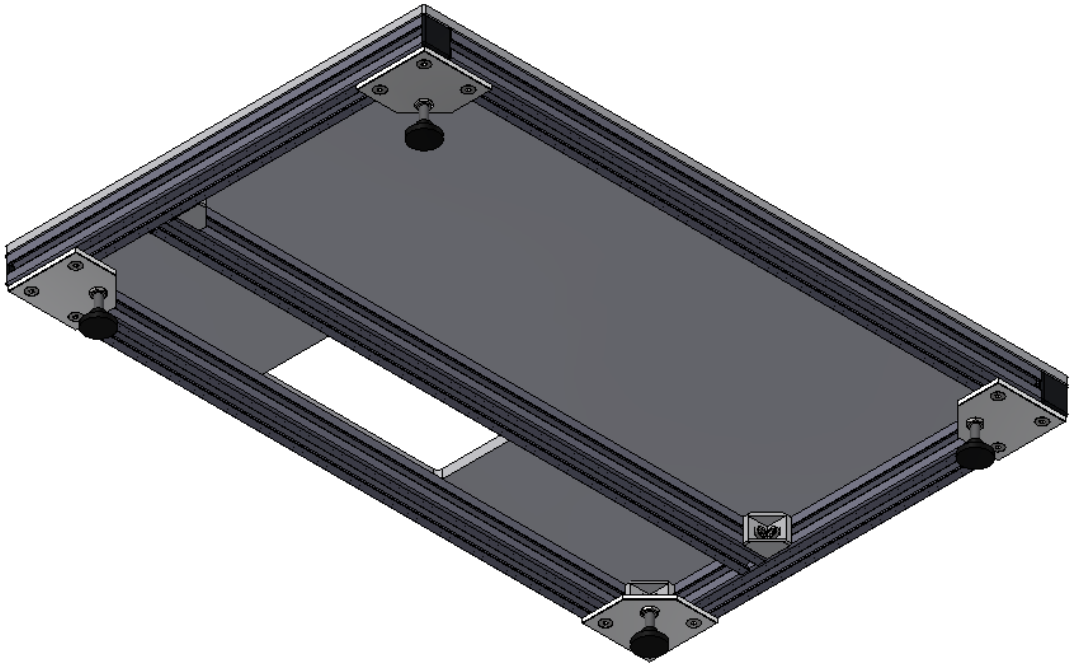
Podloga za radni prostor robota je bijeli pleksiglas („housing_1“) dimenzija 715x465 mm debljine 10 mm koji je dobiven laserskim rezanjem te se za njega u prilogu [II] nalazi tehnička dokumentacija pod brojem MB_Diplomski_11. Glavnu konstrukciju kućišta čine 3 komada *Bosch Rexroth* ekstrudiranog aluminija 3030 duljine 655 mm te 2 komada *Bosch Rexroth* ekstrudiranog aluminija 3030 duljine 459 mm. Pleksiglas je povezan s ekstrudiranim aluminijem DIN 7991 M6x16 vijcima i N8 T-NUT M6 maticama. Slika 85 i Slika 86 prikazuju izgled kućišta.



Slika 85. Kućište radnog prostora robota, pogled 1

Slika 86. Kućište radnog prostora robota, pogled 2

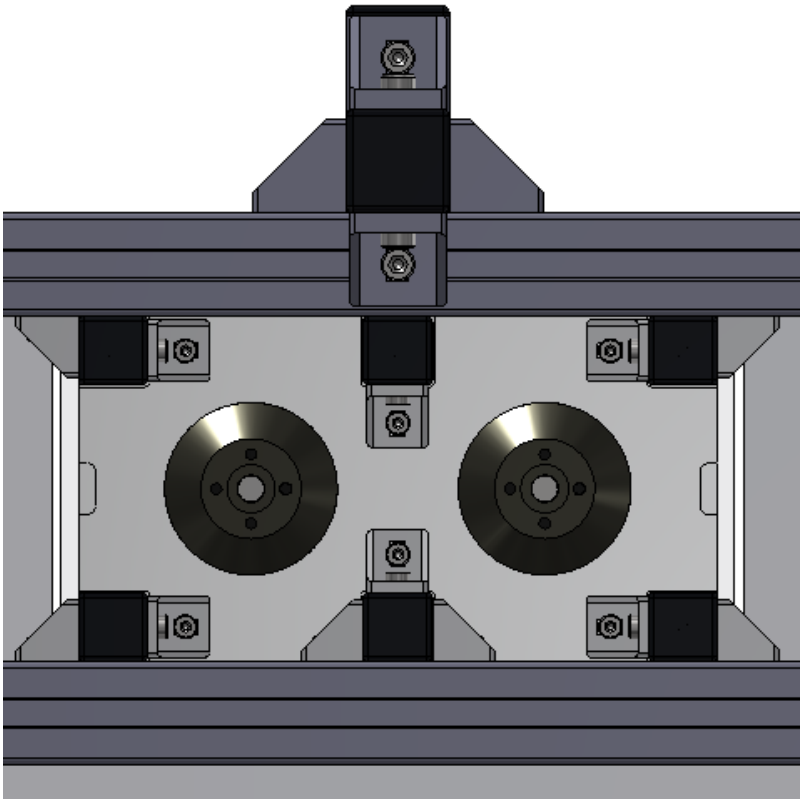
Ekstrudirani aluminiji međusobno je povezan s kutnicima, DIN 6912 M6x14 vijcima i N8 T-NUT M6 maticama. Na kućištu se nalaze i 4 nogice koje imaju mogućnost namještanja visine te tako omogućuju da se robot nalazi i na neravnom terenu. Nogice su montirane na aluminijske pločice („housing_3“) debljine 5 mm koje su isto izrađene laserskim rezanjem te se i za njih u prilogu nalazi [II] tehnička dokumentacija pod brojem MB_Diplomski_12. Pločice imaju na sebi urezan M8 navoj na koji se steže kompenzacijska nogica. Na nogicu ide i DIN 439 M8 matica koja služi protiv otpuštanja stezanog spoja. Pločice su montirane na ekstrudirani aluminij s DIN 7991 M6x12 vijcima i N8 T-NUT M6 maticama.



Slika 87. Kućište radnog prostora robota, pogled 3

Na kućište radnog prostora robota pričvršćeno je kućište motora s 2020 kutnicima, DIN 912 M4x10 vijcima, DIN 6912 M4x8 vijcima te maticama N6 T-NUT M4 i N8 T-NUT M6. Slika 88 prikazuje montirano kućište motora na kućište radnog prostora robota.

Isto tako je kućište mikroračunala i kamere pričvršćeno na kućište radnog prostora robota s 3030 kutnicima, DIN 6912 M6x14 vijcima te N8 T-NUT M8 maticama. Slika 89 prikazuje montirano kućište mikroračunala i kamere na kućište radnog prostora robota.



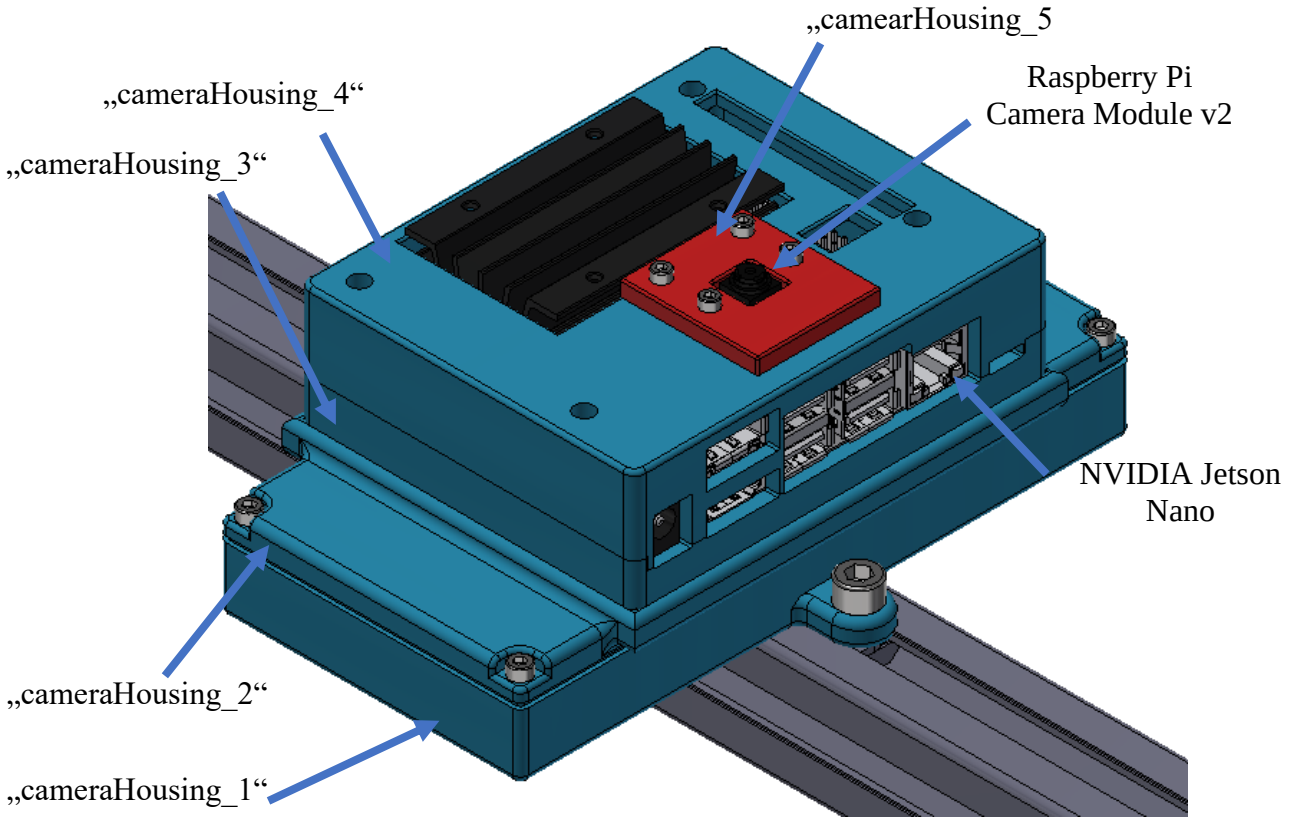
Slika 88. Montirano kućište 1

Slika 89. Montirano kućište 2

3.6.3. Kućište mikroračunala i kamere

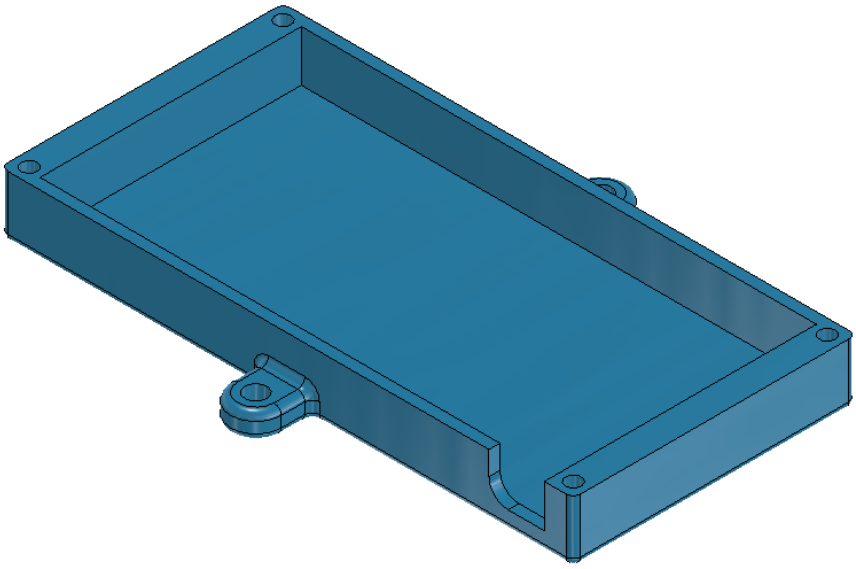
Odlučeno je da će robotom upravljati mikroračunalo *NVIDIA Jetson Nano* te je zadatkom zadana upotreba vizijskih sustava. Kamera koja se koristi za vizijski sustav je *Raspberry Pi Camera Module v2* te se ona povezuje sa *Jetson Nano* uporabom *ribbon* kabla koji je kratak. Stoga je bilo potrebno napraviti kućište za kameru i mikroračunalo koje će se nalaziti iznad samog robota.

Kućište se sastoji od 5 elemenata. „cameraHousing_1“ i „cameraHousing_2“ čine kućište za SSD disk. „cameraHousing_1“ na sebi ima prihvate pomoću kojih se montira na ekstrudirani aluminiji sa DIN 912 M6x12 vijcima i N8 T-NUT M6 maticama. „cameraHousing_2“ spaja se na „cameraHousing_1“ s DIN 912 M3x10 vijcima i M3 umetcima. „cameraHousing_3“ je donji dio kućišta za *NVIDIA Jetson Nano* te se spaja sa „cameraHousing_2“ preko DIN 912 M3x8 vijaka i M3 umetcima. Na „cameraHousing_3“ ide *Jetson Nano* i onda „cameraHousing_4“. Oni se zajedno spajaju preko DIN 912 M2,5x16 vijaka i M2,5 umetka. Na „cameraHousing_4“ se stavlja *Raspberry Pi Camera Module V2*. Kamera se zatvara s „cameraHousing_5“ poklopcem koje se povezuje s kućištem korištenjem DIN 912 M2,5x10 vijaka i M2,5 umetaka. Slika 90 prikazuje sastavljeno kućište.

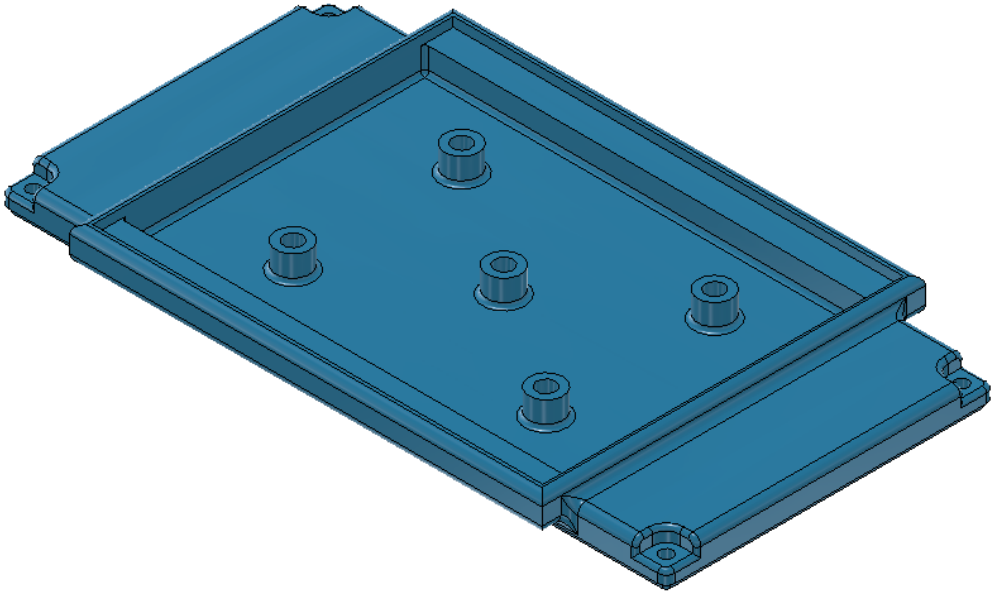


Slika 90. Kućište za mikroračunalo i kameru

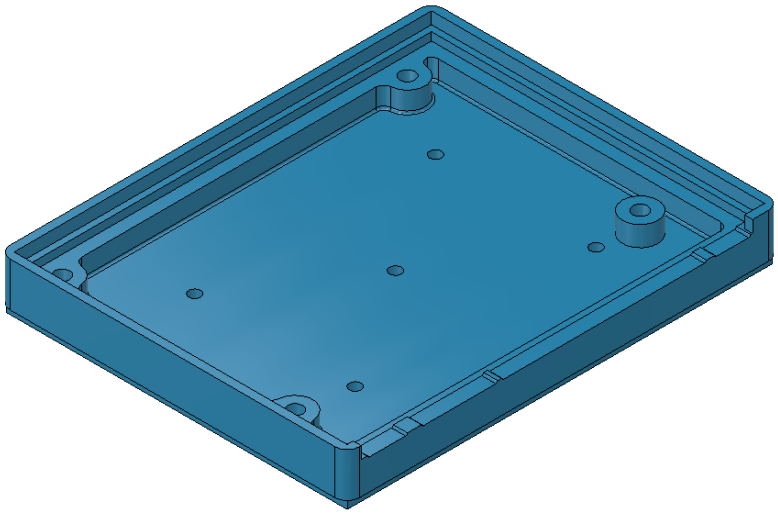
CAD model za *NVIDIA Jetson Nano* i *Raspberry Pi Camera Module v2* preuzeti su sa stranice *https://grabcad.com/library*[21][22]. Svih 5 dijelova kućišta izrađeno je aditivnom proizvodnjom korištenjem *FDM* printera. Kućište je montirano na 3030 ekstrudirane aluminijske profile duljina 460 mm i 750 mm koji su međusobno povezani s 3030 kutnicima, DIN 6912 M6x14 vijcima te N8 T-NUT M8 maticama. Slika 91, Slika 92, Slika 93, Slika 94 i Slika 95 prikazuju izgled kućišta.



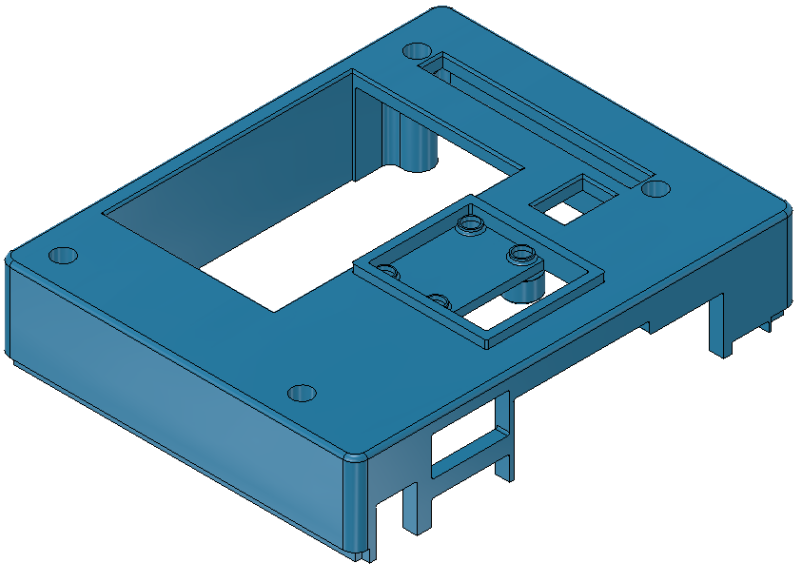
Slika 91. „cameraHousing_1“



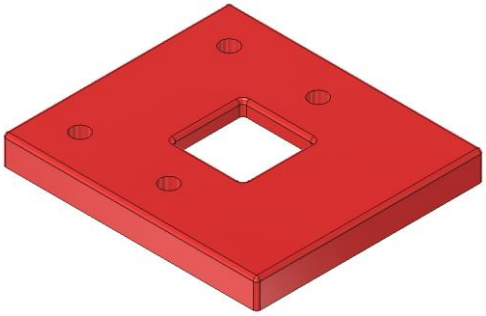
Slika 92. „cameraHousing_2“



Slika 93. „cameraHousing_3“



Slika 94. „cameraHousing_4“

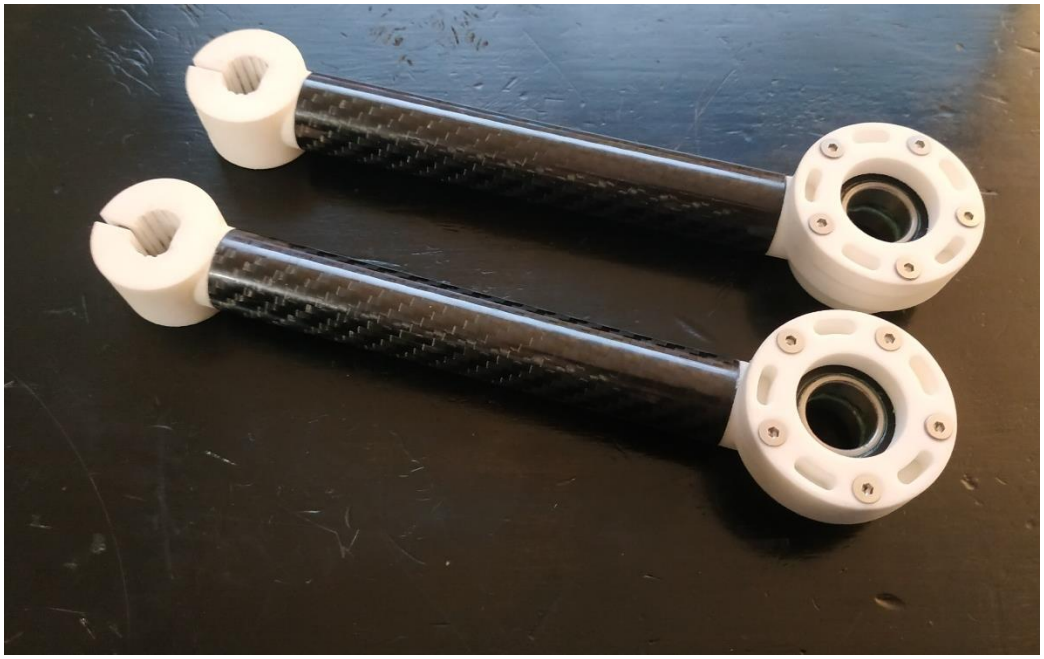


Slika 95. „cameraHousing_5“

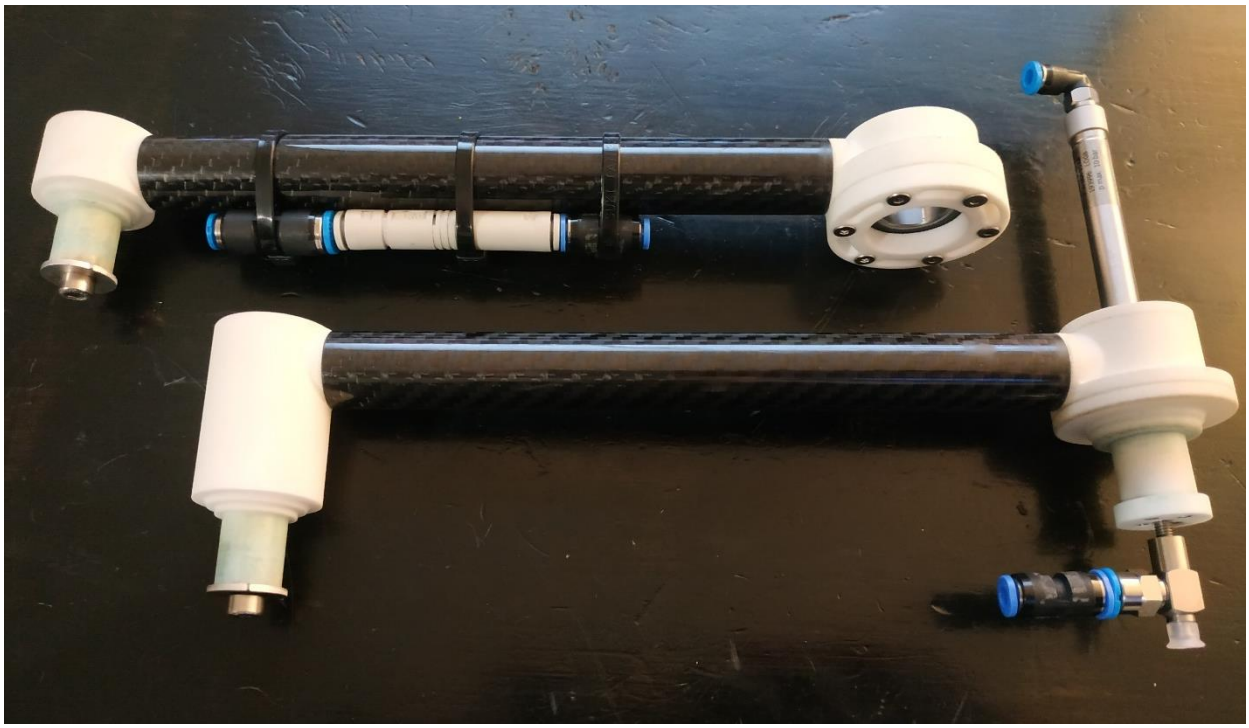
3.7. Montaža robota

Nakon što je cijeli koncept detaljno razrađen, svi dijelovi su izrađeni i robot je montiran. U ovom poglavlju je kroz nekoliko slika prikazati tijek montaže robota.

Slika 96. Montaža robota 1

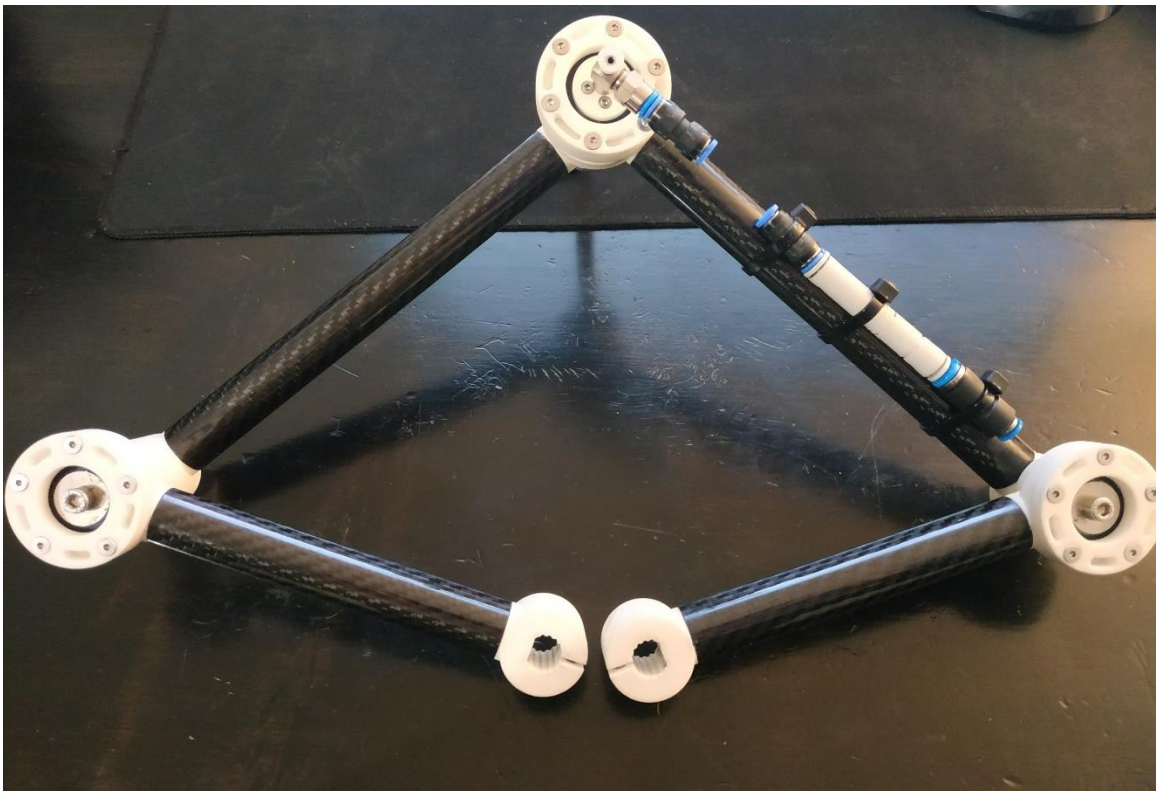


Slika 97. Montaža robota 2

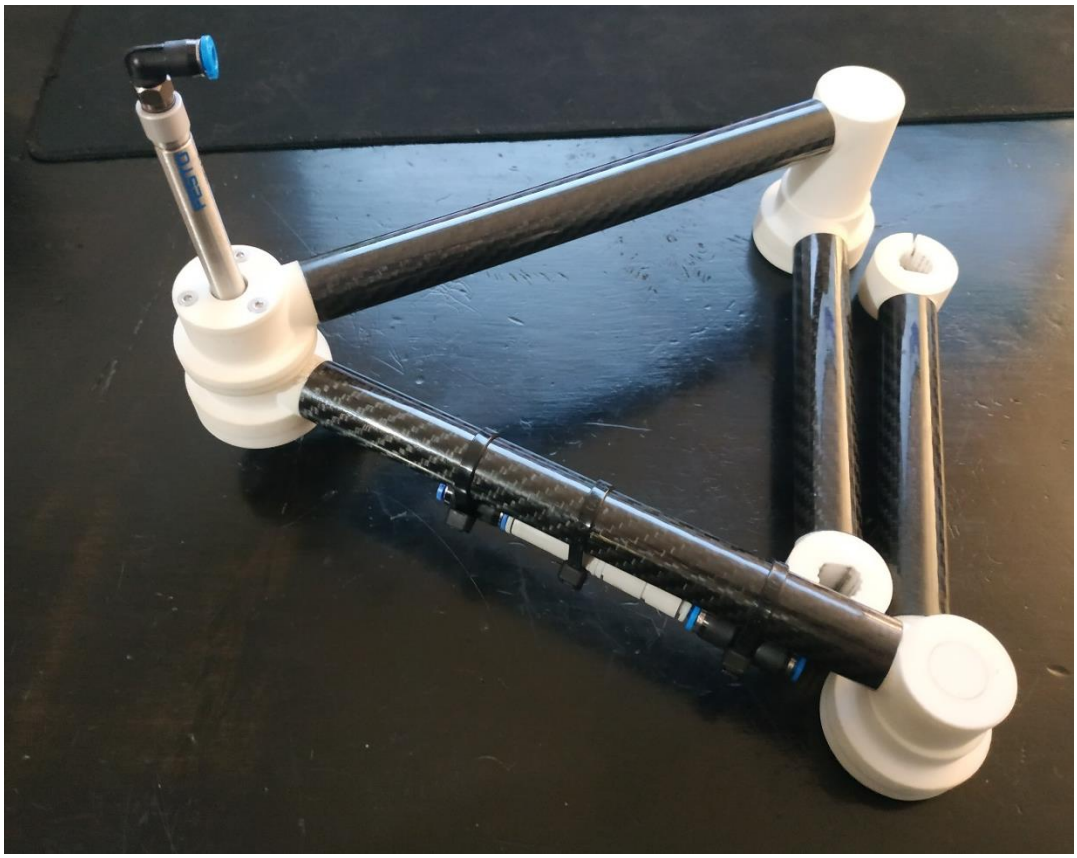


Slika 98. Montaža robota 3

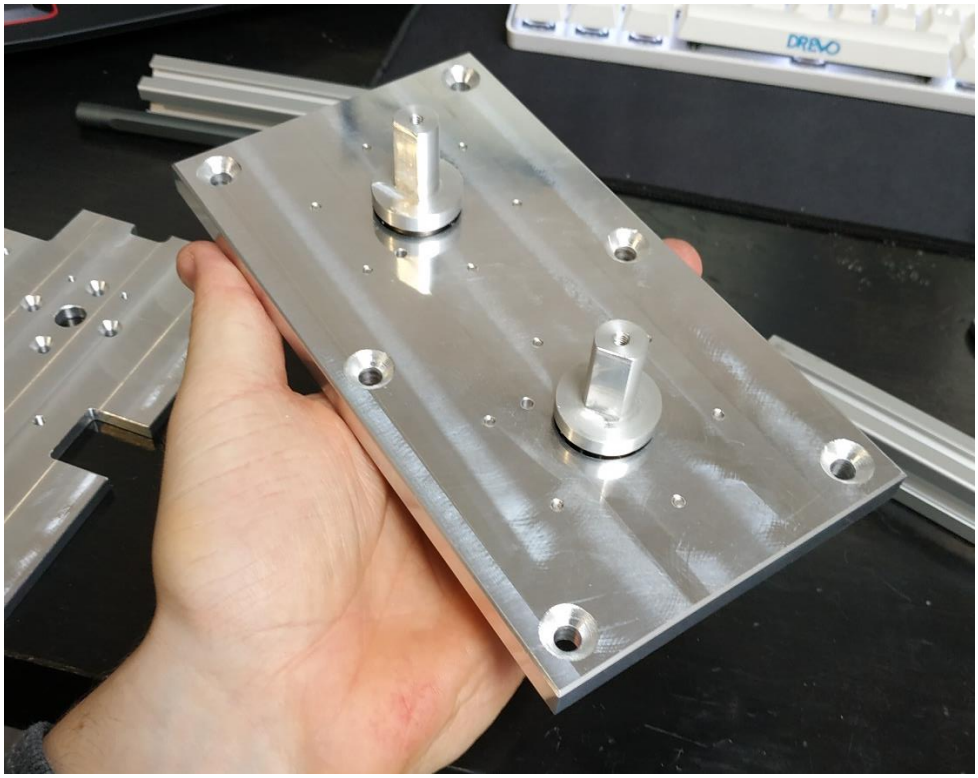
Slika 99. Montaža robota 4



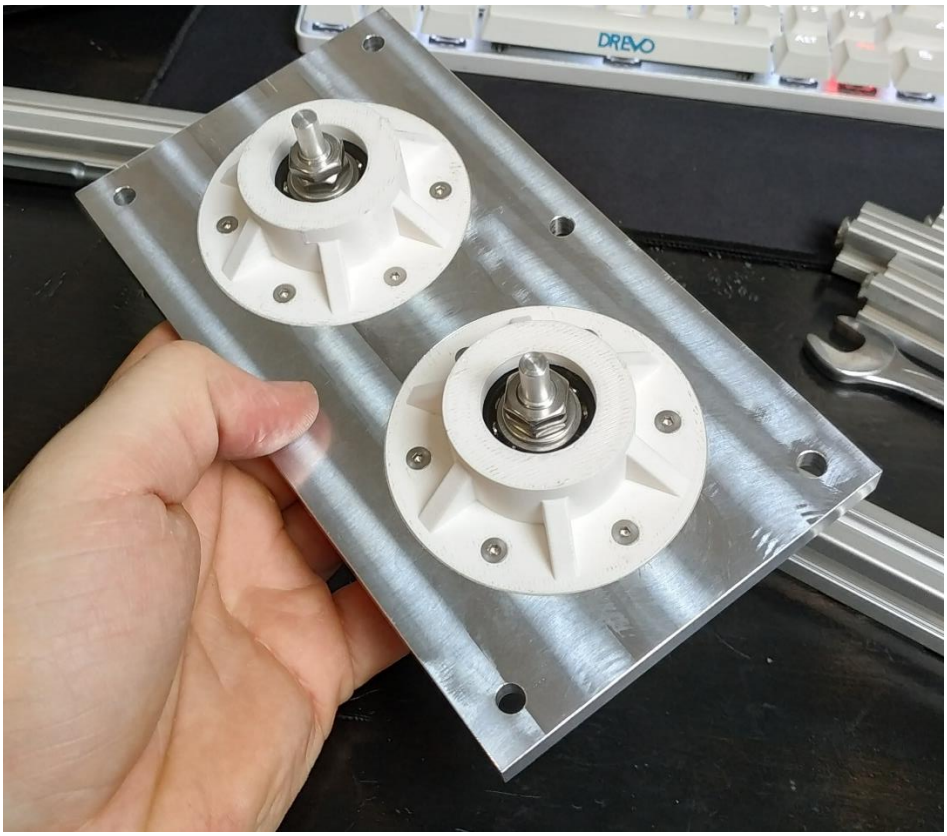
Slika 100. Montaža robota 5



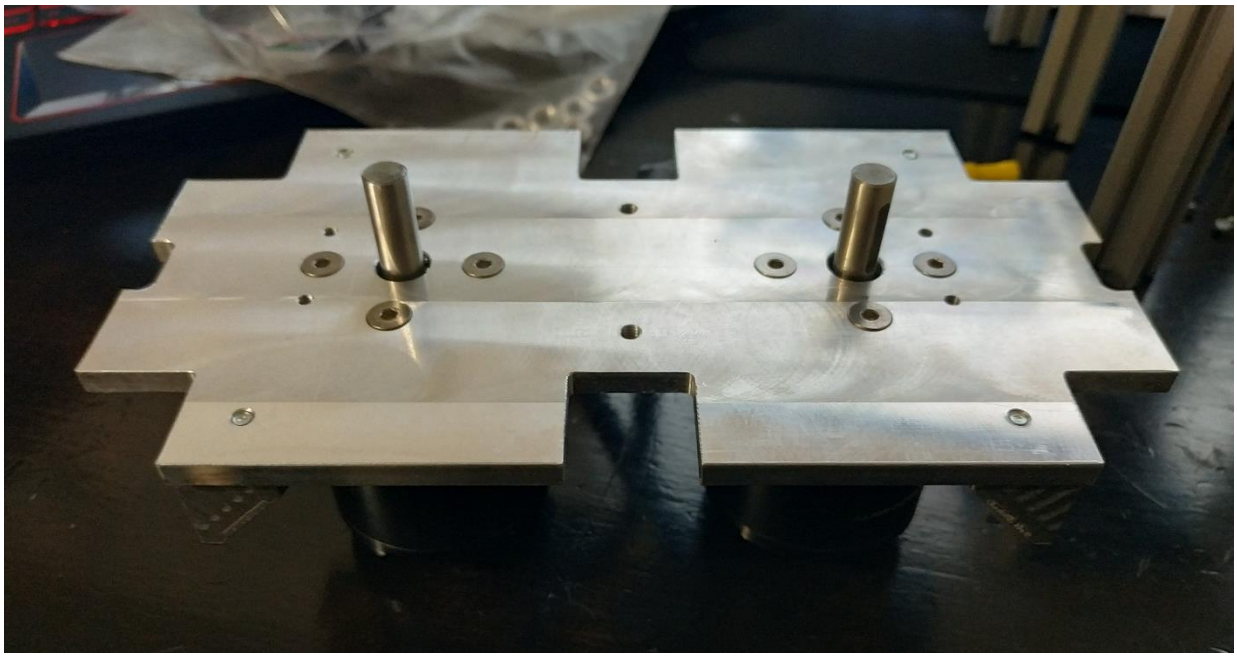
Slika 101. Montaža robota 6



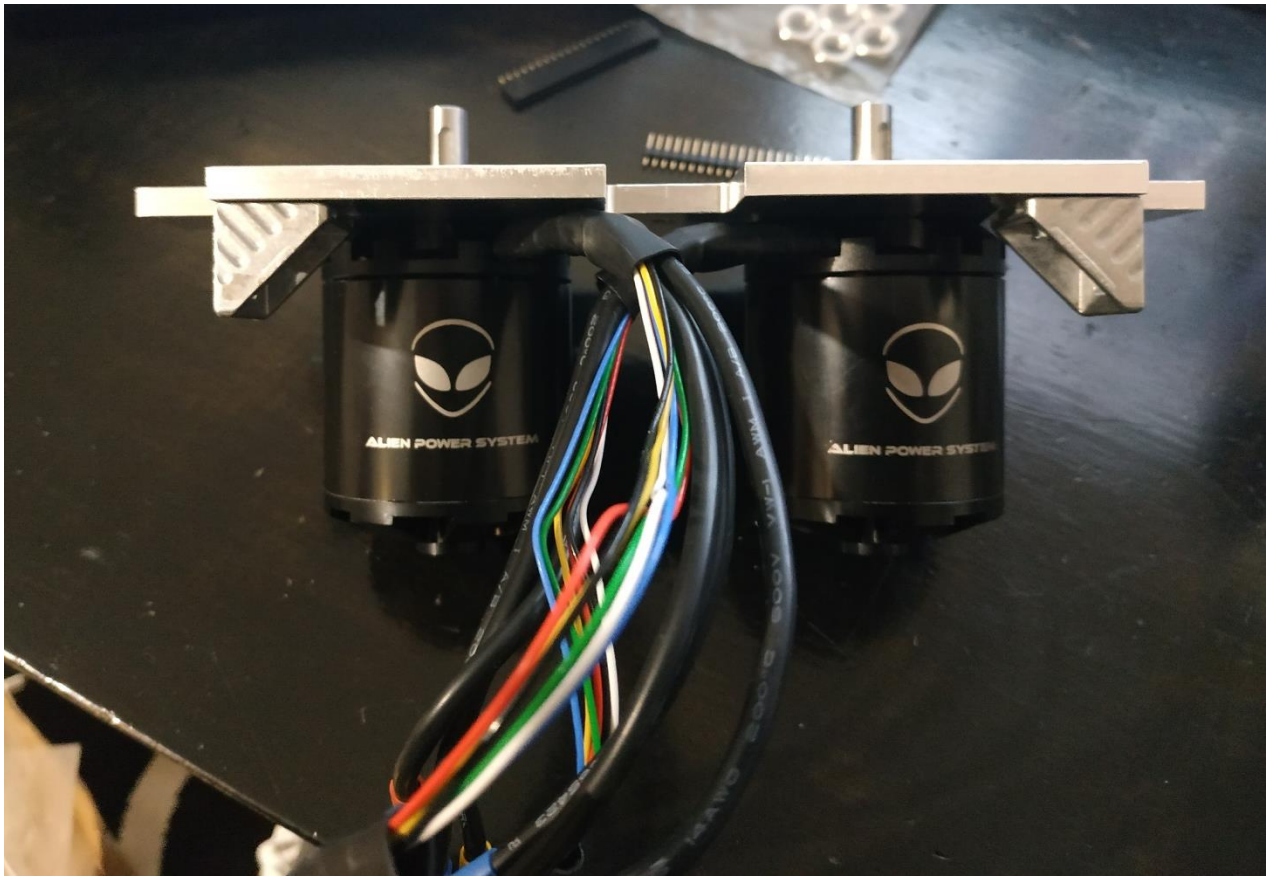
Slika 102. Montaža robota 7



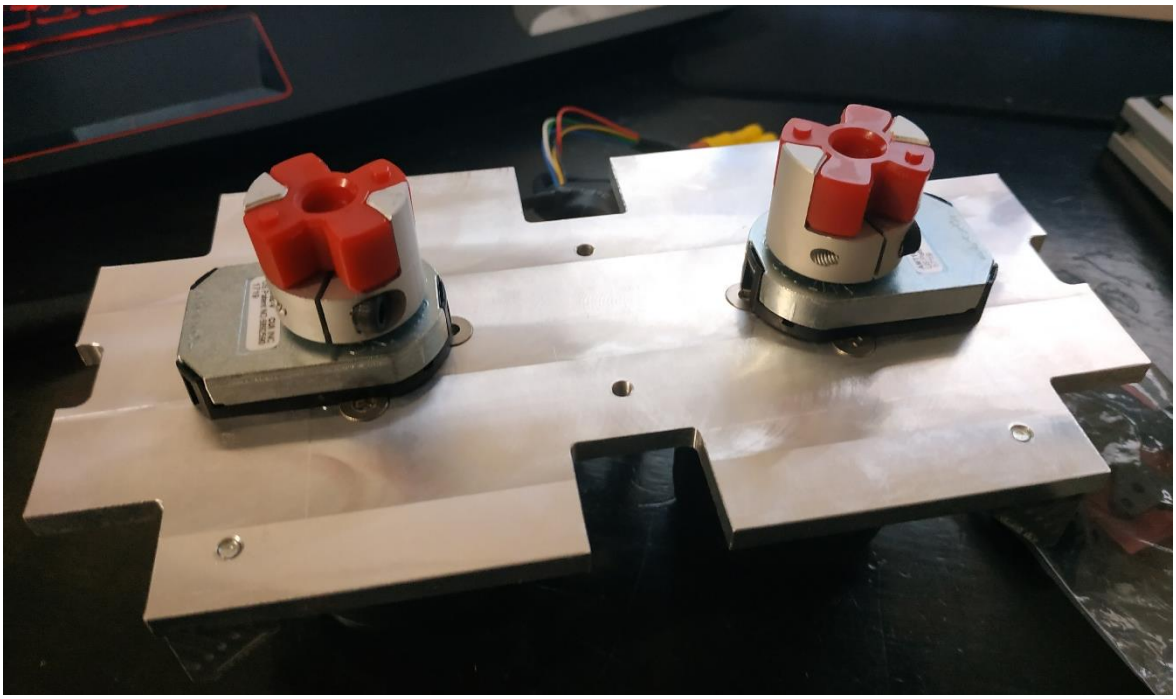
Slika 103. Montaža robota 8



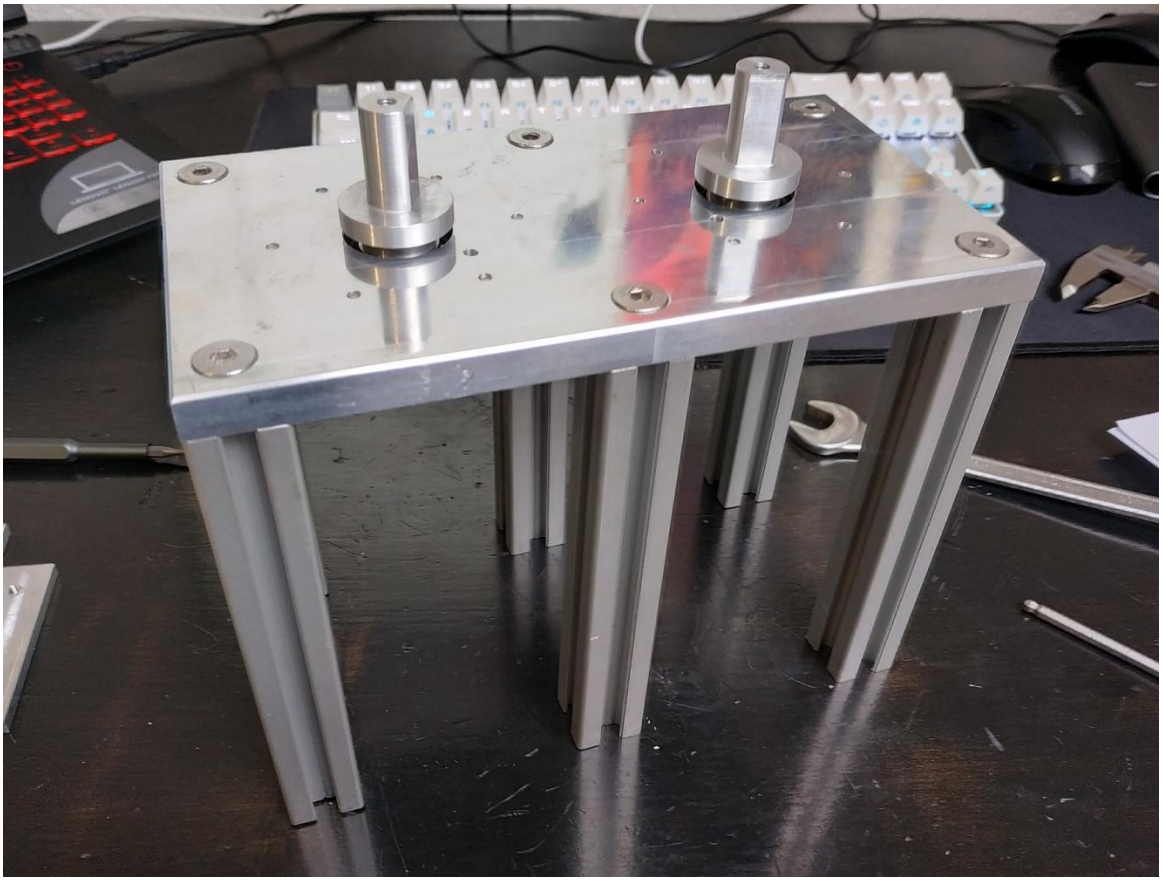
Slika 104. Montaža robota 9



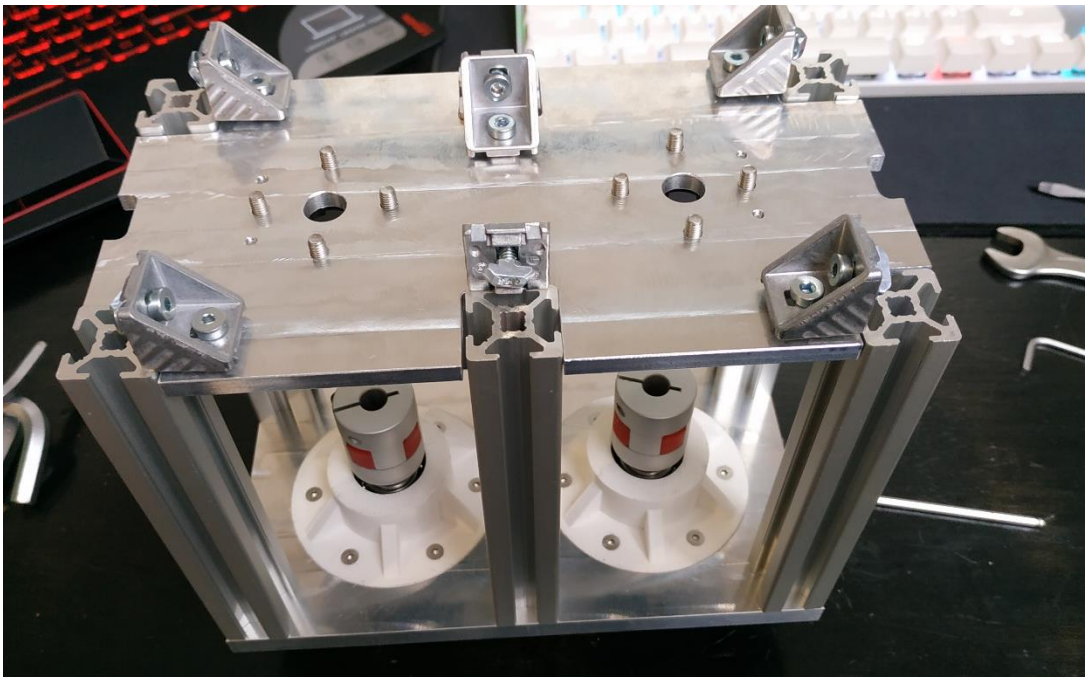
Slika 105. Montaža robota 10



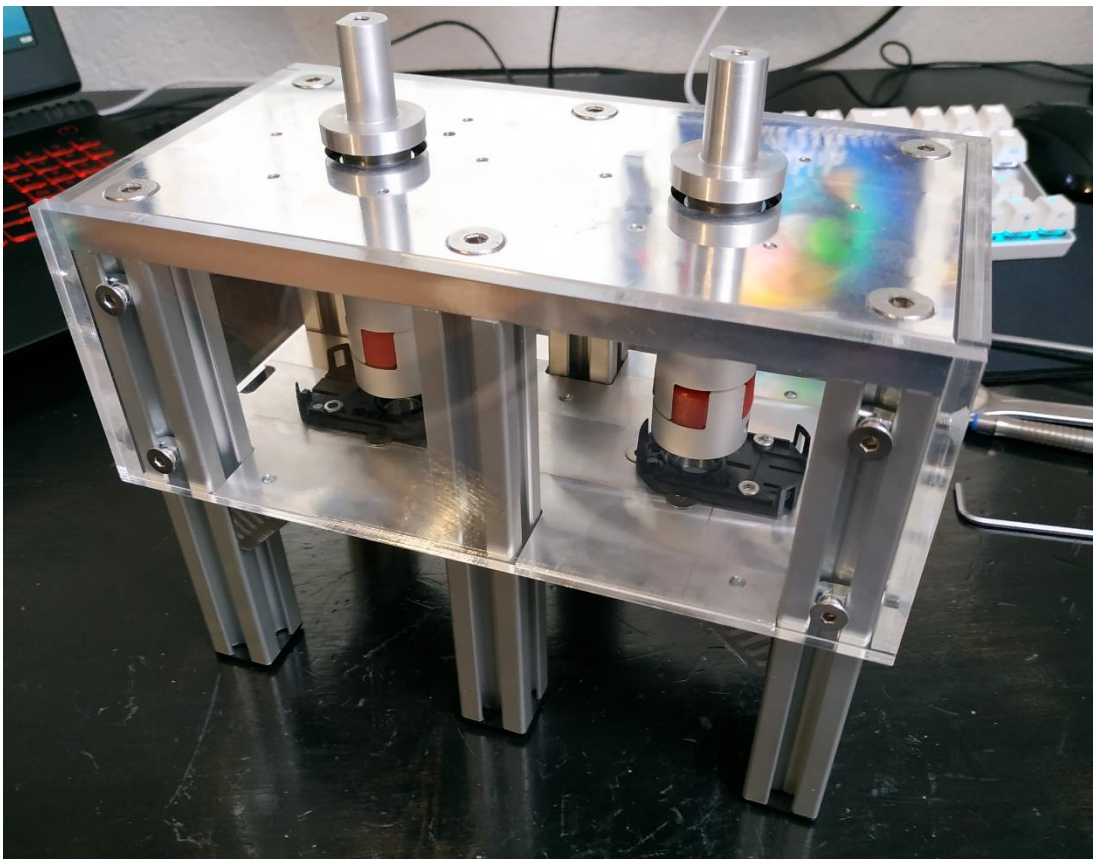
Slika 106. Montaža robota 11



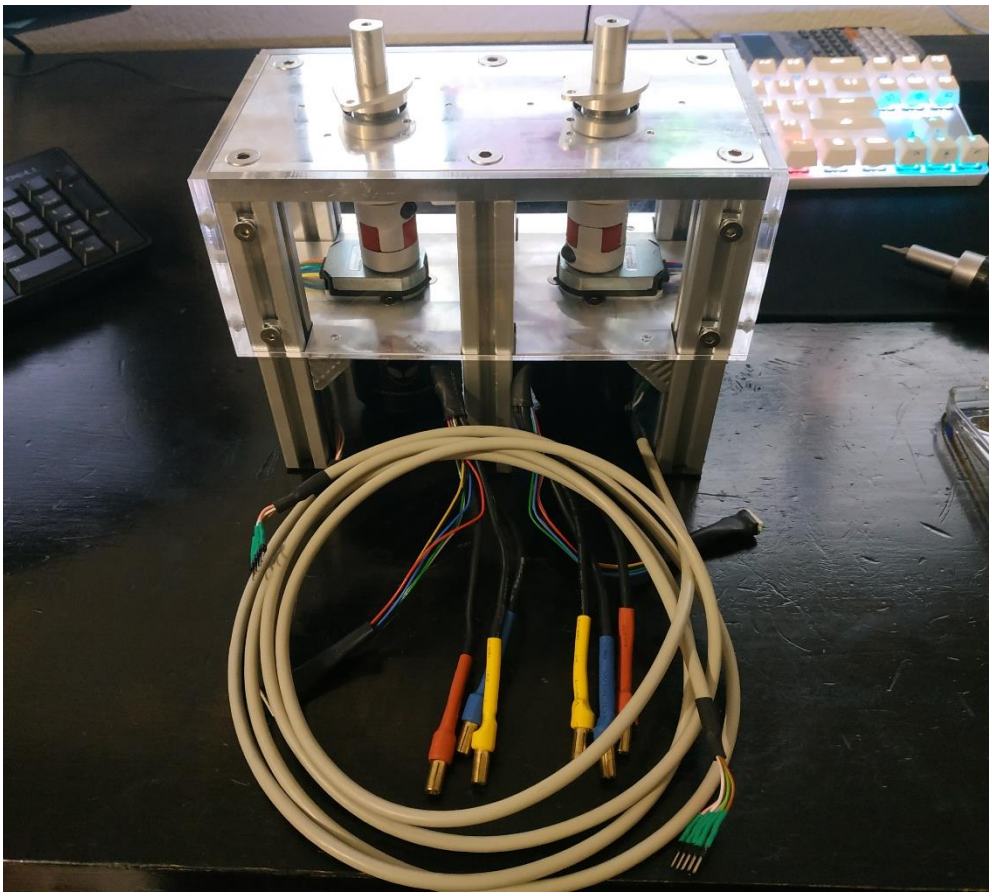
Slika 107. Montaža robota 12



Slika 108. Montaža robota 13



Slika 109. Montaža robota 14



Slika 110. Montaža robota 15

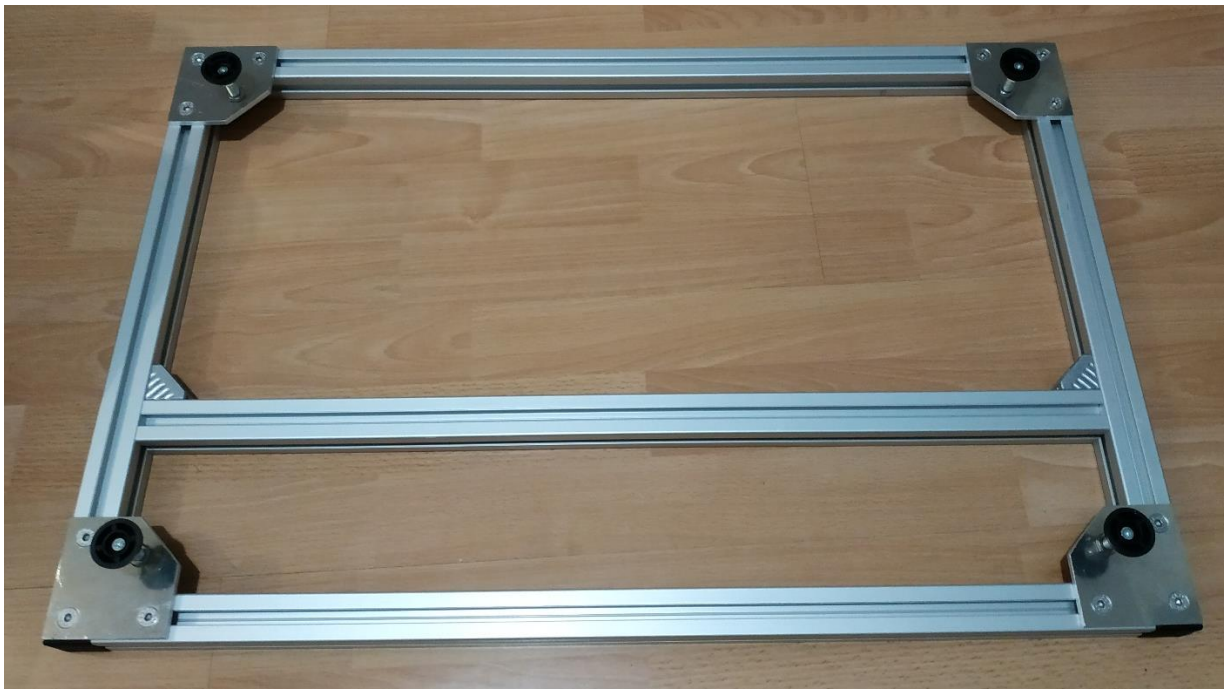
Slika 111. Montaža robota 16

Slika 112. Montaža robota 17

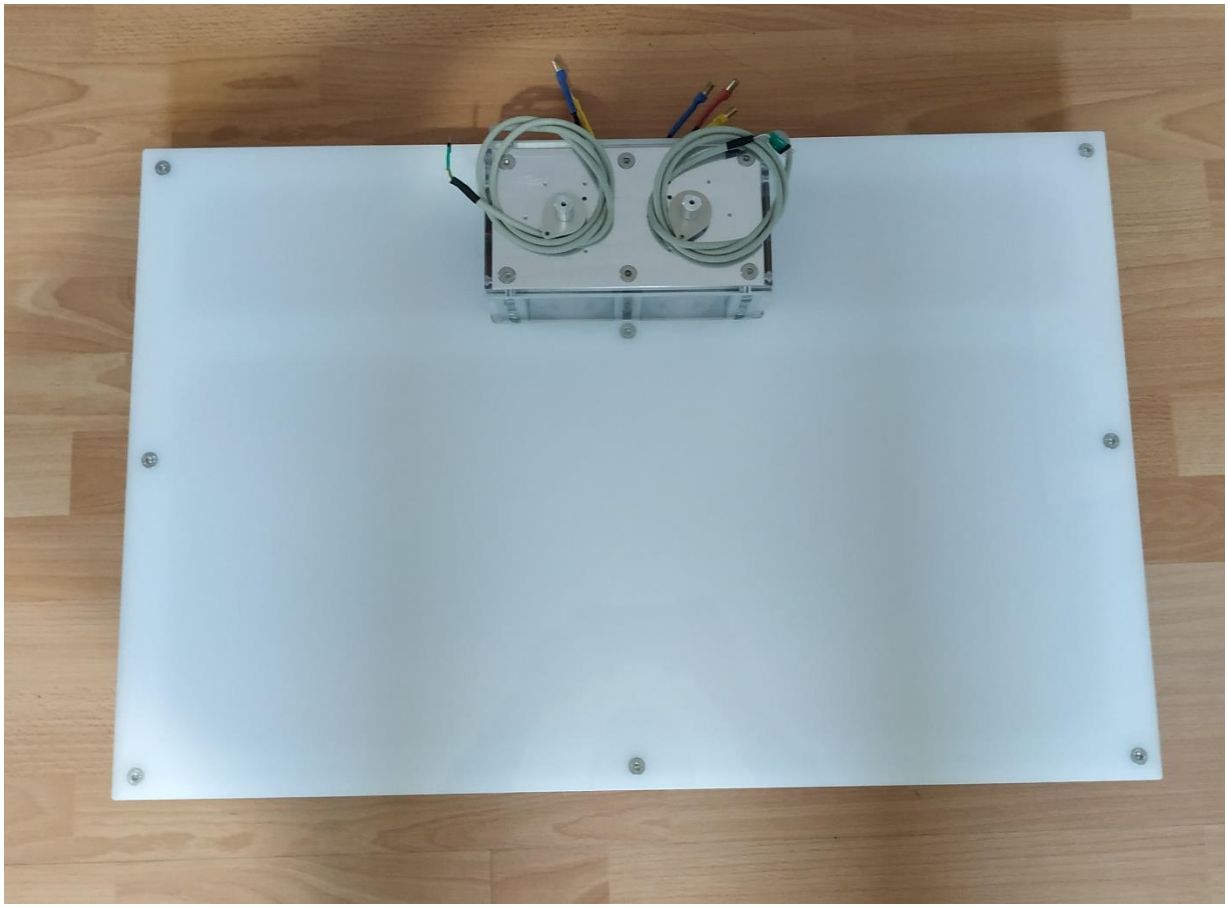
Slika 113. Montaža robota 18

Slika 114. Montaža robota 19

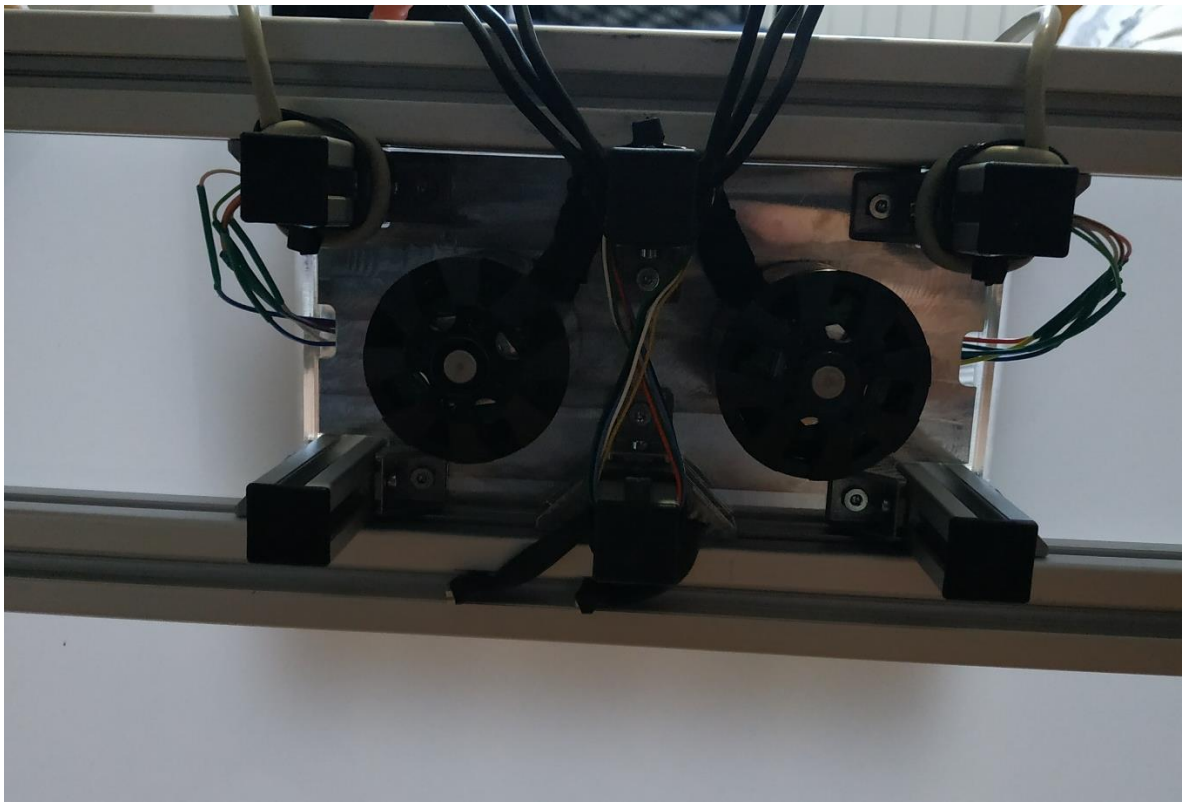
Slika 115. Montaža robota 20



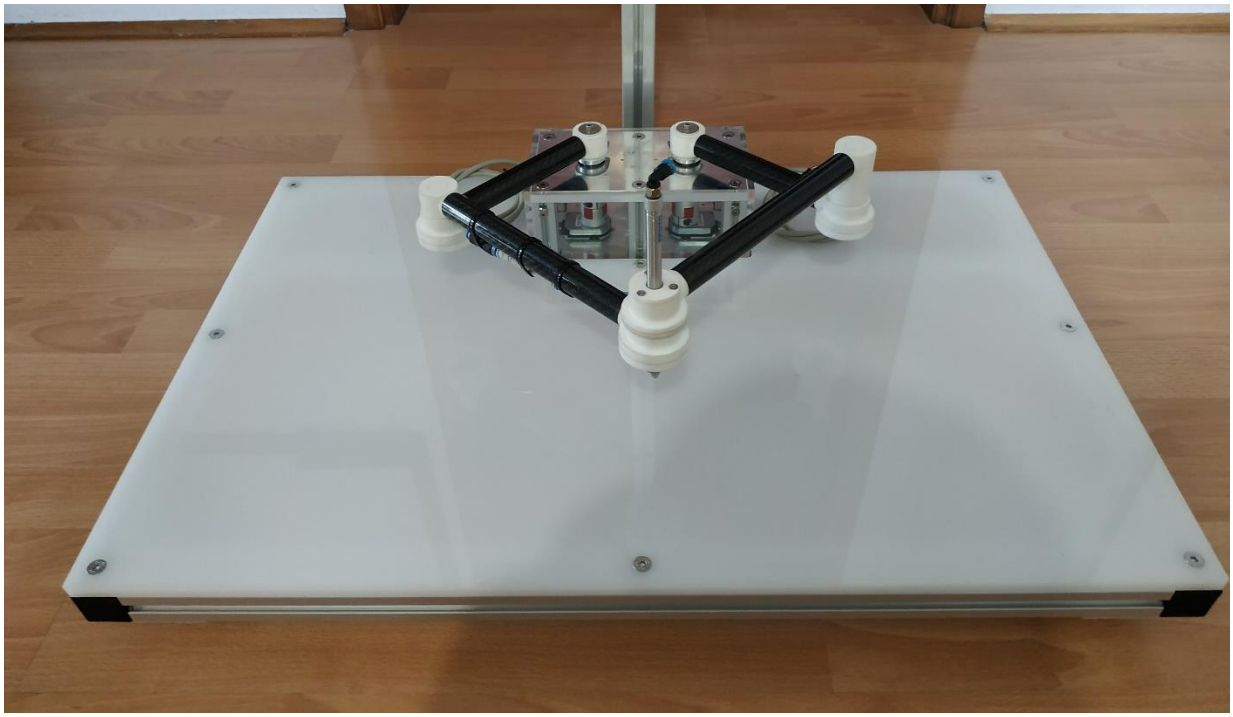
Slika 116. Montaža robota 21



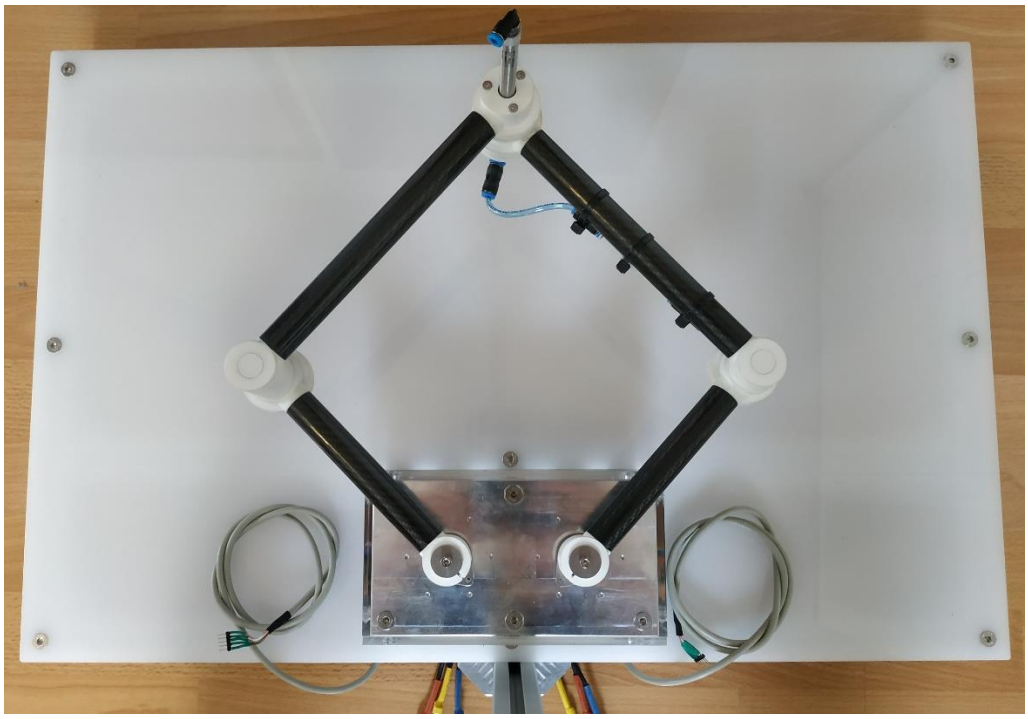
Slika 117. Montaža robota 22



Slika 118. Montaža robota 23



Slika 119. Montaža robota 24



Slika 120. Montaža robota 25

Slika 121. Montaža robota 26

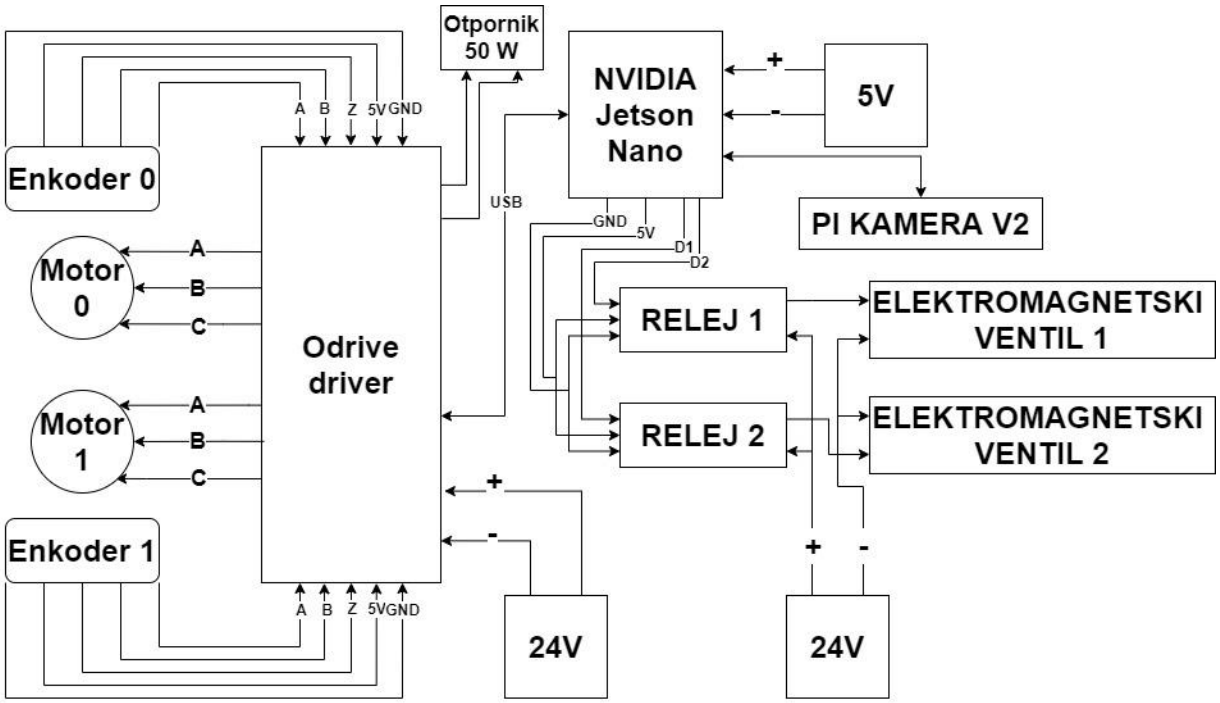
Slika 122. Montaža robota 27

4. UPRAVLJANJE

Nakon izrađenog eksperimentalnog postava robota, za isti je bilo potrebno napisati programski kod koji će upravljati njime. Kroz ovo poglavlje prvo će se pokazati kako je spojena sva elektronika koja upravlja robotom, zatim je objašnjena struktura koda i dijagram procedure rada robota. Nakon toga je svaki korak procedure rada robota detaljnije objašnjen. Kod za upravljanje robotom napisan je u programskom jeziku *Python*. Jedan od zadataka bio je napraviti grafičko sučelje, koje je izrađeno pomoću *Tkinter* modula u *Python-u*. Komunikacija između *ODrive* drivera i mikroračunala je preko *USB* protokola. Kod za slanje naredbi driveru također je napisan u Python programskom jeziku koristeći *odrivetool* modul koji je preuzet sa stranice *https://docs.odriverobotics.com*[23]. Upravljanje robotom pomoću vizijskih sustava napravljeno je pomoću *OpenCV* modula za *Python* programski jezik. Kroz poglavlje će se prikazati bitniji dijelovi koda, dok se kompletan kod nalazi u prilogu pod [I].

4.1. Shema spajanja elektronike robota

Mikroračunalo koje upravlja radom robota je *NVIDIA Jetson Nano* koje radi na *Linux* operativnom sustavu. *Jetson Nano* odabran je zbog svojih malih dimenzija, a dovoljno dobrih specifikacija za upravljanje robota. Male dimenzije i podrška za *Raspberry Pi Cameru V2* čini ga idealnim za montiranje na robota, kako prikazuje Slika 112. Za mikroračunalo potrebno je napajanje od 5V. Za upravljanje vakuum hvataljkom potrebna su dva elektromagnetska ventila koje prikazuje Slika 37. Kako upravljanje ventilima obavlja *Jetson Nano* koji radi na 5V, a ventili rade na 24V, potrebno je posebno napajanje od 24V za ventile te 2 elektromagnetska releja preko kojih će *Jetson Nano* upravljati s elektromagnetskim ventilima. Slika 123 prikazuje kako izgleda shema spajanja potrebne elektronike. Sve slike kojem imaju u svom opisu referencu [24] izrađene su korištenjem alata sa stranice *draw.io*[24].



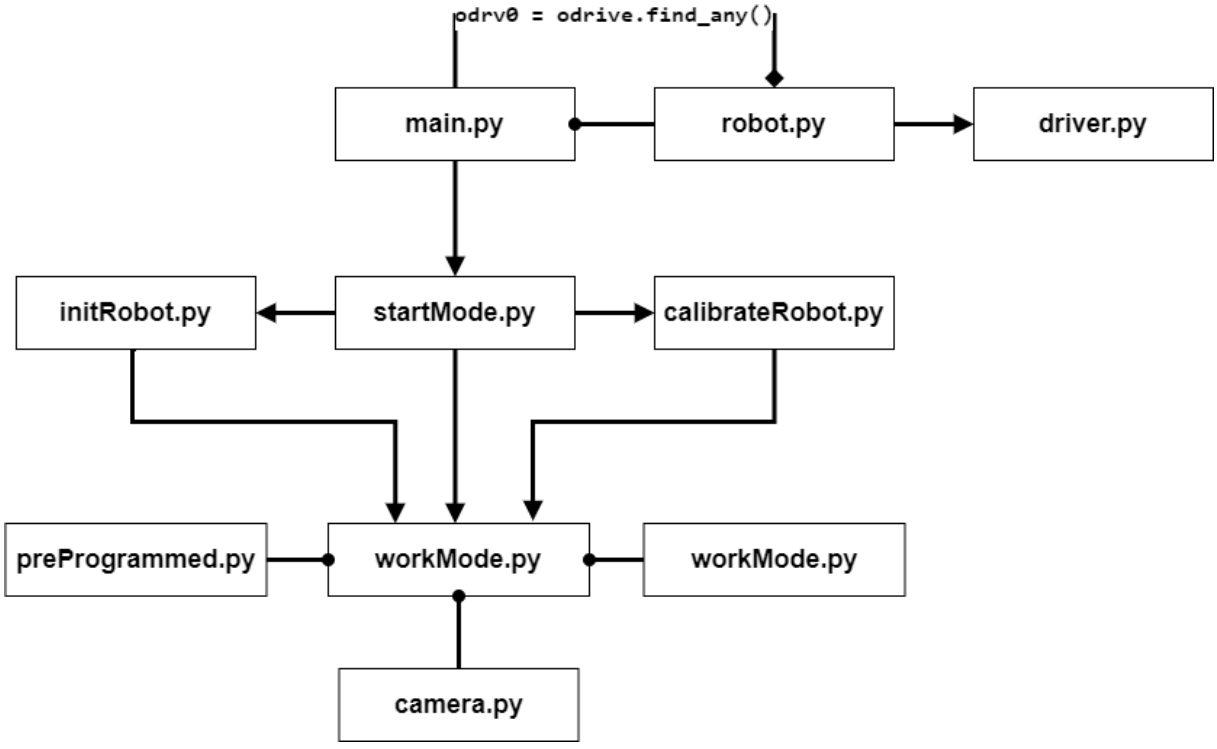
Slika 123. Shema spajanja elektronike[24]

Slika 124 prikazuje privremenu kutije za elektroniku robota.

Slika 124. Kutija za elektroniku robota

4.2. Struktura koda

Kako je *Python* objektno orijentirani programski jezik, tj. koristi klase i objekte i ovdje je korišten takav pristup kako bi se dobio modularniji i čitljiviji kod koji će kasnije biti lakše nadograditi. Tako postoje više različitih klasa. Npr. za *ODrive* driver napravljena je *Driver* klasa koja se nalazi unutar *driver.py* modula i ona služi za osnovnu komunikaciju između programa za upravljanje i drivera. Tu *Driver* klasu nasljeđuje *Robot* klasa koja se nalazi unutar *robot.py* modula. Nasljeđivanjem je omogućeno da *Robot* klasa u sebi sadrži sve atribute i metode koje su potrebne za upravljanje robotom, a opet je omogućena modularnost da *Driver* klasu također naslijedi neka druga klasa. Nadalje, postoji *Pneumatic* klasa koja se nalazi unutar *pneumatic.py* modula, a služi za operacije izuzmi i odloži koje se izvršavaju pozivanjem odgovarajućih metoda. Te metode upravljaju s dva elektroventila, čijim se paljenjem i gašenjem kontrolira rad vakuumske hvataljke. Osim *Driver*, *Robot* i *Pneumatic* klase, postoji i *Camera* klasa, koja u sebi sadrži razne atribute i metode pomoću kojih se prikazuje slika, radi postupak prepoznavanja objekata, kalibracija kamere, kalibracija koordinatnog sustava itd., a nalazi se unutar *camera.py* modula. Klase koje se nalaze unutar *tkinterWidgets.py* modula sadrže razne *Tkinter Widgets* klase koje služe za izradu grafičkog sučelja robota i imaju zadane ulazne parametre kako bi se kod napravio čitljivijim i smanjio njegov broj linija. Slika 125 prikazuje osnovnu strukturu koda**.**



Slika 125. Struktura koda[24]

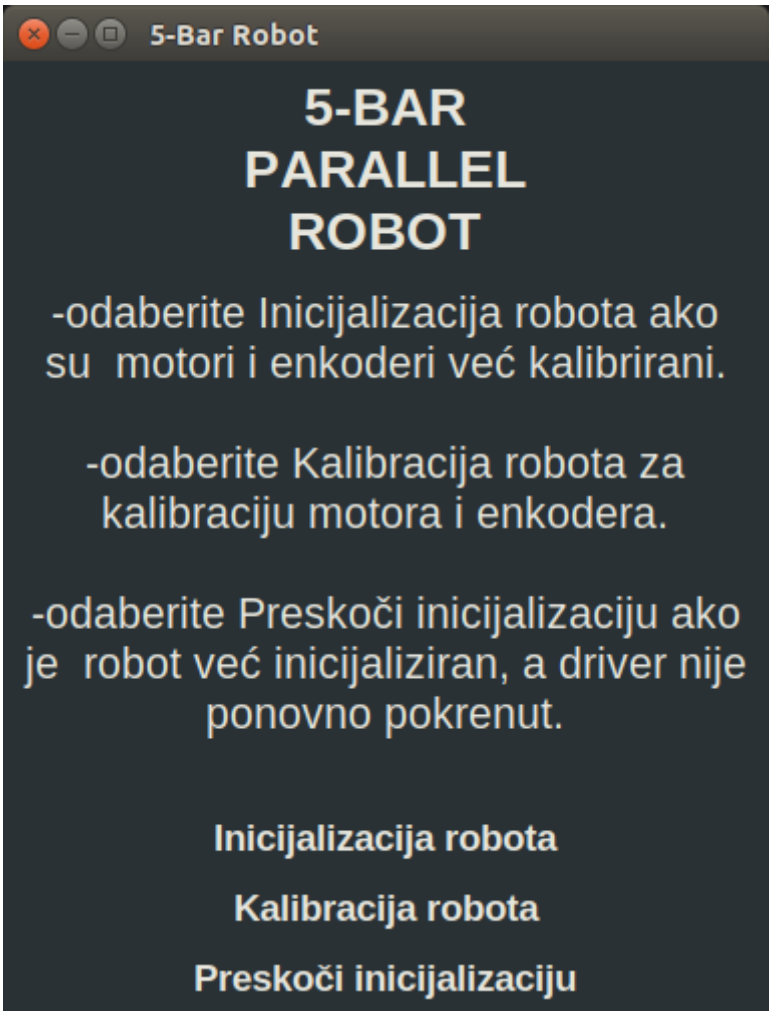
Strelicama je pokazan slijed izvršavanja koda robota, od početnog paljenja robota, inicijalizacije/kalibracije do rada robota. Program se pokreće preko *main.py* modula. Taj modul izvršava naredbu prikazanu niže.

```
odrv0 = odrive.find_any()
```

Tom naredbom dobiva se *odrv0* objekt koji se prosljeđuje *robot.py*, tj., *driver.py* modulima koji sadrže ranije spomenute *Driver* i *Robot* klase. *odrv0* omogućuje slanje raznih naredbi driveru preko *USB* kabela. Kada se inicijalizirala *Robot* klasa, program prelazi u *startMode.py* module gdje se može odabrati vrsta inicijalizacije ili kalibracije robota o čemu se više govori u poglavlju 4.3. Nakon izvršavanja inicijalizacije ili kalibracije robot prelazi u *workMode.py* module gdje se otvara glavno grafičko sučelje pomoću kojeg se upravlja robotom. U tom modulu se inicijaliziraju *Camera* i *Pneumatic* klasa te se uvozi *preProgrammed.py* module koji je kao i ostatak *workMode.py* objašnjen u 4.4.

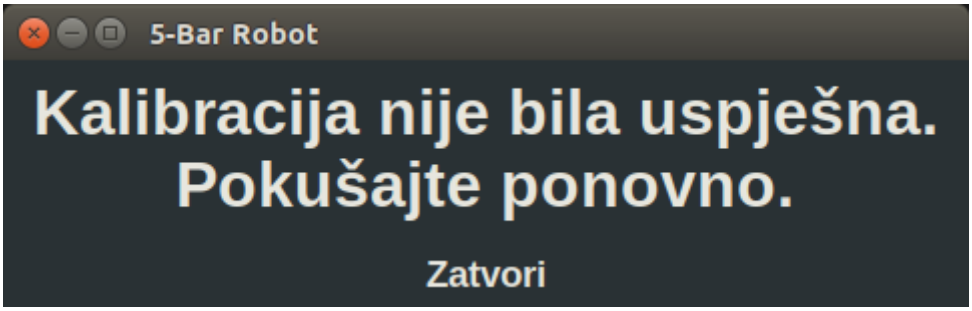
4.3. Inicijalizacija/kalibracija robota

Ranije je spomenuto kako robot koristi inkrementalne enkodere, što znači da je robot prilikom svakog paljenja drivera potrebno inicijalizirati kako bi znao gdje se nalazi u prostoru. Kada program uđe iz *main.py* u *startMode.py*, otvara se grafičko sučelje kojeg prikazuje Slika 126, gdje korisnik ima mogućnost odabira inicijalizacije robota koja je objašnjena u poglavlju 4.3.1, kalibracije robota koja je objašnjena u poglavlju 4.3.2 ili preskakivanja inicijalizacije koja je objašnjena u poglavlju 4.3.3.



Slika 126. GUI, početni prozor

Ako se bilo kada tijekom inicijalizacije ili kalibracije robota odustane, otvara se prozor kojeg prikazuje Slika 127 i daje do znanja kako je potrebno opet pokrenuti inicijalizaciju ili kalibraciju robota.

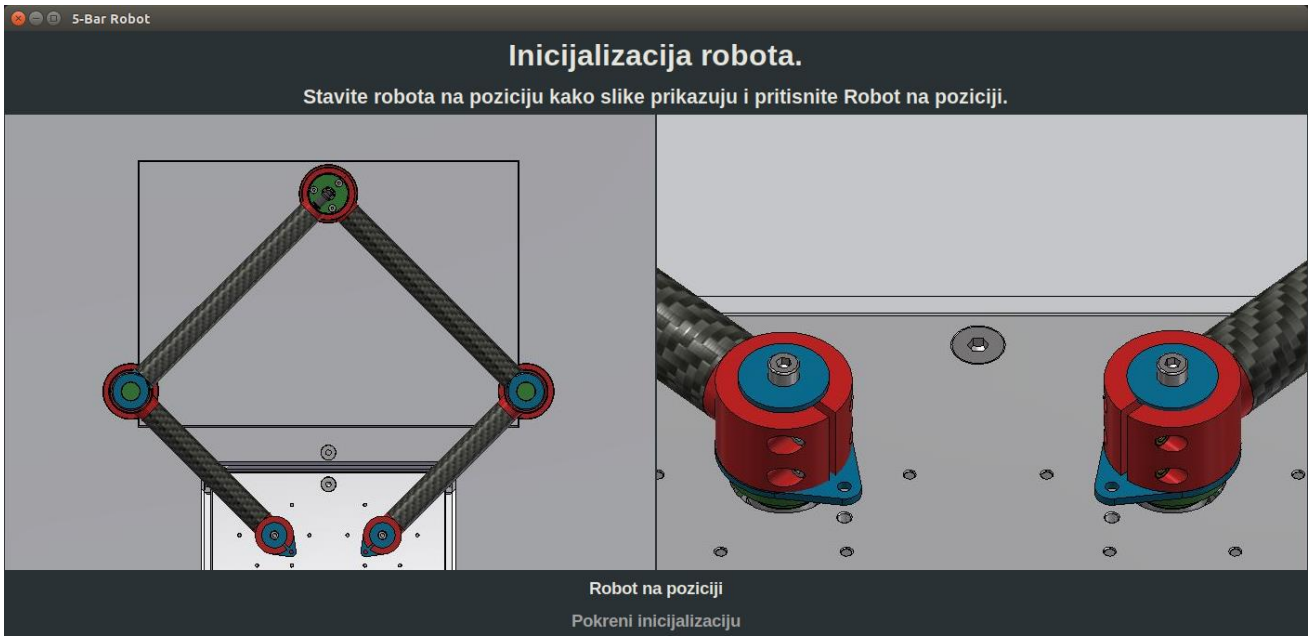


Slika 127. GUI, prekidanje kalibracije

4.3.1. Inicijalizacija robota

Kako enkoder ima Z-puls koji driver koristi za kalibraciju on je isto iskorišten za početnu inicijalizaciju robota. Slika 84 prikazuje rupe u koje idu pinovi za kalibraciju robota. Kada se poklope prolazna rupa aluminijske pločice („motorJoint_9“) i slijepa rupa aluminijske ploče od 10 mm („motorJoint_1“), tada su ulazni kutovi robota 45 ° za desni kinematski lanac (motor0) i 135 ° za lijevi kinematski lanac (motor1). Ta pozicija robota uzeta je i kao početna pozicija robota (*eng. home*). Ulazni kutovi referenciraju se prema koordinatnom sustavu kako ga Slika 76 prikazuje. Na toj slici robot se nalazi u početnoj poziciji. Potrebno je namjestiti da se Z-puls enkodera nalazi na istoj pozicija kao i kalibracijski pinovi. Kada se to jednom napravi, program zna da se nakon svake inicijalizacije robota, kada driver nađe Z-puls, enkoder0 na motoru0 ima vrijednost 0 za ulazni kut robota od 45 ° i enkoder1 na motoru1 ima vrijednost 0 za ulazni kut robota od 135 °.

Prilikom inicijalizacije robota, program prelazi u *initRobot.py* modul koji koristi metode iz *robot* objekta. Motori zbog konstrukcijskih ograničenja robota nikada ne naprave puni okret, dok driver ima ograničenje traženja Z-pulsa enkodera tako da se motor krene okretati nekoliko stupnjeva u jednu stranu, zatim okrene smjer vrtnje i traži Z-puls u drugu stranu i radi puni okret dok ga ne pronađe. Da bi se izbjegla kolizija robota sa samim sobom, grafičko sučelje napomene korisnika da stavi robota približno u poziciju koje prikazuje Slika 128, tj. u blizini Z-pulsa. Zatim pritiskom na „Robot na poziciji“ omogućuje pokretanje kalibracije. Potrebno je pritisnuti „Pokreni inicijalizaciju“ kako bi ista započela.



Slika 128. GUI, inicijalizacija robota

Inicijalizacija se izvodi tako da funkcije unutra *initRobo.py* modula pozovu metode od *robot* objekta. Prvo se poziva *sendStartParameters()* metoda koja driveru šalje zadane vrijednosti parametara motora i enkodera poput struje, brzine vrtnje itd. To se radi prilikom svakog pokretanja programa kako u driveru ne bi ostali neki ranije neželjeni parametri. Naredbama niže prikazuje se slanje parametara za struju i brzinu, dok se ostatak parametara koji se šalju može vidjeti u prilogu pod [I], *driver.py*.

```
#Struja
self.objectOdrv.axis0.motor.config.current_lim = self.AXIS_CURRENT_LIM
self.objectOdrv.axis1.motor.config.current_lim = self.AXIS_CURRENT_LIM
time.sleep(0.1)
#Brzina
self.objectOdrv.axis0.controller.config.vel_limit = self.AXIS_VEL_LIMIT
self.objectOdrv.axis1.controller.config.vel_limit = self.AXIS_VEL_LIMIT
time.sleep(0.1)
```

Kada se pošalju svi parametri driveru, pokreće se traženje Z-pulsa pozivanjem *zPulseSearch()* metode. Neki od najvažnijih naredbi metode su prikazane niže.

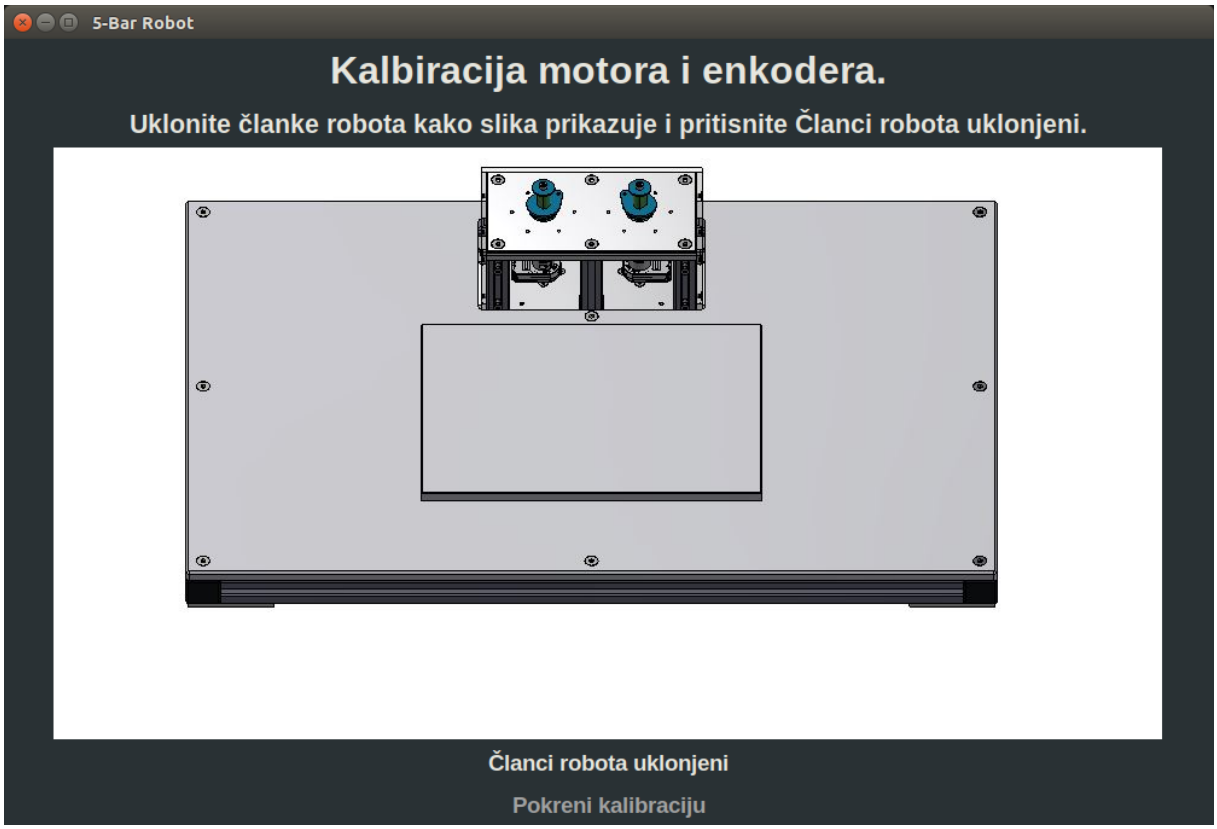
```
def zPulseSearch(self):
    self.objectOdrv.axis1.requested_state = AXIS_STATE_ENCODER_INDEX_SEARCH
    while self.objectOdrv.axis1.current_state != AXIS_STATE_IDLE:
        time.sleep(0.1)
    self.objectOdrv.axis1.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
    self.objectOdrv.axis1.controller.move_to_pos(0)
    time.sleep(0.25)
    self.objectOdrv.axis0.requested_state = AXIS_STATE_ENCODER_INDEX_SEARCH
    while self.objectOdrv.axis0.current_state != AXIS_STATE_IDLE:
        time.sleep(0.1)
    self.objectOdrv.axis0.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
    self.objectOdrv.axis0.controller.move_to_pos(0)
```

while je ovdje potreban kako program ne bi krenuo izvršavati ostale naredbe, a da traženje Z-pulsa nije završilo, što bi moglo zbuniti driver motora. Kada je driver pronašao Z-puls, motori odlaze u početnu poziciju i nakon toga u stanje mirovanja. *initRobo.py* module zatvara prozor kojeg prikazuje Slika 128 i pokreće se *workMode.py*.

4.3.2. Kalibracija robota

Ako je potrebna kalibracija motora i enkodera, u *starMode.py*, potrebno je odabrati „Kalibracija robota“ koja otvara prozor kojeg prikazuje Slika 129. Prilikom potpune kalibracije driver radi kalibraciju motora i kalibraciju enkodera. Prilikom kalibracije enkodera, motor mora napraviti jedan puni okret, a kako je ranije spomenuto to je onemogućeno zbog konstrukcijskih ograničenja robota. Stoga je potrebno ukloniti članke robota kako prikazuje Slika 129. Tek kada

se uklone članci, potrebno je potvrditi sa „Članci robota uklonjeni“ što omogućuje pokretanje kalibracije. Kalibracija se zatim mora pokrenuti sa „Pokreni kalibraciju“.



Slika 129. GUI, kalibracija robota

Program tada odlazi iz *startMode.py* u *calibrateRobot.py* modul gdje funkcije pozivaju različite metode *robot* objekta. Prvo se pozove *sendStartParameters()* metoda kao i kod inicijalizacije robota. Nakon toga program pokrene kalibraciju motora pozivajući *motorCalibration()* metodu.

```
def motorCalibration(self):
    self.objectOdrv.axis0.requested_state = AXIS_STATE_MOTOR_CALIBRATION
    while self.objectOdrv.axis0.current_state != AXIS_STATE_IDLE:
        time.sleep(0.1)
    self.objectOdrv.axis1.requested_state = AXIS_STATE_MOTOR_CALIBRATION
    while self.objectOdrv.axis1.current_state != AXIS_STATE_IDLE:
        time.sleep(0.1)
```

Nakon što je završena kalibracija motora, unutar te iste metode se govori driveru da zapamti kalibraciju motora slanjem naredbi niže.

```
    self.objectOdrv.axis0.motor.config.pre_calibrated = True
    self.objectOdrv.axis1.motor.config.pre_calibrated = True
```

Zatim je potrebno napraviti kalibraciju enkodera. To se obavlja pozivanjem *encoderCalibration()* metode. Metoda prvo zadaje driveru da se koriste enkoderi sa Z-pulsom,

zatim pokreće traženje Z-pulsa i to 2 puta za redom zbog šuma u enkoderima koji se ponekad zna javljati. Nakon što je pronađen Z-puls, pokreće se kalibracija enkodera i nakraju program zadaje driveru da zapamti kalibraciju enkodera.

```
def encoderCalibration(self):
    self.objectOdrv.axis0.encoder.config.use_index = True
    self.objectOdrv.axis1.encoder.config.use_index = True
    #pronađi z-puls
    i = 0
    while i < 2:
        self.objectOdrv.axis0.requested_state = AXIS_STATE_ENCODER_INDEX_SEARCH
        while self.objectOdrv.axis0.current_state != AXIS_STATE_IDLE:
            time.sleep(0.1)
        self.objectOdrv.axis1.requested_state = AXIS_STATE_ENCODER_INDEX_SEARCH
        while self.objectOdrv.axis1.current_state != AXIS_STATE_IDLE:
            time.sleep(0.1)
        i += 1
    #kalibracija enkodera
    self.objectOdrv.axis0.requested_state = AXIS_STATE_ENCODER_OFFSET_CALIBRATION
    while self.objectOdrv.axis0.current_state != AXIS_STATE_IDLE:
        time.sleep(0.1)
    self.objectOdrv.axis1.requested_state = AXIS_STATE_ENCODER_OFFSET_CALIBRATION
    while self.objectOdrv.axis1.current_state != AXIS_STATE_IDLE:
        time.sleep(0.1)
    self.objectOdrv.axis0.encoder.config.pre_calibrated = True
    self.objectOdrv.axis1.encoder.config.pre_calibrated = True
```

Nakraju je potrebno spremiti na driver zadane parametre i kalibraciju motora i enkodera, a to se obavlja pozivanjem *finishCalibration()* metode.

```
def finishCalibration(self):
    self.objectOdrv.axis0.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
    self.objectOdrv.axis1.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
    self.objectOdrv.axis0.controller.move_to_pos(0)
    self.objectOdrv.axis1.controller.move_to_pos(0)
    self.objectOdrv.save_configuration()
    self.objectOdrv.axis0.requested_state = AXIS_STATE_IDLE
    self.objectOdrv.axis1.requested_state = AXIS_STATE_IDLE
```

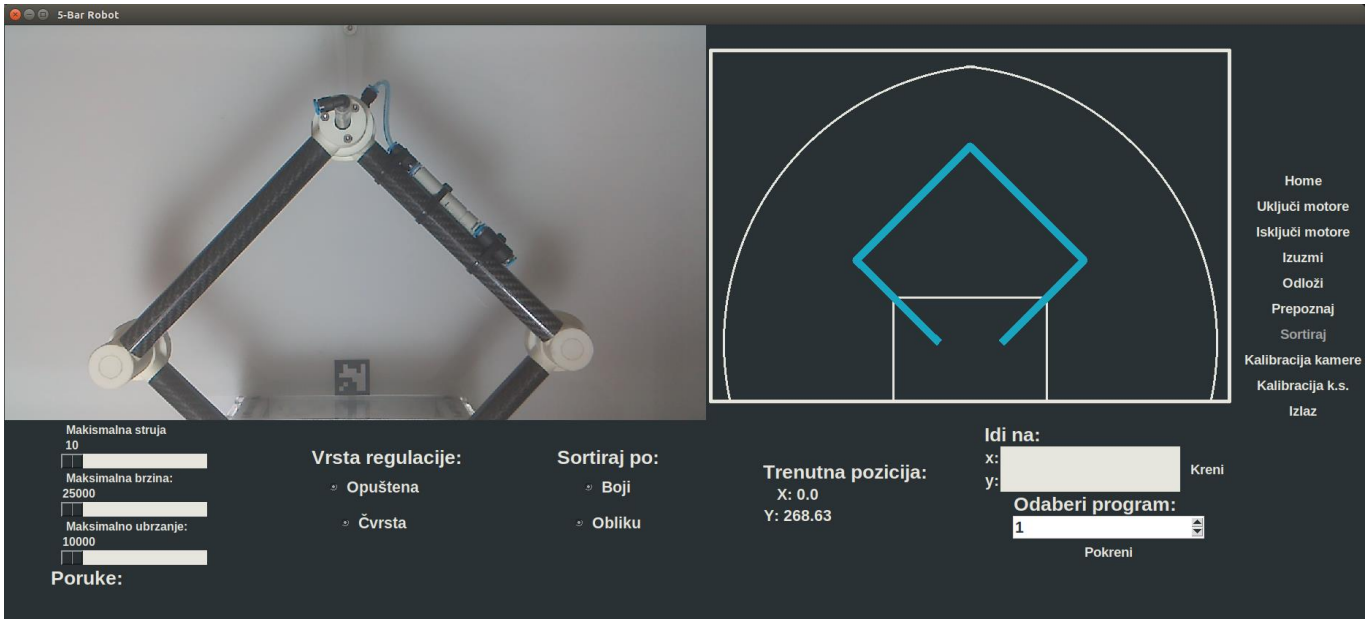
Prilikom inicijalizacije i kalibracije, svaki korak se ispisuje u grafičko sučelje, kako bi korisnik znao šta se događa.

4.3.3. Preskakivanje inicijalizacije robota

Ako je došlo do gašenja programa robota, a da pritom driver nije izgubio napajanje, moguće je preskočiti inicijalizaciju robota odabirom „Preskoči inicijalizaciju“ dok se korisnik nalazi u *startMode.py* modulu. To je moguće jer sam driver nije izgubio poziciju enkodera, bez obzira što je program za upravljanje robotom ugašen.

4.4. Upravljanje robotom kroz grafičko sučelje

Ulaskom programa u workMode.py modul, pokreće se grafičko sučelje koje Slika 130 prikazuje. Program tada inicijalizira *Camera* klasu i *Pneumatic* klasu koje koriste funkcije unutar *workMode.py* modula.



Slika 130. GUI, upravljanje robotom

Grafičko sučelje podijeljeno je na više funkcionalnih skupina. Prvo će se proći kroz funkcije za namještanje parametara programa i robota, zatim kroz načine upravljanja robotom i nakraju će se objasniti upravljački gumbi koji se nalaze desno na grafičkom sučelju robota.

Kroz grafičko sučelje moguće je namjestiti parametre robota poput maksimalne struje koja prolazi kroz motor koja je u Amperima, maksimalne brzine motora u br. impulsa/s i ubrzanja motora u br. impulsa/s^2. Za to je korištena *ScaleTemplate* klasa unutar *tkinterWidgets.py* modula. Prilikom svakog namještanja parametara poziva se *sliderToVar()* funkcija koja, ovisno da li se namješta struja, brzina ili ubrzanje, poziva različite metode od *robot* objekta i šalje zadane parametre. Za struju se poziva *maxMotorCurrent()*, za brzinu *maxVelocity()* i za ubrzanje *maxAccDel()*. Implementacija nabrojanih metoda se može vidjeti u prilogu pod [I].

```
def sliderToVar(sliderVar, objectRobot, sliderType):
    if(sliderType == "CURRENT"):
        objectRobot.maxMotorCurrent(sliderVar.get())
    if(sliderType == "VELOCITY"):
        objectRobot.maxVelocity(sliderVar.get())
    if(sliderType == "ACCLERATION"):
        objectRobot.maxAccDel(sliderVar.get())
```

Motori zajedno s enkoderima čine servomotore koji rade u zatvorenoj regulacijskoj petlji. Kako bi driver mogao uspješno regulirati motor, potrebno je dobro podesiti parametre regulatora. Driver radi u kaskadnom načinu po regulaciji pozicije koja se odabire naredbom niže,

```
self.objectOdrv.axis0.controller.config.control_mode = CTRL_MODE_POSITION_CONTROL
```

Osim po regulaciji pozicije, driver može raditi i po regulaciji brzine vrtnje i po regulaciji struje. Mogu se namještati parametri prikazani niže za motor0, za motor1 naredbe su identične.

```
self.objectOdrv.axis0.controller.config.pos_gain = self.AXIS_POSE_GAIN_T
self.objectOdrv.axis0.controller.config.vel_gain = self.AXIS_VEL_GAIN_T
self.objectOdrv.axis0.controller.config.vel_integrator_gain = self.AXIS_INTEGRATOR_GAIN_T
```

Za dobivanje najboljih mogućih performansi robota, potrebno je dobro namjestiti regulator, a za to je potrebno poznavati model motora i robota. Kako je to sam za sebe opsežan zadatak, namještanje regulatora je napravljeno prateći uputstva sa stranice *https://docs.odriverobotics.com/control*[25]. To je iterativni postupak namještanja koji za prva testiranja robota može dati prihvatljive rezultate.

U Grafičkom sučelju mogu se odabrati dvije vrste parametara, „Opušteni“ i „Čvrsti“. Kod opuštenog načina, robotu je dopuštena veća prijelazna karakteristika, dulje vrijeme smirivanja itd. Tada robot ima slabije performanse nego u „Čvrstom“ načinu rada. Odabir različitih parametra napravljen je korištenjem *RadiobuttonTemplate* klase unutar *tkinterWidgets.py* modula. Prilikom odabira nekog od načina regulacije uvijek se pozove funkcija *chooseControle()* koja šalje zadane parametre driveru pomoću *regulatorParameters()* metode od *robot* objekta. Implementacija *regulatorParameters()* metode može se vidjeti u prilogu pod [I]. *chooseControle()* prikazana je niže.

```
def chooseControle(choosenControl, objectRobot):
    if(choosenControl.get() == "Loose"):
        AXIS_POSE_GAIN = objectRobot.AXIS_POSE_GAIN_L
        AXIS_VEL_GAIN = objectRobot.AXIS_VEL_GAIN_L
        AXIS_INTEGRATOR_GAIN = objectRobot.AXIS_INTEGRATOR_GAIN_L
        objectRobot.regulatorParameters(AXIS_POSE_GAIN, AXIS_VEL_GAIN, AXIS_INTEGRATOR_GAIN)
    if(choosenControl.get() == "Tight"):
        AXIS_POSE_GAIN = objectRobot.AXIS_POSE_GAIN_T
        AXIS_VEL_GAIN = objectRobot.AXIS_VEL_GAIN_T
        AXIS_INTEGRATOR_GAIN = objectRobot.AXIS_INTEGRATOR_GAIN_T
        objectRobot.regulatorParameters(AXIS_POSE_GAIN, AXIS_VEL_GAIN, AXIS_INTEGRATOR_GAIN)
```

Korištenjem iste *RadiobuttonTemplate* klase može se napraviti objekt kojim je omogućen odabir vrste sortiranja. Zadatkom je zadano sortiranje po obliku ili po boji. Prilikom odabira

vrste sortiranja poziva se *typeSelection()* funkcija koja mijenja atribut od *camera* objekta, tj. *sortType*.

```
def typeSelection(passType, objectCamera):
    objectCamera.sortType = passType.get()
```

Grafičko sučelje također prikazuje trenutnu poziciju vrha robota. To je napravljeno pomoću *LabelTemplate* klase koja se nalazi unutar *tkinterWidgets.py* modula. Koriste se dvije instance klase, za *x* i *y* vrijednost vrha robota. Te instance se prosljeđuju u različite funkcije *workMode.py* modula. Pozivanjem tih klasa korištenjem metoda prikazanih niže, očitava se trenutna vrijednost vrha robota.

```
passCurrentXValue.config(text = "X: " + str(round(objectRobot.x_TCP, 2)))
passCurrentYValue.config(text = "Y: " + str(round(objectRobot.y_TCP, 2)))
```

Grafičko sučelje može slati korisniku različite upute kao npr. kod kalibracije kamere. To se radi korištenjem *MessageTemplate* klase koja se nalazi unutar *tkinterWidgets.py* modula. Napravi se instanca klase koja se kao parametar prosljeđuje u različite funkcije *workMode.py* modula. Pozivanjem te instance ispisuju se zadane poruke, kao npr. pomoću naredbi niže.

```
txtPass.set("Kalibracija kamere pokrenuta. Uslikajte minimalno 25 slika.
            Pritisnite 'a' za slikanje, 'f' za kraj i  'q' kako bi odustali."
            + "\nslika_" + str(objectCamera.numOfSnaps) + ".JPG")
```

U gornjem lijevom dijelu grafičkog sučelja prikazuje se uživo stanje dijela radnog područja pomoću *Raspberry Pi Camera Module v2* kamere. To je omogućeno korištenjem LabelTemplate klase koja se nalazi unutar *tkinterWidgets.py* modula i *show_frame()* metode od *camera* objekta. Navedena metoda uzima sliku s kamere svakih 10 milisekundi i prikazuje ju na grafičkom sučelju.

```
def show_frame(self, passCameraLabel):
    _, frame = self.cap.read()
    cv2image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)
    img = Image.fromarray(cv2image)
    imgtk = ImageTk.PhotoImage(image=img)
    passCameraLabel.imgtk = imgtk
    passCameraLabel.configure(image=imgtk)
    passCameraLabel.after(5, lambda : self.show_frame(passCameraLabel))
```

Navedeni prikaz radnog područja koristi se za kalibraciju kamere, koordinatnog sustava i lokalizaciju objekata po boji ili obliku.

Kako bi se robot mogao kretati, koriste se *moveAxis()* metode od *robot* objekta. Metoda radi tako da za zadani *x* i *y* prvo provjeri jesu li motori robota uključeni pomoću

driverInClosedLoopState(). Ako jesu, metoda zatim provjerava nalazi li se tražena pozicija unutar radnog područja robota pomoću *isInWorkSpace()* metode. Ako se pozicija nalazi unutar radnog područja robota, računa se inverzna kinematika u radijanima za zadanu poziciju *x* i *y* korištenjem *inverseKinematics()* metode. Dobivene radijane theta0 i theta1 potrebno je preračunati u broj impulsa enkodera pomoću *thetaToCnt()* metode. I nakraju se broj dobivenih impulsa šalje driveru koji pokreće motore pomoću *moveTraj()* metode. *moveAxis()* metoda prikazana je niže.

```
def moveAxis(self, passX, passY):
    if(not(self.driverInClosedLoopState())):
        return
    if(not(self.isInWorkSpace(passX, passY))):
        return
    self.x_TCP, self.y_TCP = passX, passY
    self.theta0, self.theta1 = self.inverseKinematics(self.x_TCP, self.y_TCP)
    self.absCnt0, self.absCnt1 = self.thetaToCnt(self.theta0, self.theta1)
    self.moveTraj(self.absCnt0, self.absCnt1)
```

Metoda provjere jesu li motori uključeni, *driverInClosedLoopState()*:

```
def driverInClosedLoopState(self):
    if(self.objectOdrv.axis0.current_state == AXIS_STATE_CLOSED_LOOP_CONTROL
            and self.objectOdrv.axis1.current_state == AXIS_STATE_CLOSED_LOOP_CONTROL):
        return True
    else:
        return False
```

Metoda provjere nalazi li se zadana točka unutar radnog područja robota, *isInWorkSpace()*:

```
def isInWorkSpace(self, passX, passY):
    if(not(((passX - 42.5)**2 + (passY - 0)**2) < 380**2)):
        return False
    if(not(((passX + 42.5)**2 + (passY - 0)**2) < 380**2)):
        return False
    if((-105 < passX < 105 ) and (0 < passY < 63)):
        return False
    if(not((-355 < passX < 355 ) and (0 < passY < 382))):
        return False
    return True
```

Metoda za računanje inverzne kinematike korištenjem formula iz poglavlja 2.3.2, *inverseKinematics()*:

```
def inverseKinematics(self, passX, passY):
    if(not(self.isInWorkSpace(passX, passY))):
        return
```

```
        beta0 = arctan2( passY, (self.l0 - passX) )
        beta1 = arctan2( passY, (self.l0 + passX) )
        alpha0_calc = (self.l1**2 + ( (self.l0 - passX)**2 + passY**2 ) - self.l2**2) /
                        (2*self.l1*sqrt( (self.l0 - passX)**2 + passY**2 ))
        alpha1_calc = (self.l1**2 + ( (self.l0 + passX)**2 + passY**2 ) - self.l2**2) /
                        (2*self.l1*sqrt( (self.l0 + passX)**2 + passY**2 ))
        alpha1 = arccos(alpha1_calc)
        alpha0 = arccos(alpha0_calc)
        if(passX >= 0 and passY < 0):
            returnTheta0 = (pi - beta0 - alpha0) - 2* pi
            returnTheta1 = beta1 + alpha1
        elif(passX < 0 and passY < 0):
            returnTheta0 = pi - beta0 - alpha0
            returnTheta1 = 2*pi + (beta1 + alpha1)
        else:
            returnTheta0 = pi - beta0 - alpha0
            returnTheta1 = beta1 + alpha1
```

Metoda preračunavanje kutova iz radijana u broj impulsa enkodera, *thetaToCnt()*:

```
    def thetaToCnt(self, passTheta0, passTheta1):
        tempCnt0 = (passTheta0 - pi/4) * self.AXIS_CPR/(2*pi)
        tempCnt1 = (passTheta1 - (3*pi)/4) * self.AXIS_CPR/(2*pi)
        return tempCnt0, tempCnt1
```

I nakraju metoda koja šalje naredbe driveru za pomicanje motora, *moveTraj()*:

```
    def moveTraj(self, newPos0, newPos1):
        if(self.objectOdrv.axis0.current_state == AXIS_STATE_CLOSED_LOOP_CONTROL and
                self.objectOdrv.axis0.current_state == AXIS_STATE_CLOSED_LOOP_CONTROL):
            self.objectOdrv.axis0.controller.move_to_pos(newPos0)
            self.objectOdrv.axis1.controller.move_to_pos(newPos1)
            return True
        else:
            return False
```

moveAxis() metoda koristi se u različitim funkcijama unutar *workMode.py* modula.

Desni gornji dio grafičkog sučelja prikazuje trenutnu poziciju robota unutar njegovog radnog područja. To je omogućeno korištenjem *CanvasRobot* klase koja se nalazi unutar *tkinterWidgets.py* modula. Kako bi se robot mogao prikazati u različitim pozicijama koriste se *drawRobotWorksapce()* i *drawRobotPostion()* metode. Prva metoda se poziva samo jednom kod inicijalizacije *CanvasRobot* klase i prikazuje radno područje robota koje se ne mijenja. Dok se *drawRobotPosition()* poziva u različitim dijelovima *workMode.py* modula te prikazuje trenutnu poziciju robota, kao što se i koristi ranije spomenuta *LabelTemplate* klasa.

```
def drawRobotPosition(self, objectRoot, drawingX, drawingY, drawingTheta0, drawingTheta1):
    self.delete('robotPosition')
    self.drawingCoordinates[4], self.drawingCoordinates[5] =
                            (drawingX + self.xOffset), (self.yOffset - drawingY) #x3, y3
    self.drawingCoordinates[2], self.drawingCoordinates[3] =
                            (-self.l0 + self.l1 * cos(drawingTheta1) + self.xOffset),
                            (self.yOffset - self.l1 * sin(drawingTheta1)) #x2, y2
    self.drawingCoordinates[6], self.drawingCoordinates[7] =
                            (self.l0 + self.l1 * cos(drawingTheta0) + self.xOffset),
                            (self.yOffset - self.l1 * sin(drawingTheta0)) #x4, y4
    self.create_line(self.drawingCoordinates, width = 10, fill = robotColor, tags='robotPosition')
    objectRoot.update()
```

Osim prikazivanja trenutne pozicije robota *CanvasRobot* može se koristiti i za upravljanje robotom. To je omogućeno korištenjem *Tkinter bind eventa*. Naredbom niže napravi se *bind* lijeve tipke računalnog miša na *CanvasRobota*.

```
robotPosition.bind("<B1-Motion>", lambda event :
    followMouseMoveRobot(event, root, robotPosition, objectRobot, currentXValue, currentYValue))
```

Sada svaki put kada korisnik sučelja pritisne lijevu tipku računalnog miša na vrh robota i držeći stisnutu tipku vuče pokazivač računalnog miša po radnom području *Canvasa robota*, poziva se *followMouseMoveRobot()* funkcija koja je prikazana niže.

```
def followMouseMoveRobot(event, objectRoot, passRobotPosition, objectRobot,
                         passCurrentXValue, passCurrentYValue):
    if(not(objectRobot.driverInClosedLoopState())):
        return
    tempX, tempY = passRobotPosition.followMouseMoveRobotFunc(event)
    if(not(objectRobot.isInWorkSpace(tempX, tempY))):
        return
    objectRobot.moveAxis(tempX, tempY)
    passRobotPosition.drawRobotPosition(objectRoot, tempX, tempY,
                                        objectRobot.theta0, objectRobot.theta1)
    passCurrentXValue.config(text = "X: " + str(round(objectRobot.x_TCP, 2)))
    passCurrentYValue.config(text = "Y: " + str(round(objectRobot.y_TCP, 2)))
```

Ta funkcija prvo provjeri jesu li motori uključeni, zatim poziva *followMouseMoveRobotFunc()* metodu *CanvasRobot* klase, koja preračunava poziciju računalnog miša u željenu točku vrha robota.

```
def followMouseMoveRobotFunc(self, event):
    currentX, currentY = (event.x - self.xOffset), (self.yOffset - event.y)
    return currentX, currentY
```

Zatim se provjerava nalazi li se željena točka unutar radnog područja robota pomoću metoda opisanih ranije. Ako je željena točka dosegljiva, program šalje driveru naredbu za pomicanjem robota pomoću *moveAxis()* metode koja je objašnjena ranije. Nakon što se robot pomakne na željenu poziciju, prikazuje se njegova pozicija.

Robotom je isto tako moguće upravljati da mu se zada samo željena točka. To je napravljeno kombinacijom *EntryTemplate* i *ButtonTemplate* klase koje se nalaze unutar tkinterWidgets.py modula. Prvo je potrebno unijeti željenu *x* i *y* vrijednost i zatim odabrati „Kreni“. Kada se odabere „Kreni“, poziva se *goToModeFunc()* koja uzima ono što se trenutno nalazi zapisano za *x* i *y* koordinate. Kako bi se onemogućilo da korisnik pošalje robot izvan radnog područja, što bi moglo dovesti do kolizije ili ulaska u singularnost, napravljena je provjera valjanosti unesenih podataka. Ako uneseni podaci nisu prihvatljivi, program izlazi iz funkcije i briše unesene podatke. *goToModeFunc()* prikazana je niže.

```
def goToModeFunc(objectRoot, objectRobot, xEntry, yEntry,
                    drawRobotPositionPass, passCurrentXValue, passCurrentYValue):
    if(not(objectRobot.driverInClosedLoopState())):
        return
    if(xEntry.get() == "" or yEntry.get() == ""):
        xEntry.delete(0, 'end')
        yEntry.delete(0, 'end')
        return
    if(not(objectRobot.isInWorkSpace(float(xEntry.get()), float(yEntry.get())))):
        xEntry.delete(0, 'end')
        yEntry.delete(0, 'end')
        return
    objectRobot.moveAxis(float(xEntry.get()), float(yEntry.get()))
    drawRobotPositionPass.drawRobotPosition(objectRoot, float(xEntry.get()), float(yEntry.get()),
                                            objectRobot.theta0, objectRobot.theta1)
    passCurrentXValue.config(text = "X: " + str(round(objectRobot.x_TCP, 2)))
    passCurrentYValue.config(text = "Y: " + str(round(objectRobot.y_TCP, 2)))
    xEntry.delete(0, 'end')
    yEntry.delete(0, 'end')
```

Ako su uneseni podaci valjani opet se poziva *moveAxis()* metoda te se prikazuje trenutna pozicija robota. Nakon izvršavanja svih naredbi, funkcija na kraju briše ono što se nalazilo zapisano za *x* i *y* vrijednosti.

Osim upravljanja računalnim mišem i zadavanjem željene točke, robotom se može upravljati zadavanjem pred programiranih točaka. To je napravljeno kombinacijom *SpinboxTemplate* i *ButtonTemplate* klasa koje se nalaze unutar *tkinterWidgets.py* modula. Osim toga korišten je i *preProgrammed.py* module u kojem se nalazi *preProgrammedFunc()* funkcija i zadani

programi. Pomoću *SpinboxTemplate* korisnik odabire željeni program iz *preProgrammed.py* modula te pritiskom na tipku „Pokreni“ poziva se *preProgrammedFunc()* funkcija koja izvršava željeni program.

```
def preProgrammedFunc(objectRoot, objectRobot, preProgramedTypesPass, buttonPass,
                      drawRobotPositionPass, passCurrentXValue, passCurrentYValue):
    chosenPPType = preProgramedTypesPass.get()
    buttonPass.configure(state='disable')
    choosenProgram = []
    if(chosenPPType == "1"):
        choosenProgram = program1
    if(chosenPPType == "2"):
        choosenProgram = program2
    for i in choosenProgram:
        objectRobot.moveAxis(i[0], i[1])
        drawRobotPositionPass.drawRobotPosition(objectRoot, objectRobot, i[0], i[1])
        passCurrentXValue.config(text =  "X: " + str(round(objectRobot.x_TCP, 2)))
        passCurrentYValue.config(text =  "Y: " + str(round(objectRobot.y_TCP, 2)))
        while(True):
            if(objectRobot.atPosition()):
                break
    buttonPass.configure(state='normal')
```

Trenutno postoje 2 zadana pred programa koje robot prati. Koriste se ranije objašnjene metode za pokretanje motora i prikazivanje trenutne pozicije robota. Dodatno je implementirana *atPosition()* metoda od *robot* objekta koja provjerava je li robot došao u željenu poziciju. Kada ne bi bilo provjere, program bi slao više naredbi nego što driver može trenutno izvršiti, što bi dovelo do negativnih posljedica, tj. da driver neke naredbe zanemari.

```
def atPosition(self):
    encoder0, encoder1 = self.encoderPosition()
    if(((self.absCnt0 - self.deviation) < encoder0 < (self.absCnt0 + self.deviation)
             and (self.absCnt1 - self.deviation) < encoder1 < (self.absCnt1 + self.deviation))):
        return True
    else:
        return False
```

Zadnje na grafičkom sučelju su upravljački gumbi koji su dobiveni korištenjem *ButtonTemplate* klase unutar *tkinterWidgets.py* modula. Ti gumbi su redom:

- Home
- Uključi motore
- Isključi motore
- Izuzmi

- Odloži
- Prepoznaj
- Sortiraj
- Kalibracija kamere
- Kalibracija k.s.
- Izlaz

Odabirom „Home“ poziva se *homeFunc()* funkcija i robot odlazi u svoju početnu poziciju koju prikazuje Slika 130.

```
def homeFunc(objectRoot, objectRobot, passCurrentXValue, passCurrentYValue, drawRobotPositionPass):
    if(not(objectRobot.homeFuncRobot())):
        return
    drawRobotPositionPass.drawRobotPosition(objectRoot, objectRobot.x_TCP, objectRobot.y_TCP,
                        objectRobot.theta0, objectRobot.theta1)
    passCurrentXValue.config(text = "X: " + str(round(objectRobot.x_TCP, 2)))
    passCurrentYValue.config(text = "Y: " + str(round(objectRobot.y_TCP, 2)))
```

homeFunc() funkcija poziva *homeFuncRobot()* metodu od *robot* objekta koja prvo provjerava jesu li motori uključeni, ako jesu, postavlja atribute objekta robota *absCnt0* i *absCnt1* na 0. Ti atributi predstavljaju trenutni broj impulsa enkodera. Zatim pomiče robot na taj zadani položaj impulsa enkodera pomoću *moveTraj()* metode od *robot* objekta.

```
def homeFuncRobot(self):
    if(not(self.driverInClosedLoopState())):
        return False
    self.absCnt0, self.absCnt1 = 0, 0
    self.moveTraj(self.absCnt0, self.absCnt1)
    self.theta0, self.theta1 = self.cntToTheta(self.absCnt0, self.absCnt1)
    self.x_TCP, self.y_TCP = self.forwardKinematics(self.theta0, self.theta1)
    while True:
        if(self.atPosition()):
            return True
```

Nakon što se robot pomaknuo na zadani položaj, program pomoću *cntToTheta()* metode preračunava zadani broj pulseva u kutove u radijanima. Iz tih kutova pomoću direktne kinematike se računa poziciju vrha robota. Formule za direktnu kinematiku prikazane su u poglavlju 2.3.1. I ovdje je korištena *atPosition()* metoda kako bi spriječila program od daljnjeg izvršavanja dok robot ne dođe u *home* poziciju.

Metoda *cntToTheta()*.

```
def cntToTheta(self, passCnt0, passCnt1):
    tempTheta0 = passCnt0 * (2*pi)/self.AXIS_CPR + pi/4
    tempTheta1 = passCnt1 * (2*pi)/self.AXIS_CPR + (3*pi)/4
    return tempTheta0, tempTheta1
```

Metoda *forwardKinematics().*

```
def forwardKinematics(self, passT0, passT1):
    x0 = self.l0 + self.l1 * cos(passT0)
    y0 = self.l1 * sin(passT0)
    x1 = -self.l0 + self.l1 * cos(passT1)
    y1 = self.l1 * sin(passT1)
    d = sqrt((x1-x0)*(x1-x0) + (y1-y0)*(y1-y0))
    if d > 2 * self.l2 :
        return None
    if d < 0:
        return None
    if d == 0:
        return None
    else:
        a=(d**2)/(2*d)
        h=sqrt(self.l2**2-a**2)
        x2=x0+a*(x1-x0)/d
        y2=y0+a*(y1-y0)/d
        x3=x2+h*(y1-y0)/d
        y3=y2-h*(x1-x0)/d
        if(not(self.isInWorkSpace(x3, y3))):
            return None
        else:
            return(x3,y3)
```

U više metoda prikazana je provjera ima li robot uključene ili isključene motore. U ovom slučaju uključeni motori znače da rade u zatvorenoj regulacijskoj petlji te pokušavaju zadržati zadanu poziciju, tj. rade u servo načinu rada. Dok ugašeni motori znače da se mogu slobodno rotirati i ne pokušavaju zadržati svoju poziciju, tj. rade u *idle* načinu rada. Kako bi se mogle koristiti funkcionalnosti robota, potrebno je uključiti motore, tj. dovesti ih u servo način rada. Odabirom „Uključi motore“ poziva se *closedLoopFunc()* koja pomoću *closedLoopFuncRobot()* metode *robot* objekta pali motore robota. Osim toga *closedLoopFunc()* prikazuje i trenutnu poziciju robota.

```
def closedLoopFunc(objectRoot, objectRobot,
                   passCurrentXValue, passCurrentYValue, drawRobotPositionPass):
    if(not(objectRobot.closedLoopFuncRobot())):
        return
```

```
        drawRobotPositionPass.drawRobotPosition(objectRoot, objectRobot.x_TCP, objectRobot.y_TCP,
                                                objectRobot.theta0, objectRobot.theta1)
        passCurrentXValue.config(text = "X: " + str(round(objectRobot.x_TCP, 2)))
        passCurrentYValue.config(text = "Y: " + str(round(objectRobot.y_TCP, 2)))
```

Metoda *closedLoopFuncRobot()* poziva pod metodu *goToClosedLoop()* koja šalje naredbe driveru za paljenje motora te očitava trenutnu poziciju enkodera pomoću *enkoder.pos_estimate* i sprema ih u atribute *absCnt0* i *absCnt1*, zatim se pomoću ranije pokazanih metoda *cntToTheta()* i *forwardKinemtacis()* mijenjaju ostali atributi robota.

```
    def closedLoopFuncRobot(self):
        if(self.driverInClosedLoopState()):
            return False
        self.goToClosedLoop()
        self.absCnt0, self.absCnt1 =
                self.objectOdrv.axis0.encoder.pos_estimate, self.objectOdrv.axis1.encoder.pos_estimate
        self.theta0, self.theta1 = self.cntToTheta(self.absCnt0, self.absCnt1)
        self.x_TCP, self.y_TCP = self.forwardKinematics(self.theta0, self.theta1)
        return True
```

Metoda *goToClosedLoop()*.

```
    def goToClosedLoop(self):
        self.objectOdrv.axis0.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
        self.objectOdrv.axis1.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
```

Ako se žele isključiti motori, odabire se „Isključi motore“ te se poziva funkcija *idleFunc()* koja poziva *goToIdle()* metodu koja šalje naredbe driveru da isključi motore.

```
    def goToIdle(self):
        self.objectOdrv.axis0.requested_state = AXIS_STATE_IDLE
        self.objectOdrv.axis1.requested_state = AXIS_STATE_IDLE
```

Odabirom „Izuzmi“ i „Odloži“ pozivaju se *pickFunc()* i *placeFunc()* funkcije koje pozivaju *pick()* i *place()* metodu *pnemautic* objekta. Te metode nalaze se unutar *Pneumatic* klase koja upravlja radom vakuumske hvataljke. Kako je ranije objašnjeno, *Jetson Nano* pali i gasi elektromagnetske releje koji pale i gase elektroventile. Paljenje i gašenje releja obavlja se preko pinova koji se nalaze na *Jetson Nano* mikroračunalu. Kako bi se moglo upravljati s tim pinovima potrebno je koristiti *Jetson.GPIO* modul. Zatim je potrebno definirati koji se pinovi koriste i kako će se koristiti, to je napravljeno unutar *__init__()* metode *Pneumatic* klase.

```
    def __init__(self):
        GPIO.setmode(GPIO.BOARD)
```

```
        GPIO.setup(self.cylinder_pin, GPIO.OUT)
        GPIO.setup(self.vacuum_pin, GPIO.OUT)
        GPIO.output(self.cylinder_pin, GPIO.LOW)
        GPIO.output(self.vacuum_pin, GPIO.LOW)
```

pick() metoda.

```
    def pick(self):
        if(self.inPickedState == 1):
            return
        GPIO.output(self.cylinder_pin, GPIO.HIGH)
        time.sleep(0.5)
        GPIO.output(self.vacuum_pin, GPIO.HIGH)
        time.sleep(1)
        GPIO.output(self.cylinder_pin, GPIO.LOW)
        time.sleep(1)
        self.inPickedState = 1
```

place() metoda.

```
    def place(self):
        if(self.inPickedState == 0):
            return
        GPIO.output(self.cylinder_pin, GPIO.HIGH)
        time.sleep(1)
        GPIO.output(self.vacuum_pin, GPIO.LOW)
        time.sleep(0.25)
        GPIO.output(self.cylinder_pin, GPIO.LOW)
        self.inPickedState = 0
```

„Prepoznaj“, „Sortiraj“, „Kalibracija kamere“ i „Kalibracija k.s.“ objašnjeni su u poglavlju 4.4.1.

I zadnje „Izlaz“ poziva funkciju *quitFunc()* koja gasi motore robota pomoću *goToIdle()* metode i zatvara grafičko sučelje robota.

```
    def quitFunc(objectRoot, objectRobot):
        objectRobot.goToIdle()
        objectRoot.destroy()
```

4.4.1. Vizijski sustavi

Osim ranije spomenutog upravljanja, robotom je moguće i upravljati preko vizijskih sustava. Koristeći *Raspberry Pi Camera Module v2* kameru potrebno je napravit lokalizaciju predmeta koji se nalaze unutar radnog područja robota te ih sortirati, ovisno kako korisnik odluči, po boji ili obliku. Kako bi kamera mogla uspješno obaviti navedeni zadatak, prvo je potrebno kameru

kalibrirati, zatim povezati koordinatni sustav kamere s koordinatnim sustavom robota. Nakraju, nakon uspješne kalibracije, potrebno je lokalizirati predmete, pronaći njihova središta u obliku *u* i *v* koordinata robota i transformirati te koordinate u koordinatni sustav robota kako bi robot mogao razvrstati predmete. Sve funkcije korištene za vizijske sustave su iz *OpenCV Python* modula i *Numpy Python* modula.

4.4.1.1. Kalibracija kamere

Kalibracija kamere napravljena je po uputstvima sa stranice *https://opencv-python-tutroals.readthedocs.io*[26], otkud su i iskorišteni neki dijelovi koda. Kako svaka kamera ima neku distorziju, potrebno je kod aplikacija za vizijski sustav napraviti njihovu kalibraciju. Dvije najčešće vrste distorzija su tangencijalna i radijalna distorzija. Tangencijalna distorzija događa se kada kamera nije savršeno ravna s ravninom čiju sliku projicira pa se neki dijelovi slike čine bliže nego što jesu. Uslijed radijalne distorzije, ravne crte u stvarnosti čine se kosima na slici i taj se efekt sve više povećava kako su pikseli udaljeniji od centra kamere. Radijalna distorzija se rješava kako pokazuju jednadžbe (28) i (29), dok se tangencijalne rješavaju kako pokazuju jednadžbe (30) i (31).[26]

$$x_{rad\_ispravljeno} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad \textbf{(28)}$$

$$y_{rad\_ispravljeno} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad \textbf{(29)}$$

$$x_{rad\_ispravljeno} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \quad \textbf{(30)}$$

$$y_{rad\_ispravljeno} = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \quad \textbf{(31)}$$

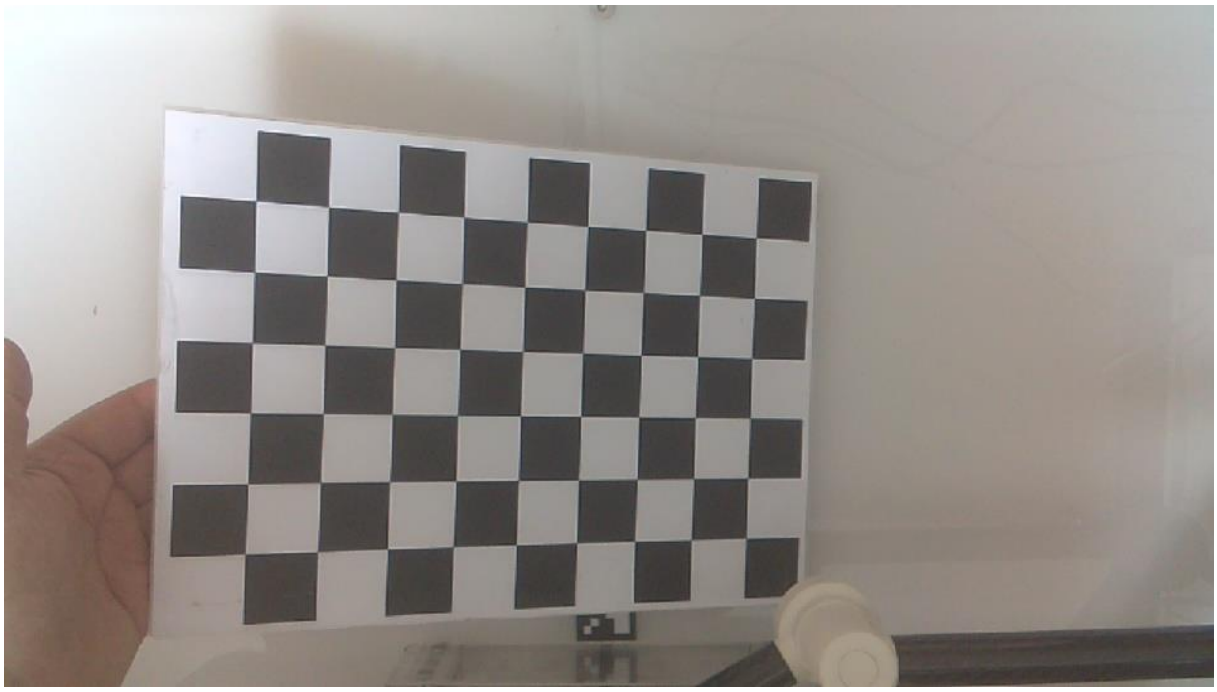
Potrebno je pronaći pet parametara koji se nazivaju distorzijski koeficijenti, prikazuje ih jednadžba (32).

$$\boldsymbol{dst} = [k_1\ k_2\ p_1\ p_2\ k_3] \quad \textbf{(32)}$$

Osim navedenih pet parametara, potrebno je pronaći i intrinzične i ekstrinzične parametre kamere. Ekstrinzični parametri su translacijski i rotacijski vektori koji translatiraju točku u stvarnom svijetu, u točku kamere. O tome se govori u 4.4.1.2. Intrinzični parametri kamere specifični su za svaku kameru. Ti parametri su fokalna duljina (f_x, f_y) u pikselima i optički centar (c_x, c_y) u pikselima. Parametri kamere stavljaju se u 3x3 matricu i takva se matrica naziva matrica kamere, a prikazuje ju jednadžba (33).

$$\boldsymbol{mtx} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(33)}$$

Kalibracija se izvodi pomoću kalibracijske mreže. Kalibracijska mreža može biti raznih oblika, no ovdje je korištena šahovska ploča. Šahovsku ploču potrebno je staviti unutar vidnog polja kamere i uslikati sliku, kako Slika 131 prikazuje. Potrebno je uslikati minimalno 25 slika kalibracijske mreže u različitim pozicijama za dobivanje što točnijih rezultata.

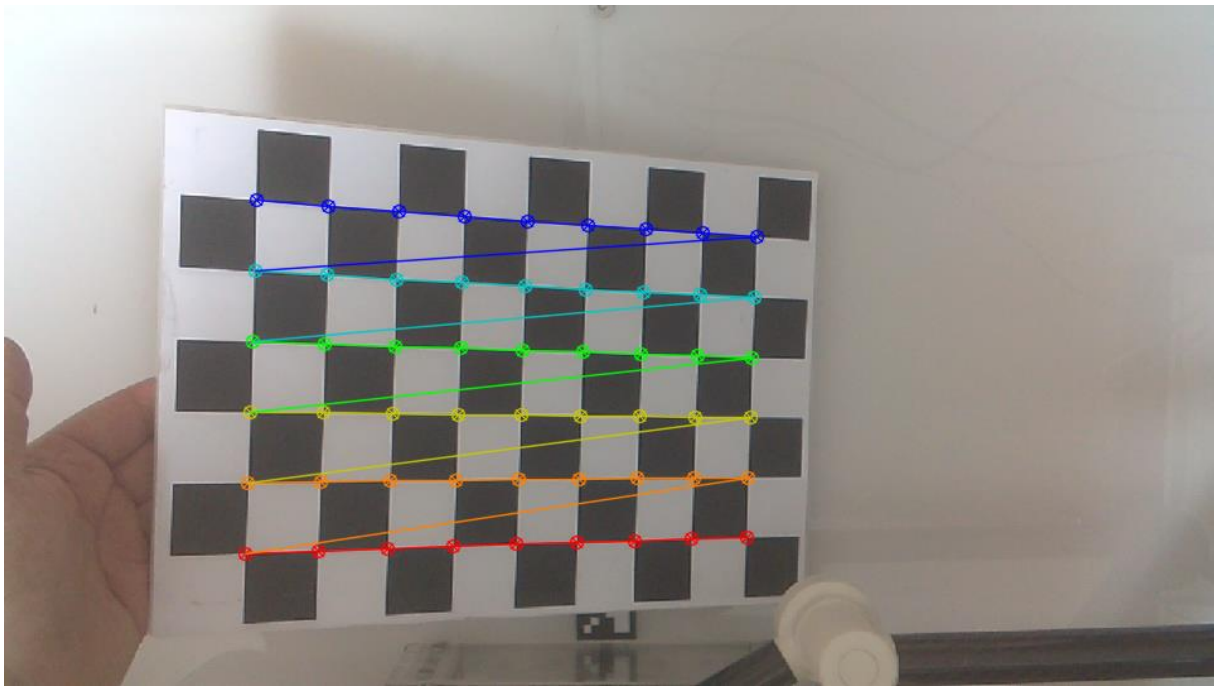


Slika 131. Kalibracija kamere, slikanje kalibracijske mreže

Kalibracija se pokreće odabirom „Kalibracija kamere“ iz grafičkog sučelja kojeg Slika 130 prikazuje. Program tada pokreće *calibrateCameraFunc()*. Prvo se poziva funkcija *moveRobotForCameraView()* koja šalje naredbu robotu da se pomakne izvan vidnog polja kamere. Zatim se radi *bind* tipki na tipkovnici pomoću kojih se kontrolira kalibracija kamere. Tako se pritiskom tipke 'a' poziva funkcija *calibrateCameraTakeImage()* koja uzima trenutno prikazanu sliku kamere i sprema ju u zadanu mapu. Pritiskom tipke 'f' kalibracija se nastavlja dalje, ali samo ako je uslikano minimalno 25 slika i nakraju ako korisnik hoće odustati od kalibracije pritišće tipku 'q'. Nakon što se uslika minimalno 25 slika i nastavi kalibracija, program pokreće funkciju *calibrateCameraSequence()* koja poziva *calibrateCamera()* metodu od *camera* objekta. *calibrateCamera()* metoda ulazi u mapu gdje se nalaze prethodno spremljene slike kalibracije, kao npr. Slika 131. Program uzima svaku sliku kalibracije i pokušava pronaći zadani broj točaka u *x* i *y* smjeru pomoću *cv2.findChessboardCorners()* funkcije. Ako je program uspio pronaći zadane točke, iscrtava ih na slici te tu sliku prikazuje korisniku sučelja, kako Slika 132 prikazuje. Korisniku se daje mogućnost prihvaćanja pronađenih točaka pritiskom tipke 'a', odbijanjem pronađenih točaka pritiskom tipke 'r' ili odustajanja od kalibracije pritiskom tipke 'q'. Ako korisnik prihvati pronađene točke, njihove *u*

i *v* koordinate spremaju se u *imgpoints* listu. Osim *imgpoints*, koristi se i *objpoints* lista koja sadrži lokacije točaka na kalibracijskoj mreži u *x* i *y* smjeru, u ovom slučaju korištena je (6x9) mreža. Ti podaci spremaju se u *objpoints* kao (0,0,0), (1,0,0) itd. gdje su (*x*, *y*, *z*) normalizirane lokacije točaka u prostoru. Normalizirane znači da se ne uzima stvarna veličina kvadrata šahovske ploče jer ona u ovom slučaju kod kalibracije nije potrebna.

Kada je program prošao kroz sve prethodno uslikane slike, poziva se *cv2.calibrateCamera()* funkcija koja računa matricu kamere i distorzijske koeficijente te ih ispisuje korisniku grafičkog sučelja.



Slika 132. Kalibracija kamere, pronalazak točaka

Bitno je napomenuti kako *Raspberry Pi Camera Module v2* kamera može raditi u više različitih rezolucija. Ovdje je odabrana radna rezolucija od 1280x720 piksela te je skalirana na veličinu od 960x540 piksela. Inicijalizacija kamere obavlja se naredbom prikazanom niže.

```
self.cap = cv2.VideoCapture(self.gstreamer_pipeline(), cv2.CAP_GSTREAMER)
```

Gdje je *gstreamer_pipeline()* metoda kojom se namješta rezolucija, broj *FPS-a* itd., a potrebno ju je koristiti kako bi *Jetson Nano* uspješno prikazivao sliku s kamere.

Nakon kalibracije kamere za spomenutu rezoluciju i skaliranje dobiveni su sljedeći rezultati:

$$\boldsymbol{dst} = [0{,}11304 \;\; 0{,}48364 \;\; -0{,}00565 \;\; -0{,}001672 \;\; -2{,}88852], \tag{34}$$

$$\boldsymbol{mtx} = \begin{bmatrix} 975{,}73 & 0 & 476{,}32 \\ 0 & 976{,}93 & 274{,}12 \\ 0 & 0 & 1 \end{bmatrix} \text{ pikseli.} \tag{35}$$

Implementacija *calibrateCamera()* metode prikazana je niže.

```
def calibrateCamera(self):
    xNumOfSquares, yNumOfSquares = 6, 9
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
    objp = np.zeros((xNumOfSquares * yNumOfSquares,3), np.float32)
    objp[:,:2] = np.mgrid[0:yNumOfSquares,0:xNumOfSquares].T.reshape(-1,2)
    objpoints = []
    imgpoints = []
    images = glob.glob(self.path + '*.JPG')
    for fname in images:
        img = cv2.imread(fname)
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        ret, corners = cv2.findChessboardCorners(gray, (9,6), None)
        if ret == True:
            corners2 = cv2.cornerSubPix(gray,corners, (11,11), (-1,-1), criteria)
            cv2.drawChessboardCorners(img, (9,6), corners2, ret)
            comapreValue = self.repeatingFunc(img)
            if(comapreValue == 'a'):
                objpoints.append(objp)
                imgpoints.append(corners)
            if(comapreValue == 'r'):
                continue
            if(comapreValue == 'q'):
                return False
    cv2.destroyAllWindows()
    _, self.mtx, self.dist, _, _ = cv2.calibrateCamera(objpoints, imgpoints,
                                                       gray.shape[::-1], None, None)
    print("mtx:\n", self.mtx)
    print("dist:\n", self.dist)
    return True
```

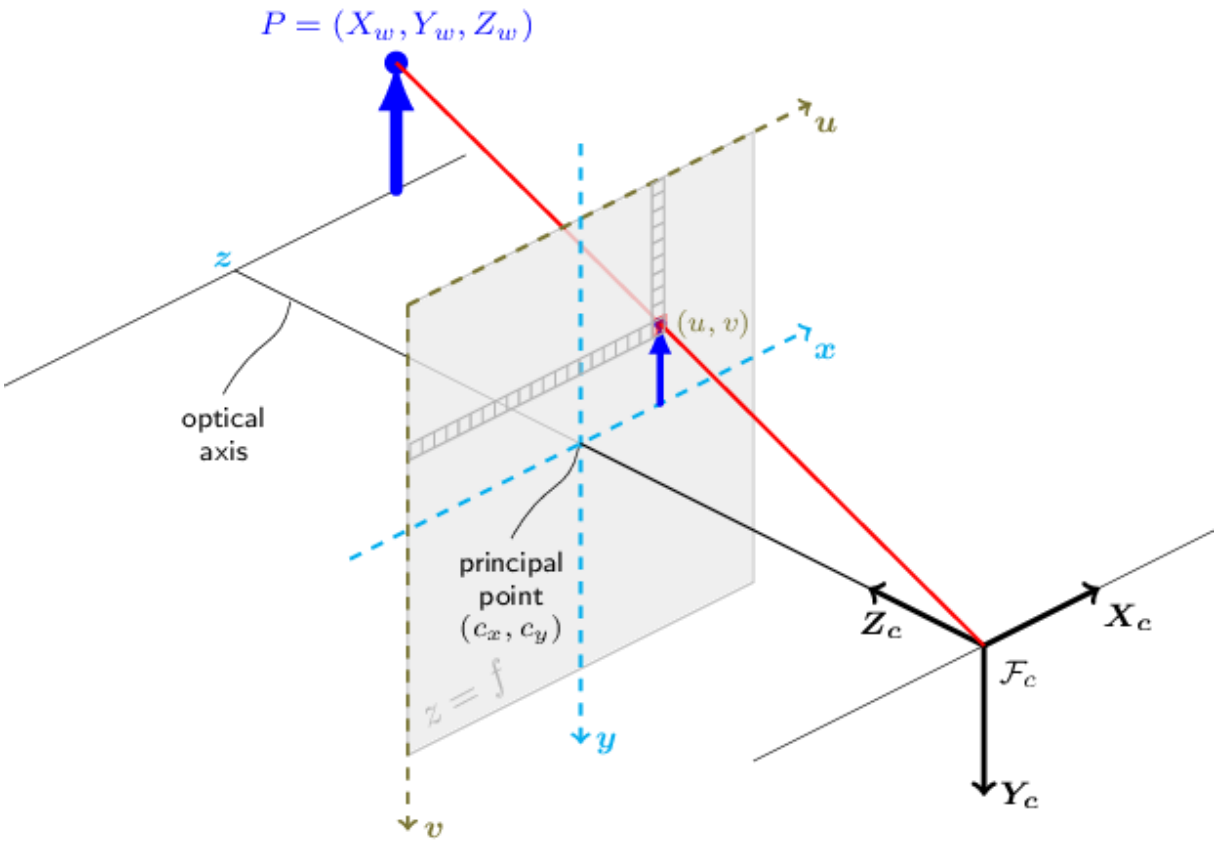
Implementacija ostalih spomenutih funkcija i metoda nalazi se u prilogu pod [I].Dobiveni parametri kalibracije koriste se kod povezivanja koordinatnog sustava kamere s koordinatnim sustavom robota što je pokazano u poglavlju 4.4.1.2. Osim toga dobiveni parametri koriste se za uklanjanje distorzija s kamere, što je implementirano u *unDistort()* metodi *camera* objekta. Ta metoda uzima sliku s kamere, uklanja distorziju te vraća dobivenu sliku.

```
def unDistort(self, frame):
    h,  w = frame.shape[:2]
    newcameramtx, roi=cv2.getOptimalNewCameraMatrix(self.mtx, self.dist, (w,h), 1, (w,h))
    mapx,mapy = cv2.initUndistortRectifyMap(self.mtx, self.dist, None, newcameramtx,(w,h),5)
    dst = cv2.remap(frame,mapx,mapy,cv2.INTER_LINEAR)
    x,y,w,h = roi
    dst = dst[y:y+h, x:x+w]
    return dst
```

4.4.1.2. Kalibracija koordinatnog sustava

Nakon kalibracije kamere, tj. pronalaska distorzijskih faktora i matrice kamere, potrebno je pronaći i ekstrinzične parametre kamere. Ekstrinzični parametri povezuju koordinatni sustav kamere s koordinatnim sustavom robota. Ti parametri potrebni su kako bi program nakon pronalaska središta lokaliziranih predmeta koji su u obliku (u, v) koordinata slike, mogao prebaciti te koordinate u koordinatni sustav kamere u obliku (x, y, z) koordinata. Jednadžba (36) povezuje koordinate piksela kamere s koordinatama u realnom svijetu.[27] Takav model kamere naziva se *pinhole model* i prikazuje ga Slika 133.

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{23} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix} \quad \textbf{(36)}$$



Slika 133. *Pinhole* model kamere[27]

Iz jednadžbe (36) može se uočiti kako su matrica [***rotM***|***tvec***] i faktor skaliranja *s* nepoznanice. ***rotM*** predstavlja matricu rotacije, tj. opisuje kako je koordinatni sustav kamere orijentiran u odnosu na koordinatni sustava robota i izražava se u radijanima. ***tvec*** predstavlja vektor translacije, tj. opisuje položaj koordinatnog sustava kamere u odnosu na koordinatni sustav

robota te se izražava u mm. Kako se ovdje promatra samo 2D slučaj, tj. potrebno je pronaći *x* i *y* koordinate, faktor skaliranja *s* predstavlja visinu kamere u odnosu na scenu koja se promatra.

Za pronalazak X_W, Y_W i Z_W koordinata potrebno je riješiti inverz matrice [***rotM***|***tvec***], no može se uočiti da ona nije kvadratna. Ako se pretpostavi da visina scene na koju kamera gleda, *z*, nije 0, model se može pojednostaviti kako ga jednadžba (37) prikazuje.

$$\begin{bmatrix} X_W \\ Y_W \\ Z_W \end{bmatrix} = \left[s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \boldsymbol{mtx}^{-1} - \boldsymbol{tvec} \right] \boldsymbol{rotM}^{-1} \tag{37}$$

Kalibracija koordinatnog sustava radi tako da kamera uslika radno područje na kojem se nalazi marker kojem se znaju pozicije njegova četiri vrha u koordinatnom sustavu robota. Program prepoznaje marker i pronalazi položaj njegova četiri vrha u koordinatnom sustavu slike, tj. *u* i *v*. Korišteni su *Aruco* markeri za prepoznavanje koji su kao i funkcije preuzeti sa stranice *https://mecaruco2.readthedocs.io*[28]. *calibrateCSDetectMarkers()* metoda *camera* objekta služi za prepoznavanje markera i njegova četiri vrha.

```
def calibrateCSDetectMarkers(self):
    _, frame = self.cap.read()
    aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
    parameters =  aruco.DetectorParameters_create()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    corners, ids, _ = aruco.detectMarkers(gray, aruco_dict, parameters=parameters)
    frame_markers = aruco.drawDetectedMarkers(frame.copy(), corners, ids)
    return frame_markers, corners, ids
```

Koordinate četiri vrha markera u stvarnom svijetu i na slici, zajedno sa ***mtx*** i ***dst*** matricama iz (34) i (35) ulaze kao parametri u *cv2.solvePnP()* funkciju koja pronalazi vektor translacije ***tvec*** i vektor rotacije ***rvec***. Za jednadžbu (37) potrebna je matrica rotacije, stoga se vektor rotacije *rvec* prebacuje u matricu rotacije ***rotM*** pomoću *cv2.Rodrigues()* funkcije.

Za kalibraciju koordinatnog sustava, korisnik sučelja odabire „Kalibracija k.s.“ te program pokreće funkciju *calibrateCoordinateSystemFunc()*. Funkcija prvo pomiče robota izvan vidnog područja kamere, zatim poziva *calibrateCSDetectMarkers()* metodu koja slika radno područje robota, prepoznaje marker, pronalazi njegova četiri vrha i ocrtava ih kako Slika 134 prikazuje. Ta slika se pokaže korisniku sučelja koji pritiskom tipke 'a' na tipkovnici može prihvatiti detekciju markera, pritiskom tipke 'r' može ponoviti detekciju ili pritiskom tipke 'q' može odustati od kalibracije.

Slika 134. Kalibracija koordinatnog sustava, detekcija markera

Ako je korisnik prihvatio sliku, poziva se *calibrateCSSequence()* metoda *camera* objekta.

```
def calibrateCSSequence(self, passCorners):
    self.imagePoints = passCorners[0][0]
    _, self.rvec, self.tvec = cv2.solvePnP(self.objectPoints, self.imagePoints,
                                           self.mtx, self.dist, cv2.SOLVEPNP_IPPE)
    self.rotM = cv2.Rodrigues(self.rvec)[0]
    print("imagePoints\n", self.imagePoints)
    print("rvec:\n", self.rvec)
    print("tvec:\n", self.tvec)
    print("rotM:\n", self.rotM)
```

Metoda računa i ispisuje *tvec*, *rvec* i *rotM* parametre.

$$\boldsymbol{tvec} = \begin{bmatrix} 2{,}92 \\ 203{,}36 \\ 625{,}33 \end{bmatrix} \text{ mm} \tag{38}$$

$$\boldsymbol{rvec} = \begin{bmatrix} -3{,}15 \\ 0{,}05 \\ 0{,}17 \end{bmatrix} \text{ rad} \tag{39}$$

$$\boldsymbol{rotM} = \begin{bmatrix} 0{,}99 & -0{,}03 & -0{,}11 \\ -0{,}03 & -0{,}99 & -0{,}01 \\ -0{,}11 & -0{,}02 & -0{,}99 \end{bmatrix} rad \tag{40}$$

Dobiveni vektor translacije i matricu rotacije koristi *pixelCSToRobotCS()* metoda *camera* objekta kako bi transformirala koordinate iz koordinatnog sustava slike u koordinatni sustav robota.

```
    def pixelCSToRobotCS(self, u, v):
        uV = np.array([[u, v, 1]], np.float32)
        uV = self.scaleFactor * uV.T
        xyz = inv(self.mtx).dot(uV)
        xyz_c = np.subtract(xyz, self.tvec.T)
        invRotM = inv(self.rotM)
        XYZ = invRotM.dot(xyz_c)
        return int(XYZ[0]), int(XYZ[1])
```

4.4.1.3. Lokalizacija predmeta rada na temelju boje i oblika

Nakon kalibracije kamere i povezivanja koordinatnog sustava, potrebno je napraviti lokalizaciju objekta po boji i obliku. Objekti koji će se prepoznavati su osnovni geometrijski oblici trokut, kvadrat, pravokutnik, peterokut, šesterokut i kružnica. Za svaki od oblika postoji jedna plava, jedna crvena i jedna zelena verzija. Slika 135 prikazuje navedene elemente.

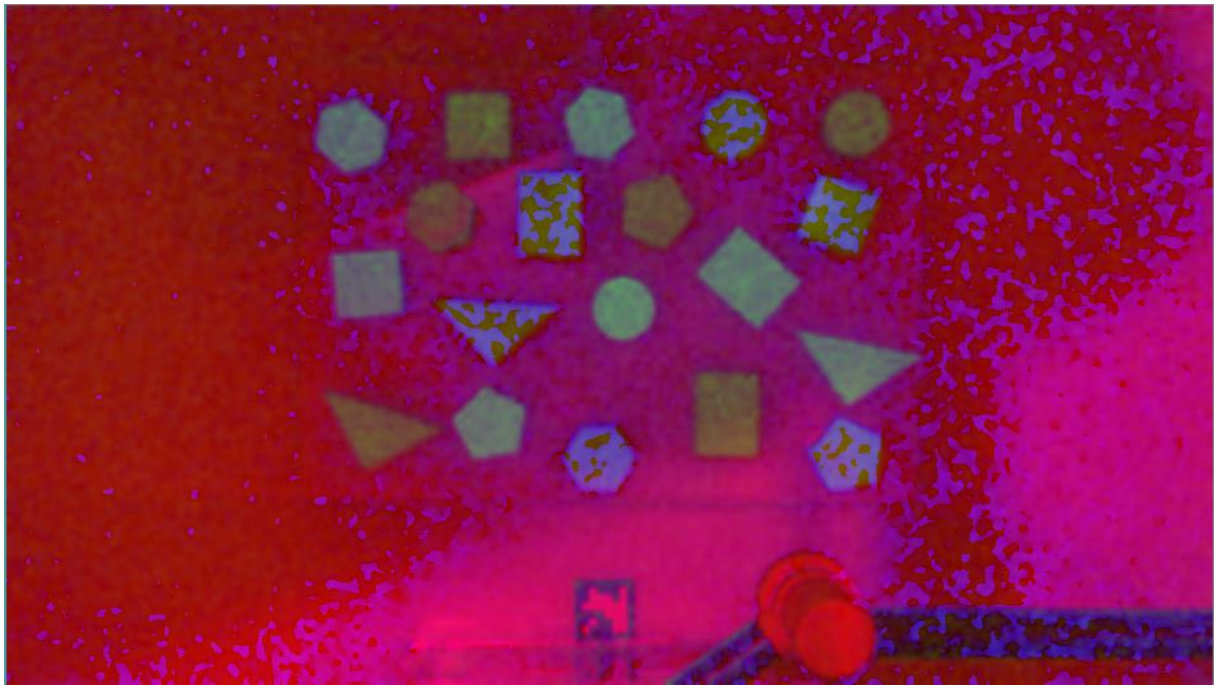
Slika 135. Osnovni geometrijski elementi za prepoznavanje

Proces lokalizacije kreće tako da korisnik sučelja odabire „Prepoznaj“ koje prikazuje Slika 130. Program pokreće *detectFunc()* funkciju koja prvo pomiče robot izvan vidnog područja kamere kako bi kamera mogla snimiti sliku. Zatim se uslikana slika prenosi u *detectFuncCamera()* metodu *camera* objekta.

Kako je *detectFuncCamera()* metoda dugačka, pokazat će se dijelovi metode za prepoznavanje plave boje, dok se implementacija cijele metode nalazi u prilogu pod [I]. Za primjer detekcije koristi se Slika 135.

Metoda na početku kreira praznu listu *detectedShapes* gdje posprema sve detektirane objekte, tj. njihov oblik, boju i središte u koordinatnom sustavu robota. Na stranici *https://opencv-python-tutroals.readthedocs.io*[29] pokazno je kako je moguće napraviti detekciju boje koristeći *HSV* prostor boja, stoga je taj pristup korišten. Dobivena slika s kamere transformira se iz *RGB* prostora boja u *HSV* prostor boja te se dobiva rezultat kojeg prikazuje Slika 136.

```
hsv = cv2.cvtColor(blurredHSV, cv2.COLOR_BGR2HSV)
```



Slika 136. Slika u *HSV* prostoru boja

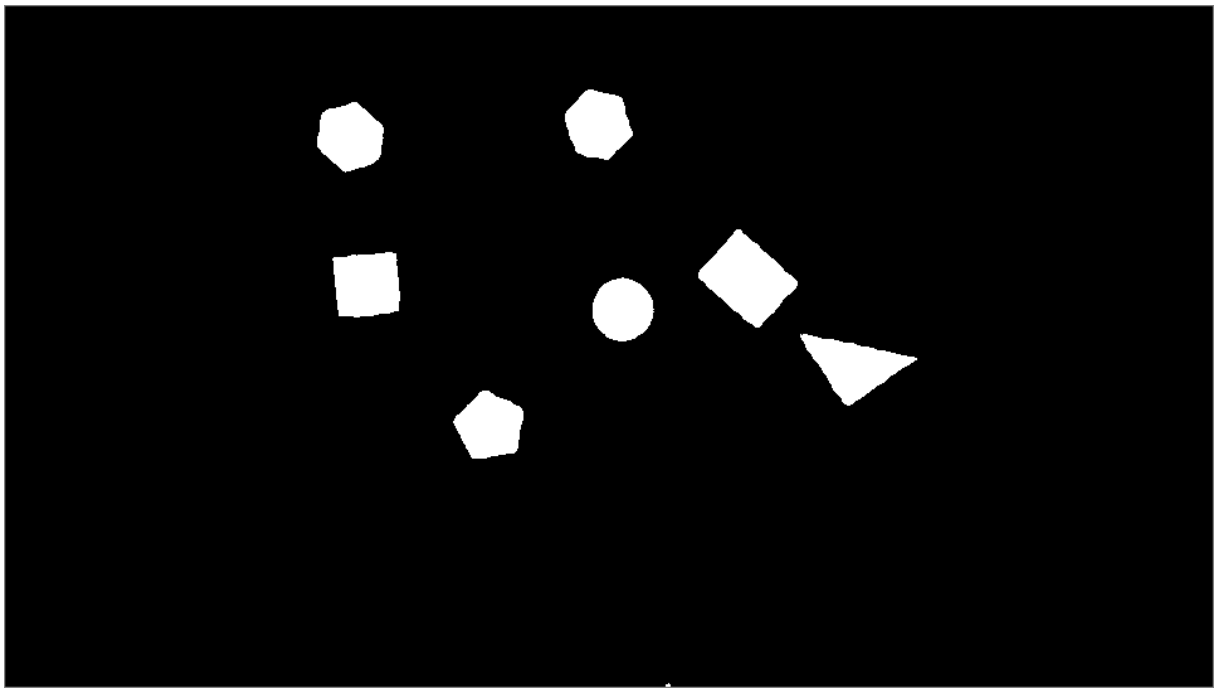
Zatim se napravi raspon plave boje u HSV prostoru.

```
lower_blue = np.array([75,30,30])
upper_blue = np.array([130,255,255])
```

Kada se raspon gornje i donje granice plave boje te slika u *HSV* prostoru boja uvrste u *cv2.inRange* funkcije dobivaju se rezultati koje prikazuje Slika 137.

```
maskBlue = cv2.inRange(hsv, lower_blue, upper_blue)
```

cv2.inRange funkcija uspoređuje gornje i donje vrijednosti plave boje s pikselima sa slikom u *HSV* prostoru boja. Svim pikselima unutar zadanih granica plave boje se postavlja vrijednost na 1, dok svim ostalim pikselima postavlja vrijednost na 0. Time su se dobila potpuno bijela područja na slici koja predstavljaju sve plave oblike koje prikazuje Slika 135.



Slika 137. Maska za plavu boju

Nakon što se pronašla maska za plavu boju, traže se sve konture koje prikazuje Slika 137. Za to se koristi *cv2.findContours()* funkcija.

```
cntsB = cv2.findContours(maskBlue.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Konturama se mogu nazivati krivulje koje zatvaraju neki prostor jednakih točaka, u ovom slučaju sva bijela područja na slici. Za pronalazak kontura se preporuča korištenje binariziranih slika, tj. gdje su svi pikseli ili 0 ili 1.[30] To je ovdje i napravljeno upotrebom maske, kako Slika 137 pokazuje. Jednom kada se pronađu konture, s njima se mogu raditi razne operacije kako je vidljivo na stranici *https://opencv-python-tutroals.readthedocs.io*[31].

```
for c in cntsB[0]:
    if(cv2.contourArea(c) < minArea):
        continue
    epsilon = 0.035*cv2.arcLength(c,True)
    approx = cv2.approxPolyDP(c,epsilon,True)
    shape = self.detectShape(approx)
    M = cv2.moments(approx)
    if (M["m00"] == 0):
        M["m00"]=1
    cX = int(M["m10"] / M["m00"])
    cY = int(M["m01"] / M["m00"])
    rX, rY = self.pixelCSToRobotCS(cX, cY)
    centerCo = str(rX) + ", " + str(rY)
```

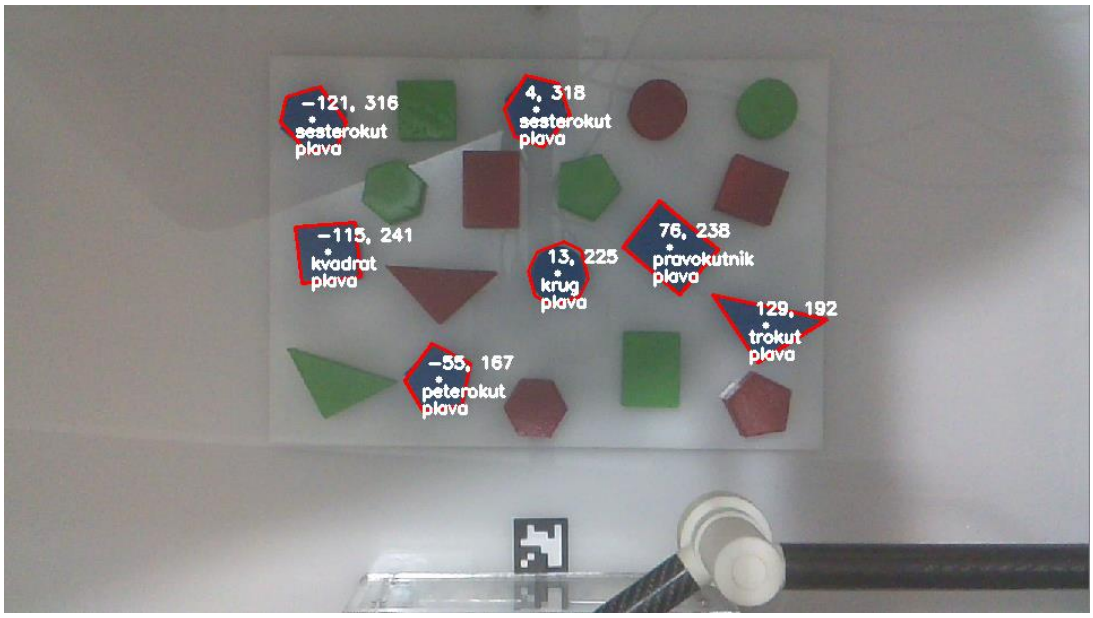
```
        cv2.drawContours(frame,[approx],0,(0,0,255),2)
        cv2.circle(frame, (cX, cY), 3, (255, 255, 255), -1)
        cv2.putText(frame, centerCo,  (cX - 10, cY - 10),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
        cv2.putText(frame, shape, (cX - 15, cY + 15),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
        cv2.putText(frame, "plava", (cX - 15, cY + 30),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
        self.detectedShapes.append([shape, "blue", (rX, rY)])
```

Dobivene konture prvo se aproksimiraju, tj. pojednostavljuju kako bi se dobili pravilniji oblici i uklonio nastali šum na rubovima kojeg prikazuje i Slika 137. Za to se koristi *cv2.arcLentgth()*. Dobivene aproksimirane konture koriste se za prepoznavanje oblika koje obavlja *detectShape()* metoda unutar *camera* objekta. Metoda za ulazne aproksimirane konture vraća podatak radi li se o trokutu, krugu ili nekom drugom od objekata koje je potrebno prepoznati.

```
    def detectShape(self, approx):
        shape = ""
        if len(approx) == 3:
            shape = "trokut"
        elif len(approx) == 4:
            rect = cv2.minAreaRect(approx)
            box = cv2.boxPoints(rect)
            box = np.int0(box)
            w = np.sqrt((box[0][0]-box[1][0])*(box[0][0]-box[1][0]) +
                        (box[0][1]-box[1][1])*(box[0][1]-box[1][1]))
            h = np.sqrt((box[1][0]-box[2][0])*(box[1][0]-box[2][0]) +
                        (box[1][1]-box[2][1])*(box[1][1]-box[2][1]))
            if 0.9 <= w/h <= 1.1:
                shape = "kvadrat"
            else:
                shape = "pravokutnik"
        elif len(approx) == 5:
            shape = "peterokut"
        elif len(approx) == 6:
            shape = "sesterokut"
        else:
            shape = "krug"
        return shape
```

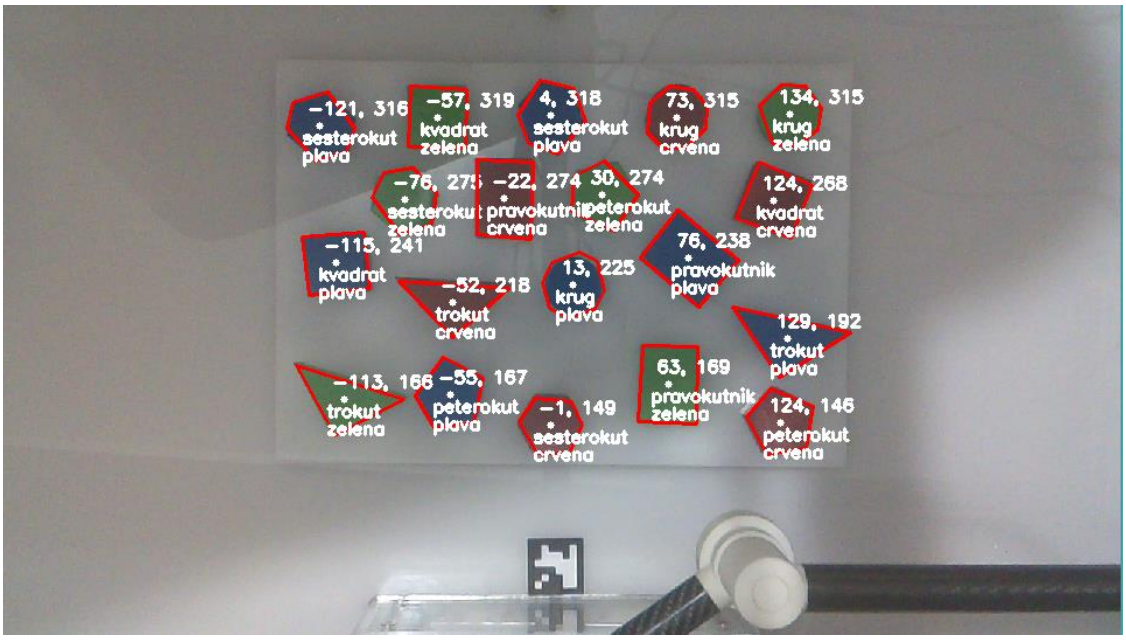
Osim prepoznavanja oblika, računaju se i momenti slike pomoću *cv2.moments()* funkcije. Momenti slike služe za pronalazak središta aproksimiranih kontura. Dobivena središta potrebno je prebaciti iz koordinatnog sustava slike u koordinatni sustav robota za što se koristi ranije spomenuta *pixelsToRobotCS()* metoda. Zbog tog razloga bilo je potrebno napraviti kalibraciju

koordinatnog sustava. Nakraju se crtaju prepoznate konture, središta, boja i oblik na sliku. Dobiveni rezultat za plavu boju prikazuje Slika 138.



Slika 138. Prepoznavanje plavih objekata

Isti postupak napravi se i za prepoznavanje zelenih i crvenih objekata te se dobiju rezultati koje prikazuje Slika 139. Postupak izvođenja *detectFuncCamera()* metode prikazuje Slika 140. Nakon što se završi detekcija svih objekata, *detectFuncCamera()* metoda vraća sliku s prepoznatim objektima te ranije spomenutu listu *detectedShapes*. Korisnik sučelja pritiskom tipke 'a' na tipkovnici može prihvatiti detektirane objekte, ponoviti detekciju pritiskom tipke 'r' ili odustati od detekcije pritiskom tipke 'q'.



Slika 139. Prepoznavanje svih objekata

Slika 140. Proces prepoznavanja boje i oblika objekta

Ako korisnik sučelja prihvati detektirane objekte isti se mogu i sortirati, što je objašnjeno u poglavlju 4.4.1.4.

4.4.1.4. Upravljanje robotom vizijskim sustavima kroz grafičko sučelje

Nakon prihvaćanja detektiranih objekata, omogućuje se korisniku odabir „Sortiraj“ funkcionalnosti. Prije pokretanja sortiranja, potrebno je odabrati vrstu sortiranja inače grafičko sučelje napominje korisnika da to napravi.

Odabirom „Sortiraj“ program pokreće *sortFunc()* funkciju koja ovisno o vrsti sortiranja pokreće *sortByColor()* metodu ili *sortByShape()* metodu *camera* objekta. Te metode uzimaju ranije spomenutu *detectedShapes* listu i sortira predmete unutar te liste po boji ili obliku. Kada se lista sortira u manje podliste, pokreću se funkcije *sortSequenceColor()* ili *sortSequenceShape()*. U slučaju sortiranja po boji, *sortSequenceColor()* ulazi u sortirane podliste po boji i to redom za plavu, zelenu i nakraju crvenu boju. Ako se sortira po obliku, sortSequenceShape() ulazi u sortirane podliste po obliku i to redom za trokut, kvadrat, pravokutnik, peterokut, šesterokut i nakraju krug. U oba slučaja program za svaki prepoznati objekt poziva *sortSequenceMoveRobot()* funkciju koja obavlja operacije izuzmi-odloži za proslijeđene koordinate iz sortirane liste. Implementacija *sortSequenceMoveRobot()* funkcije je prikazana niže, dok se implementacija ostalih metoda i funkcija nalazi u prilogu pod [I]

```
def sortSequenceMoveRobot(objectRoot, objectRobot, objectPneumatic,
                          drawRobotPositionPass, passCurrentXValue, passCurrentYValue,
                          passSortedX, passSortedY, passGroupX, passGroupY):
    objectRobot.goToClosedLoop()
    objectRobot.moveAxis(passSortedX, passSortedY)
    drawRobotPositionPass.drawRobotPosition(objectRoot, passSortedX, passSortedY,
                                            objectRobot.theta0, objectRobot.theta1)
    passCurrentXValue.config(text = "X: " + str(round(objectRobot.x_TCP, 2)))
    passCurrentYValue.config(text = "Y: " + str(round(objectRobot.y_TCP, 2)))
    while True:
        objectRoot.update()
        if(objectRobot.atPosition()):
            break
    objectRoot.update()
    objectPneumatic.pick()
    objectRobot.moveAxis(passGroupX, passGroupY)
    drawRobotPositionPass.drawRobotPosition(objectRoot, passGroupX, passGroupY,
                                            objectRobot.theta0, objectRobot.theta1)
    passCurrentXValue.config(text = "X: " + str(round(objectRobot.x_TCP, 2)))
    passCurrentYValue.config(text = "Y: " + str(round(objectRobot.y_TCP, 2)))
    while(True):
        objectRoot.update()
        if(objectRobot.atPosition()):
            break
    objectRoot.update()
    objectPneumatic.place()
```

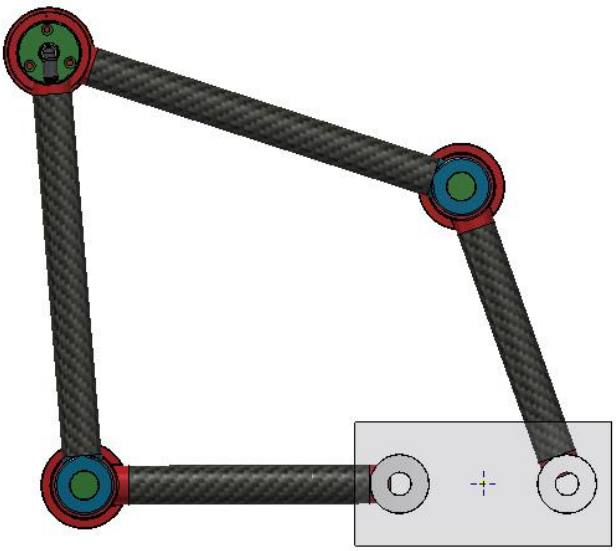
Kada se završi operacija sortiranja, robot odlazi u početnu poziciju tj. *home* te se motori robota gase.

5. MJERENJE

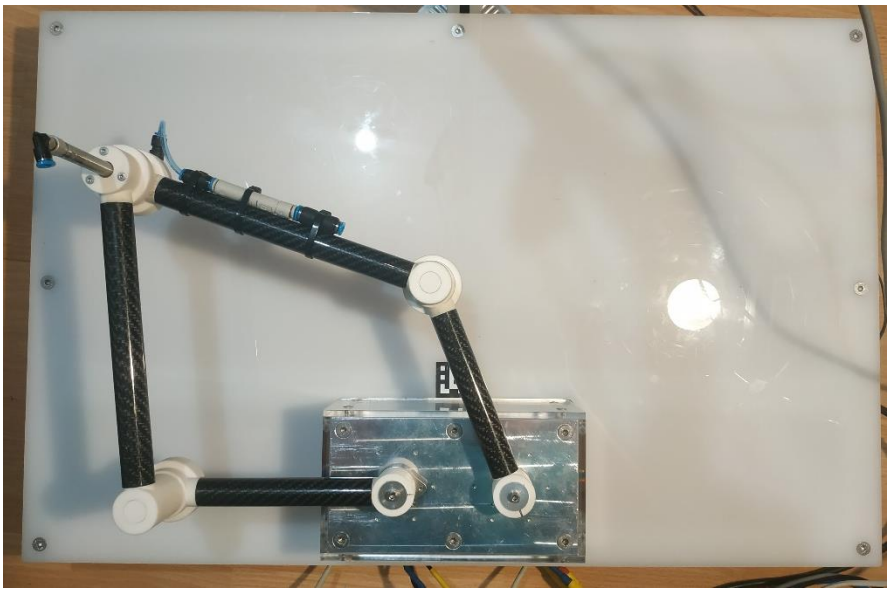
U ovom poglavlju napravljeno je mjerenje brzine, ubrzanja, ponovljivosti i nosivosti na eksperimentalnom postavu.

5.1. Mjerenje brzine i ubrzanja robota

Mjerenje brzine i ubrzanja napravljeno je korištenjem izlaznih podataka s drivera i simulacije iz poglavlja 3.4.2. Za ulazne parametre odabrane su dvije točke u koordinatnom sustavu robota. Prva ulazna točka je (-220, 200) i robot se tada nalazi u poziciji kako prikazuju Slika 141 i Slika 142.

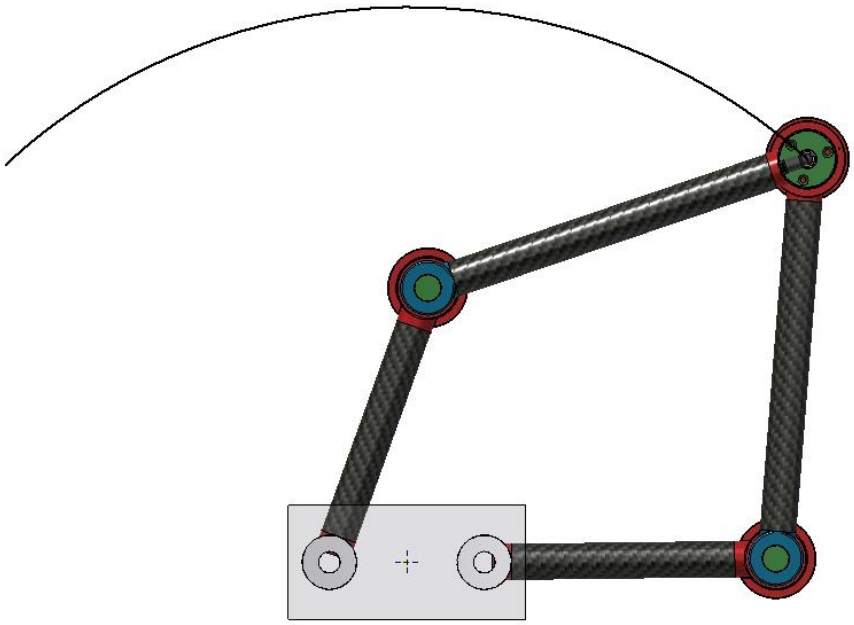


Slika 141. Mjerenje brzine i ubrzanja, pozicija 1 u simulaciji

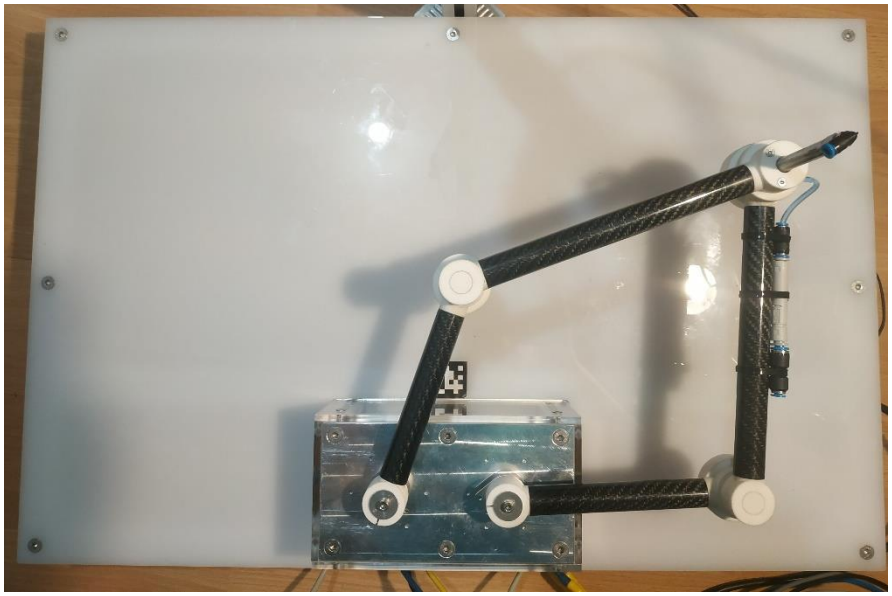


Slika 142. Mjerenje brzine i ubrzanja, pozicija 1, eksperimentalni postav

Druga odabrana ulazna točka je (220,220) i robot se tada nalazi u poziciji kako prikazuju Slika 143 i Slika 144



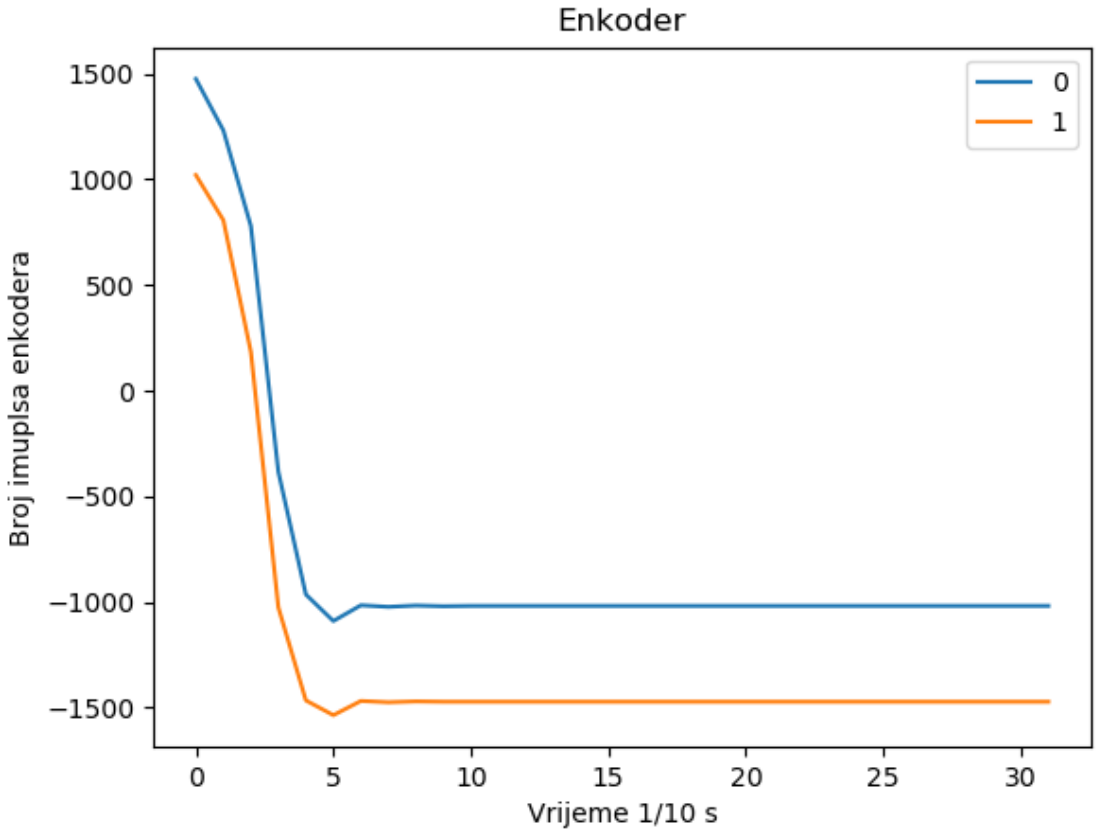
Slika 143. Mjerenje brzine i ubrzanja, pozicija 2 u simulaciji



Slika 144. Mjerenje brzine i ubrzanja, pozicija 2, eksperimentalni postav

Za početnu i krajnju točku potrebno je pomoću inverzne kinematike dobiti početni i krajnji kut, koji za točku iz pozicije 1 (-220, 220) iznosi (109.61 °, 179.75 °) i pozicije 2 (220, 220) iznosi (0.25 °, 70.39 °). Kako driver za poziciju uzima broj impulsa enkodera u apsolutnoj referenci, što je ranije objašnjeno, potrebno je ulazne kutove prebaciti u impulse koji za poziciju 1 iznose (1470.18, 1018.32) i poziciju 2 (-1018.32, -1470.18). Graf pozicije enkodera motora0 i motora1

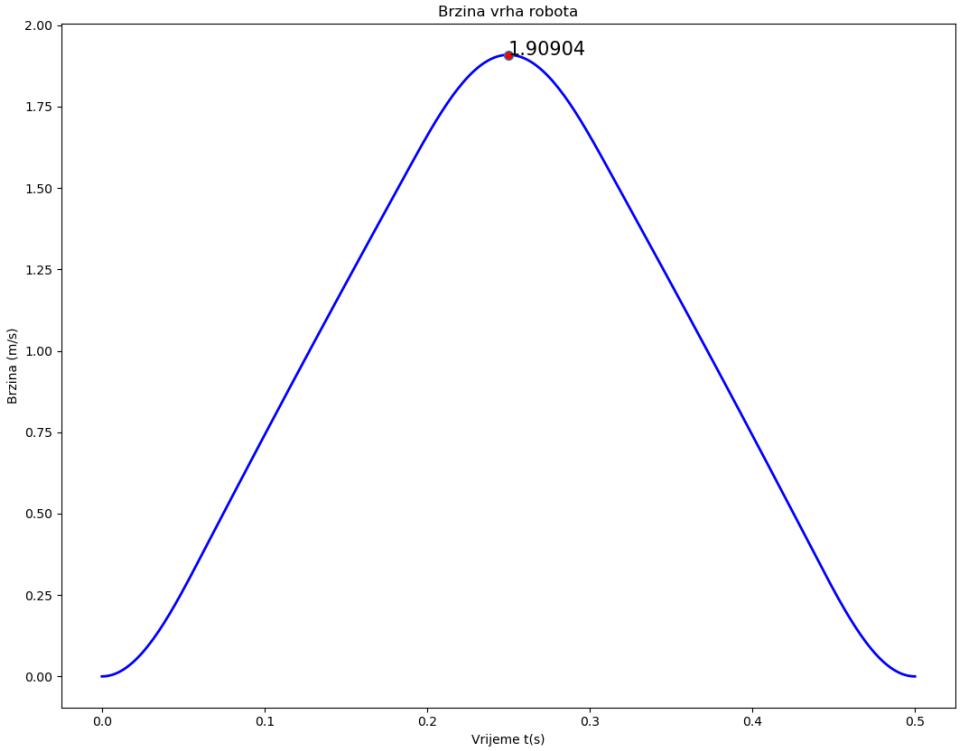
prikazuje Slika 145. Graf je dobiven korištenjem *start_liveplotter()* funkcije koja je dio *odrivetool* modula.



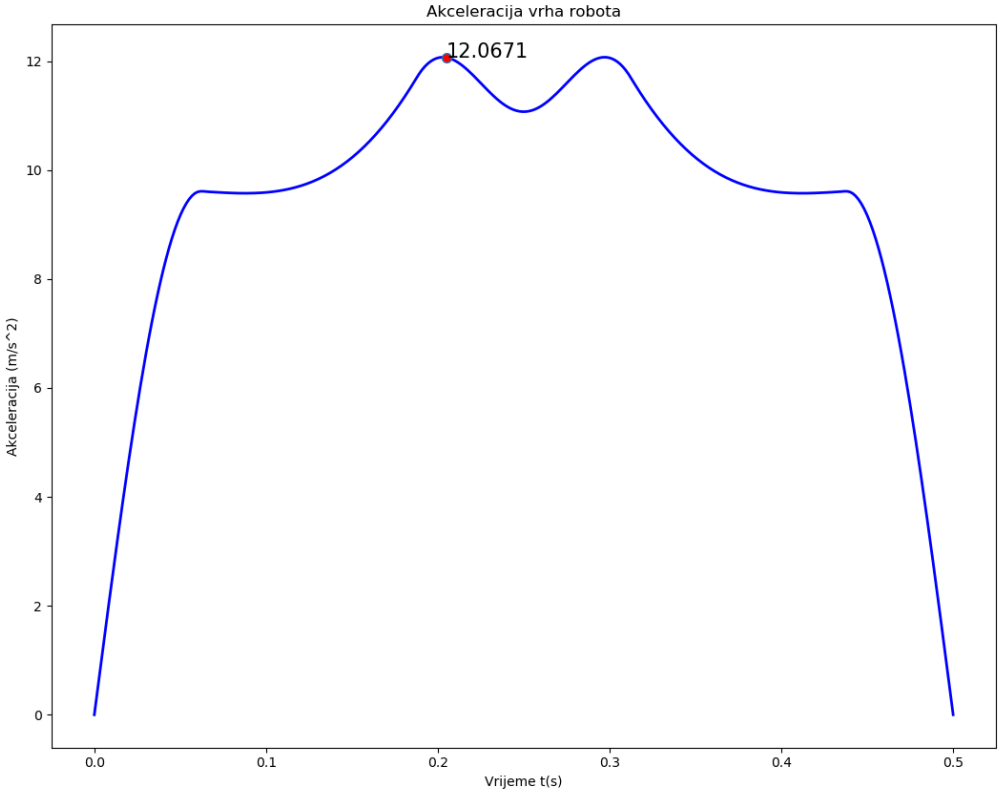
Slika 145. Pomaci enkodera motora0 i motora1

Prilikom mjerenja brzine i ubrzanja robota zadana je maksimalna struja motora do 55 A, brzina do 150000 impuls/s i ubrzanje do 150000 impuls/s^2. *x* os grafa prikazuje 10 uzoraka po jednoj sekundi, a *y* os prikazuje broj impulsa enkodera. Iz grafa se može očitati da će robot iz početne pozicije u impulsima (1470.18, 1018.32), u konačnu poziciju u impulsima (-1018.32, -1470.18) stići za 0.5 sekundi.

Ako se spomenuti kutovi dobiveni inverznom kinematikom uvrste u *Inventor Dynamic Simulation* i odredi se potrebno vrijeme prelaska iz pozicije 1 u poziciju 2 od 0,5 s, dobivaju se grafovi brzine i ubrzanja koje prikazuje Slika 146 i Slika 147. Može se očitati kako je maksimalna linearna brzina vrha robota 1.90904 m/s, dok je maksimalna akceleracija 12.0671 m/s^2. U zadatku je zadana minimalna brzina vrha robota od 1 m/s i akceleracija od 9.81 m/s^2, što znači da robot zadovoljava postavljeni uvjet.



Slika 146. Maksimalna brzina vrha robota



Slika 147. Maksimalno ubrzanje vrha robota

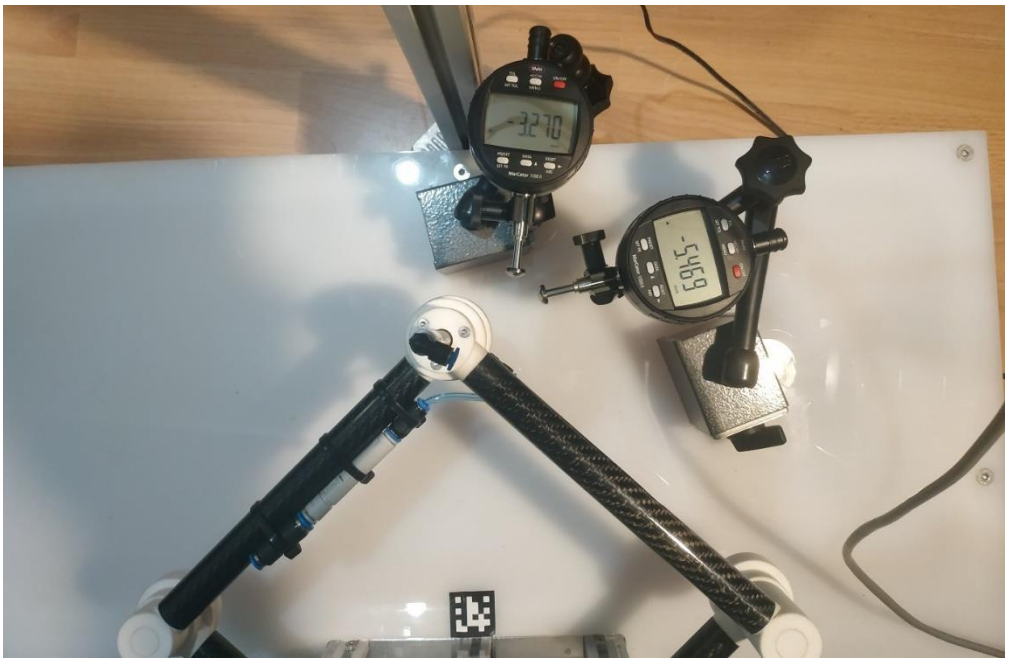
5.2. Mjerenje ponovljivosti robota

Ponovljivost robota mjerena je s dva komparatora *Mahr MarCator 1086R* i magnetskim držačem, koje prikazuje Slika 148**.**

Slika 148. Mahr MarCator 1086R

Mjerenje ponovljivosti napravljeno je tako da je robot postavljen u zadanu poziciju, npr. (40, 270) za poziciju jedan, kako Slika 149 prikazuje**.** Na toj poziciji pomično mjerilo se namjesti na 0. Zatim se robot odmaknuo na drugu poziciju kako Slika 150 prikazuje. Nakon toga se vrati na početnu poziciju kako Slika 151 prikazuje. Dok je robot na početnoj poziciji očita se razlika u pomaku s pomičnih mjerila. Napravljeno je 3 mjerenja po svakoj poziciji te su mjerenja napravljena na 4 različite pozicije kako Slika 149, Slika 152, Slika 153 i Slika 154 prikazuju.

Slika 149. Mjerenje ponovljivosti, pozicija 1



Slika 150. Mjerenje ponovljivosti, pozicija 1, robot odmaknut

Slika 151. Mjerenje ponovljivosti, pozicija 1, robot primaknut

Slika 152. Mjerenje ponovljivosti, pozicija 2

Slika 153. Mjerenje ponovljivosti, pozicija 3

Slika 154. Mjerenje ponovljivosti, pozicija 4

Tablica 1 prikazuje rezultate mjerenja ponovljivosti robota.

Tablica 1. Rezultati mjerenja ponovljivosti.

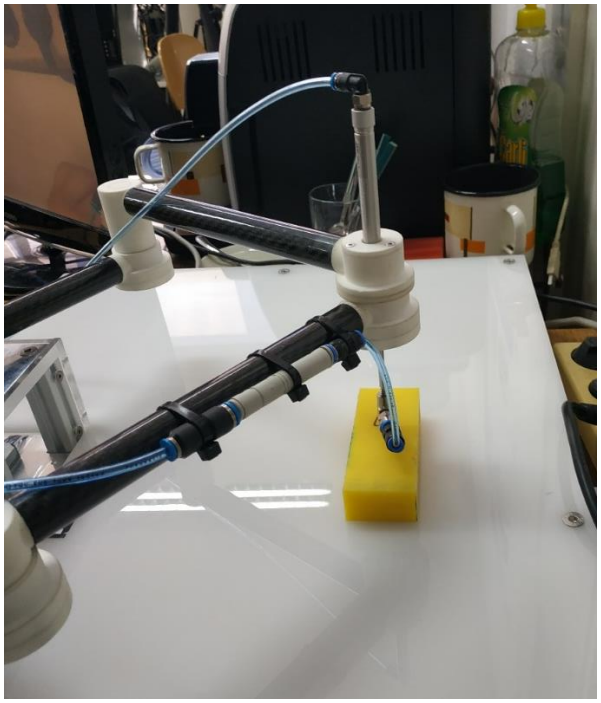
Pozicija	Mjerna točka	Točka odmaka	Mjerenje pomaka 1[mm]	Mjerenje pomaka 2 [mm]	Mjerenje pomaka 3 [mm]	Pomično mjerilo
1	(40, 270)	(0, 250)	-0,033	-0,100	-0,060	1
			-0,048	-0,104	-0,060	2
2	(-160, 165)	(0, 120)	0,017	0,102	0,031	1
			0,006	0.105	0.021	2
3	(0, 100)	(0, 200)	0.065	0.035	-0.101	1
			-0.058	-0.072	-0.107	2
4	(130, 90)	(0, 75)	0.036	-0.084	0.064	1
			0.071	0.095	-0.035	2

Može se vidjeti kako je najveće odstupanje komparatora -0.107 mm. Ovim mjerenjem ne može se izračunati niti pokazati točna ponovljivost vrha robota. Mjerenje je napravljeno kako bi se vidjela kolika je maksimalna ponovljivost u različitim smjerovima i pozicijama na *xy* ravnini.

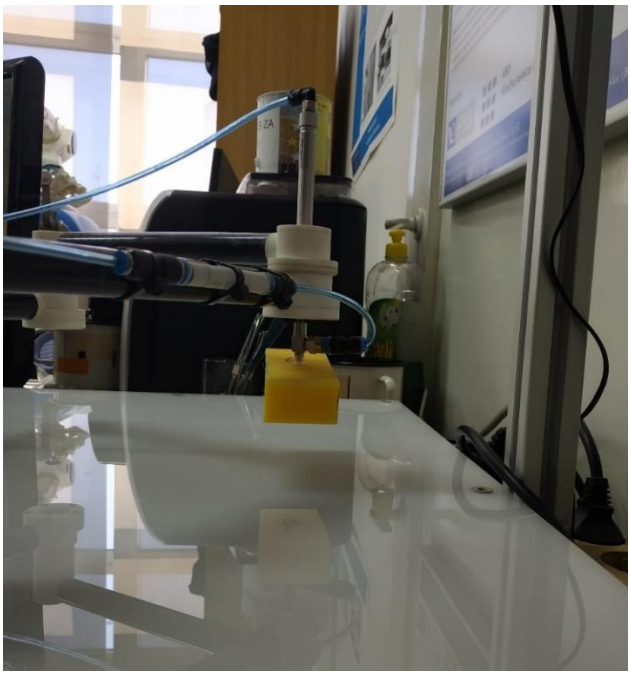
5.3. Mjerenje nosivosti robota

Mjerenje nosivosti robota provedeno je tako da robot proba napraviti operaciju izuzmi-odloži predmeta mase koja je veća od zadatkom zadane minimalne nosivosti robota, tj. 50 g. Zato je odabran predmet kojeg prikazuje Slika 155. Njegova masa iznosi 103 g te je izrađen od polimera. Slika 156, Slika 157 i Slika 158 prikazuju postupak izuzmi-odloži na predmetu mase 103g.

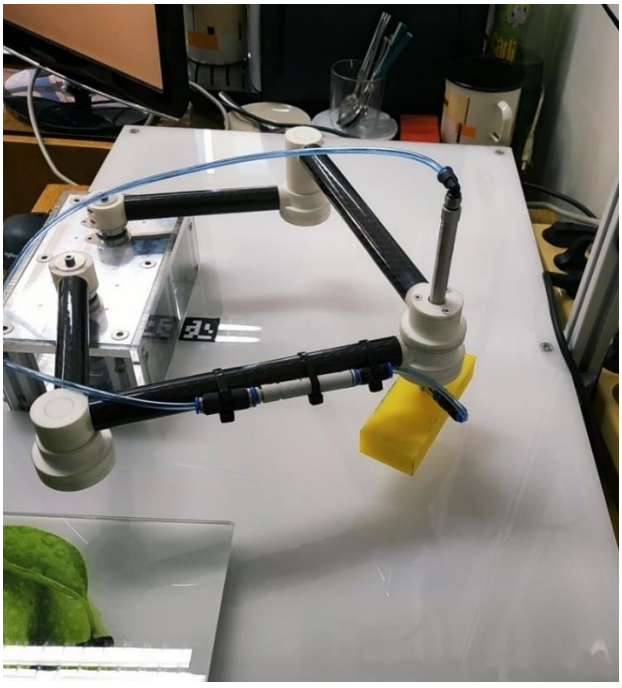
Slika 155. Masa predmeta za testiranje nosivosti



Slika 156. Mjerenje nosivosti robota 1



Slika 157. Mjerenje nosivosti robota 2



Slika 158. Mjerenje nosivosti robota 3

Robot je sposoban raditi operacije izuzmi-odloži na predmetima većim od 50 g, no potrebno je napomenuti kako je ovaj predmet od 103g bio gornja granica nosivosti robota. Zbog malog promjera vakuumske kapice, sila držanja bila je dovoljna samo za rukovanje predmetom malim brzinama. No robot je namijenjen za operacije rukovanja predmetima manjih masa i dimenzija, kakve Slika 135 prikazuje.

6. ZAKLJUČAK

Kroz ovaj diplomski rad pokazan je kompletan razvoj jednog manjeg i jednostavnijeg robota paralelne kinematike. Prvo se kroz uvod pokazalo neke od poznatijih vrsta robota paralelne kinematike te koje su im prednosti i mane. Zatim su se ukratko pokazale formule za direktnu i inverznu kinematiku koje su našle svoju primjenu kasnije, kod upravljanja robotom. Isto je pokazano kako i robotu sa samo 2 stupnja slobode gibanja mogu raditi probleme područja singularnosti. No pokazano je kako se odabirom pravilnih parametara robota mogu izbjeći područja singularnosti uz zadovoljavanje minimalnog radnog prostora. Nakon upoznavanja s robotima paralelne kinematike, bilo je potrebno razraditi cijeli koncept u *CAD* programu. Tu je pokazan tijek razvoja proizvoda. Od početnih nekoliko koncepata do završnog proizvoda, kojeg je prije same izrade potrebno validirati sa simulacijama. Isto tako definirano je od kojeg materijala su dijelovi robota izrađeni te kojim tehnikama izrade su dobiveni. Pokazan je i odabir standardnih dijelova poput ležajeva, vakuumske hvataljke itd. Potom je iz rezultata simulacije odabrana vrsta aktuatora za koju je bilo potrebno razraditi ostatak kućišta. Nakon razrade cijelog robota, bilo ga je potrebno izraditi i potom montirati. Nakon montaže robota napravljen je i program kojim je robot upravljan. Kako bi robot bio što više intuitivan za krajnjeg korisnika, napravljeno je cijelo grafičko sučelje kako bi upravljanje robotom bilo što lakše. Osim ručnog upravljanja robotom, napravljeno je prepoznavanje predmeta po boji i/ili obliku te njihovo razvrstavanje primjenom vizijskih sustava. Nakraju su napravljena neka informativna mjerenja kako bi se vidjelo zadovoljava li robot zadatkom postavljene uvjete, poput brzine, ubrzanja, ponovljivosti i nosivosti. Kada se napravi sve gore navedeno, može se vidjeti koliko je to dugačak i kompleksan proces, gdje uspješnost razvoja robota ovisi o različitim faktorima. Ovaj diplomski rad predstavlja spoj različitih područja od konstruiranja do programiranja, stoga je kroz njegovu izradu i mnogo naučeno. Za dobiveni robot nije razvijeno najbolje moguće upravljanje niti je napravljena najbolja vizija, ali to nije ni bio cilj ovog rada. Cilj je bio robota napraviti iz nule kao otvorenu platformu za njegov daljnji napredak i usavršavanje kroz druge diplomske i završne radove, što je u ovom diplomskom radu i postignuto.

LITERATURA

[1] Parallel manipulator – Wikipedia, https://en.wikipedia.org/wiki/Parallel_manipulator, Zadnji pristup: 5.3.2020.

[2] DeltaRamki - Delta robot – Wikipedia, https://en.wikipedia.org/wiki/Delta_robot#/media/File:DeltaRamki.gif, Zadnji pristup: 5.3.2020.

[3] Simulator-flight-compartment - Stewart platform – Wikipedia, https://en.wikipedia.org/wiki/Stewart_platform#/media/File:Simulator-flight-compartment.jpeg, Zadnji pristup: 5.3.2020.

[4] Robots industriales de cinemática paralela FANUC M-2, https://www.fanuc.eu/es/es/robots/p%C3%A1gina-filtro-robots/serie-m2, Zadnji pristup: 5.3.2020.

[5] Robot SCARA _ 4 ejes _ para tiempo de ciclo rápido - RP series - Mitsubishi Electric Europe B.V., https://www.directindustry.es/prod/mitsubishi-electric-europe-bv/product-12225-412261.html, Zadnji pristup: 5.3.2020.

[6] Geometrija – GeoGebra, https://www.geogebra.org/geometry, Zadnji pristup: 5.3.2020.

[7] Circle, Cylinder, Sphere, http://paulbourke.net/geometry/circlesphere/, Zadnji pristup: 5.3.2020.

[8] Le, Tien & Kang, Hee-Jun & Doan Quang, Vinh. (2013). A method for optimal kinematic design of five-bar planar parallel manipulators. 2013 International Conference on Control, Automation and Information Sciences, ICCAIS 2013. 7-11. 10.1109/ICCAIS.2013.6720521.

[9] Gosselin, Clément & Angeles, Jorge. (1990). Singularity Analysis of Closed-Loop Kinematic Chains. Robotics and Automation, IEEE Transactions on. 6. 281 - 290. 10.1109/70.56660.

[10] Figielski, Alexandre & Bonev, Ilian & Bigras, Pascal. (2007). Towards development of a 2-DOF planar parallel robot with optimal workspace use. 1562 - 1566. 10.1109/ICSMC.2007.4413840.

[11] doc. dr. sc. Krešimir Vučković, Ležajevi, ak. god. 2018/19, Zadnji pristup: 4.3.2020.

[12] DSNU_EN, https://www.festo.com/cat/hr_hr/data/doc_engb/PDF/EN/DSNU_EN.PDF, Zadnji pristup: 4.3.2020.

[13] ESG_EN, https://www.festo.com/cat/hr_hr/data/doc_engb/PDF/EN/ESG_EN.PDF, Zadnji pristup: 4.3.2020.

[14] Home _ Festo Croatia, https://www.festo.com/cms/hr_hr/index.htm, Zadnji pristup: 4.3.2020.

[15] robot joint reducer gearbox, robot arm speed reducer - Xernt.com, https://www.xernt.com/robot-joint-reducer-gearbox-robot-arm-speed-reducer.html, Zadnji pristup: 4.3.2020.

[16] Inside The Arm of a Motoman – YouTube, https://www.youtube.com/watch?v=lQqHbf_qVWc, Zadnji pristup: 4.3.2020.

[17] Harmonic Drive® strain wave gear, https://harmonicdrive.de/en/glossary/harmonic-driver-strain-wave-gear, Zadnji pristup: 4.3.2020.

[18] APS 5065 Outrunner brushless motor 90KV 1800W, https://alienpowersystem.com/shop/brushless-motors/50mm/aps-5065-outrunner-brushless-motor-90kv-1800w/, Zadnji pristup: 4.3.2020.

[19] Things in Motion_ How to select the right power source for a hobby BLDC (PMSM) motor, https://things-in-motion.blogspot.com/2018/12/how-to-select-right-power-source-for.html, Zadnji pristup: 4.3.2020.

[20] MJC-20-GR-6-8 _ CAD _ NBK, https://webassistants.partcommunity.com/cgi-bin/vbshtmlcgi?script=$CADENAS_DATA/23d-libs/nbk_assistant/webassistant/format.vbb&info=nbk/motion_control_parts/couplicon_mini/jaw_type_mjc/mjc/mjc_asmtab.prj&varset={D1=6},{D2=8},{CN=MJC-20-GR},{A=20},{L=10},{W=30},{B=8},{C=1.0},{F=5.0},{M=M3},{WT=0.7}&lina=MJC-20-GR-6-8&language=GB&dxf2d=n, Zadnji pristup 1.3.2020.

[21] NVIDIA Jetson Nano _ 3D CAD Model Library _ GrabCAD, https://grabcad.com/library/nvidia-jetson-nano-2, Zadnji pristup: 2.3.2020.

[22] Raspberry Pi Camera Module v2 _ 3D CAD Model Library _ GrabCAD, https://grabcad.com/library/raspberry-pi-camera-module-v2-1, Zadnji pristup: 2.3.2020.

[23] Getting Started _ ODrive, https://docs.odriverobotics.com/, Zadnji pristup: 2.3.2020.

[24] draw.io, https://www.draw.io/, Zadnji pristup: 5.3.2020.

[25] Control _ ODrive, https://docs.odriverobotics.com/control, Zadnji pristup: 2.3.2020.

[26] Camera Calibration — OpenCV-Python Tutorials 1 documentation, https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html, Zadnji pristup: 2.3.2020.

[27] Camera Calibration and 3D Reconstruction — OpenCV 2.4.13.7 documentation, https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html, Zadnjiu pristup: 1.3.2020.

[28] ARUCO markers_ basics — Scientific Python_ a collection of science oriented python examples documentation, https://mecaruco2.readthedocs.io/en/latest/notebooks_rst/Aruco/aruco_basics.html, Zadnji pristup: 1.3.2020.

[29] Changing Colorspaces — OpenCV-Python Tutorials 1 documentation, https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_colorspaces/py_colorspaces.html#converting-colorspaces, Zadnji pristup: 1.3.2020.

[30] Contours _ Getting Started — OpenCV-Python Tutorials 1 documentation, https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_contours_begin/py_contours_begin.html#contours-getting-started, Zadnji pristup: 1.3.2020.

[31] Contour Features — OpenCV-Python Tutorials 1 documentation, https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_contours/py_contour_features/py_contour_features.html#contour-features, Zadnji pristup: 1.3.2020.

PRILOZI

I. Programski kod za upravljanje radom robota

II. Tehnička dokumentacija

III. Poliamid *PA220*, mehanička svojstva

IV. *CD-R disc*

Programski kod za upravljanje robotom:

main.py:

```
import odrive
import startMode
import workMode
import robot

def main():
    odrv0 = odrive.find_any()
    robotObject = robot.Robot(odrv0)
    if(startMode.startModeInit(robotObject)):
        workMode.workModeInit(robotObject)
    else:
        startMode.unSuccesfulCalibration()

main()
```

startMode.py:

```
import tkinterWidgets as tW
import initRobot as iR
import calibrateRobot as cR

calibrationSuccesful = False

def startModeInit(objectRobot):
    rootSM = tW.Tk()
    rootSM.title("5-Bar Robot")
    rootSM.configure(background = tW.backgroundColor)
    startMsgTxt1 = tW.StringVar()
    startMsgTxt2 = tW.StringVar()
    startMsgTxt1.set("5-BAR PARALLEL ROBOT")
    startMsgTxt2.set("-odaberite Inicijalizacija robota ako su  motori i
enkoderi već kalibrirani.\n\n-odaberite Kalibracija robota za
kalibraciju motora i enkodera.\n\n-odaberite Preskoči inicijalizaciju
ako je  robot već inicijaliziran, a driver nije  ponovno pokrenut.\n")
    startMsg1 = tW.MessageTemplate(rootSM, textvariable = startMsgTxt1,
                                   font = ("Helvetica", "20", "bold"))
    startMsg2 = tW.MessageTemplate(rootSM, textvariable = startMsgTxt2,
                                   font = ("Helvetica", "16"))
    startMsg1.pack(expand = True, fill = tW.X)
    startMsg2.pack(expand = True, fill = tW.X)
    initRobotButton = tW.ButtonTemplate(rootSM,
                          text = "Inicijalizacija robota", command = lambda:
                          initRobotFunc(rootSM, objectRobot))
```

```
    fullCalibrationButton = tW.ButtonTemplate(rootSM,
                            text = "Kalibracija robota", command = lambda:
                            fullCalibrationFunc(rootSM, objectRobot))
    skipCalibrationButton = tW.ButtonTemplate(rootSM,
                            text = "Preskoči inicijalizaciju",
                            command = lambda: skipCalibrationFunc(rootSM,
                            objectRobot))
    initRobotButton.pack(expand = True, fill = tW.X)
    fullCalibrationButton.pack(expand = True, fill = tW.X)
    skipCalibrationButton.pack(expand = True, fill = tW.X)
    rootSM.mainloop()
    if(iR.calibrationSuccesful or cR.calibrationSuccesful
                    or calibrationSuccesful):
        return True
    else:
        return False

def initRobotFunc(objectRoot, objectRobot):
    objectRoot.destroy()
    iR.initRobot(objectRobot)

def fullCalibrationFunc(objectRoot, objectRobot):
    objectRoot.destroy()
    cR.calibrateRobot(objectRobot)

def skipCalibrationFunc(objectRoot, objectRobot):
    global calibrationSuccesful
    objectRobot.sendStartParameters()
    calibrationSuccesful = True
    objectRoot.destroy()

def unSuccesfulCalibration():
    rootMUS = tW.Tk()
    rootMUS.title("5-Bar Robot")
    rootMUS.configure(background = tW.backgroundColor)
    startMsgTxt1 = tW.StringVar()
    startMsgTxt1.set("Kalibracija nije bila uspješna.\nPokušajte ponovno.")
    startMsg1 = tW.MessageTemplate(rootMUS,
                    textvariable = startMsgTxt1,
                    font = ("Helvetica", "24", "bold"), width = 1000)
    startMsg1.pack()
    quitButton = tW.ButtonTemplate(rootMUS,
                    text = "Zatvori", command = lambda: quitRoot(rootMUS))
    quitButton.pack(expand = True)
    rootMUS.mainloop()
```

```
def quitRoot(objectRoot):
    objectRoot.destroy()
```

initRobot.py:

```
import tkinterWidgets as tW
from PIL import Image, ImageTk
import time

calibrationSuccesful = False
delayLength = 3

def initRobot(objectRobot):
    rootiR = tW.Tk()
    rootiR.title("5-Bar Robot")
    rootiR.configure(background = tW.backgroundColor)
    iRTxt1 = tW.StringVar()
    iRTxt1.set("Inicijalizacija robota.")
    iRMsg1 = tW.MessageTemplate(rootiR, textvariable = iRTxt1,
                        font = ("Helvetica", "24", "bold"), width = 1000)
    iRMsg1.pack()
    iRTxt2 = tW.StringVar()
    iRTxt2.set("Stavite robota na poziciju kako slike prikazuju
                        i pritisnite Robot na poziciji.")
    iRMsg2 = tW.MessageTemplate(rootiR, textvariable = iRTxt2,
                        font = ("Helvetica", "16", "bold"), width = 1000)
    iRMsg2.pack()
    photoFrame = tW.FrameTemplate(rootiR)
    photoFrame.pack(expand = True, fill = tW.X)
    photoCanvas1 = tW.CanvasTempalte(photoFrame, width=700, height=490)
    photoCanvas2 = tW.CanvasTempalte(photoFrame, width=700, height=490)
    img1 = ImageTk.PhotoImage(Image.open(
      '/home/matej/Desktop/MatejBozic/5BarRobot/slike/robotPosition.JPG'))
    img2 = ImageTk.PhotoImage(Image.open(
      '/home/matej/Desktop/MatejBozic/5BarRobot/slike/robotPins.JPG'))
    photoCanvas1.create_image(700/2, 490/2, image = img1)
    photoCanvas2.create_image(700/2, 490/2, image = img2)
    photoCanvas1.pack(side = tW.LEFT, expand = True, fill = tW.X)
    photoCanvas2.pack(side = tW.RIGHT, expand = True, fill = tW.X)
    robotInPlaceButton = tW.ButtonTemplate(rootiR,
                    text = "Robot na poziciji", command = lambda:
                    enableHandCalibration(startInitializationButton))
    startInitializationButton = tW.ButtonTemplate(rootiR,
                    text = "Pokreni inicijalizaciju", state = tW.DISABLED,
                    command = lambda: startInitialization(rootiR ,objectRobot,
                    robotInPlaceButton , startInitializationButton, iRTxt2))
    robotInPlaceButton.pack(expand = True, fill = tW.X)
```

```
    startInitializationButton.pack(expand = True, fill = tW.X)
    rootiR.mainloop()

def enableHandCalibration(buttonPass):
    buttonPass.config(state = tW.NORMAL)

def startInitialization(objectRoot, objectRobot, buttonPass1,
                        buttonPass2, passTxt):
    global calibrationSuccesful
    buttonPass1.config(state = tW.DISABLED)
    buttonPass2.config(state = tW.DISABLED)
    #Obavijesti korisnika
    passTxt.set("Inicijalizacija je započela.")
    objectRoot.update()
    time.sleep(delayLength)
    #Pošalji parametre driveru
    passTxt.set("Parametri motora se šalju driveru.")
    objectRoot.update()
    time.sleep(delayLength)
    objectRobot.sendStartParameters()
    #Traženje Z-pulsa
    passTxt.set("Traženje Z-pulsa.")
    objectRoot.update()
    time.sleep(delayLength)
    objectRobot.zPulseSearch()
    #Kalibracija gotova
    passTxt.set("Z-puls pronađen. Kraj inicijalizacije")
    objectRoot.update()
    time.sleep(delayLength)
    calibrationSuccesful = True
    objectRoot.destroy()
```

calibrateRobot.py:

```
import tkinterWidgets as tW
from PIL import Image, ImageTk
import time

calibrationSuccesful = False
delayLength = 3

def calibrateRobot(objectRobot):
    rootcR = tW.Tk()
    rootcR.title("5-Bar Robot")
    rootcR.configure(background = tW.backgroundColor)
    cRTxt1 = tW.StringVar()
    cRTxt1.set("Kalbiracija motora i enkodera.")
```

```
    cRMsg1 = tW.MessageTemplate(rootcR, textvariable = cRTxt1,
             font = ("Helvetica", "24", "bold"), width = 1000)
    cRMsg1.pack()
    cRTxt2 = tW.StringVar()
    cRTxt2.set("Uklonite članke robota kako slika prikazuje i
                   pritisnite Članci robota uklonjeni.")
    cRMsg2 = tW.MessageTemplate(rootcR, textvariable = cRTxt2,
             font = ("Helvetica", "16", "bold"), width = 1000)
    cRMsg2.pack()
    photoCanvas1 = tW.CanvasTempalte(rootcR, width=1000, height=490)
    img1 = ImageTk.PhotoImage(Image.open(
      '/home/matej/Desktop/MatejBozic/5BarRobot/slike/robotPositionFull.JPG'))
    photoCanvas1.create_image(1000/2, 490/2, image = img1)
    photoCanvas1.pack(expand = True, fill = tW.X)
    armLinkagesRemovedButton = tW.ButtonTemplate(rootcR,
             text = "Članci robota uklonjeni",
             command = lambda: enableFullCalibration(startCalibrationButton))
    startCalibrationButton = tW.ButtonTemplate(rootcR,
             text =  "Pokreni kalibraciju", state = tW.DISABLED,
             command = lambda: startCalibration(rootcR ,objectRobot,
             armLinkagesRemovedButton, startCalibrationButton, cRTxt2))
    armLinkagesRemovedButton.pack(expand = True, fill = tW.X)
    startCalibrationButton.pack(expand = True, fill = tW.X)
    rootcR.mainloop()

def enableFullCalibration(buttonPass):
    buttonPass.config(state = tW.NORMAL)

def startCalibration(objectRoot, objectRobot, buttonPass1, buttonPass2, passTxt):
    global calibrationSuccesful
    buttonPass1.config(state = tW.DISABLED)
    buttonPass2.config(state = tW.DISABLED)
    #Obavijesti korisnika
    passTxt.set("Kalibracija je započela.")
    objectRoot.update()
    time.sleep(delayLength)
    #Pošalji parametre driveru
    passTxt.set("Parametri motora se šalju driveru.")
    objectRoot.update()
    time.sleep(delayLength)
    objectRobot.sendStartParameters()
    #Kalibracija motora
    passTxt.set("Kalibracija motora.")
    objectRoot.update()
    time.sleep(delayLength)
    objectRobot.motorCalibration()
    #Kalibracija enkodera
```

```
    passTxt.set("Kalibracija enkodera.")
    objectRoot.update()
    time.sleep(delayLength)
    objectRobot.encoderCalibration()
    #Kalibracija gotova
    passTxt.set("Spremanje parametara. Kraj kalibracije.")
    objectRoot.update()
    time.sleep(delayLength)
    objectRobot.finishCalibration()
    calibrationSuccesful = True
    objectRoot.destroy()
```

workMode.py:

```
import tkinterWidgets as tW
import preProgrammed as pP
import pneumatic
import camera

def workModeInit(objectRobot):
    objectPneumatic = pneumatic.Pneumatic()
    objectCamera = camera.Camera()
    root = tW.Tk()
    root.title("5-Bar Robot")
    root.configure(background = tW.backgroundColor)
    cameraLabel = tW.LabelTemplate(root, width = 960, height = 540)
    cameraLabel.grid(row = 0, column = 0, rowspan = 2, columnspan = 4)
    objectCamera.show_frame(cameraLabel)
    robotPosition = tW.CanvasRobot(root, width= 720, height= 540)
    robotPosition.grid(row = 0, column = 4, rowspan = 2, columnspan = 3)
    robotPosition.drawRobotWorksapce()
    robotPosition.bind("<B1-Motion>", lambda event :
                    followMouseMoveRobot(event, root, robotPosition,
                    objectRobot, currentXValue, currentYValue))
    slidersFrame = tW.FrameTemplate(root, width= 240, height= 270)
    slidersFrame.grid(row = 2, column = 0, rowspan = 1, columnspan = 2)
    varSliderC = tW.IntVar()
    varSliderV = tW.IntVar()
    varSliderA = tW.IntVar()
    currtenSlider = tW.ScaleTemplate(slidersFrame, label = "Makismalna struja",
                    from_ = 10, to  = 55, var = varSliderC,
                    command = lambda sliderType:
                    sliderToVar(varSliderC, objectRobot, "CURRENT"))
    trayVelSlider = tW.ScaleTemplate(slidersFrame, label = "Maksimalna brzina:",
                    from_ = 25000, to  = 150000, var = varSliderV,
                    command = lambda sliderType:
                    sliderToVar(varSliderV, objectRobot, "VELOCITY"))
```

```
    trayAccSlider = tW.ScaleTemplate(slidersFrame, label = "Maksimalno ubrzanje:",
                    from_ = 10000, to  = 150000, var = varSliderA,
                    command = lambda sliderType:
                    sliderToVar(varSliderA, objectRobot, "ACCLERATION"))
    currtenSlider.grid(row = 0, column = 0)
    trayVelSlider.grid(row = 1, column = 0)
    trayAccSlider.grid(row = 2, column = 0)
    varControl = tW.StringVar()
    controlTypeFrame = tW.LabelFrameTemplate(root,
                    text = "Vrsta regulacije:", width= 240, height= 270)
    controlTypeFrame.grid(row = 2, column = 2, rowspan = 1, columnspan = 1)
    looseControl = tW.RadiobuttonTemplate(controlTypeFrame,
                          text = "Opuštena", variable=varControl,
                          value="Loose", command = lambda:
                          chooseControle(varControl, objectRobot))
    tightControl = tW.RadiobuttonTemplate(controlTypeFrame,
                          text = "Čvrsta", variable=varControl,
                          value="Tight", command = lambda:
                          chooseControle(varControl, objectRobot))
    looseControl.grid(row = 0, column = 0)
    tightControl.grid(row = 1, column = 0)
    varType = tW.StringVar()
    sortTypeFrame =tW.LabelFrameTemplate(root, text = "Sortiraj po:",
                                         width= 240, height= 270)
    sortTypeFrame.grid(row = 2, column = 3, rowspan = 1, columnspan = 1)
    colorSort = tW.RadiobuttonTemplate(sortTypeFrame, text = "Boji",
                          variable=varType, value="Color", command = lambda:
                          typeSelection(varType, objectCamera))
    shapeSort = tW.RadiobuttonTemplate(sortTypeFrame, text = "Obliku",
                          variable=varType, value="Shape",
                          command = lambda: typeSelection(varType, objectCamera))
    colorSort.grid(row = 0, column = 0)
    shapeSort.grid(row = 1, column = 0)
    curentXYFrame = tW.LabelFrameTemplate(root, width= 240, height= 270,
                                          text = "Trenutna pozicija:")
    curentXYFrame.grid(row = 2, column = 4, rowspan = 1, columnspan = 1)
    currentXValue = tW.LabelTemplate(curentXYFrame, text = "X: " +
             str(round(objectRobot.x_TCP, 2)), font =("Helvetica", "16", "bold"))
    currentYValue = tW.LabelTemplate(curentXYFrame, text = "Y: " +
             str(round(objectRobot.x_TCP, 2)), font =("Helvetica", "16", "bold"))
    currentXValue.grid(row = 0, column = 0)
    currentYValue.grid(row = 1, column = 0)
    gTPPFrame = tW.FrameTemplate(root, width= 480, height= 270)
    gTPPFrame.grid(row = 2, column = 5, rowspan = 1, columnspan = 2)
    goToMode = tW.LabelFrameTemplate(gTPPFrame, width= 240, height= 270,
                                     text = "Idi na:")
    goToMode.grid(row = 0, column = 0, rowspan = 1, columnspan = 1)
```

```
    xLabel = tW.LabelTemplate(goToMode, font =("Helvetica", "16", "bold"),
                                        text = "x:")
    yLabel = tW.LabelTemplate(goToMode, font =("Helvetica", "16", "bold"),
                                        text = "y:")
    xEntry = tW.EntryTemplate(goToMode)
    yEntry = tW.EntryTemplate(goToMode)
    goToButton = tW.ButtonTemplate(goToMode, text = "Kreni", command =
                                lambda: goToModeFunc(root, objectRobot,
                                xEntry, yEntry, robotPosition,
                                currentXValue, currentYValue))
    xLabel.grid(row = 0, column = 0)
    yLabel.grid(row = 1, column = 0)
    xEntry.grid(row = 0, column = 1)
    yEntry.grid(row = 1, column = 1)
    goToButton.grid(row = 0, column = 2, rowspan = 2)
    preProgramedMode = tW.LabelFrameTemplate(gTPPFrame, width= 240, height= 270, t
ext = "Odaberi program:")
    preProgramedMode.grid(row = 1, column = 0, rowspan = 1, columnspan = 1)
    preProgramedTypes = tW.SpinboxTemplate(preProgramedMode,
                                        to = pP.numOfPrograms)
    preProgramedButton = tW.ButtonTemplate(preProgramedMode, text = "Pokreni",
                    command = lambda: pP.preProgrammedFunc(root, objectRobot,
                    preProgramedTypes, preProgramedButton, robotPosition, currentXV
                    alue, currentYValue))
    preProgramedTypes.grid(row = 0, column = 0, sticky = tW.S)
    preProgramedButton.grid(row = 1, column = 0, sticky = tW.S)
    buttonsFrame = tW.FrameTemplate(root, width= 240, height= 270)
    buttonsFrame.grid(row = 0, column = 7, rowspan = 3, columnspan = 1)
    homeButton = tW.ButtonTemplate(buttonsFrame, text = "Home",
                    command = lambda: homeFunc(root, objectRobot,
                    currentXValue, currentYValue, robotPosition))
    closedLoopButton = tW.ButtonTemplate(buttonsFrame, text = "Uključi motore",
                    command = lambda: closedLoopFunc(root, objectRobot,
                    currentXValue, currentYValue, robotPosition))
    idleButton = tW.ButtonTemplate(buttonsFrame, text = "Isključi motore",
                    command = lambda: idleFunc(objectRobot))
    pickButton = tW.ButtonTemplate(buttonsFrame, text = "Izuzmi",
                    command = lambda: pickFunc(objectPneumatic))
    placeButton = tW.ButtonTemplate(buttonsFrame, text = "Odloži",
                    command = lambda: placeFunc(objectPneumatic))
    detectButton = tW.ButtonTemplate(buttonsFrame, text = "Prepoznaj",
                    command = lambda: detectFunc(root, objectCamera, objectRobot,
                    wMTxt1, sortButton, robotPosition,
                    currentXValue, currentYValue))
```

```
    sortButton = tW.ButtonTemplate(buttonsFrame, text = "Sortiraj",
                    state = 'disabled', command = lambda: sortFunc(root,
                    objectCamera, objectRobot, objectPneumatic, robotPosition,
                    wMTxt1, sortButton, currentXValue, currentYValue))
    calibrateCameraButton = tW.ButtonTemplate(buttonsFrame,
                    text = "Kalibracija kamere", command = lambda:
                    calibrateCameraFunc(root, objectRobot, objectCamera,
                    robotPosition, wMTxt1, currentXValue, currentYValue))
    calibrateCoordinateSystemButton = tW.ButtonTemplate(buttonsFrame,
                    text = "Kalibracija k.s.", command = lambda:
                    calibrateCoordinateSystemFunc(root, objectCamera,
                    objectRobot, robotPosition, wMTxt1,
                    currentXValue, currentYValue ))
    quitButton = tW.ButtonTemplate(buttonsFrame, text = "Izlaz", command = lambda:
                     quitFunc(root, objectRobot))
    homeButton.grid(row = 0, column = 0)
    closedLoopButton.grid(row = 1, column = 0)
    idleButton.grid(row = 2, column = 0)
    pickButton.grid(row = 3, column = 0)
    placeButton.grid(row = 4, column = 0)
    detectButton.grid(row = 5, column = 0)
    sortButton.grid(row = 6, column = 0)
    calibrateCameraButton.grid(row = 7, column = 0)
    calibrateCoordinateSystemButton.grid(row = 8, column = 0)
    quitButton.grid(row = 9, column = 0)
    messageFrame = tW.LabelFrameTemplate(root, width= 1920, height= 270,
                                            text = "Poruke:")
    messageFrame.grid(row = 10, column = 0, rowspan = 8, columnspan = 1,
                                            sticky = tW.S)
    wMTxt1 = tW.StringVar()
    wMTxt1.set("")
    wMMsg1 = tW.MessageTemplate(messageFrame, textvariable = wMTxt1,
                    font = ("Helvetica", "18", "bold"), width = 960)
    wMMsg1.pack()
    root.mainloop()

def sliderToVar(sliderVar, objectRobot, sliderType):
    if(sliderType == "CURRENT"):
        objectRobot.maxMotorCurrent(sliderVar.get())
    if(sliderType == "VELOCITY"):
        objectRobot.maxVelocity(sliderVar.get())
    if(sliderType == "ACCLERATION"):
        objectRobot.maxAccDel(sliderVar.get())

def chooseControle(choosenControl, objectRobot):
    if(choosenControl.get() == "Loose"):
        AXIS_POSE_GAIN = objectRobot.AXIS_POSE_GAIN_L
```

```
        AXIS_VEL_GAIN = objectRobot.AXIS_VEL_GAIN_L
        AXIS_INTEGRATOR_GAIN = objectRobot.AXIS_INTEGRATOR_GAIN_L
        objectRobot.regulatorParameters(AXIS_POSE_GAIN,
                    AXIS_VEL_GAIN, AXIS_INTEGRATOR_GAIN)
    if(choosenControl.get() == "Tight"):
        AXIS_POSE_GAIN = objectRobot.AXIS_POSE_GAIN_T
        AXIS_VEL_GAIN = objectRobot.AXIS_VEL_GAIN_T
        AXIS_INTEGRATOR_GAIN = objectRobot.AXIS_INTEGRATOR_GAIN_T
        objectRobot.regulatorParameters(AXIS_POSE_GAIN,
                    AXIS_VEL_GAIN, AXIS_INTEGRATOR_GAIN)

def followMouseMoveRobot(event, objectRoot, passRobotPosition, objectRobot, passCu
rrentXValue, passCurrentYValue):
    if(not(objectRobot.driverInClosedLoopState())):
        return
    tempX, tempY = passRobotPosition.followMouseMoveRobotFunc(event)
    if(not(objectRobot.isInWorkSpace(tempX, tempY))):
        return
    objectRobot.moveAxis(tempX, tempY)
    passRobotPosition.drawRobotPosition(objectRoot, tempX, tempY,
                    objectRobot.theta0, objectRobot.theta1)
    passCurrentXValue.config(text = "X: " + str(round(objectRobot.x_TCP, 2)))
    passCurrentYValue.config(text = "Y: " + str(round(objectRobot.y_TCP, 2)))

def typeSelection(passType, objectCamera):
    objectCamera.sortType = passType.get()

def goToModeFunc(objectRoot, objectRobot, xEntry, yEntry,
            drawRobotPositionPass, passCurrentXValue, passCurrentYValue):
    if(not(objectRobot.driverInClosedLoopState())):
        return
    if(xEntry.get() == "" or yEntry.get() == ""):
        xEntry.delete(0, 'end')
        yEntry.delete(0, 'end')
        return
    if(not(objectRobot.isInWorkSpace(float(xEntry.get()), float(yEntry.get())))):
        xEntry.delete(0, 'end')
        yEntry.delete(0, 'end')
        return
    objectRobot.moveAxis(float(xEntry.get()), float(yEntry.get()))
    drawRobotPositionPass.drawRobotPosition(objectRoot,
            float(xEntry.get()), float(yEntry.get()),
            objectRobot.theta0, objectRobot.theta1)
    passCurrentXValue.config(text = "X: " + str(round(objectRobot.x_TCP, 2)))
    passCurrentYValue.config(text = "Y: " + str(round(objectRobot.y_TCP, 2)))
    xEntry.delete(0, 'end')
    yEntry.delete(0, 'end')
```

```
def homeFunc(objectRoot, objectRobot, passCurrentXValue,
                passCurrentYValue, drawRobotPositionPass):
    if(not(objectRobot.homeFuncRobot())):
        return
    drawRobotPositionPass.drawRobotPosition(objectRoot,
            objectRobot.x_TCP, objectRobot.y_TCP,
            objectRobot.theta0, objectRobot.theta1)
    passCurrentXValue.config(text = "X: " + str(round(objectRobot.x_TCP, 2)))
    passCurrentYValue.config(text = "Y: " + str(round(objectRobot.y_TCP, 2)))

def closedLoopFunc(objectRoot, objectRobot, passCurrentXValue, passCurrentYValue,
                           drawRobotPositionPass):
    if(not(objectRobot.closedLoopFuncRobot())):
        return
    drawRobotPositionPass.drawRobotPosition(objectRoot,
            objectRobot.x_TCP, objectRobot.y_TCP,
            objectRobot.theta0, objectRobot.theta1)
    passCurrentXValue.config(text = "X: " + str(round(objectRobot.x_TCP, 2)))
    passCurrentYValue.config(text = "Y: " + str(round(objectRobot.y_TCP, 2)))

def idleFunc(objectRobot):
    objectRobot.goToIdle()

def pickFunc(objectPneumatic):
    objectPneumatic.pick()

def placeFunc(objectPneumatic):
    objectPneumatic.place()

def detectFunc(objectRoot, objectCamera, objectRobot, txtPass, buttonPass,
                   drawRobotPositionPass, passCurrentXValue, passCurrentYValue):
    moveRobotForCameraView(objectRobot, objectRoot, drawRobotPositionPass,
                   passCurrentXValue, passCurrentYValue)
    txtPass.set("Pritisnite 'a' za prihvaćanje detektiranih objekata,
                   'r' kako bi ponovili detekciju ili 'q' kako bi odustali.")
    objectRoot.update()
    _, frame = objectCamera.cap.read()
    dSFrame = objectCamera.detectFuncCamera(frame)
    compareValue = objectCamera.repeatingFunc(dSFrame)
    if(compareValue == 'a'):
        txtPass.set("Pritisnite Sortiraj za pokretanje sortiranja.")
        buttonPass.configure(state = 'normal')
        objectRoot.update()
        return
    if(compareValue == 'r'):
        detectFunc(objectRoot, objectCamera, objectRobot, txtPass, buttonPass,
                   drawRobotPositionPass, passCurrentXValue, passCurrentYValue)
```

```
    if(compareValue == 'q'):
        objectCamera.detectedShapes = []
        txtPass.set("")
        objectRoot.update()
        return

def sortFunc(objectRoot, objectCamera, objectRobot, objectPneumatic, drawRobotPosi
tionPass, txtPass, buttonPass, passCurrentXValue, passCurrentYValue):
    objectRobot.goToClosedLoop()
    if(objectCamera.sortType == "none"):
        txtPass.set("Prvo odaberite način sortiranja.")
        objectRoot.update()
        return
    buttonPass.configure(state = 'disabled')
    objectRoot.update()
    if(objectCamera.sortType == "Color"):
        txtPass.set("Predmeti se sortiraju po boji.")
        objectRoot.update()
        sortedBlue, sortedGreen, sortedRed = objectCamera.sortByColor()
        sortSequenceColor(objectRoot, objectRobot, objectPneumatic,
                    drawRobotPositionPass, passCurrentXValue, passCurrentYValue,
                    sortedBlue, sortedGreen, sortedRed)
    if(objectCamera.sortType == "Shape"):
        txtPass.set("Predmeti se sortiraju po obliku.")
        objectRoot.update()
        sortedTriangle, sortedSquare, sortedRectangle, sortedPentagon,
                    sortedHexagon, sortedCircle = objectCamera.sortByShape()
        sortSequenceShape(objectRoot, objectRobot, objectPneumatic,
             drawRobotPositionPass, passCurrentXValue, passCurrentYValue,
             sortedTriangle, sortedSquare, sortedRectangle, sortedPentagon,
             sortedHexagon, sortedCircle)
    homeFunc(objectRoot, objectRobot, passCurrentXValue, passCurrentYValue,
                          drawRobotPositionPass)
    objectRobot.goToIdle()

def sortSequenceColor(objectRoot, objectRobot, objectPneumatic, drawRobotPositionP
ass, passCurrentXValue, passCurrentYValue, passSortedBlue, passSortedGreen, passSo
rtedRed):
    blueGroupX, blueGroupY = 240, 45
    greenGroupX, greenGroupY = 240, 155
    redGroupX, redGroupY = -240,155
    for i in passSortedBlue:
        sortedX, sortedY = i[2][0], i[2][1]
        sortSequenceMoveRobot(objectRoot, objectRobot, objectPneumatic,
             drawRobotPositionPass, passCurrentXValue, passCurrentYValue,
             sortedX, sortedY, blueGroupX, blueGroupY)
```

```
    for i in passSortedGreen:
        sortedX, sortedY = i[2][0], i[2][1]
        sortSequenceMoveRobot(objectRoot, objectRobot, objectPneumatic,
             drawRobotPositionPass, passCurrentXValue, passCurrentYValue,
             sortedX, sortedY, greenGroupX, greenGroupY)
    for i in passSortedRed:
        sortedX, sortedY = i[2][0], i[2][1]
        sortSequenceMoveRobot(objectRoot, objectRobot, objectPneumatic,
              drawRobotPositionPass, passCurrentXValue, passCurrentYValue,
             sortedX, sortedY, redGroupX, redGroupY)

def sortSequenceShape(objectRoot, objectRobot, objectPneumatic, drawRobotPositionP
ass, passCurrentXValue, passCurrentYValue, passSortedTriangle, passSortedSquare, p
assSortedRectangle, passSortedPentagon, passSortedHexagon, passSortedCircle):
    triangleGroupX, trangleGroupY = -220, 50
    squareGroupX, squareGroupY = -220, 150
    rectangleGroupX, rectangleGroupY = -220, 250
    pentagonGroupX, pentagonGroupY = 220, 50
    hexagonGroupX, hexagonGroupY = 220, 150
    circleGroupX, circleGroupY = 220, 250
    for i in passSortedTriangle:
        sortedX, sortedY = i[2][0], i[2][1]
        sortSequenceMoveRobot(objectRoot, objectRobot, objectPneumatic,
             drawRobotPositionPass, passCurrentXValue, passCurrentYValue,
             sortedX, sortedY, triangleGroupX, trangleGroupY)
    for i in passSortedSquare:
        sortedX, sortedY = i[2][0], i[2][1]
        sortSequenceMoveRobot(objectRoot, objectRobot, objectPneumatic,
             drawRobotPositionPass, passCurrentXValue, passCurrentYValue,
             sortedX, sortedY, squareGroupX, squareGroupY)
    for i in passSortedRectangle:
        sortedX, sortedY = i[2][0], i[2][1]
        sortSequenceMoveRobot(objectRoot, objectRobot, objectPneumatic,
             drawRobotPositionPass, passCurrentXValue, passCurrentYValue,
             sortedX, sortedY, rectangleGroupX, rectangleGroupY)
    for i in passSortedPentagon:
        sortedX, sortedY = i[2][0], i[2][1]
        sortSequenceMoveRobot(objectRoot, objectRobot, objectPneumatic,
             drawRobotPositionPass, passCurrentXValue, passCurrentYValue,
             sortedX, sortedY, pentagonGroupX, pentagonGroupY)
    for i in passSortedHexagon:
        sortedX, sortedY = i[2][0], i[2][1]
        sortSequenceMoveRobot(objectRoot, objectRobot, objectPneumatic,
             drawRobotPositionPass, passCurrentXValue, passCurrentYValue,
             sortedX, sortedY, hexagonGroupX, hexagonGroupY)
```

```
    for i in passSortedCircle:
        sortedX, sortedY = i[2][0], i[2][1]
        sortSequenceMoveRobot(objectRoot, objectRobot, objectPneumatic,
             drawRobotPositionPass, passCurrentXValue, passCurrentYValue,
             sortedX, sortedY, circleGroupX, circleGroupY)

def sortSequenceMoveRobot(objectRoot, objectRobot, objectPneumatic,
                  drawRobotPositionPass, passCurrentXValue, passCurrentYValue,
                  passSortedX, passSortedY, passGroupX, passGroupY):
    objectRobot.goToClosedLoop()
    objectRobot.moveAxis(passSortedX, passSortedY)
    drawRobotPositionPass.drawRobotPosition(objectRoot, passSortedX, passSortedY,
                         objectRobot.theta0, objectRobot.theta1)
    passCurrentXValue.config(text = "X: " + str(round(objectRobot.x_TCP, 2)))
    passCurrentYValue.config(text = "Y: " + str(round(objectRobot.y_TCP, 2)))
    while True:
        objectRoot.update()
        if(objectRobot.atPosition()):
            break
    objectRoot.update()
    objectPneumatic.pick()
    objectRobot.moveAxis(passGroupX, passGroupY)
    drawRobotPositionPass.drawRobotPosition(objectRoot, passGroupX, passGroupY,
                        objectRobot.theta0, objectRobot.theta1)
    passCurrentXValue.config(text = "X: " + str(round(objectRobot.x_TCP, 2)))
    passCurrentYValue.config(text = "Y: " + str(round(objectRobot.y_TCP, 2)))
    while(True):
        objectRoot.update()
        if(objectRobot.atPosition()):
            break
    objectRoot.update()
    objectPneumatic.place()

def calibrateCameraFunc(objectRoot, objectRobot, objectCamera,
      drawRobotPositionPass, txtPass, passCurrentXValue, passCurrentYValue):
    moveRobotForCameraView(objectRobot, objectRoot, drawRobotPositionPass,
                        passCurrentXValue, passCurrentYValue)
    objectRoot.bind('a', lambda e: objectCamera.calibrateCameraTakeImage())
    objectRoot.bind('f', lambda e: objectCamera.calibrateCameraStartCalibration())
    objectRoot.bind('q', lambda e: objectCamera.calibrateCameraQuitCalibration())
    txtPass.set("Kalibracija kamere pokrenuta. Uslikajte minimalno 25 slika.
             Pritisnite 'a' za slikanje, 'f' za kraj i  'q' kako bi odustali.")
    objectRoot.update()
```

```
    while True:
        if(objectCamera.takeImage):
            txtPass.set("Kalibracija kamere pokrenuta. Uslikajte minimalno
             25 slika. Pritisnite 'a' za slikanje, 'f' za kraj i  'q' kako bi
             odustali." + "\nslika_" + str(objectCamera.numOfSnaps) + ".JPG")
            objectRoot.update()
            objectCamera.takeImage = False
        if(objectCamera.startCalibration):
            if(objectCamera.numOfSnaps > objectCamera.minNumOfSnaps):
                objectRoot.unbind('a')
                objectRoot.unbind('f')
                objectRoot.unbind('q')
                objectCamera.startCalibration = False
                txtPass.set("")
                objectRoot.update()
                calibrateCameraSequence(objectRoot, objectCamera, txtPass)
                objectCamera.numOfSnaps = 0
                break
            else:
                txtPass.set("Potrebno je uslikati još: "
                   + str(objectCamera.minNumOfSnaps - objectCamera.numOfSnaps)
                   + " slika.")
                objectRoot.update()
        if(objectCamera.quitCalibration):
            txtPass.set("Kalibracija kamere prekinuta")
            objectRoot.update()
            objectCamera.quitCalibration = False
            objectRoot.unbind('a')
            objectRoot.unbind('f')
            objectRoot.unbind('q')
            break
        objectRoot.update()

def calibrateCameraSequence(objectRoot, objectCamera, txtPass):
    txtPass.set("Pritisnite 'a' za prihvaćanje slike, 'r' kako bi odbili
             sliku ili 'q' kako bi odustali od kalibracije.")
    objectRoot.update()
    retValue = objectCamera.calibrateCamera()
    if(retValue == True):
        txtPass.set("Kalibracija je gotova i parametri kalibracije su ispisani.")
        objectRoot.update()
    if(retValue == False):
        txtPass.set("Kalibracija je prekinuta.")
        objectRoot.update()

def calibrateCoordinateSystemFunc(objectRoot, objectCamera, objectRobot,
      drawRobotPositionPass, txtPass, passCurrentXValue, passCurrentYValue):
```

```
    moveRobotForCameraView(objectRobot, objectRoot, drawRobotPositionPass,
                   passCurrentXValue, passCurrentYValue)
    txtPass.set("Pritisnite 'a' za prihvaćanje detektiranog markera, 'r' kako
                   bi ponovili detekciju ili 'q' kako bi odustali.")
    objectRoot.update()
    frame_markers, tempCorners, tempIds = objectCamera.calibrateCSDetectMarkers()
    comapreValue = objectCamera.repeatingFunc(frame_markers)
    if(comapreValue == 'a'):
        if(tempIds is None):
            txtPass.set("Kalibracija nije uspjela, pokušajte ponovno.")
            objectRoot.update()
            return
        txtPass.set("Kalibracija je gotova i parametri kalibracije su ispisani.")
        objectCamera.calibrateCSSequence(tempCorners)
        objectRoot.update()
        return
    if(comapreValue == 'r'):
        calibrateCoordinateSystemFunc(objectRoot, objectCamera, objectRobot,
             drawRobotPositionPass, txtPass, passCurrentXValue, passCurrentYValue)
    if(comapreValue == 'q'):
        txtPass.set("")
        objectRoot.update()
        return

def quitFunc(objectRoot, objectRobot):
    objectRobot.goToIdle()
    objectRoot.destroy()

def moveRobotForCameraView(objectRobot, objectRoot, drawRobotPositionPass,
              passCurrentXValue, passCurrentYValue):
    objectRobot.closedLoopFunc()
    movedX, movedY = 310, 85
    objectRobot.moveAxis(movedX, movedY)
    drawRobotPositionPass.drawRobotPosition(objectRoot, movedX, movedY,
                                objectRobot.theta0, objectRobot.theta1)
    passCurrentXValue.config(text = "X: " + str(round(objectRobot.x_TCP, 2)))
    passCurrentYValue.config(text = "Y: " + str(round(objectRobot.y_TCP, 2)))
    while(True):
        objectRoot.update()
        if(objectRobot.atPosition()):
            break
    objectRobot.goToIdle()
```

preProgrammed.py:

```
import time

program1 = [(0, 268), (0, 290), (-150, 290), (-75, 290),(-150, 220),
            (-150, 180), (-150, 100), (-75, 100), (0, 100), (75, 100),
            (150, 100), (150, 180), (150, 220), (150, 290), (75, 290),
            (0, 290), (0, 268)]
program2 = [(0, 268), (0, 290), (0, 150), (0, 290),(0, 150), (0, 290),
             (0, 150), (0, 290), (0, 150), (0, 268)]
numOfPrograms = 2

def preProgrammedFunc(objectRoot, objectRobot, preProgramedTypesPass, buttonPass,
drawRobotPositionPass, passCurrentXValue, passCurrentYValue):
    chosenPPType = preProgramedTypesPass.get()
    buttonPass.configure(state='disable')
    choosenProgram = []
    if(chosenPPType == "1"):
        choosenProgram = program1
    if(chosenPPType == "2"):
        choosenProgram = program2
    for i in choosenProgram:
        objectRobot.moveAxis(i[0], i[1])
        drawRobotPositionPass.drawRobotPosition(objectRoot, objectRobot,
                                                i[0], i[1])
        passCurrentXValue.config(text =  "X: " + str(round(objectRobot.x_TCP, 2)))
        passCurrentYValue.config(text =  "Y: " + str(round(objectRobot.y_TCP, 2)))
        while(True):
            if(objectRobot.atPosition()):
                break
    buttonPass.configure(state='normal')
```

driver.py:

```
from odrive.enums import *
import odrive
import time

class Driver:
    AXIS_CURRENT_LIM = 15
    AXIS_VEL_LIMIT = 100000
    AXIS_CALIBRATION_CURRENT = 10
    AXIS_POLE_PAIRS = 7
    AXIS_MOTOR_TYPE = MOTOR_TYPE_HIGH_CURRENT
    AXIS_CPR = 8192
    AXIS_ENCODER_MODE = ENCODER_MODE_INCREMENTAL
    ODRV0_BRAKE_RESISTANCE = 0.5
    AXIS_VEL_LIMIT_TOLERANCE = 0
```

```
    AXIS_POSE_GAIN_L = 20
    AXIS_VEL_GAIN_L = 0.0005000000237487257
    AXIS_INTEGRATOR_GAIN_L = 0.0010000000474974513
    AXIS_POSE_GAIN_T = 24
    AXIS_VEL_GAIN_T = 0.0040786536410450935
    AXIS_INTEGRATOR_GAIN_T = 0.00020393267914187163
    AXIS_CALIBRATION_LOCKIN_VEL = 40
    AXIS_CALIBRATION_LOCKIN_RAMP_DISTANCE = -3.1415927410125732
    AXIS_CALIBRATION_LOCKIN_ACCEL = 20
    AXIS_TRAP_CONFIG_VEL_LIMIT = 25000.0
    AXIS_TRAP_CONFIG_ACCEL_LIMIT = 10000.0
    AXIS_TRAP_CONFIG_DECEL_LIMIT = 10000.0
    def __init__(self, passObjectOdrv):
        self.objectOdrv = passObjectOdrv
    def __del__(self):
        self.sendStartParameters()
    def sendStartParameters(self):
        #Struja
        self.objectOdrv.axis0.motor.config.current_lim = self.AXIS_CURRENT_LIM
        self.objectOdrv.axis1.motor.config.current_lim = self.AXIS_CURRENT_LIM
        time.sleep(0.1)
        #Brzina
        self.objectOdrv.axis0.controller.config.vel_limit = self.AXIS_VEL_LIMIT
        self.objectOdrv.axis1.controller.config.vel_limit = self.AXIS_VEL_LIMIT
        time.sleep(0.1)
        #Struja pri kalibraciji
        self.objectOdrv.axis0.motor.config.calibration_current =
                                    self.AXIS_CALIBRATION_CURRENT
        self.objectOdrv.axis1.motor.config.calibration_current =
                                    self.AXIS_CALIBRATION_CURRENT
        time.sleep(0.1)
        #Broj polova motora
        self.objectOdrv.axis0.motor.config.pole_pairs = self.AXIS_POLE_PAIRS
        self.objectOdrv.axis1.motor.config.pole_pairs = self.AXIS_POLE_PAIRS
        time.sleep(0.1)
        #Vrsta motora
        self.objectOdrv.axis0.motor.config.motor_type = self.AXIS_MOTOR_TYPE
        self.objectOdrv.axis1.motor.config.motor_type = self.AXIS_MOTOR_TYPE
        time.sleep(0.1)
        #CPR enkodera
        self.objectOdrv.axis0.encoder.config.cpr = self.AXIS_CPR
        self.objectOdrv.axis1.encoder.config.cpr = self.AXIS_CPR
        time.sleep(0.1)
        #Vrsta enkodera
        self.objectOdrv.axis0.encoder.config.mode = self.AXIS_ENCODER_MODE
        self.objectOdrv.axis0.encoder.config.mode = self.AXIS_ENCODER_MODE
        time.sleep(0.1)
```

```
        #Otpornik za regenerativno kocenje
        self.objectOdrv.config.brake_resistance = self.ODRV0_BRAKE_RESISTANCE
        time.sleep(0.1)
        #Velocity limit tolerance
        self.objectOdrv.axis0.controller.config.vel_limit_tolerance =
                                            self.AXIS_VEL_LIMIT_TOLERANCE
        self.objectOdrv.axis1.controller.config.vel_limit_tolerance =
                                            self.AXIS_VEL_LIMIT_TOLERANCE
        time.sleep(0.1)
        #Parametri regulatora
        self.objectOdrv.axis0.controller.config.pos_gain = self.AXIS_POSE_GAIN_T
        self.objectOdrv.axis0.controller.config.vel_gain = self.AXIS_VEL_GAIN_T
        self.objectOdrv.axis0.controller.config.vel_integrator_gain =
                                                self.AXIS_INTEGRATOR_GAIN_T
        time.sleep(0.1)
        self.objectOdrv.axis1.controller.config.pos_gain = self.AXIS_POSE_GAIN_T
        self.objectOdrv.axis1.controller.config.vel_gain = self.AXIS_VEL_GAIN_T
        self.objectOdrv.axis1.controller.config.vel_integrator_gain =
                                                self.AXIS_INTEGRATOR_GAIN_T
        time.sleep(0.1)
        #Z-puls parametri
        self.objectOdrv.axis0.config.calibration_lockin.vel =
                                            self.AXIS_CALIBRATION_LOCKIN_VEL
        self.objectOdrv.axis0.config.calibration_lockin.ramp_distance =
                                    self.AXIS_CALIBRATION_LOCKIN_RAMP_DISTANCE
        self.objectOdrv.axis0.config.calibration_lockin.accel =
                                            self.AXIS_CALIBRATION_LOCKIN_ACCEL
        time.sleep(0.1)
        self.objectOdrv.axis1.config.calibration_lockin.vel =
                                            self.AXIS_CALIBRATION_LOCKIN_VEL
        self.objectOdrv.axis1.config.calibration_lockin.ramp_distance =
                                    self.AXIS_CALIBRATION_LOCKIN_RAMP_DISTANCE
        self.objectOdrv.axis1.config.calibration_lockin.accel =
                                            self.AXIS_CALIBRATION_LOCKIN_ACCEL
        time.sleep(0.1)
        #Pracenje trajektorije parametri
        self.objectOdrv.axis0.trap_traj.config.vel_limit =
                                            self.AXIS_TRAP_CONFIG_VEL_LIMIT
        self.objectOdrv.axis1.trap_traj.config.vel_limit =
                                            self.AXIS_TRAP_CONFIG_VEL_LIMIT
        time.sleep(0.1)
        self.objectOdrv.axis0.trap_traj.config.accel_limit =
                                            self.AXIS_TRAP_CONFIG_ACCEL_LIMIT
        self.objectOdrv.axis1.trap_traj.config.accel_limit =
                                            self.AXIS_TRAP_CONFIG_ACCEL_LIMIT
        time.sleep(0.1)
```

```
        self.objectOdrv.axis0.trap_traj.config.decel_limit =
                                            self.AXIS_TRAP_CONFIG_DECEL_LIMIT
        self.objectOdrv.axis1.trap_traj.config.decel_limit =
                                            self.AXIS_TRAP_CONFIG_DECEL_LIMIT
        time.sleep(0.1)
    def encoderPosition(self):
        pos0 = self.objectOdrv.axis0.encoder.pos_estimate
        pos1 = self.objectOdrv.axis1.encoder.pos_estimate
        return pos0, pos1
    def driverInClosedLoopState(self):
        if(self.objectOdrv.axis0.current_state == AXIS_STATE_CLOSED_LOOP_CONTROL
       and self.objectOdrv.axis1.current_state == AXIS_STATE_CLOSED_LOOP_CONTROL):
            return True
        else:
            return False
    def goToIdle(self):
        self.objectOdrv.axis0.requested_state = AXIS_STATE_IDLE
        self.objectOdrv.axis1.requested_state = AXIS_STATE_IDLE
    def goToClosedLoop(self):
        self.objectOdrv.axis0.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
        self.objectOdrv.axis1.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
    def moveTraj(self, newPos0, newPos1):
        if(self.objectOdrv.axis0.current_state == AXIS_STATE_CLOSED_LOOP_CONTROL
            and self.objectOdrv.axis0.current_state ==
                                AXIS_STATE_CLOSED_LOOP_CONTROL):
            self.objectOdrv.axis0.controller.move_to_pos(newPos0)
            self.objectOdrv.axis1.controller.move_to_pos(newPos1)
            return True
        else:
            return False
    def movePos(self, newPos0, newPos1):
        if(self.objectOdrv.axis0.current_state == AXIS_STATE_CLOSED_LOOP_CONTROL
            and self.objectOdrv.axis0.current_state ==
                                AXIS_STATE_CLOSED_LOOP_CONTROL):
            self.objectOdrv.axis0.controller.pos_setpoint = newPos0
            self.objectOdrv.axis1.controller.pos_setpoint = newPos1
            return True
        else:
            return False
    def maxMotorCurrent(self, passCurrent):
        self.objectOdrv.axis0.motor.config.current_lim = passCurrent
        self.objectOdrv.axis1.motor.config.current_lim = passCurrent
    def maxVelocity(self, passVel):
        self.objectOdrv.axis0.trap_traj.config.vel_limit = passVel
        self.objectOdrv.axis1.trap_traj.config.vel_limit = passVel
```

```
    def maxAccDel(self, passAccDel):
        self.objectOdrv.axis0.trap_traj.config.accel_limit = passAccDel
        self.objectOdrv.axis1.trap_traj.config.accel_limit = passAccDel
        self.objectOdrv.axis0.trap_traj.config.decel_limit = passAccDel
        self.objectOdrv.axis1.trap_traj.config.decel_limit = passAccDel
    def regulatorParameters(self, passAXIS_POSE_GAIN, passAXIS_VEL_GAIN,
                                            passAXIS_INTEGRATOR_GAIN):
        self.objectOdrv.axis0.controller.config.pos_gain = passAXIS_POSE_GAIN
        self.objectOdrv.axis0.controller.config.vel_gain = passAXIS_VEL_GAIN
        self.objectOdrv.axis0.controller.config.vel_integrator_gain =
                                                        passAXIS_INTEGRATOR_GAIN
        self.objectOdrv.axis1.controller.config.pos_gain = passAXIS_POSE_GAIN
        self.objectOdrv.axis1.controller.config.vel_gain = passAXIS_VEL_GAIN
        self.objectOdrv.axis1.controller.config.vel_integrator_gain =
                                                        passAXIS_INTEGRATOR_GAIN
    def zPulseSearch(self):
        self.objectOdrv.axis1.requested_state = AXIS_STATE_ENCODER_INDEX_SEARCH
        while self.objectOdrv.axis1.current_state != AXIS_STATE_IDLE:
            time.sleep(0.1)
        self.objectOdrv.axis1.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
        self.objectOdrv.axis1.controller.move_to_pos(0)
        time.sleep(0.25)
        self.objectOdrv.axis0.requested_state = AXIS_STATE_ENCODER_INDEX_SEARCH
        while self.objectOdrv.axis0.current_state != AXIS_STATE_IDLE:
            time.sleep(0.1)
        self.objectOdrv.axis0.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
        self.objectOdrv.axis0.controller.move_to_pos(0)
        time.sleep(1)
        self.objectOdrv.axis1.requested_state = AXIS_STATE_IDLE
        self.objectOdrv.axis1.requested_state = AXIS_STATE_ENCODER_INDEX_SEARCH
        while self.objectOdrv.axis1.current_state != AXIS_STATE_IDLE:
            time.sleep(0.1)
        self.objectOdrv.axis1.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
        self.objectOdrv.axis1.controller.move_to_pos(0)
        time.sleep(0.25)
        self.objectOdrv.axis0.requested_state = AXIS_STATE_IDLE
        self.objectOdrv.axis0.requested_state = AXIS_STATE_ENCODER_INDEX_SEARCH
        while self.objectOdrv.axis0.current_state != AXIS_STATE_IDLE:
            time.sleep(0.1)
        self.objectOdrv.axis0.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
        self.objectOdrv.axis0.controller.move_to_pos(0)
        time.sleep(0.25)
        self.objectOdrv.axis0.requested_state = AXIS_STATE_IDLE
        self.objectOdrv.axis1.requested_state = AXIS_STATE_IDLE
    def motorCalibration(self):
        self.objectOdrv.axis0.requested_state = AXIS_STATE_MOTOR_CALIBRATION
```

```
        while self.objectOdrv.axis0.current_state != AXIS_STATE_IDLE:
            time.sleep(0.1)
        self.objectOdrv.axis1.requested_state = AXIS_STATE_MOTOR_CALIBRATION
        while self.objectOdrv.axis1.current_state != AXIS_STATE_IDLE:
            time.sleep(0.1)
        self.objectOdrv.axis0.motor.config.pre_calibrated = True
        self.objectOdrv.axis1.motor.config.pre_calibrated = True
    def encoderCalibration(self):
        self.objectOdrv.axis0.encoder.config.use_index = True
        self.objectOdrv.axis1.encoder.config.use_index = True
        #pronađi z-puls
        i = 0
        while i < 2:
            self.objectOdrv.axis0.requested_state =
                                AXIS_STATE_ENCODER_INDEX_SEARCH
            while self.objectOdrv.axis0.current_state != AXIS_STATE_IDLE:
                time.sleep(0.1)
            self.objectOdrv.axis1.requested_state =
                                AXIS_STATE_ENCODER_INDEX_SEARCH
            while self.objectOdrv.axis1.current_state != AXIS_STATE_IDLE:
                time.sleep(0.1)
            i += 1
        #kalibracija enkodera
        self.objectOdrv.axis0.requested_state =
                                AXIS_STATE_ENCODER_OFFSET_CALIBRATION
        while self.objectOdrv.axis0.current_state != AXIS_STATE_IDLE:
            time.sleep(0.1)
        self.objectOdrv.axis1.requested_state =
                                AXIS_STATE_ENCODER_OFFSET_CALIBRATION
        while self.objectOdrv.axis1.current_state != AXIS_STATE_IDLE:
            time.sleep(0.1)
        self.objectOdrv.axis0.encoder.config.pre_calibrated = True
        self.objectOdrv.axis1.encoder.config.pre_calibrated = True
    def finishCalibration(self):
        self.objectOdrv.axis0.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
        self.objectOdrv.axis1.requested_state = AXIS_STATE_CLOSED_LOOP_CONTROL
        self.objectOdrv.axis0.controller.move_to_pos(0)
        self.objectOdrv.axis1.controller.move_to_pos(0)
        self.objectOdrv.save_configuration()
        self.objectOdrv.axis0.requested_state = AXIS_STATE_IDLE
        self.objectOdrv.axis1.requested_state = AXIS_STATE_IDLE
```

robot.py:

```
import driver
from numpy import pi, sqrt, sin, cos, arccos, arctan2
import time

class Robot(driver.Driver):
    x_TCP = 0.0
    y_TCP = 0.0
    absCnt0 = 0.0
    absCnt1 = 0.0
    theta0 = 0.0
    theta1 = 0.0
    l0 = 42.5
    l1 = 160
    l2 = 220
    deviation = 12.5
    def __init__(self, passObjectOdrv):
        super().__init__(passObjectOdrv)
    def atPosition(self):
        encoder0, encoder1 = self.encoderPosition()
        if(((self.absCnt0 - self.deviation) < encoder0 <
                    (self.absCnt0 + self.deviation) and
                    (self.absCnt1 - self.deviation) < encoder1 <
                    (self.absCnt1 + self.deviation))):
            return True
        else:
            return False
    def isInWorkSpace(self, passX, passY):
        if(not(((passX - 42.5)**2 + (passY - 0)**2) < 380**2)):
            return False
        if(not(((passX + 42.5)**2 + (passY - 0)**2) < 380**2)):
            return False
        if((-105 < passX < 105 ) and (0 < passY < 63)):
            return False
        if(not((-355 < passX < 355 ) and (0 < passY < 382))):
            return False
        return True
    def forwardKinematics(self, passT0, passT1):
        x0 = self.l0 + self.l1 * cos(passT0)
        y0 = self.l1 * sin(passT0)
        x1 = -self.l0 + self.l1 * cos(passT1)
        y1 = self.l1 * sin(passT1)
        d = sqrt((x1-x0)*(x1-x0) + (y1-y0)*(y1-y0))
        if d > 2 * self.l2 :
            return None
        if d < 0:
            return None
```

```
        if d == 0:
            return None
        else:
            a=(d**2)/(2*d)
            h=sqrt(self.l2**2-a**2)
            x2=x0+a*(x1-x0)/d
            y2=y0+a*(y1-y0)/d
            x3=x2+h*(y1-y0)/d
            y3=y2-h*(x1-x0)/d
            if(not(self.isInWorkSpace(x3, y3))):
                return None
            else:
                return(x3,y3)
    def inverseKinematics(self, passX, passY):
        if(not(self.isInWorkSpace(passX, passY))):
            return
        beta0 = arctan2( passY, (self.l0 - passX) )
        beta1 = arctan2( passY, (self.l0 + passX) )
        alpha0_calc = (self.l1**2 + ( (self.l0 - passX)**2 + passY**2 )
                        - self.l2**2) / (2*self.l1*sqrt( (self.l0 - passX)**2
                        + passY**2 ))
        alpha1_calc = (self.l1**2 + ( (self.l0 + passX)**2 + passY**2 )
                        - self.l2**2) / (2*self.l1*sqrt( (self.l0 + passX)**2
                        + passY**2 ))
        alpha1 = arccos(alpha1_calc)
        alpha0 = arccos(alpha0_calc)
        if(passX >= 0 and passY < 0):
            returnTheta0 = (pi - beta0 - alpha0) - 2* pi
            returnTheta1 = beta1 + alpha1
        elif(passX < 0 and passY < 0):
            returnTheta0 = pi - beta0 - alpha0
            returnTheta1 = 2*pi + (beta1 + alpha1)
        else:
            returnTheta0 = pi - beta0 - alpha0
            returnTheta1 = beta1 + alpha1
        return(returnTheta0, returnTheta1)
    def thetaToCnt(self, passTheta0, passTheta1):
        tempCnt0 = (passTheta0 - pi/4) * self.AXIS_CPR/(2*pi)
        tempCnt1 = (passTheta1 - (3*pi)/4) * self.AXIS_CPR/(2*pi)
        return tempCnt0, tempCnt1
    def cntToTheta(self, passCnt0, passCnt1):
        tempTheta0 = passCnt0 * (2*pi)/self.AXIS_CPR + pi/4
        tempTheta1 = passCnt1 * (2*pi)/self.AXIS_CPR + (3*pi)/4
        return tempTheta0, tempTheta1
    def moveAxis(self, passX, passY):
        if(not(self.driverInClosedLoopState())):
            return
```

```
        if(not(self.isInWorkSpace(passX, passY))):
            return
        self.x_TCP, self.y_TCP = passX, passY
        self.theta0, self.theta1 = self.inverseKinematics(self.x_TCP, self.y_TCP)
        self.absCnt0, self.absCnt1 = self.thetaToCnt(self.theta0, self.theta1)
        self.moveTraj(self.absCnt0, self.absCnt1)
    def closedLoopFuncRobot(self):
        if(self.driverInClosedLoopState()):
            return False
        self.goToClosedLoop()
        self.absCnt0, self.absCnt1 = self.objectOdrv.axis0.encoder.pos_estimate,
                                     self.objectOdrv.axis1.encoder.pos_estimate
        self.theta0, self.theta1 = self.cntToTheta(self.absCnt0, self.absCnt1)
        self.x_TCP, self.y_TCP = self.forwardKinematics(self.theta0, self.theta1)
        return True
    def homeFuncRobot(self):
        if(not(self.driverInClosedLoopState())):
            return False
        self.absCnt0, self.absCnt1 = 0, 0
        self.moveTraj(self.absCnt0, self.absCnt1)
        self.theta0, self.theta1 = self.cntToTheta(self.absCnt0, self.absCnt1)
        self.x_TCP, self.y_TCP = self.forwardKinematics(self.theta0, self.theta1)
        while True:
            if(self.atPosition()):
                return True
```

prenumatic.py:

```
import Jetson.GPIO as GPIO
import time

class Pneumatic:
    inPickedState = 0
    cylinder_pin = 12
    vacuum_pin = 18
    def __init__(self):
        GPIO.setmode(GPIO.BOARD)
        GPIO.setup(self.cylinder_pin, GPIO.OUT)
        GPIO.setup(self.vacuum_pin, GPIO.OUT)
        GPIO.output(self.cylinder_pin, GPIO.LOW)
        GPIO.output(self.vacuum_pin, GPIO.LOW)
    def pick(self):
        if(self.inPickedState == 1):
            return
        GPIO.output(self.cylinder_pin, GPIO.HIGH)
        time.sleep(0.5)
        GPIO.output(self.vacuum_pin, GPIO.HIGH)
```

```
        time.sleep(1)
        GPIO.output(self.cylinder_pin, GPIO.LOW)
        time.sleep(1)
        self.inPickedState = 1
    def place(self):
        if(self.inPickedState == 0):
            return
        GPIO.output(self.cylinder_pin, GPIO.HIGH)
        time.sleep(1)
        GPIO.output(self.vacuum_pin, GPIO.LOW)
        time.sleep(0.25)
        GPIO.output(self.cylinder_pin, GPIO.LOW)
        self.inPickedState = 0
```

camera.py:

```
from PIL import ImageTk, Image
from numpy.linalg import inv
from cv2 import aruco
import numpy as np
import cv2
import os
import glob

class Camera:
    dist = np.array([1.13040870e-01, 4.83643123e-01, -5.64991267e-03,
                     -1.67241759e-03, -2.88852146e+00], np.float32)
    mtx = np.array([[975.73130976, 0.0, 476.31978043],
                    [0.0, 976.93157338, 274.124011],
                    [0.0, 0.0, 1.0]], np.float32)
    rvec = np.array([-3.15036483, 0.05164624, 0.17272349], np.float32)
    tvec =  np.array([[2.91749029, 203.36206627, 625.33317957]], np.float32)
    rotM = np.array([[0.99347229, -0.03191661, -0.10951772],
                     [-0.03344112, -0.9993673, -0.01211135],
                     [-0.10906188, 0.01569469, -0.99391105]], np.float32)
    scaleFactor = tvec[0][2]
    imagePoints = np.array([(453., 456.),(500.,  456.),(500., 503.),
                            (453., 503.)], np.float32)
    objectPoints = np.array([(-15, 87.5, 0), (15, 87.5, 0), (15, 57.5, 0),
                            (-15, 57.5, 0)], np.float32)
    sortType = "none"
    detectedShapes = []
    takeImage = False
    startCalibration = False
    quitCalibration = False
    path = '/home/matej/Desktop/MatejBozic/5BarRobot/slikeZaKalibraciju/'
    numOfSnaps = 0
```

```
    minNumOfSnaps = 26
    def __init__(self):
        self.cap = cv2.VideoCapture(self.gstreamer_pipeline(), cv2.CAP_GSTREAMER)
    def __del__(self):
        if(self.cap.isOpened()):
            self.cap.release()
        cv2.destroyAllWindows()
    def gstreamer_pipeline(
        self,
        capture_width=1280,
        capture_height=720,
        display_width=960,
        display_height=540,
        framerate=21,
        flip_method=0,
    ):
        return (
            "nvarguscamerasrc ! "
            "video/x-raw(memory:NVMM), "
            "width=(int)%d, height=(int)%d, "
            "format=(string)NV12, framerate=(fraction)%d/1 ! "
            "nvvidconv flip-method=%d ! "
            "video/x-raw, width=(int)%d, height=(int)%d, format=(string)BGRx ! "
            "videoconvert ! "
            "video/x-raw, format=(string)BGR ! appsink"
            % (
                capture_width,
                capture_height,
                framerate,
                flip_method,
                display_width,
                display_height,
            )
        )
    def show_frame(self, passCameraLabel):
        _, frame = self.cap.read()
        cv2image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)
        img = Image.fromarray(cv2image)
        imgtk = ImageTk.PhotoImage(image=img)
        passCameraLabel.imgtk = imgtk
        passCameraLabel.configure(image=imgtk)
        passCameraLabel.after(5, lambda : self.show_frame(passCameraLabel))
    def calibrateCameraTakeImage(self):
        self.numOfSnaps += 1
        _, frame = self.cap.read()
        fileName = "slika_" + str(self.numOfSnaps) + '.JPG'
        cv2.imwrite(os.path.join(self.path, fileName), frame)
```

```
        self.takeImage = True
    def calibrateCameraStartCalibration(self):
        self.startCalibration = True
    def calibrateCameraQuitCalibration(self):
        self.quitCalibration = True
    def calibrateCamera(self):
        xNumOfSquares, yNumOfSquares = 6, 9
        criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
        objp = np.zeros((xNumOfSquares * yNumOfSquares,3), np.float32)
        objp[:,:2] = np.mgrid[0:yNumOfSquares,0:xNumOfSquares].T.reshape(-1,2)
        objpoints = []
        imgpoints = []
        images = glob.glob(self.path + '*.JPG')
        for fname in images:
            img = cv2.imread(fname)
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            ret, corners = cv2.findChessboardCorners(gray, (9,6), None)
            if ret == True:
                corners2 = cv2.cornerSubPix(gray,corners,
                          (11,11), (-1,-1), criteria)
                cv2.drawChessboardCorners(img, (9,6), corners2, ret)
                comapreValue = self.repeatingFunc(img)
                if(comapreValue == 'a'):
                    objpoints.append(objp)
                    imgpoints.append(corners)
                if(comapreValue == 'r'):
                    continue
                if(comapreValue == 'q'):
                    return False
        cv2.destroyAllWindows()
        _, self.mtx, self.dist, _, _ = cv2.calibrateCamera(objpoints, imgpoints,
                                                gray.shape[::-1], None, None)
        print("mtx:\n", self.mtx)
        print("dist:\n", self.dist)
        return True
    def calibrateCSDetectMarkers(self):
        _, frame = self.cap.read()
        aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
        parameters =  aruco.DetectorParameters_create()
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        corners, ids, _ = aruco.detectMarkers(gray, aruco_dict,
                                                parameters=parameters)
        frame_markers = aruco.drawDetectedMarkers(frame.copy(), corners, ids)
        return frame_markers, corners, ids
    def calibrateCSSequence(self, passCorners):
        self.imagePoints = passCorners[0][0]
```

```
        _, self.rvec, self.tvec = cv2.solvePnP(self.objectPoints,
                    self.imagePoints, self.mtx, self.dist, cv2.SOLVEPNP_IPPE)
        self.rotM = cv2.Rodrigues(self.rvec)[0]
        print("imagePoints\n", self.imagePoints)
        print("rvec:\n", self.rvec)
        print("tvec:\n", self.tvec)
        print("rotM:\n", self.rotM)
    def unDistort(self, frame):
        h,  w = frame.shape[:2]
        newcameramtx, roi=cv2.getOptimalNewCameraMatrix(self.mtx, self.dist,
                                                        (w,h), 1, (w,h))
        mapx,mapy = cv2.initUndistortRectifyMap(self.mtx, self.dist, None,
                                                newcameramtx,(w,h),5)
        dst = cv2.remap(frame,mapx,mapy,cv2.INTER_LINEAR)
        x,y,w,h = roi
        dst = dst[y:y+h, x:x+w]
        return dst
    def repeatingFunc(self, passFrame):
        retVal = ""
        while True:
            cv2.imshow("passFrame", passFrame)
            key = cv2.waitKey(0)
            if(key == ord('a')):
                retVal = 'a'
                break
            if(key == ord('r')):
                retVal = 'r'
                break
            if(key == ord('q')):
                retVal = 'q'
                break
        cv2.destroyAllWindows()
        return retVal
    def detectShape(self, approx):
        shape = ""
        if len(approx) == 3:
            shape = "trokut"
        elif len(approx) == 4:
            rect = cv2.minAreaRect(approx)
            box = cv2.boxPoints(rect)
            box = np.int0(box)
            w = np.sqrt((box[0][0]-box[1][0])*(box[0][0]-box[1][0]) +
                        (box[0][1]-box[1][1])*(box[0][1]-box[1][1]))
            h = np.sqrt((box[1][0]-box[2][0])*(box[1][0]-box[2][0]) +
                        (box[1][1]-box[2][1])*(box[1][1]-box[2][1]))
            if 0.9 <= w/h <= 1.1:
                shape = "kvadrat"
```

```
            else:
                shape = "pravokutnik"
        elif len(approx) == 5:
            shape = "peterokut"
        elif len(approx) == 6:
            shape = "sesterokut"
        else:
            shape = "krug"
        return shape
    def detectFuncCamera(self, frame):
        self.detectedShapes = []
        minArea = 1000
        blurredHSV = cv2.GaussianBlur(frame, (5, 5), 5)
        hsv = cv2.cvtColor(blurredHSV, cv2.COLOR_BGR2HSV)
        #plava
        lower_blue = np.array([75,30,30])
        upper_blue = np.array([130,255,255])
        maskBlue = cv2.inRange(hsv, lower_blue, upper_blue)
        cntsB = cv2.findContours(maskBlue.copy(), cv2.RETR_EXTERNAL,
                                                  cv2.CHAIN_APPROX_SIMPLE)
        #zelena
        lower_green = np.array([40, 30, 75])
        upper_green = np.array([80, 255, 255])
        maskGreen = cv2.inRange(hsv, lower_green, upper_green)
        cntsG = cv2.findContours(maskGreen.copy(), cv2.RETR_EXTERNAL,
                                                  cv2.CHAIN_APPROX_SIMPLE)
        #crvena
        lower_red1 = np.array([120, 35, 0])
        upper_red1 = np.array([180, 255, 255])
        lower_red2 = np.array([0, 25, 0])
        upper_red2 = np.array([10, 255, 255])
        maskRed = cv2.inRange(hsv, lower_red1, upper_red1) + cv2.inRange(hsv,
                                                    lower_red2, upper_red2)
        cntsR = cv2.findContours(maskRed.copy(), cv2.RETR_EXTERNAL,
                                                    cv2.CHAIN_APPROX_SIMPLE)
        for c in cntsB[0]:
            if(cv2.contourArea(c) < minArea):
                continue
            epsilon = 0.035*cv2.arcLength(c,True)
            approx = cv2.approxPolyDP(c,epsilon,True)
            shape = self.detectShape(approx)
            M = cv2.moments(approx)
            if (M["m00"] == 0):
                M["m00"]=1
            cX = int(M["m10"] / M["m00"])
            cY = int(M["m01"] / M["m00"])
            rX, rY = self.pixelCSToRobotCS(cX, cY)
```

```
            centerCo = str(rX) + ", " + str(rY)
            cv2.drawContours(frame,[approx],0,(0,0,255),2)
            cv2.circle(frame, (cX, cY), 3, (255, 255, 255), -1)
            cv2.putText(frame, centerCo,  (cX - 10, cY - 10),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
            cv2.putText(frame, shape, (cX - 15, cY + 15),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
            cv2.putText(frame, "plava", (cX - 15, cY + 30),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
            self.detectedShapes.append([shape, "blue", (rX, rY)])
        for c in cntsG[0]:
            if(cv2.contourArea(c) < minArea):
                continue
            epsilon = 0.04*cv2.arcLength(c,True)
            approx = cv2.approxPolyDP(c,epsilon,True)
            shape = self.detectShape(approx)
            M = cv2.moments(approx)
            if (M["m00"] == 0):
                M["m00"]=1
            cX = int(M["m10"] / M["m00"])
            cY = int(M["m01"] / M["m00"])
            rX, rY = self.pixelCSToRobotCS(cX, cY)
            centerCo = str(rX) + ", " + str(rY)
            cv2.drawContours(frame,[approx],0,(0,0,255),2)
            cv2.circle(frame, (cX, cY), 3, (255, 255, 255), -1)
            cv2.putText(frame, centerCo, (cX - 10, cY - 10),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
            cv2.putText(frame, shape, (cX - 15, cY + 15),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
            cv2.putText(frame, "zelena", (cX - 15, cY + 30),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
            self.detectedShapes.append([shape, "green", (rX, rY)])
        for c in cntsR[0]:
            if(cv2.contourArea(c) < minArea):
                continue
            epsilon = 0.035*cv2.arcLength(c,True)
            approx = cv2.approxPolyDP(c,epsilon,True)
            shape = self.detectShape(approx)
            M = cv2.moments(approx)
            if (M["m00"] == 0):
                M["m00"]=1
            cX = int(M["m10"] / M["m00"])
            cY = int(M["m01"] / M["m00"])
            rX, rY = self.pixelCSToRobotCS(cX, cY)
            centerCo = str(rX) + ", " + str(rY)
            cv2.drawContours(frame,[approx],0,(0,0,255),2)
            cv2.circle(frame, (cX, cY), 3, (255, 255, 255), -1)
```

```
            cv2.putText(frame, centerCo, (cX - 10, cY - 10),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
            cv2.putText(frame, shape, (cX - 15, cY + 15),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
            cv2.putText(frame, "crvena", (cX - 15, cY + 30),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
            self.detectedShapes.append([shape, "red", (rX, rY)])
        return frame
    def pixelCSToRobotCS(self, u, v):
        uV = np.array([[u, v, 1]], np.float32)
        uV = self.scaleFactor * uV.T
        xyz = inv(self.mtx).dot(uV)
        xyz_c = np.subtract(xyz, self.tvec.T)
        invRotM = inv(self.rotM)
        XYZ = invRotM.dot(xyz_c)
        return int(XYZ[0]), int(XYZ[1])
    def sortByColor(self):
        blue = []
        green = []
        red = []
        for i in self.detectedShapes:
            if(i[1] == "blue"):
                blue.append(i)
            if(i[1] == "green"):
                green.append(i)
            if(i[1] == "red"):
                red.append(i)
        return blue, green, red
    def sortByShape(self):
        triangle = []
        square = []
        rectangle = []
        pentagon = []
        hexagon = []
        circle = []
        for i in self.detectedShapes:
            if(i[0] == "trokut"):
                triangle.append(i)
            if(i[0] == "kvadrat"):
                square.append(i)
            if(i[0] == "pravokutnik"):
                rectangle.append(i)
            if(i[0] == "peterokut"):
                pentagon.append(i)
            if(i[0] == "sesterokut"):
                hexagon.append(i)
```

```
            if(i[0] == "krug"):
                circle.append(i)
        return triangle, square, rectangle, pentagon, hexagon, circle
```

tkinterWidgets.py:

```
from tkinter import *
from PIL import Image, ImageTk
from numpy import sin, cos

backgroundColor = '#293134'
lettersColor = '#E7E6DE'
robotColor = '#19a4bf'

class FrameTemplate(Frame):
    def __init__(self, *args, **kwargs):
        Frame.__init__(self, *args, **kwargs)
        self['bg'] = backgroundColor
        self['bd'] = 0
        self['highlightbackground'] = backgroundColor

class ButtonTemplate(Button):
    def __init__(self, *args, **kwargs):
        Button.__init__(self, *args, **kwargs)
        self['bg'] = backgroundColor
        self['fg'] = lettersColor
        self['highlightbackground'] = backgroundColor
        self['highlightcolor'] = backgroundColor
        self['activebackground'] = backgroundColor
        self['activeforeground'] = lettersColor
        self['relief'] = FLAT
        self['font'] = ("Helvetica", "14", "bold")

class MessageTemplate(Message):
    def __init__(self, *args, **kwargs):
        Message.__init__(self, *args, **kwargs)
        self['bg'] = backgroundColor
        self['fg'] = lettersColor
        self['relief'] = FLAT
        self['justify'] = CENTER

class CanvasTempalte(Canvas):
    def __init__(self, *args, **kwargs):
        Canvas.__init__(self, *args, **kwargs)
        self['bg'] = backgroundColor
        self['highlightbackground'] = backgroundColor
        self['bd'] = 0
```

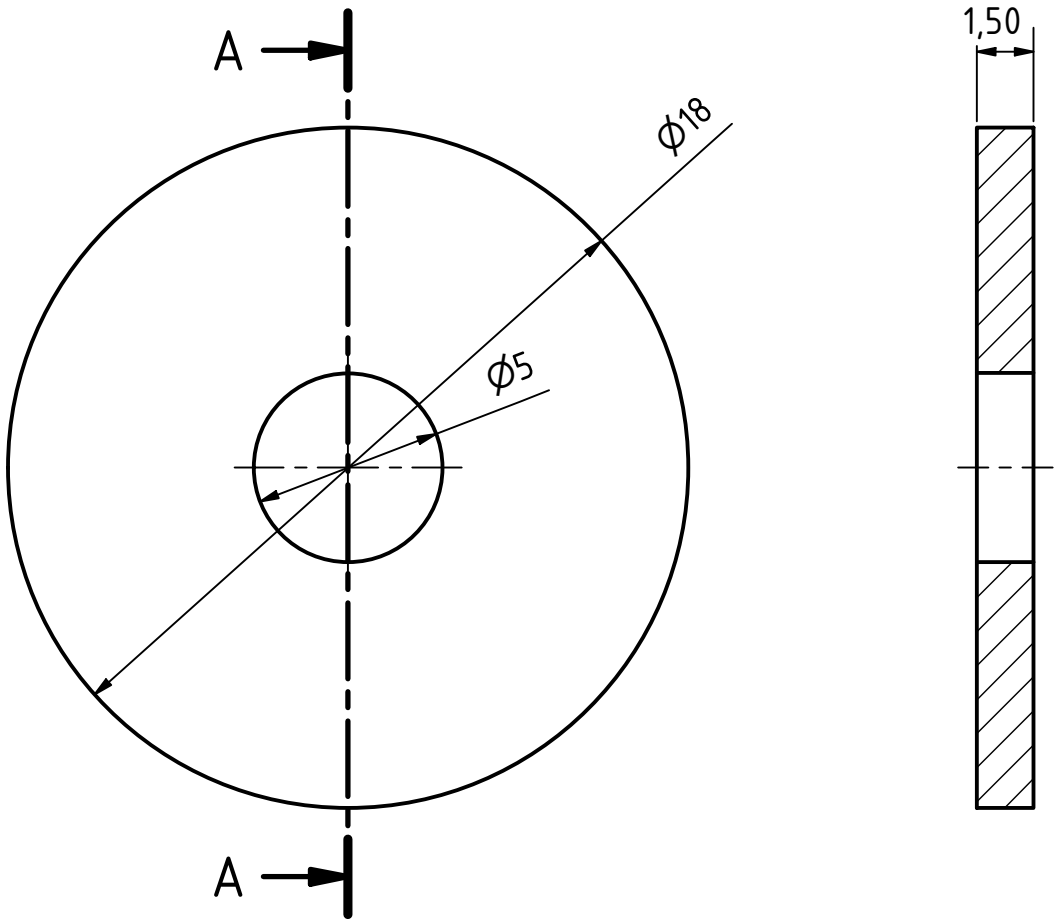
```
class CanvasRobot(Canvas):
    drawingCoordinates = [317.5, 435, 0, 0, 0, 0, 0, 0, 402.5, 435 ]
    xOffset = 360
    yOffset = 435
    l0 = 42.5
    l1 = 160
    def __init__(self, *args, **kwargs):
        Canvas.__init__(self, *args, **kwargs)
        self['bg'] = backgroundColor
        self['highlightbackground'] = backgroundColor
        self['bd'] = 0
        self['cursor'] = "dot"
    def drawRobotWorksapce(self):
        self.create_line(5, 35, 715, 35, 715, 515, 5, 515, 5, 35,
                         width = 5, fill = lettersColor)
        self.create_line(255, 372.5, 495, 372.5, 495, 515, 255, 515, 255, 372.5,
                         width = 3, fill = lettersColor)
        self.create_arc(-62.5, 55, 697.5, 815, start = -12.5,
                         extent = 97, style = ARC, width = 3,
                         outline = lettersColor)
        self.create_arc(22.5, 55, 782.5, 815, start = 192.5,
                         extent = -97, style = ARC, width = 3,
                         outline = lettersColor)
    def drawRobotPosition(self, objectRoot, drawingX, drawingY,
                         drawingTheta0, drawingTheta1):
        self.delete('robotPosition')
        self.drawingCoordinates[4], self.drawingCoordinates[5] =
                         (drawingX + self.xOffset), (self.yOffset - drawingY)
        self.drawingCoordinates[2], self.drawingCoordinates[3] =
(-self.l0 + self.l1 * cos(drawingTheta1) + self.xOffset),
                         (self.yOffset - self.l1 * sin(drawingTheta1))
        self.drawingCoordinates[6], self.drawingCoordinates[7] =
                         (self.l0 + self.l1 * cos(drawingTheta0) + self.xOffset),
                         (self.yOffset - self.l1 * sin(drawingTheta0))
        self.create_line(self.drawingCoordinates, width = 10,
                         fill = robotColor, tags='robotPosition')
        objectRoot.update()
    def followMouseMoveRobotFunc(self, event):
        currentX, currentY = (event.x - self.xOffset), (self.yOffset - event.y
        return currentX, currentY

class LabelTemplate(Label):
    def __init__(self, *args, **kwargs):
        Label.__init__(self, *args, **kwargs)
        self['bg'] = backgroundColor
        self['fg'] = lettersColor
        self['highlightbackground'] = backgroundColor
```

```
        self['highlightcolor'] = backgroundColor
        self['relief'] = FLAT
        self['activebackground'] = backgroundColor

class LabelFrameTemplate(LabelFrame):
    def __init__(self, *args, **kwargs):
        LabelFrame.__init__(self, *args, **kwargs)
        self['bg'] = backgroundColor
        self['highlightbackground'] = backgroundColor
        self['fg'] = lettersColor
        self['font'] = ("Helvetica", "20", "bold")
        self['bd'] = 0

class ScaleTemplate(Scale):
    def __init__(self, *args, **kwargs):
        Scale.__init__(self, *args, **kwargs)
        self['bg'] = backgroundColor
        self['highlightbackground'] = backgroundColor
        self['highlightcolor'] = backgroundColor
        self['activebackground'] = lettersColor
        self['troughcolor'] = lettersColor
        self['fg'] = lettersColor
        self['font'] = ("Helvetica", "20", "bold")
        self['bd'] = 0
        self['orient'] = HORIZONTAL
        self['repeatdelay'] = 1000
        self['font'] = ("Helvetica", "12", "bold")
        self['bd'] = 0
        self['width'] = 20
        self['sliderlength'] = 30
        self['length'] = 200

class RadiobuttonTemplate(Radiobutton):
    def __init__(self, *args, **kwargs):
        Radiobutton.__init__(self, *args, **kwargs)
        self['bd'] = 0
        self['font'] = ("Helvetica", "16", "bold")
        self['borderwidth'] = 10
        self['bg'] = backgroundColor
        self['activebackground'] = backgroundColor
        self['highlightbackground'] = backgroundColor
        self['highlightcolor'] = backgroundColor
        self['fg'] = lettersColor
        self['selectcolor'] = 'black'
```

```
class EntryTemplate(Entry):
    def __init__(self, *args, **kwargs):
        Entry.__init__(self, *args, **kwargs)
        self['bg'] = lettersColor
        self['fg'] = backgroundColor
        self['font'] = ("Helvetica", "16", "bold")
        self['highlightbackground'] = lettersColor
        self['highlightcolor'] = lettersColor
        self['relief'] = FLAT

class SpinboxTemplate(Spinbox):
    def __init__(self, *args, **kwargs):
        Spinbox.__init__(self, *args, **kwargs)
        self['fg'] = backgroundColor
        self['font'] = ("Helvetica", "16", "bold")
        self['relief'] = FLAT
        self['from_'] = 1
```

6.3

A-A (5 : 1)



Mjerilo: M5:1
Broj komada: 2
Materijal: AlMgSiCu (AL 6061)

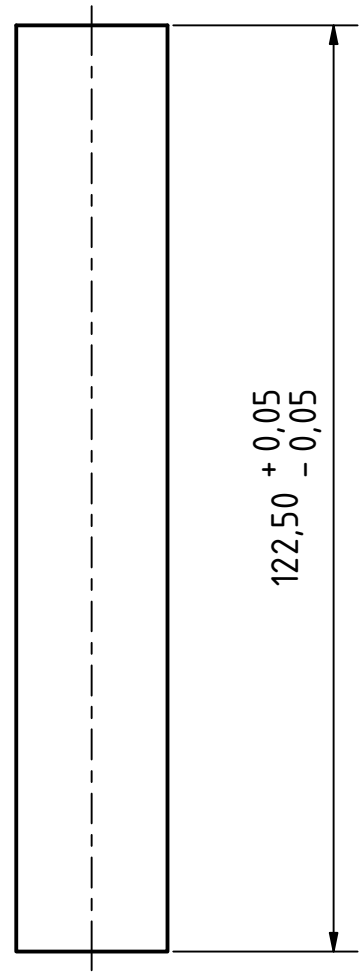
					Date	Name
				Drawn	13.1.2020.	Matej
				Checked		
				Standard		
State	Changes	Date	Name			

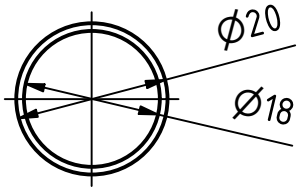
linkageA1_3/B1_3

MB_Diplomski_00

1

A4





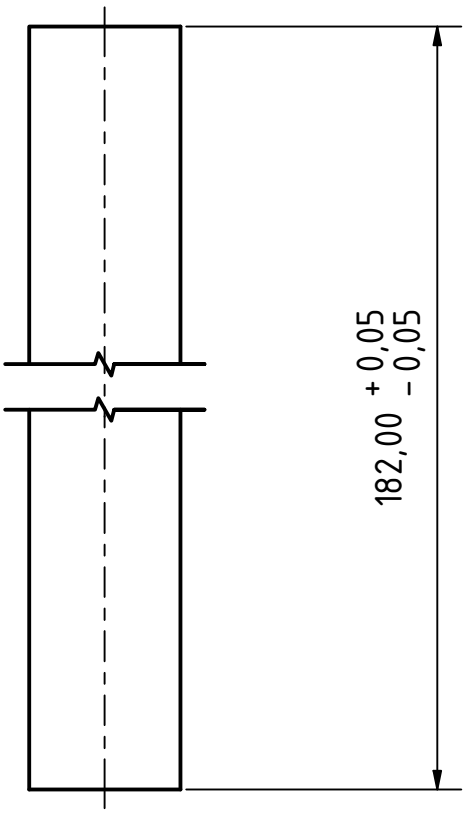
M1:1
Broj komada: 2
Materijal: Dostavljena karbonska cijev

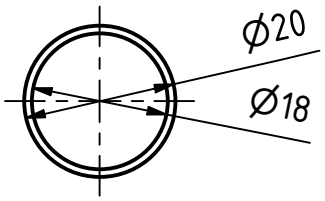
					Date	Name
				Drawn	14.1.2020.	Matej
				Checked		
				Standard		
State	Changes	Date	Name			

CarbonTube_122.5

MB_Diplomski_01

1
A4



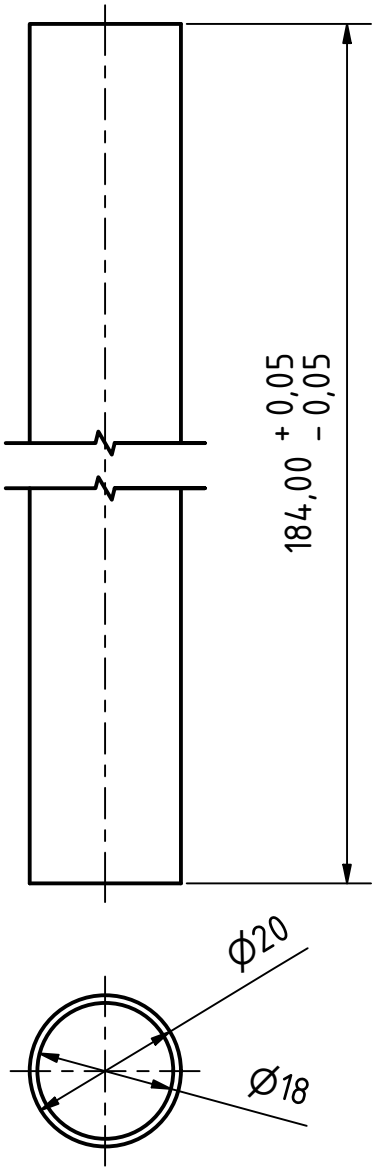


M1:1
Broj komada: 1
Materijal: Dostavljena karbonska cijev

					Date	Name
				Drawn	14.1.2020.	Matej
				Checked		
				Standard		
State	Changes	Date	Name			

CarbonTube_182

MB_Diplomski_02 | 1 | A4



M1:1

Broj komada: 1

Materijal: Dostavljena karbonska cijev

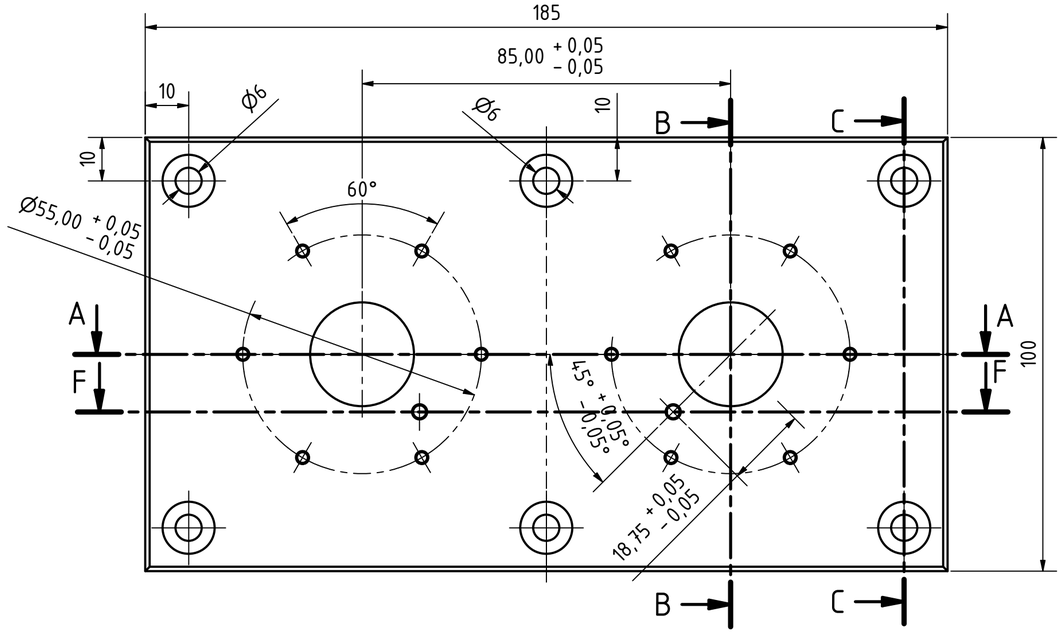
					Date	Name
				Drawn	14.1.2020.	Matej
				Checked		
				Standard		
State	Changes	Date	Name			

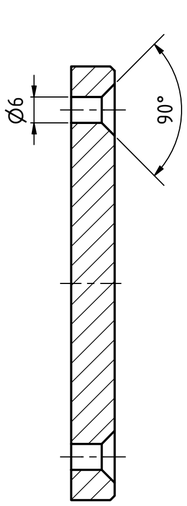
CarbonTube_184

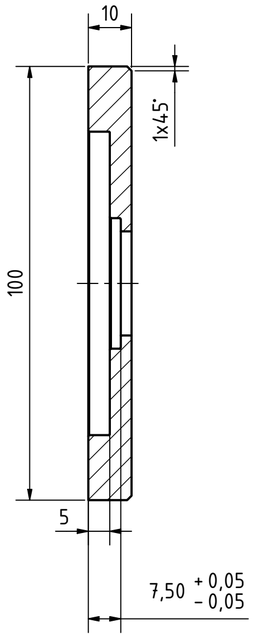
MB_Diplomski_03

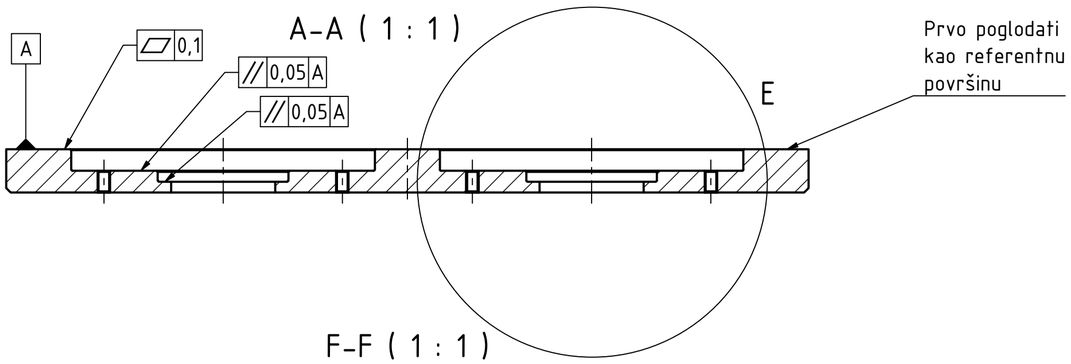
1

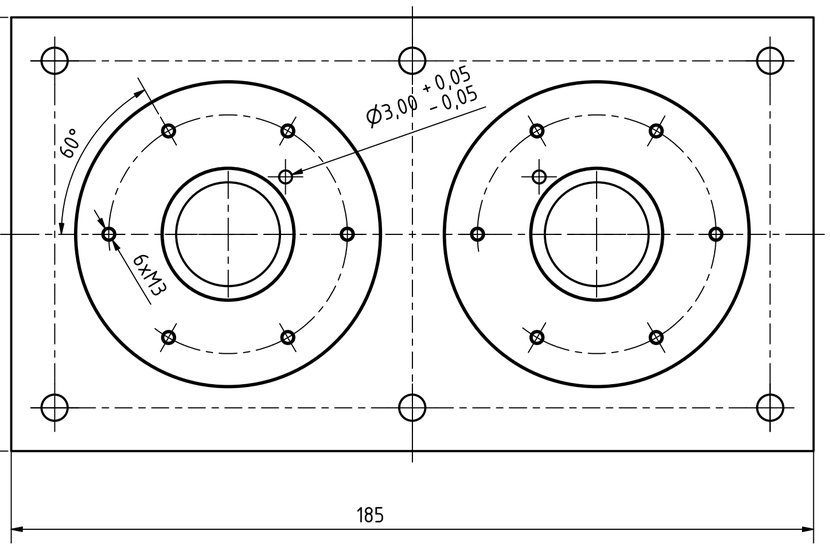
A4

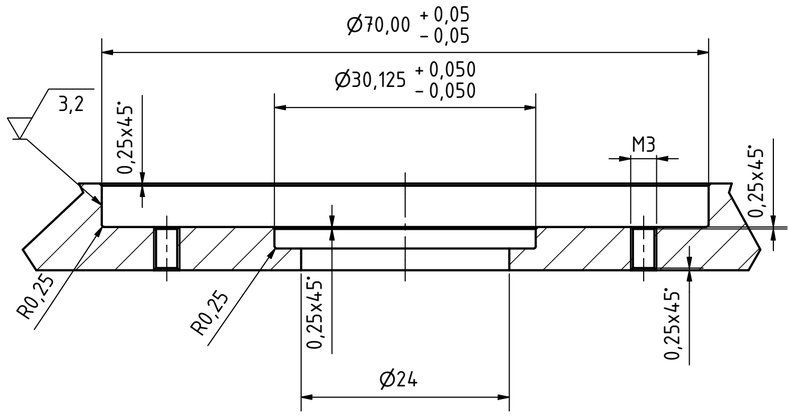


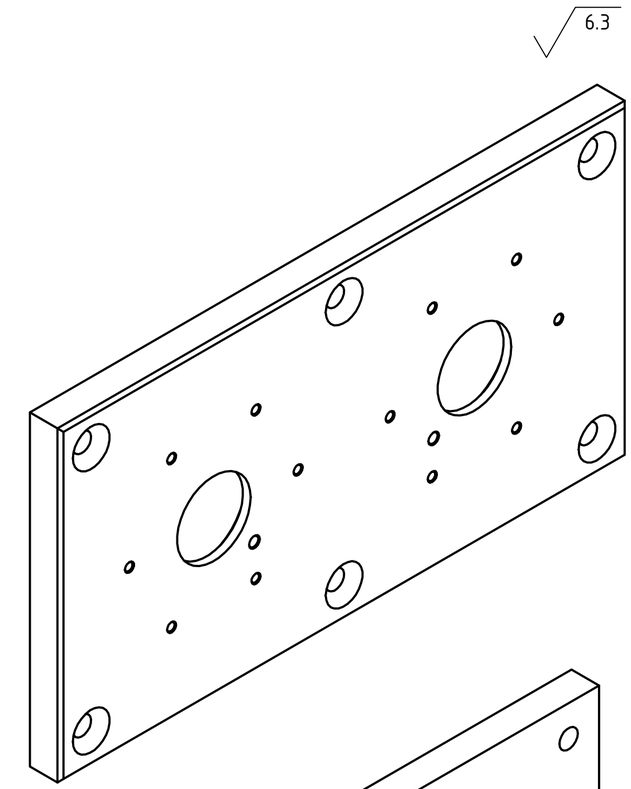


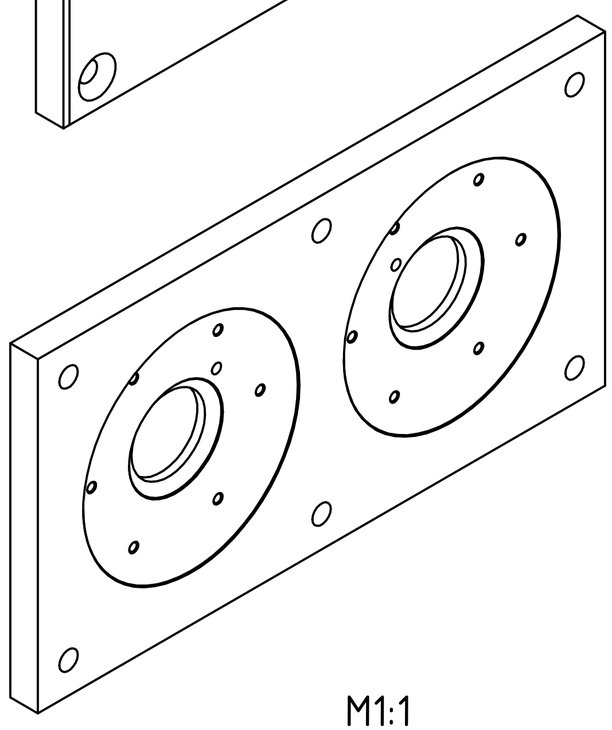












M1:1

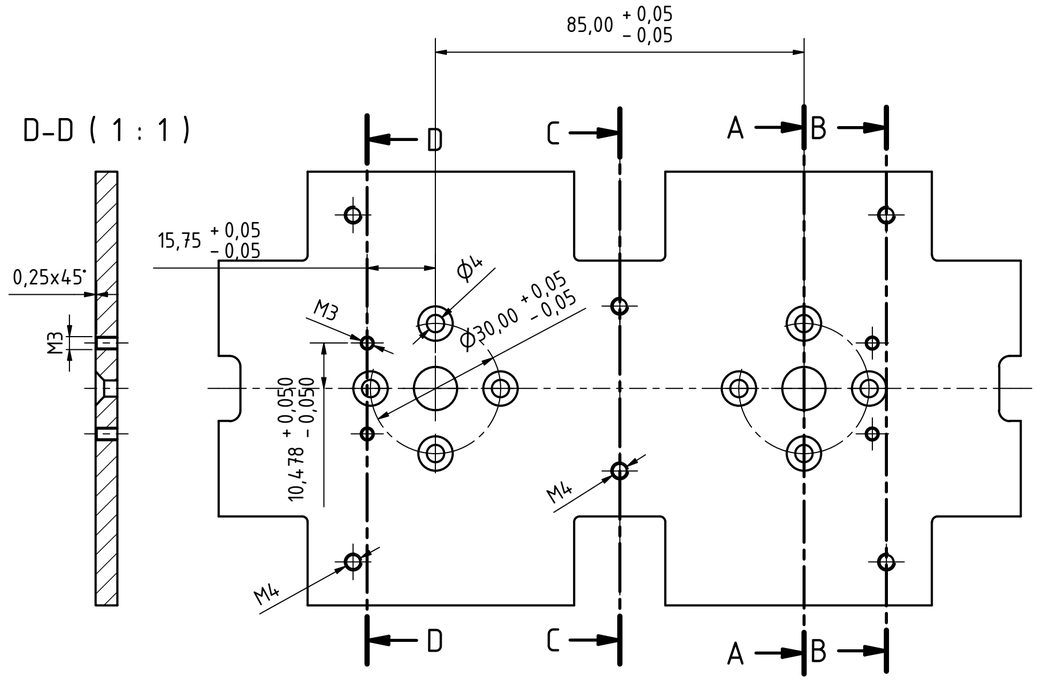
Broj komada: 1

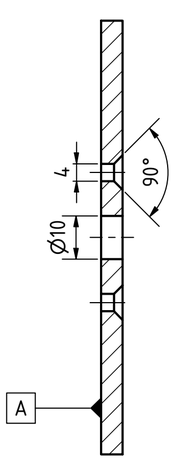
Materijal: AlMgSiCu (AL 6061)

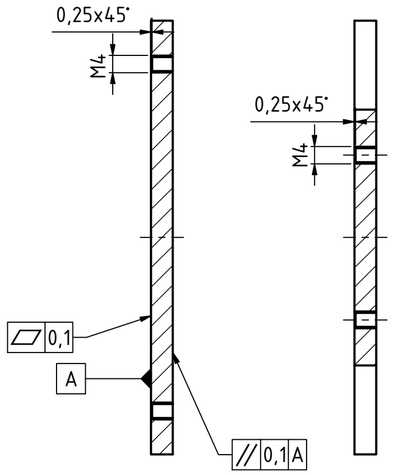
					Date	Name	motorJoint_1
				Drawn	10.1.2020.	Matej	
				Checked			
				Standard			
							MB_Diplomski_04
State	Changes	Date	Name				

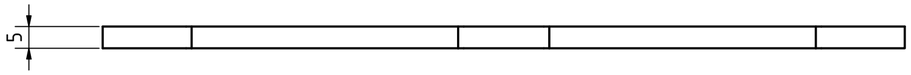
1

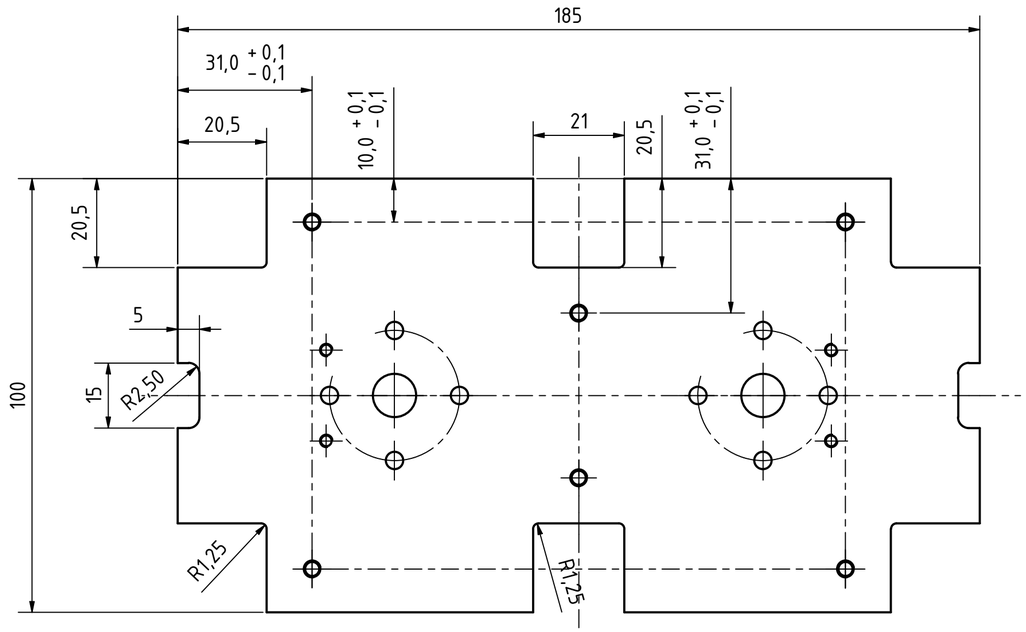
A2

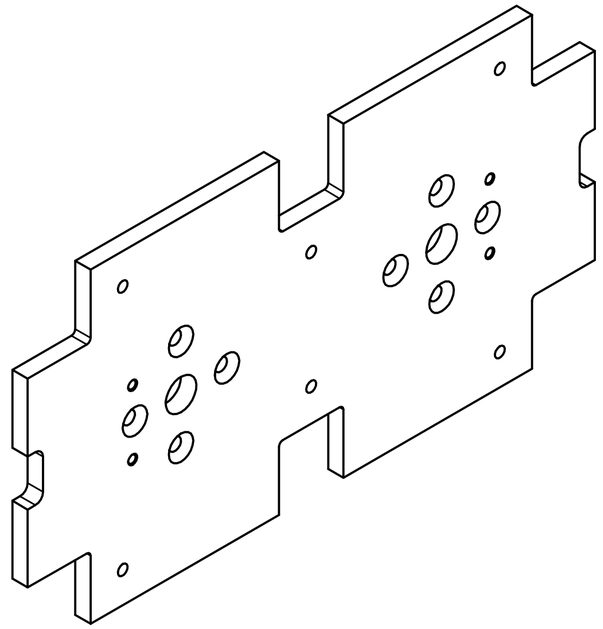












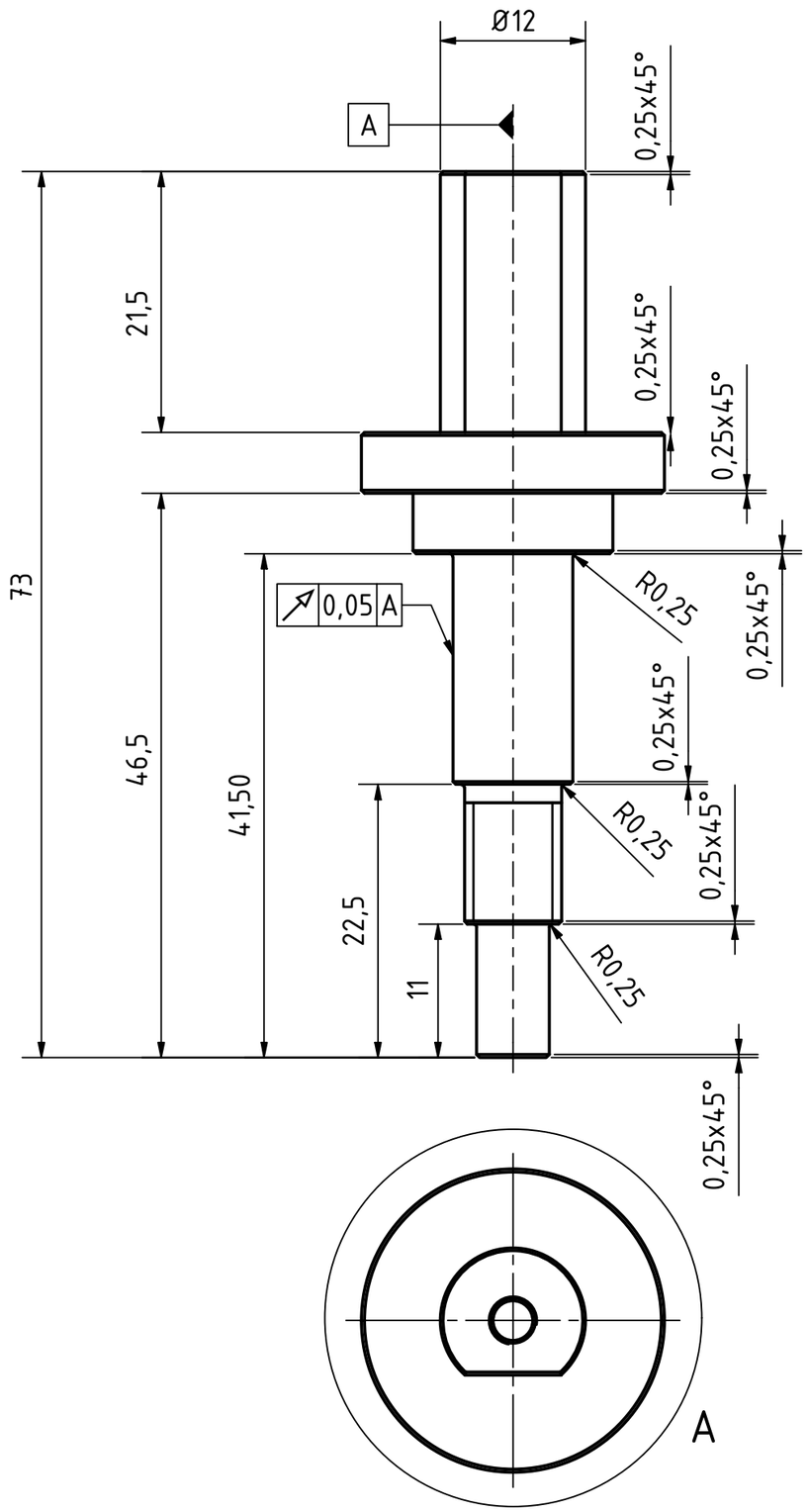
6.3

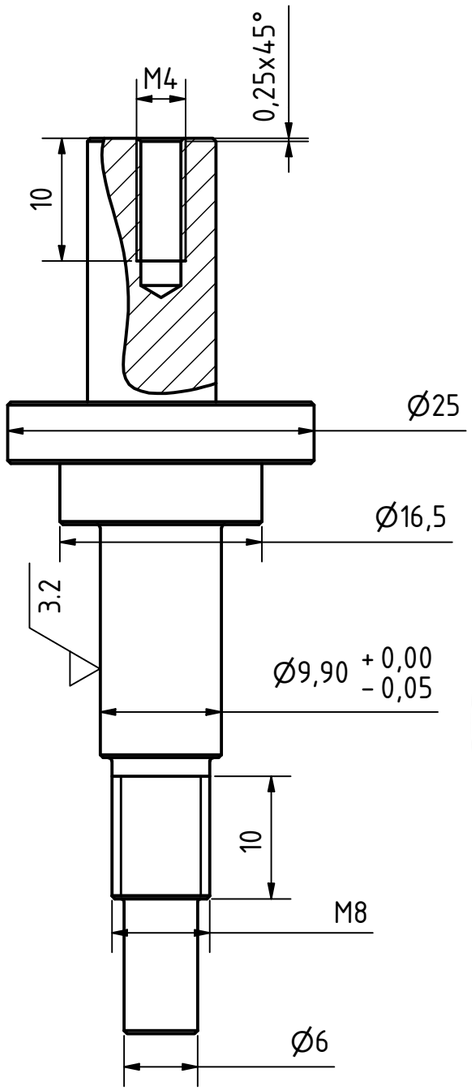
Mjerilo: M1:1

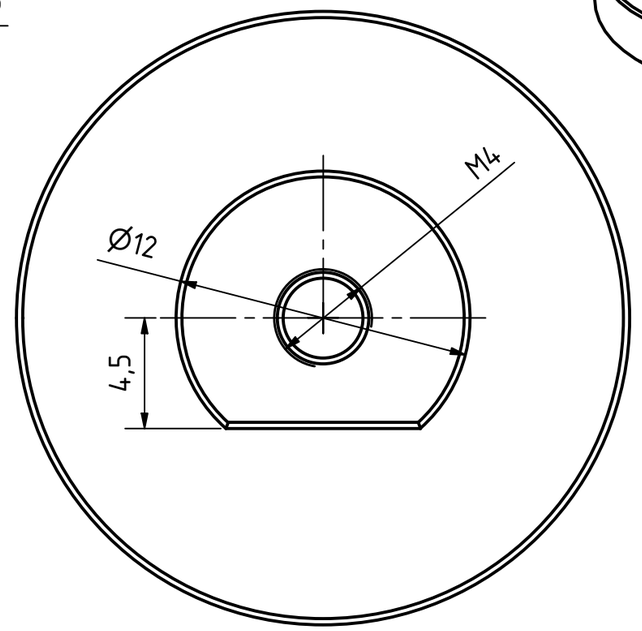
Broj komada: 1
Materijal: AlMgSiCu (AL 6061)

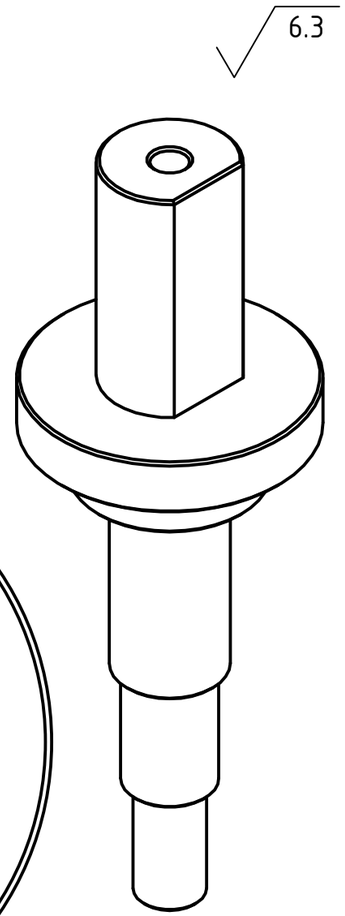
					Date	Name	motorJoint_2
				Drawn	10.1.2020.	Matej	
				Checked			
				Standard			
							MB_Diplomski_05
State	Changes	Date	Name				A2

1









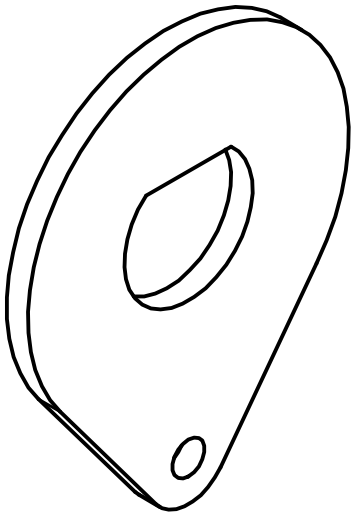
M2:1

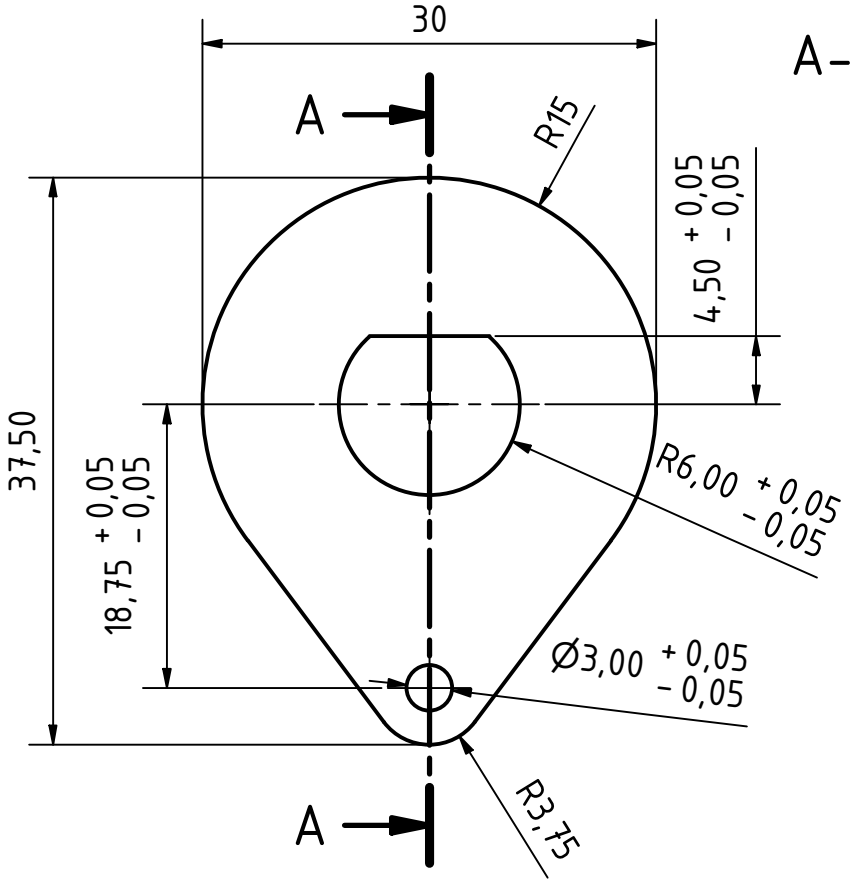
Broj komada: 2

Materijal: AlMgSiCu (AL 6061)

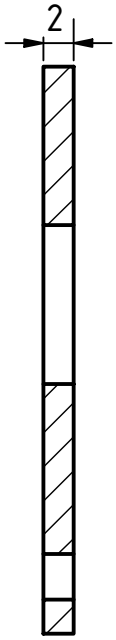
						Date	Name		
					Drawn	7.1.2020.	Matej	motorJoint_8	
					Checked				
					Standard				
								MB_Diplomski_06	1
State		Changes	Date	Name					A3

6.3





A-A (2 : 1)



Mjerilo: M2:1
Broj komada: 2
Materijal: AlMgSiCu (AL 6061)

					Date	Name
				Drawn	13.1.2020.	Matej
				Checked		
				Standard		
State	Changes	Date	Name			

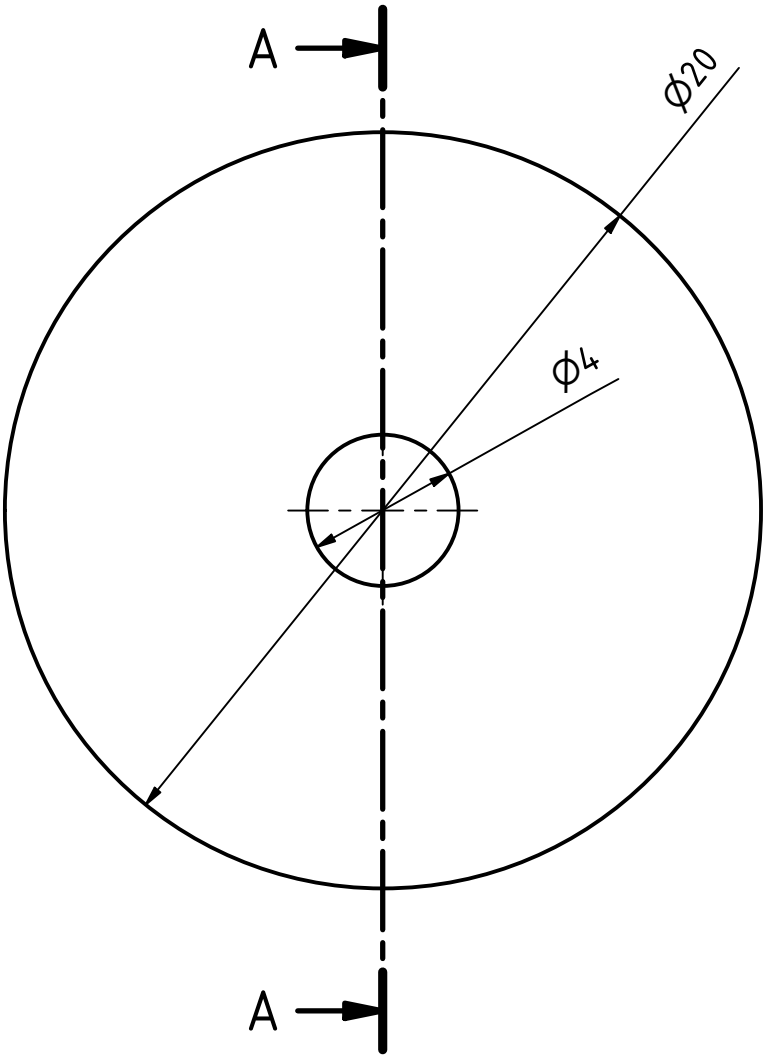
motorJoint_9

MB_Diplomski_07

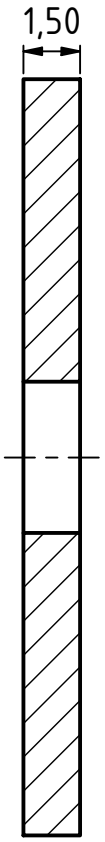
1

A4

6.3



A-A (5: 1)



Mjerilo: M5:1
Broj komada: 2
Materijal: AlMgSiCu (AL 6061)

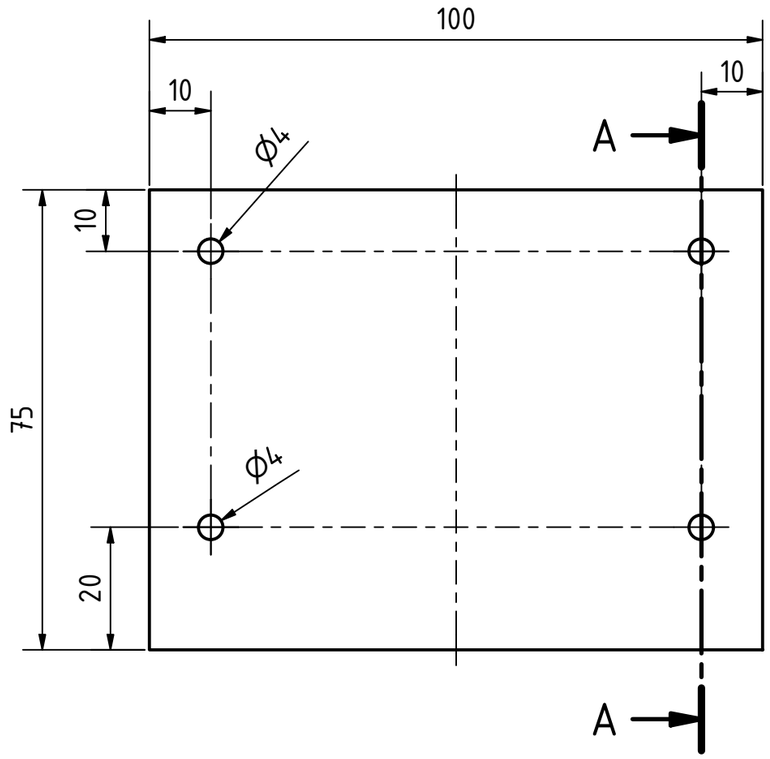
					Date	Name
				Drawn	13.1.2020.	Matej
				Checked		
				Standard		
State	Changes	Date	Name			

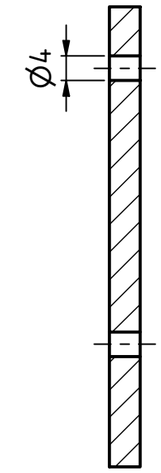
motorJoint_10

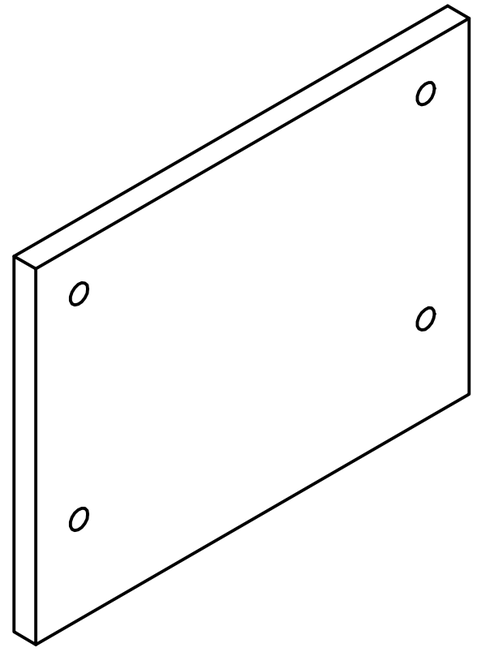
MB_Diplomski_08

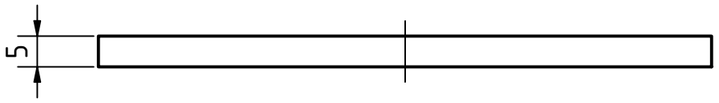
1

A4





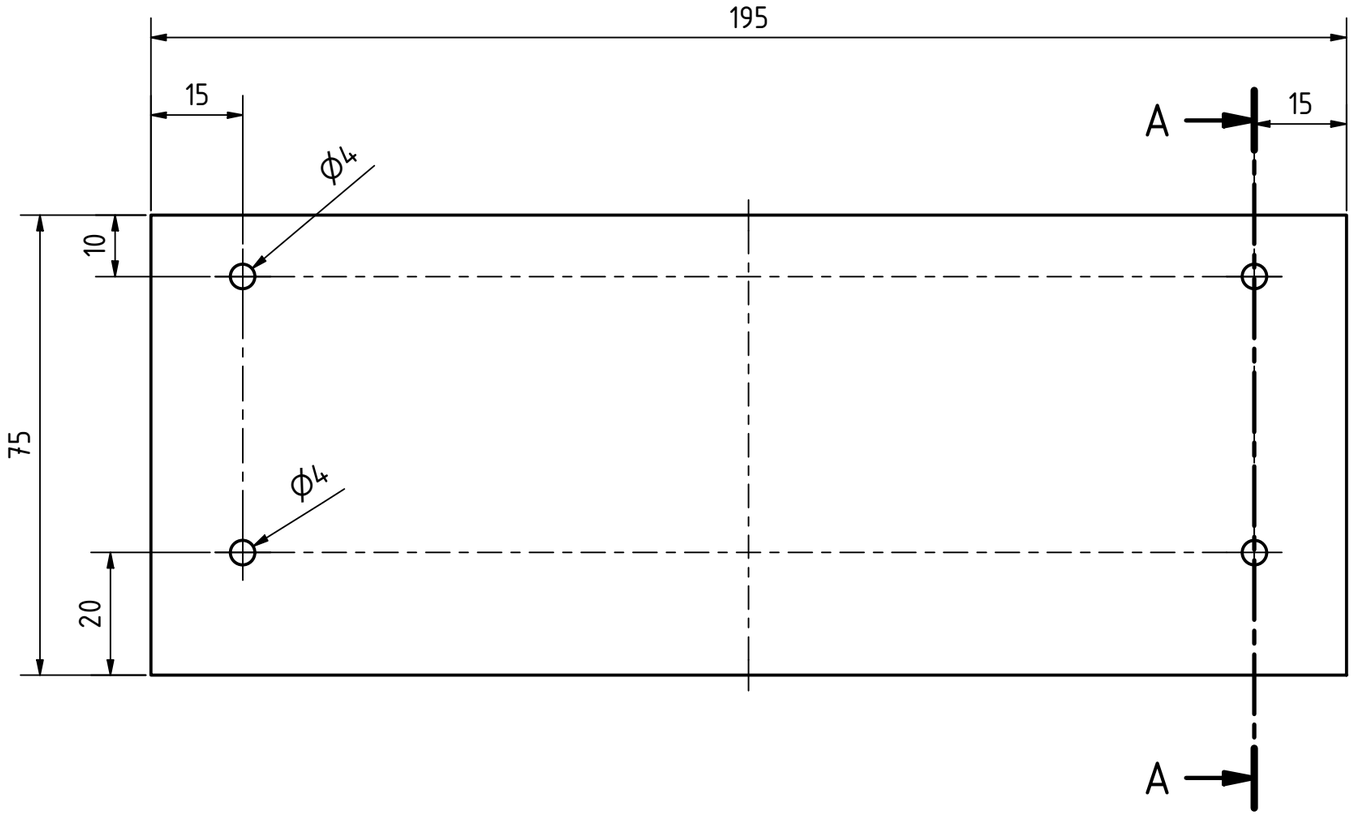




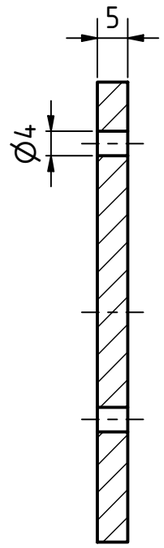
Mjerilo: M1:1
Broj komada: 2
Materijal: Pleksiglas Prozirni

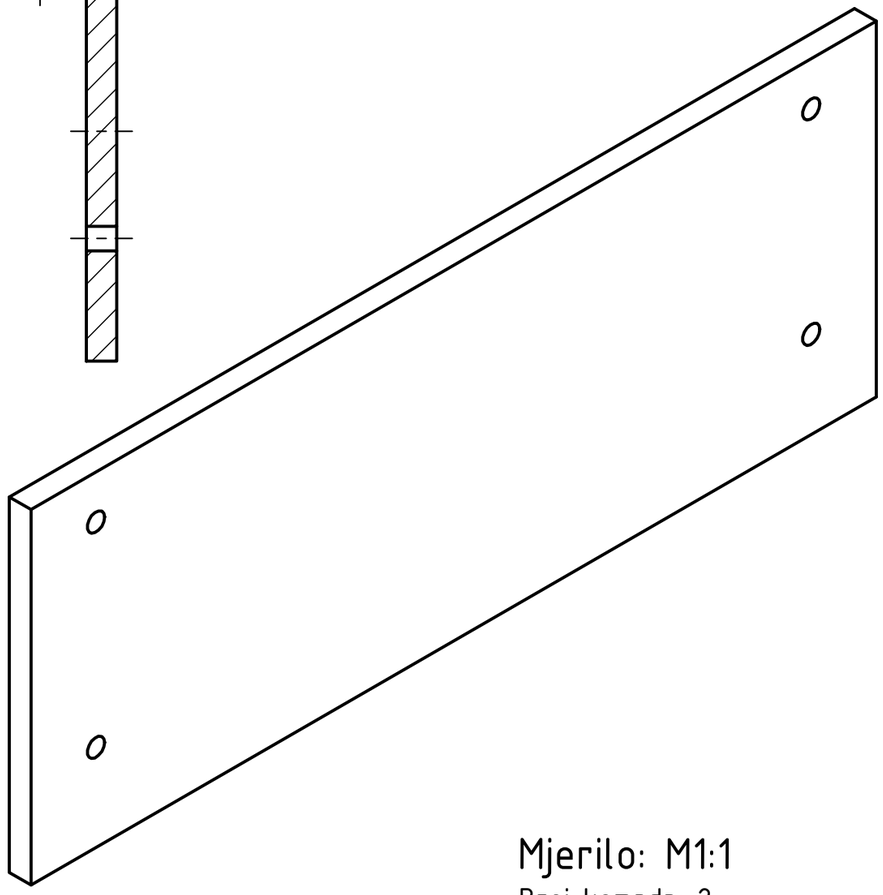
					Date	Name	motorJoint_6
				Drawn	23.1.2020.	Matej	
				Checked			
				Standard			
							MB_Diplomski_09
State	Changes	Date	Name				

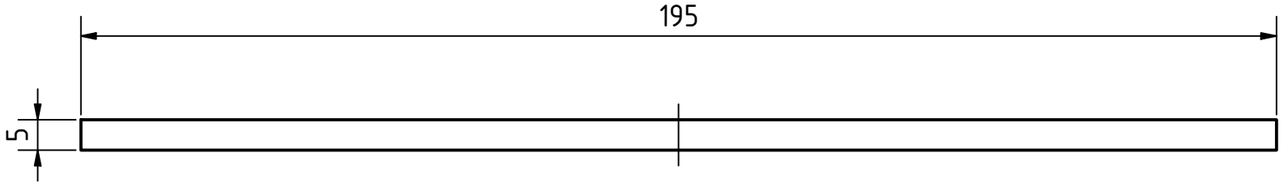
1
A3



A-A (1 : 1)







Mjerilo: M1:1

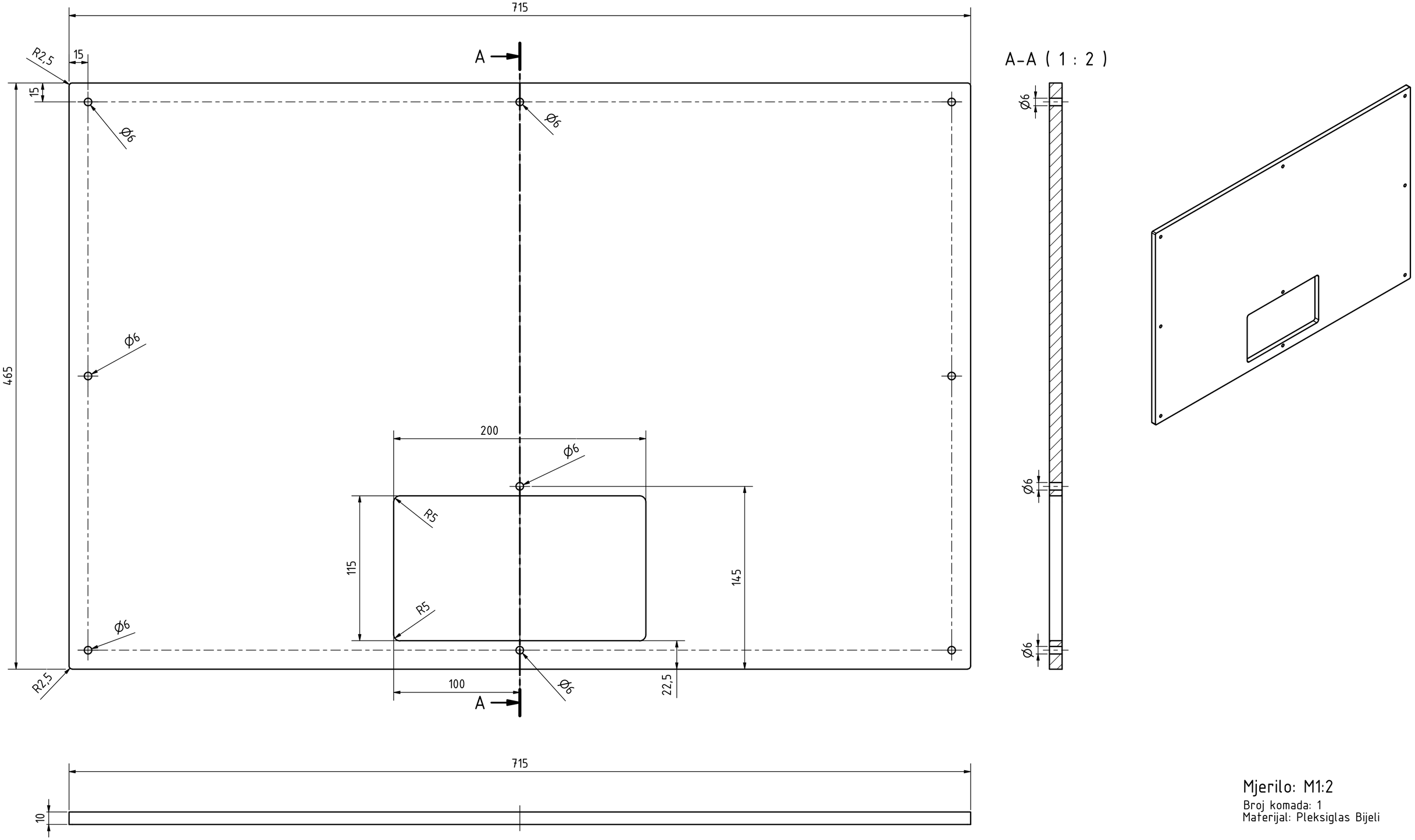
Broj komada: 2

Materijal: Pleksiglas Prozirni

					Date	Name
				Drawn	23.1.2020.	Matej
				Checked		
				Standard		
State	Changes	Date	Name			

motorJoint_7

MB_Diplomski_10 | 1 | A3



Mjerilo: M1:2
Broj komada: 1
Materijal: Pleksiglas Bijeli

	Date	Name
Drawn	23.1.2020.	Matej
Checked		
Standard		

housing_1

MB_Diplomski_11

1

A2

State	Changes	Date	Name

MATERIJAL: PA2200

Tolerancija slobodnih mjera / GENERAL TOLERANCES

				Radijus do: RADII UP TO:
A	Dimenzije vezane uz os Z DIMENSIONS RELATED TO Z AXIS			0,6±0,3 1,0±0,4
B	Dimenzije vezane uz osi X i Y DIMENSIONS RELATED TO X,Y AXIS			1,6±0,5 2,5±0,8
		A	B	5,0±1,0
do:	1	±0,18	±0,08	10 ±1,5
UP TO:	3	±0,19	±0,09	30 ±2,0
	6	±0,20	±0,10	200 ±3,0
	10	±0,21	±0,11	>200 ±5,0
	15	±0,23	±0,13	
	22	±0,25	±0,15	Debljina rebara:
	30	±0,27	±0,17	RIB THICKNESS:
	40	±0,30	±0,20	<=3,0±0,35/-0,2
	53	±0,34	±0,24	>3,0+0,60/-0,3
	70	±0,38	±0,28	
	90	±0,44	±0,34	Kut / ANGLE
	120	±0,51	±0,41	kraća stranica do:
	160	±0,60	±0,50	SHORTER LEG UP TO:
	200	±0,70	±0,60	10mm ± 2,0°
	250	±0,90	±0,80	50mm ± 1,5°
	315	±1,10	±1,00	400mm ± 1,0°

MATERIJAL: PA3200-GF

Tolerancija slobodnih mjera / GENERAL TOLERANCES

				Radijus do: RADII UP TO:
A	Dimenzije vezane uz os Z DIMENSIONS RELATED TO Z AXIS			0,6±0,3 1,0±0,4
B	Dimenzije vezane uz osi X i Y DIMENSIONS RELATED TO X,Y AXIS			1,6±0,5 2,5±0,8
		A	B	5,0±1,0
do:	1	±0,20	±0,10	10 ±1,5
UP TO:	3	±0,21	±0,11	30 ±2,0
	6	±0,22	±0,12	200 ±3,0
	10	±0,24	±0,14	>200 ±5,0
	15	±0,27	±0,17	
	22	±0,30	±0,20	Debljina rebara:
	30	±0,34	±0,24	RIB THICKNESS:
	40	±0,38	±0,28	<=3,0±0,35/-0,2
	53	±0,43	±0,33	>3,0+0,60/-0,3
	70	±0,50	±0,40	
	90	±0,60	±0,50	Kut / ANGLE
	120	±0,70	±0,60	kraća stranica do:
	160	±0,85	±0,75	SHORTER LEG UP TO:
	200	±1,05	±0,95	10mm ± 2,0°
	250	±1,25	±1,15	50mm ± 1,5°
	315	±1,55	±1,45	400mm ± 1,0°

Svojstva materijala / MATERIAL PROPERTIES: PA2200

Mehanička svojstva / MECHANICAL PROPERTIES	**Vrijednost VALUE**	**Mjerna jedinica UNIT**	**Standard ispitivanja TEST STANDARD**
Modul elastičnosti, 23°C / YOUNGS MODULUS, 23°C	1500	MPa	ISO 1
Zarezna žilavost - IZOD (23°C) / IZOD IMPACT NOTCHED, 23°C	4,4	kJ/m²	ISO 180/1A
Tvrdoća Shore D / SHORE D HARDNESS (15s)	75	-	ISO 868
3D podaci / 3D DATA	**Vrijednost VALUE**	**Mjerna jedinica UNIT**	**Standard ispitivanja TEST STANDARD**
Mehanička svojstva dijelova prizvedenih pomoću aditivne tehnologije (npr. lasersko sinteriranje - SLS, stereolitografija - SLA, taložno sraščivanje - FDM, 3D printanje) su ovisna o smjeru izgradnje obzirom na njihov sloj po sloj način izrade.			
THE PROPERTIES OF PARTS MANUFACTURED USING ADDITIVE MANUFACTURING TECHNOLOGY (E.G. LASER SINTERING, STEREOLITOGRAPHY, FUSED DEPOSITION MODELLING, 3D PRINTING) ARE, DUE TO THEIR LAYER-BY-LAYER PRODUCTION, TO SOME EXTENT DIRECTION DEPENDENT. THIS HAS TO BE CONSIDERED WHEN DESIGNINGTHE PART AND DEFINING THE BUILD ORIENTATION.			
Modul elastičnosti (smjer X) / TENSILE MODULUS (X DIRECTION)	1700	MPa	ISO 527-1/-2
Modul elastičnosti (smjer Y) / TENSILE MODULUS (Y DIRECTION)	1700	MPa	ISO 527-1/-2
Modul elastičnosti (smjer Z) / TENSILE MODULUS (Z DIRECTION)	1700	MPa	ISO 527-1/-2
Vlačna čvrstoća (smjer X) / TENSILE STRENGTH (X DIRECTION)	50	MPa	ISO 527-1/-2
Vlačna čvrstoća (smjer X) / TENSILE STRENGTH (X DIRECTION)	50	MPa	ISO 527-1/-2
Vlačna čvrstoća (smjer X) / TENSILE STRENGTH (X DIRECTION)	50	MPa	ISO 527-1/-2
Istezljivost pri lomu (smjer X) / STRAIN AT BREAK (X DIRECTION)	20	%	ISO 527-1/-2
Istezljivost pri lomu (smjer Y) / STRAIN AT BREAK (Y DIRECTION)	20	%	ISO 527-1/-2
Istezljivost pri lomu (smjer Z) / STRAIN AT BREAK (Z DIRECTION)	10	%	ISO 527-1/-2
Žilavost-Charpy (+23°C, smjer X) / CHARPY IMPACT STRENGTH (+23°C, X DIRECTION)	53	kJ/m²	ISO 179/1eU
Zarezna žilavost-Charpy (+23°C, smjer X) / CHARPY NOTCHED IMPACT STRENGTH (+23°C, X DIRECTION)	4,8	kJ/m²	ISO 179/1eA
Modul elastičnosti (23°C, smjer X) / FLEXURAL MODULUS (23°C, X DIRECTION)	1500	MPa	ISO 178
Toplinska svojstva / THERMAL PROPERTIES	**Vrijednost VALUE**	**Mjerna jedinica UNIT**	**Standard ispitivanja TEST STANDARD**
Temperatura taljenja (10°C/min) / MELTING TEMPERATURE (10°C/min)	176	°C	ISO 11357-1/-3
Temperatura omekšavanja (50°C/h 50N) / VICAT SOFTENING TEMPERATURE (50°C/h 50N)	163	°C	ISO 306
Druga svojstva / OTHER PROPERTIES	**Vrijednost VALUE**	**Mjerna jedinica UNIT**	**Standard ispitivanja TEST STANDARD**
Gustoća (laserski sinterirano) / DENSITY (LASERSINTERED)	930	kg/m³	EOS Method

Svojstva materijala / MATERIAL PROPERTIES: PA3200-GF

Mehanička svojstva / MECHANICAL PROPERTIES	**Vrijednost VALUE**	**Mjerna jedinica UNIT**	**Standard ispitivanja TEST STANDARD**
Modul elastičnosti, 23°C / YOUNGS MODULUS, 23°C	2900	MPa	ISO 1
Zarezna žilavost - IZOD (23°C) / IZOD IMPACT NOTCHED, 23°C	4,2	kJ/m²	ISO 180/1A
Tvrdoća Shore D / SHORE D HARDNESS (15s)	80	-	ISO 868
3D podaci / 3D DATA	**Vrijednost VALUE**	**Mjerna jedinica UNIT**	**Standard ispitivanja TEST STANDARD**
Mehanička svojstva dijelova prizvedenih pomoću aditivne tehnologije (npr. lasersko sinteriranje - SLS, stereolitografija - SLA, taložno sraščivanje - FDM, 3D printanje) su ovisna o smjeru izgradnje obzirom na njihov sloj po sloj način izrade.			
THE PROPERTIES OF PARTS MANUFACTURED USING ADDITIVE MANUFACTURING TECHNOLOGY (E.G. LASER SINTERING, STEREOLITOGRAPHY, FUSED DEPOSITION MODELLING, 3D PRINTING) ARE, DUE TO THEIR LAYER-BY-LAYER PRODUCTION, TO SOME EXTENT DIRECTION DEPENDENT. THIS HAS TO BE CONSIDERED WHEN DESIGNINGTHE PART AND DEFINING THE BUILD ORIENTATION.			
Modul elastičnosti (smjer X) / TENSILE MODULUS (X DIRECTION)	3200	MPa	ISO 527-1/-2
Modul elastičnosti (smjer Y) / TENSILE MODULUS (Y DIRECTION)	3200	MPa	ISO 527-1/-2
Modul elastičnosti (smjer Z) / TENSILE MODULUS (Z DIRECTION)	2500	MPa	ISO 527-1/-2
Vlačna čvrstoća (smjer X) / TENSILE STRENGTH (X DIRECTION)	51	MPa	ISO 527-1/-2
Vlačna čvrstoća (smjer X) / TENSILE STRENGTH (X DIRECTION)	51	MPa	ISO 527-1/-2
Vlačna čvrstoća (smjer X) / TENSILE STRENGTH (X DIRECTION)	47	MPa	ISO 527-1/-2
Istezljivost pri lomu (smjer X) / STRAIN AT BREAK (X DIRECTION)	9	%	ISO 527-1/-2
Žilavost-Charpy (+23°C, smjer X) / CHARPY IMPACT STRENGTH (+23°C, X DIRECTION)	35	kJ/m²	ISO 179/1eU
Zarezna žilavost-Charpy (+23°C, smjer X) / CHARPY NOTCHED IMPACT STRENGTH (+23°C, X DIRECTION)	5,4	kJ/m²	ISO 179/1eA
Toplinska svojstva / THERMAL PROPERTIES	**Vrijednost VALUE**	**Mjerna jedinica UNIT**	**Standard ispitivanja TEST STANDARD**
Temperatura taljenja (10°C/min) / MELTING TEMPERATURE (10°C/min)	176	°C	ISO 11357-1/-3
Temperatura omekšavanja A / VICAT SOFTENING TEMPERATURE A	179	°C	ISO 306
Temperatura omekšavanja (50°C/h 50N) / VICAT SOFTENING TEMPERATURE (50°C/h 50N)	163	°C	ISO 306
Druga svojstva / OTHER PROPERTIES	**Vrijednost VALUE**	**Mjerna jedinica UNIT**	**Standard ispitivanja TEST STANDARD**
Gustoća (laserski sinterirano) / DENSITY (LASERSINTERED)	1220	kg/m³	EOS Method

KLEX d.o.o. MB: 1744569 OIB: 61910837843 VAT Nr. HR61910837843

KLARIĆ EXPLORER Tel. +385 1 458 00 88 ZAGREBAČKA BANKA Direktor:
Veselišće 18 Fax. +385 1 457 27 49 2360000-1101677051 Igor Klarić dipl.ing
HR - 10000 Zagreb eMail: klex@klex.hr SWIFT:ZABAHR2X Gordana Klarić dipl.ing
IBAN: HR9423600001101677051