Kinematičko upravljanje robotskom rukom za teleoperaciju korištenjem mobilnog uređaja

Furdi, Evelyn

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 712

KINEMATIČKO UPRAVLJANJE ROBOTSKOM RUKOM ZA TELEOPERACIJU KORIŠTENJEM MOBILNOG UREĐAJA

Evelyn Furdi

Zagreb, veljača 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 712

KINEMATIČKO UPRAVLJANJE ROBOTSKOM RUKOM ZA TELEOPERACIJU KORIŠTENJEM MOBILNOG UREĐAJA

Evelyn Furdi

Zagreb, veljača 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 30. rujna 2024.

DIPLOMSKI ZADATAK br. 712

Pristupnica:	**Evelyn Furdi (0036518852)**
Studij:	Računarstvo
Profil:	Računalno inženjerstvo
Mentorica:	prof. dr. sc. Marija Seder

Zadatak: **Kinematičko upravljanje robotskom rukom za teleoperaciju korištenjem mobilnog uređaja**

Opis zadatka:

Teleoperacija je ključan element u robotičkim sustavima koji omogućuje upravljanje robotima na daljinu, posebno kada je riječ o izvođenju složenih zadataka. Ovaj rad fokusira se na razvoj aplikacije u operacijskom sustavu Android za teleoperaciju robotske ruke koja se zasniva na Robotskom Operacijskom Sustavu (ROS). Poseban naglasak stavlja se na upravljanje robotskom rukom kroz implementaciju vlastitog algoritma kinematičkog upravljanja i inverzne kinematike. Aplikacija će ostvariti intuitivno korisničko sučelje kroz koje će korisnici moći zadavati naredbe za pozicioniranje i orijentaciju izvršnog elementa robotske ruke, koje će potom biti izvršene korištenjem razvijenog algoritma. Razvijene komponente bit će testirane na robotskoj ruci Kinova Jaco unutar simulacijskog okruženja Gazebo te u stvarnom laboratorijskom okruženju.

Rok za predaju rada: 14. veljače 2025.

Sadržaj

Uvod

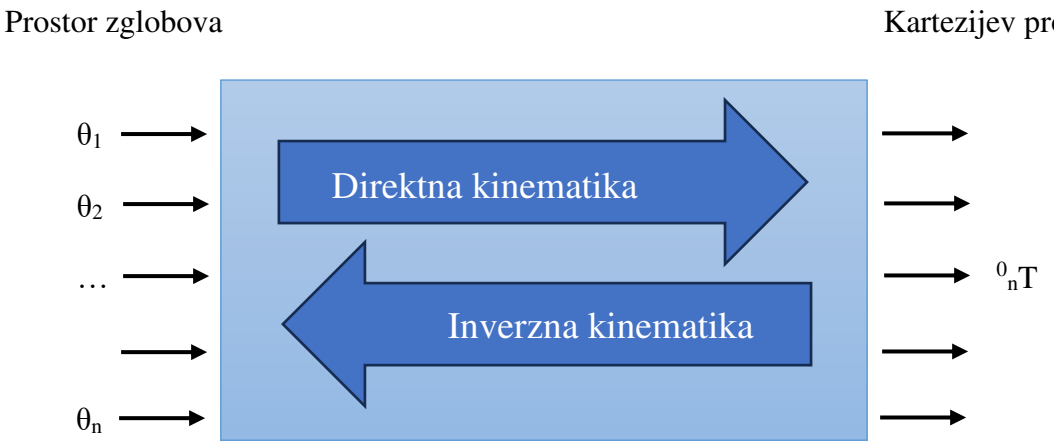
Teleoperacija je ključan element u robotičkim sustavima koji omogućuje upravljanje robotima na daljinu, posebno kada je riječ o izvođenju složenih zadataka. Koristi se za primjenu u različitim područjima, od industrijske automatizacije i medicine do istraživanja nepristupačnih područja. Brzim razvojem i unapređenjem dostupnih tehnologija, osobito mobilnih uređaja i robotskih operacijskih sustava (ROS), teleoperacija postaje sve pristupačnija i lakša za korištenje krajnjim korisnici.

Ovaj rad fokusira se na razvoj Android aplikacije za teleoperaciju robotske ruke koja se temelji na ROS sustavu. Poseban naglasak stavlja se na upravljanje robotskom rukom kroz implementaciju vlastitog algoritma kinematičkog upravljanja i inverzne kinematike. Kao sustav kojim se upravlja odabran je model robotske ruke tvrtke Kinova naziva JACO Gen 2 zbog svoje modularnosti i prilagodljivosti.

Ideja je ostvariti intuitivno korisničko sučelje kroz koje će korisnici moći zadavati naredbe za pozicioniranje i orijentaciju izvršnog elementa robotske ruke, koje će potom biti izvršene korištenjem razvijenog algoritma. U poglavljima koja slijede najprije će biti objašnjeno samo upravljanje robotskom rukom gdje će se pojasniti dva primarna koncepta upravljanja rukom, a to su direktna i inverzna kinematika. Zatim slijedi poglavlje koje pojašnjava izrađenu Android aplikaciju koja se koristi za teleoperaciju rukom i njezino povezivanje sa skriptom koja upravlja rukom. Nakon toga biti će dati pregled korištenih tehnologija u samoj implementaciji rješenja gdje spadaju ROS, Gazebo i sama robotska ruka te upute za pokretanje cijelog implementacijskog rješenja. Naposljetku napravit će se pregled rezultata iz simulacije u Gazebo simulacijskom okruženju i biti će dati pregled stvarnih rezultata iz laboratorija na fizičkoj ruci.

1. Upravljanje robotskom rukom

Kinematika ([1]) je grana mehanike koja je temeljna tehnika za proučavanje kretanja robota manipulatora u robotici. Proučava gibanje manipulatora ne uzimajući pritom u obzir sile i momente koji uzrokuju kretanje. Uglavnom se u kinematici koriste dva prostora modeliranja, a to su Kartezijev i kvaternionski prostor. Kinematika robota se dijeli na direktnu kinematiku i inverznu kinematiku.



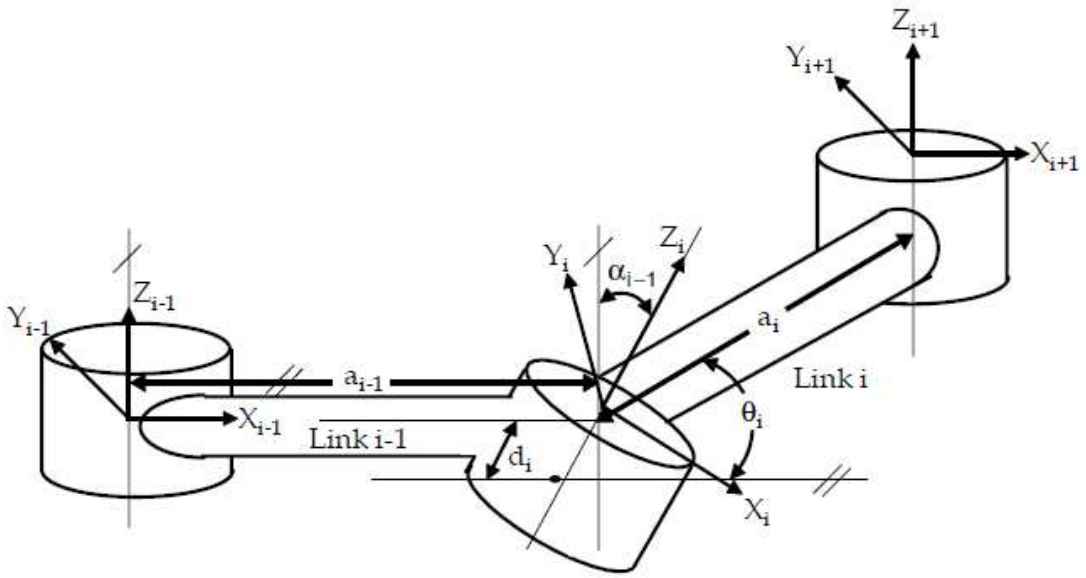
Sl 1.1 Shematski prikaz direktne i inverzne kinematike

1.1. Direktna kinematika

Direktna kinematika ([1]) koristi se za izračun orijentacije i položaja izvršnog člana robotske ruke. Za opisivanje kinematike robota najčešće se koristi Denavit-Hartenberg metoda s četiri parametara:

- $a_{i-1,}$ duljina uda
- α_{i-1}, zakret uda
- d_i, odmak uda
- θ_i, kut zgloba

U svrhu određivanja DH parametara, za svaki spoj na slici (Sl 1.2) je pričvršćen koordinatni sustav.



Sl 1.2 Dodjela koordinatnog sustava za opći manipulator a. [1]

$$
{}^{i-1}_{i}T = R_x(\alpha_{i-1})D_x(a_{i-1})R_z(\theta_i)Q_i(d_i)
$$

$$
= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\alpha_{i-1} & -s\alpha_{i-1} & 0 \\ 0 & s\alpha_{i-1} & c\alpha_{i-1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_{i-1} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
= \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1}d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Sl 1.3. Opća matrica transformacije T. [1]

Slika (Sl 1.3) prikazuje način izračunavanja transformacijske matrice gdje R_x i R_z predstavljaju rotacije, a D_x i Q_i translacije. Konačno direktna kinematika izvršnog člana ruke u odnosu na osnovni okvir dobiva se množenjem svih ${}^{i-1}_{i}T$ matrica. Složenost izvođenja jednadžbi direktne kinematike je jednostavna pa samim time uvijek postoji rješenje direktne kinematike manipulatora.

1.2. Jakobijan matrica

Jakobijan matrica ([7]) je nazvana po matematičaru Jacobiju, a u robotici se koristi za izražavanje odnosa između linearne i kutne brzine izvršnog člana ruke i kutne brzine zglobova. Formula glasi: $\dot{p}_{ee} = J_{ee} \cdot \dot{q}$ ([7])

Kutna brzina zglobova lako se izračunava uzeći u obzir kut koji smo pomaknuli u određenom vremenskom rasponu. Kad bismo npr. zglob q_1 pomaknuli u jednoj sekundi za puni krug, taj zglob bi se kretao s rotacijskom brzinom $\dot{q}_1 = 2\pi$ *rad/s* ([7]). Kutna brzina izvršnog člana ruke v_{ee} može se razdvojiti u dvije komponente, linearna brzina u x i linearna brzina u y smjeru. Položaj izvršnog člana ruke ako se gleda da možemo pomicati samo q_1 zglob može se prikazati

$$p_{ee} = \begin{pmatrix} l_0 + l_1 \cdot cos(q_1) \\ l_0 + l_1 \cdot sin(q_1) \end{pmatrix} \text{ ([7])}$$

gdje se q_1 mijenja kroz vrijeme dok su l_0, l_1 konstante. Izračunamo li vremensku derivaciju dobivamo brzinu izvršnog člana $v_{ee} = \begin{pmatrix} -l_1 \cdot sin(q_1) \\ l_1 \cdot cos(q_1) \end{pmatrix}$ ([7])

J_{ee} je matrica koja se sastoji od parcijalnih derivacija kinetičkih funkcija, prikazana je na slici (Sl 1.4)

$$J_{ee} = \begin{bmatrix} \frac{\partial x}{\partial q_0} & \frac{\partial x}{\partial q_1} & \frac{\partial x}{\partial q_2} \\ \frac{\partial y}{\partial q_0} & \frac{\partial y}{\partial q_1} & \frac{\partial y}{\partial q_2} \\ \frac{\partial \phi}{\partial q_0} & \frac{\partial \phi}{\partial q_1} & \frac{\partial \phi}{\partial q_2} \end{bmatrix}$$

Parcijalne derivacije za položaj (x, y)

Parcijalne derivacije za kut ϕ izvršnog člana

Sl 1.4 Matrica parcijalnih derivacija kinetičkih funkcija [7]

Kad raspišemo parcijalne derivacije po x (Sl 1.5) i parcijalne derivacije po y (Sl 1.6) i uzmemo u obzir da su sve u zadnjem redu = 1, sad možemo izračunati linearnu i kutnu brzinu izvršnog člana ruke.

$$\frac{\partial x}{\partial q_0} = -l_0 \cdot sin(q_0) - l_1 \cdot sin(q_0 + q_1) - l_2 \cdot sin(q_0 + q_1 + q_2)$$

$$\frac{\partial x}{\partial q_1} = -l_1 \cdot sin(q_0 + q_1) - l_2 \cdot sin(q_0 + q_1 + q_2)$$

$$\frac{\partial x}{\partial q_2} = -l_2 \cdot sin(q_0 + q_1 + q_2)$$

Sl 1.5 Parcijalne derivacije po x [7]

$$\frac{\partial y}{\partial q_0} = l_0 \cdot cos(q_0) + l_1 \cdot cos(q_0 + q_1) + l_2 \cdot cos(q_0 + q_1 + q_2)$$

$$\frac{\partial y}{\partial q_1} = l_1 \cdot cos(q_0 + q_1) + l_2 \cdot cos(q_0 + q_1 + q_2)$$

$$\frac{\partial y}{\partial q_2} = l_2 \cdot cos(q_0 + q_1 + q_2)$$

Sl 1.6 Parcijalne derivacije po y [7]

1.3. Inverzna kinematika

Inverzna kinematika ([1]) koristi se za pronalaženje vrijednosti na koje treba postaviti pojedini od zglobova ruke kako bi se postigao željeni položaj izvršnog člana ruke. Kao ulaz prima koordinate položaja i orijentaciju izvršnog člana ruke te mora izračunati na koje vrijednosti treba postaviti zglobove da bi se izvršni član doveo u tu željenu poziciju. Postoji više pristupa u rješavanju problema inverzne kinematike, a onaj korišten u ovom diplomskom radu naziva se kinetičko upravljanje. Za daljnje objašnjavanje inverzne kinematike iskoristit ćemo isječke kodova ROS čvorova korištenih u ovom diplomskom radu koji koriste inverznu kinematiku kako bi upravljali rukom bilo u simulacijskom ili laboratorijskom okruženju.

1.3.1. Primjeri korištenja za upravljanje robotskom rukom u simulatoru

Inverzna kinematika koristi se u ROS čvoru move_robot_cartesian.py kako bi se omogućilo upravljanje rukom u Kartezijevom koordinatnom sustavu. Kada ROS čvor iz aplikacije iz kontrolera u Kartezijevom koordinatnom sustavu primi neku od naredbi kao što su „x_pos, y_pos, z_pos“ potrebno je zamaknuti izvršni čvor ruke za npr. naredbu

„x_pos“ na poziciju (x_trenutni + delta, y_trenutni, z_trenutni), a za to nam treba inverzna kinematika kako bismo izračunali za koliko treba promijeniti vrijednost pojedinog zgloba da bi se izvršni čvor ruke pomaknuo u pozitivnom smjeru x osi za (+ delta) vrijednost. Isti princip vrijedi i za naredbe „x_neg, y_neg, z_neg“ samo se njihove vrijednosti mijenjaju za (-delta). Kad je primljena jedna od tih naredbi, najprije se poziva funkcija `differential_movement(np.array([delta, 0.0, 0.0]), prefix, nbJoints)`

Funkcija za postizanje diferencijalnih pomaka prikazana na slici (Sl 1.7) koristi Jakobijan matricu za postizanje malih diferencijalnih pomaka izvršnog člana ruke u željenom smjeru u Kartezijevom prostoru. Na početku funkcije najprije se iz ROS teme /j2n6s300/joint_states dohvaća trenutna konfiguracija robota koja ovdje predstavlja početnu poziciju zglobova robota.

```
def differential_movement(command, prefix, nbJoints):
    msg = rospy.wait_for_message('/j2n6s300/joint_states', JointState)
    current_conf = np.array(msg.position)[0:6]
    current_pos = forward_kinematics(current_conf)
    next_pos = current_pos + command
    vdot = next_pos - current_pos
    qdot = np.linalg.pinv(numerical_jacobian(current_conf)).dot(vdot)
    next_conf = current_conf + qdot
    moveJoint(next_conf, prefix, nbJoints)
```

Sl 1.7 Isječak koda funkcije za postizanje diferencijalnih pomaka izvršnog člana ruke

Zatim se uz pomoć funkcije za direktnu kinematiku (Sl 1.8) predajući joj trenutnu konfiguraciju kao parametar izračunava trenutna pozicija izvršnog čvora ruke.

```
def forward_kinematics(conf):
    D1 = 0.2755
    D2 = 0.41
    D3 = 0.2073
    D4 = 0.0741
    D5 = 0.0741
    D6 = 0.16

    e2 = 0.0098
    aa = 30 * np.pi / 180
    sa = np.sin(aa)
    s2a = np.sin(2 * aa)
    d4b = D3 + (sa / s2a) * D4
    d5b = (sa / s2a) * D4 + (sa / s2a) * D5
    d6b = (sa / s2a) * D5 + D6
    al = np.array([np.pi / 2, np.pi, np.pi / 2, 2 * aa, 2 * aa, np.pi])
    a = np.array([0, D2, 0, 0, 0, 0])
    d = np.array([D1, 0, -e2, -d4b, -d5b, -d6b])
    th = np.array([-conf[0], conf[1] - np.pi / 2, conf[2] + np.pi / 2, conf[3], conf[4] - np.pi, conf[5] + np.pi / 2])
    T = np.eye(4)
    for i in range(6):
        T = np.dot(T, dh_matrix(th[i], d[i], a[i], al[i]))
    fk = T[:3, 3]
    return fk
```

Sl 1.8 Isječak koda funkcije direktne kinematike za izračun trenutne pozicije izvršnog čvora

. Funkcija za direktnu kinematiku uz pomoć fizičkih parametara za našu Jaco ruku j2n6s300 korištenu u ovom diplomskom radu, zapisuje tablicu Denavit-Hartenberg parametara za izračun krajnje pozicije (x, y, z). Nakon toga inicijalizira matricu transformacije (jedinična matrica) te za svaki stupanj slobode robotske ruke množimo matricu transformacija da dobijemo transformaciju cijelog kinematičkog lanca. Konačno iz matrice izvlačimo samo krajnji desni redak jer u obzir ne uzimamo rotaciju radi kompleksnosti i taj redak nam predstavlja (x, y, z) poziciju izvršnog čvora ruke u Kartezijevom koordinatnom sustavu.

Sada kada imamo izračunatu trenutnu poziciju izvršnog čvora ruke, sljedeću odnosno traženu poziciju možemo prikazati kao trenutna_poziciju + np.array([delta, 0.0, 0.0]) (prvi predani parametar te funkcije). Izračunamo razliku trenutne i tražene pozicije da bismo odredili željeni pomak. Koristimo pseudoinverz Jacobijan matrice, koja se numerički računa funkcijom čiji je kod prikazan na slici (Sl 1.9), da nađemo brzinu u konfiguracijskom prostoru koja će ostvariti pomak u željenom smjeru npr. (+ delta) u x smjeru. Konačno možemo odrediti konfiguraciju robota koja ostvaruje taj željeni pomak i nju poslati robotu na izvršavanje. Ova konfiguracija je zapravo aproksimacija i ne ostvaruje potpuno željenu poziciju izvršnog čvora, ali je za male pomake dovoljno dobra.

```
def numerical_jacobian(conf):
    eps = 0.00001
    J = np.zeros((3, 6))
    for i in range(6):
        delta = np.zeros(6)
        delta[i] = delta[i] + eps
        J[:, i] = (forward_kinematics(conf + delta) - forward_kinematics(conf - delta)) / (2 * eps)
    return J
```

Sl 1.9 Isječak koda funkcije za numerički izračun Jacobijan matrice

Kada ROS čvor iz aplikacije iz kontrolera u Kartezijevom koordinatnom sustavu primi naredbu „SetPose x y z“ za postavljanje izvršnog čvora ruke na konkretne predane vrijednosti za izračun inverzne kinematike poziva se funkcija za iterativni izračun inverzne kinematike `inverse_kinematics(x, y, z, prefix, nbJoints)`

Kao i u funkciji za postizanje diferencijalnih pomaka, na početku funkcije za izračun inverzne kinematike čiji se kod nalazi na slici (Sl 1.10), najprije se iz ROS teme /j2n6s300/joint_states dohvaća trenutna konfiguracija robota koja predstavlja početnu poziciju zglobova robota. Nakon toga iterativno se uspoređuje trenutna pozicija izvršnog čvora s željenom pozicijom. Ako je razlika manja od unaprijed definiranog praga epsilon, postupak se prekida. Razlika između ciljane i trenutne pozicije koristi se za izračunavanje

potrebnih promjena u konfiguraciji zglobova pomoću pseudoinverza Jakobijan matrice isto kao i kod funkcije za postizanje diferencijalnih pomaka. Konačno možemo odrediti konfiguraciju robota koja ostvaruje taj željeni pomak i nju poslati robotu na izvršavanje.

```
def inverse_kinematics(x, y, z, prefix, nbJoints):
    msg = rospy.wait_for_message('/j2n6s300/joint_states', JointState)
    current_conf = np.array(msg.position)[0:6]
    eps = 0.01
    pos = np.array([x, y, z])
    for i in range(10000):
        current_pos = forward_kinematics(current_conf)
        if np.sum(np.abs(current_pos - pos)) < eps:
            break
        vdot = (pos - current_pos)
        qdot = np.linalg.pinv(numerical_jacobian(current_conf)).dot(vdot)
        current_conf = current_conf + qdot * 0.1
    moveJoint(current_conf, prefix, nbJoints)
```

Sl 1.10 Isječak koda funkcije za iterativni postupak izračuna inverzne kinematike

1.3.2. Primjeri korištenja za upravljanje robotskom rukom u laboratoriju

Za upravljanje u Kartezijevom koordinatnom sustavu u ROS čvoru joints_pose_client.py koristi se inverzna kinematika. Kada ROS čvor iz aplikacije iz kontrolera u Kartezijevom koordinatnom sustavu primi neku od naredbi kao što su „x_pos, y_pos, z_pos“ potrebno je zamaknuti izvršni čvor ruke za npr. naredbu „x_pos“ na poziciju (x_trenutni + delta, y_trenutni, z_trenutni), a za to nam treba inverzna kinematika kako bismo izračunali za koliko treba promijeniti vrijednost pojedinog zgloba da bi se izvršni čvor ruke pomaknuo u pozitivnom smjeru x osi za (+ delta) vrijednost. Isti princip vrijedi i za naredbe „x_neg, y_neg, z_neg“ samo se njihove vrijednosti mijenjaju za (- delta). U ovom ROS čvoru, za izračunavanje inverzne kinematike koristi se već gotova datoteka pose_action_client.py iz `catkin_ws/src/kinova_demo/nodes/kinova_demo` direktorija. Njezin poziv za npr. x_pos izgleda ovako:

```
command = f'python3 {script_path}pose_action_client.py
   j2n6s300 mrad {x+deltaP} {y} {z} {roll} {pitch} {yaw}'
os.system(command)
```

Modul os omogućava da naš program pokrene vanjski proces (u ovom slučaju drugu Python skriptu). Naredba za pokretanje skripte koristi f-string za dinamičko popunjavanje varijabli. Vrijednosti koje se šalju naredbom su:

- {script_path} - puna putanja do datoteke pose_action_client.py, koja se koristi za slanje Kartezijevih ciljeva.
- j2n6s300 - tip robota (Kinova Jaco 6 DOF model).
- mrad - jedinica za ulazne vrijednosti, u ovom slučaju metri za poziciju i radijani za orijentaciju.
- {x+deltaP}, {y}, {z} - pozicija izvršnog člana ruke u Kartezijevom prostoru (uzimaju se trenutne vrijednosti s pomakom delta u pozitivnom smjeru x-osi).
- {roll}, {pitch}, {yaw} - orijentacija izvršnog čvora izražena Eulerovim kutovima u radijanima.

Primanjem naredbe skripta pose_action_client.py najprije poziva funkciju `getcurrentCartesianCommand(prefix)` (Sl 1.11) kako bi dohvatila i spremila trenutnu poziciju i orijentaciju izvršnog čvora.

```
def getcurrentCartesianCommand(prefix_):
    # wait to get current position
    topic_address = '/' + prefix_ + 'driver/out/cartesian_command'
    rospy.Subscriber(topic_address, kinova_msgs.msg.KinovaPose, setcurrentCartesianCommand)
    rospy.wait_for_message(topic_address, kinova_msgs.msg.KinovaPose)
    print('position listener obtained message for Cartesian pose. ')
```

Sl 1.11 Prikaz funkcije za dohvaćanje trenutnog stanja izvršnog čvora ruke

Nakon što smo spremili trenutno stanje izvršnog čvora, poziva se naredba

`pose_mq, pose_mdeg, pose_mrad = unitParser(args.unit, args.pose_value, args.relative)`

Funkcija `unitParser(unit_, pose_value_, relative_)` konvertira ulazne vrijednosti pozicije i orijentacije u odgovarajući format. Podržava tri različite vrste ulaza:

- mq: pozicija (u metrima) + Kvaternion.
- mdeg: pozicija (u metrima) + Eulerovi kutovi (u stupnjevima).
- mrad: pozicija (u metrima) + Eulerovi kutovi (u radijanima)

Ako su ulazni podaci zadani u Eulerovim kutovima onda orijentaciju konvertira u kvaternion. Kad su svi ulazni podaci dobro konvertirani, podaci o trenutnoj poziciji uzimaju se iz pose_mq i zatim se to šalje kao ulazni parametri u funkciju `cartesian_pose_client(poses[:3], poses[3:])`

Funkcija `cartesian_pose_client(position, orientation)` šalje kartezijevu poziciju i orijentaciju izvršnog čvora ruke, u obliku kvaterniona, akcijskom poslužitelju na ROS temu /j2n6s300_driver/pose_action/tool_pose kao što je prikazano u kodu na slici (Sl 1.12). Akcijski poslužitelj obrađuje kartezijevu naredbu i koristi ugrađene algoritme za rješavanje inverzne kinematike, kako bi izračunao potrebne kutove zglobova. ROS zatim šalje te kutove upravljačkim sustavima robota, koji zatim pokreću zglobove kako bi izvršni čvor došao u zadanu poziciju i orijentaciju.

```
def cartesian_pose_client(position, orientation):
    """Send a cartesian goal to the action server."""
    action_address = '/' + prefix + 'driver/pose_action/tool_pose'
    client = actionlib.SimpleActionClient(action_address, kinova_msgs.msg.ArmPoseAction)
    client.wait_for_server()

    goal = kinova_msgs.msg.ArmPoseGoal()
    goal.pose.header = std_msgs.msg.Header(frame_id=(prefix + 'link_base'))
    goal.pose.pose.position = geometry_msgs.msg.Point(
        x=position[0], y=position[1], z=position[2])
    goal.pose.pose.orientation = geometry_msgs.msg.Quaternion(
        x=orientation[0], y=orientation[1], z=orientation[2], w=orientation[3])

    # print('goal.pose in client 1: {}'.format(goal.pose.pose)) # debug

    client.send_goal(goal)

    if client.wait_for_result(rospy.Duration(10.0)):
        return client.get_result()
    else:
        client.cancel_all_goals()
        print('        the cartesian action timed-out')
        return None
```

Sl 1.12 Funkcija za slanje Kartezijevih vrijednosti na ROS poslužitelj i vraćanje odgovarajuće vrijednosti kutova zglobova ruke

Pristup korištenjem skripte pose_action_client.py omogućuje jednostavno i modularno upravljanje robotom. Glavni program ostaje odvojen od složene logike upravljanja zglobovima i inverzne kinematike, dok ROS akcijski poslužitelj osigurava preciznost i pouzdanost u izvršavanju pokreta. Ovaj sustav omogućuje intuitivno upravljanje robotskom rukom, čineći ga pogodnim za širok raspon primjena, uključujući teleoperaciju i autonomne zadatke.

2. Android aplikacija za teleoperaciju

Kao sučelje za teleoperaciju odabrana je Android aplikacija iz razloga što velika većina današnjih uređaja koristi baš Android operacijski sustav. Za razvoj aplikacije odabrano je razvojno okruženje Android studio koje je posebno dizajnirano za razvoj Android aplikacija. Android studio ([4]) nudi niz značajki koje značajno olakšavaju izradu Android aplikacija, neke od njih su:

- jedinstveno okruženje u kojem se mogu razvijati aplikacije za sve Android uređaje
- brz emulator s bogatim brojem značajka
- fleksibilan sustav izrade temeljen na Gradle-u
- Live Edit koji omogućuje ažuriranje komponenata u emulatorima i fizičkim uređajima u stvarnom vremenu

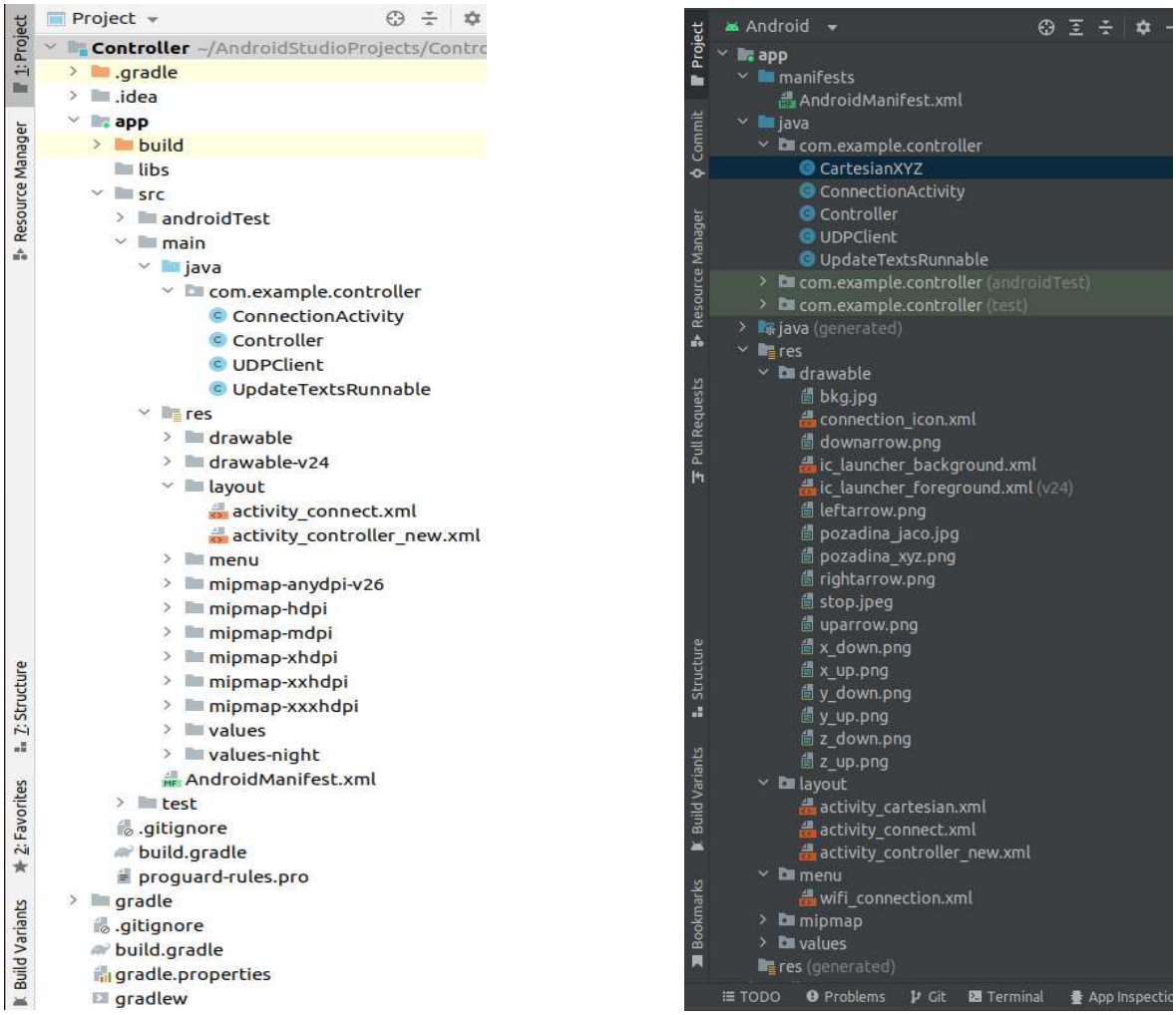
2.1. Razvoj aplikacije

Aplikacija je nastala kao nadogradnja aplikacije iz mog vlastitog završnog rada ([3]), a ostvarena kroz Android studio verzije Koala | 2024.1.1 i pisana u razvojnom jeziku Java. Nije mijenjan minimalni SDK (Software Development Kit) već je ostavljen API 21: Android 5.0 (Lolipop) jer je on podržan na 94,1% uređaja. Na slici (Sl 2.1) nalazi se prikaz Android projekta aplikacije iz završnog rada i struktura projekta nadograđene aplikacije. Osim što je vidljivo u strukturi da su neke klase dodane, promijenjen je čitav dizajn aplikacije. Prikaz strukture projekta organiziran je po modulima radi bržeg pristupa datotekama projekt. Svaki od modula u aplikaciji sadrži direktorije ([4]):

- manifesti
- java
- res

Manifest sadrži AndroidManifes.xml datoteku u kojoj je definiran naziv same aplikacije, sve njezine aktivnosti i odabrana je glavna aktivnost, usluge i dopuštenja koja su potrebna za pokretanje aplikacije. U nadograđenoj aplikaciji u manifest je dodana nova aktivnost za teleoperaciju u Kartezijevom koordinatnom sustavu dok ostalo nije mijenjano.

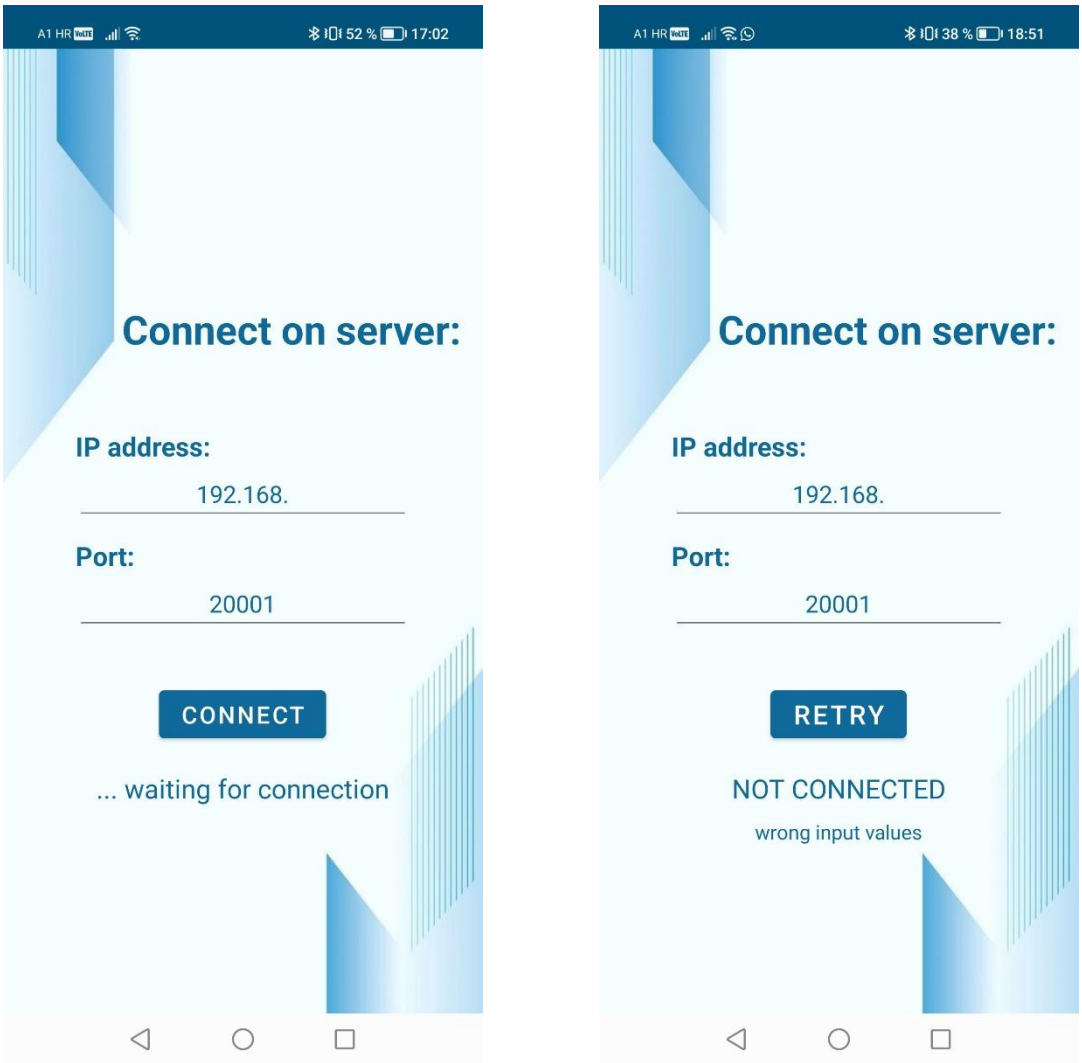
Java direktorij sadrži sve datoteke izvornog koda u Javi, a tu spadaju sve aktivnosti i klase koje aplikacija koristi. Aktivnosti u Android aplikaciji su klase koje direktno preko svojeg korisničkog sučelja komuniciraju s korisnicima. Postoji samo jedna glavna aktivnost (MainActivity) a to je kao i u prethodnoj verziji aplikacije ConnectionActivity i ona se otvara kao početna aktivnosti tj. početni zaslon kojeg korisnik vidi kad pokrene aplikaciju. U java direktoriju također se nalaze i testni kodovi koji služe samom testiranju koda aplikacije. Na slici (Sl 2.1) može se vidjeti da se u java direktorij dodala klasa CartesianXYZ koja služi za teleoperaciju u Kartezijevom koordinatnom sustavu. Zadnji od direktorija je res direktorij koji sadrži sve ostale resurse kao što su slike, prikaze rasporeda komponenti (dizajn) za svaku od aktivnosti, izbornike i sve ostalo povezano sa samim izgledom i dizajnom unutar aplikacije. Resursima se može pristupati kroz aktivnosti i klase preko njihovih definiranih id-eva kroz klasu R (npr. R.button1). Na strukturi projekta vidljivo je da je dodan novi raspored komponenti (layout) activity_cartesian.xml gdje je definiran raspored komponenti nove aktivnosti kontrolera CartesianXYZ.



Sl 2.1 Struktura projekta aplikacije iz završnog rada (lijevo) i nadograđene aplikacije (desno)

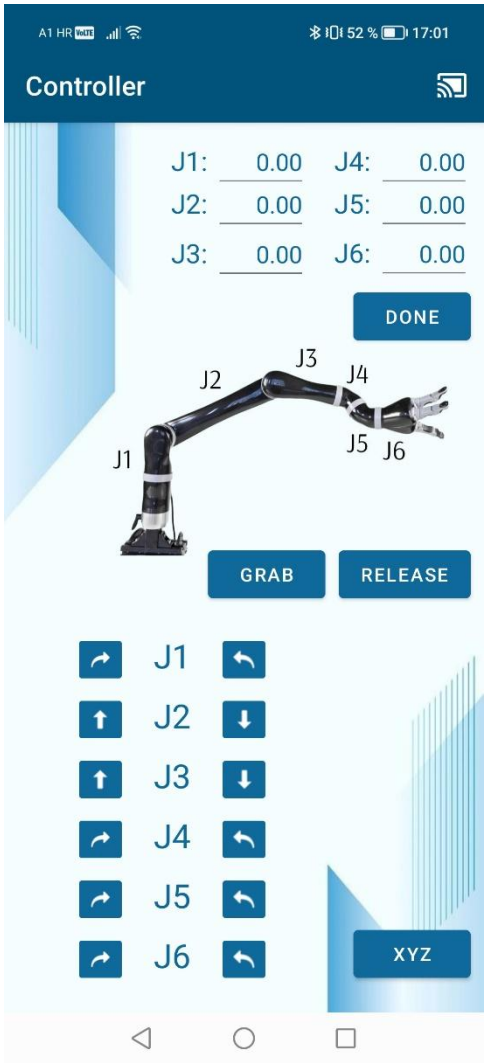
2.2. Izgled sučelja i opis funkcionalnosti aplikacije

Novo implementirana aplikacija sastoji se od tri aktivnosti. Prilikom ulaska u aplikaciju otvara se glavna aktivnost a to je ConnectionActivity (Sl 2.2). Funkcionalnosti ove aktivnosti u odnosu na istu aktivnost iz završnog rada nisu promijenjene, promijenjen je samo dizajn korisničkog sučelja. Ova aktivnost omogućuje nam da se povežemo na računalo na kojem se nalazi ROS kako bismo mogli kroz ostale dvije aktivnosti koje nam služe kao kontroleri, slati naredbe i tako pokretati ruku bilo u Gazebo simulatoru ili fizičku ruku u laboratoriju. Na korisničkom sučelju ove aktivnosti nalaze se dva polja za unos i jedan gumb za slanje tih upisanih podataka na server. Kako bi povezivanje bilo uspješno, treba unijeti ispravu IP adresu servera i port na kojem on sluša. Ako uneseni podaci nisu valjani, prikazuje se gumb za ponovni pokušaj koji resetira polja za unos.



Sl 2.2 Glavna aktivnost aplikacije za povezivanje sa serverom

Kad su uneseni podaci valjani i povezivanje uspije, aplikacija automatski otvara aktivnost kontrolera u koordinatnom sustavu robota. Izgled aktivnosti tog kontrolera koje je u strukturi projekta pod nazivnom Controller prikazan je na slici (Sl 2.3). Aktivnost kontrolera sastoji se od nekoliko ključnih komponenti. Na vrhu u desnom kutu u zaglavlju aktivnosti postoji izbornik koji nam omogućuje da prekinemo vezu sa serverom klikom na gumb i aplikacija se vraća na glavnu aktivnost za povezivanje na server. Ispod zaglavlja na svijetlom dijelu zaslona postoji nekoliko komponenti za teleoperaciju robotskom rukom postavljenjem njezinih zglobova (joints). Ruka koja se u ovom projektu koristi je ruka Jaco Gen 2 tipa j2n6s300 koja ima 6 zglobova i 3 prsta pa zato kontroler sadrži komponente prilagođene tim specifikacijama. Gore desno imamo polja za upis konkretnih vrijednosti za svaki od šest zglobova naše robotske ruke. Klikom na gumb DONE šalju se preko servera sve te upisane vrijednosti do ROS čvora koji zatim svaki od zglobova ruke postavlja na točno za njega poslanu vrijednosti.

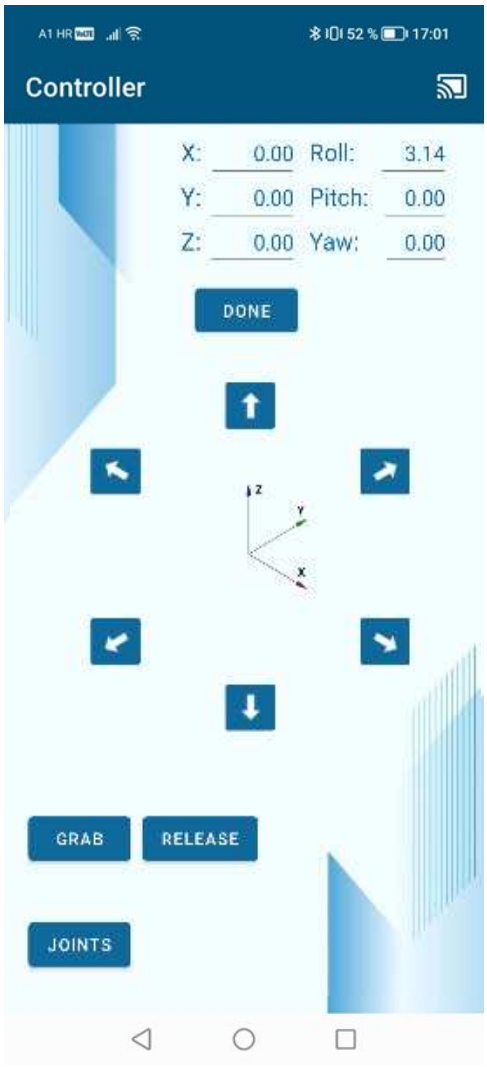


Sl 2.3 Izgled aktivnosti kontrolera u koordinatnom sustavu robota

Na desnom donjem dijelu aktivnosti nalazi se dvanaest strelica, po dvije za svaki od zglobova koje ga ovisno o tome koja je pritisnuta pokreću gore/dolje ili zakreću ulijevo ili udesno. Za lakše snalaženje sa strelicama, u sredini zaslona postoji slika robotske ruke sa naznačenim imenima zglobova. Ispod slike ruke s desne strane nalaze se dva gumba koji služe za otvaranje/zatvaranje šake to su gumbi s tekstom grab i release. Skroz dolje desno nalazi se gumb s natpisom XYZ. Klikom na taj gumb otvara se aktivnost kontrolera u Kartezijevom koordinatom sustavu (Sl 2.4).

Aktivnost u Kartezijevom koordinatom sustavu implementirana je kako bi se korisnicima koji se prvi put susreću s robotskom rukom ili ne znaju dobro građu robota i raspored zglobova, omogućilo lakšu teleoperaciju po x, y, z koordinatnim osima koje se koriste u svakodnevnom životu kao gore/dolje, naprijed/natrag, lijevo/desno. Ovakvo upravljanje za većinu ljudi intuitivnije je i lakše za shvatiti od upravljanja u nekom drugom, nama neprirodnom, koordinatnom sustavu. Ova aktivnost također se sastoji od nekoliko komponenata koje omogućuju upravljanjem po x, y, z osi na više načina. Prvi način upravljanja je unošenjem konkretnih vrijednosti u tekstualna polja x, y, z, roll, pitch i yaw koja se nalaze u gornjem desnom dijelu aktivnost. Klikom na gumb DONE šalju se preko servera sve te upisane vrijednosti do ROS čvora koji zatim preko inverzne kinematike izračunava vrijednosti na koje mora postaviti pojedini od zglobova da se izvršni član ruke postavi na točno zadanu vrijednosti.

U središnjem dijelu aktivnosti nalazi se slika triju osi Kartezijevog koordinatnog sustava i oko nje postoji 6 strelica, po dvije u svakom smjeru koje omogućuju kretanje izvršnog člana ruke gore/dolje, naprijed/natrag ili lijevo/desno. U donjem dijelu nalaze se tri gumba. Prva dva su kao i na prethodno opisanoj aktivnosti kontrolera u koordinatnom sustavu robota, gumbi za otvaranje/zatvaranje šake, a posljednji je gumb JOINTS koji nas vodi na aktivnost kontrolera u koordinatnom sustavu robota.

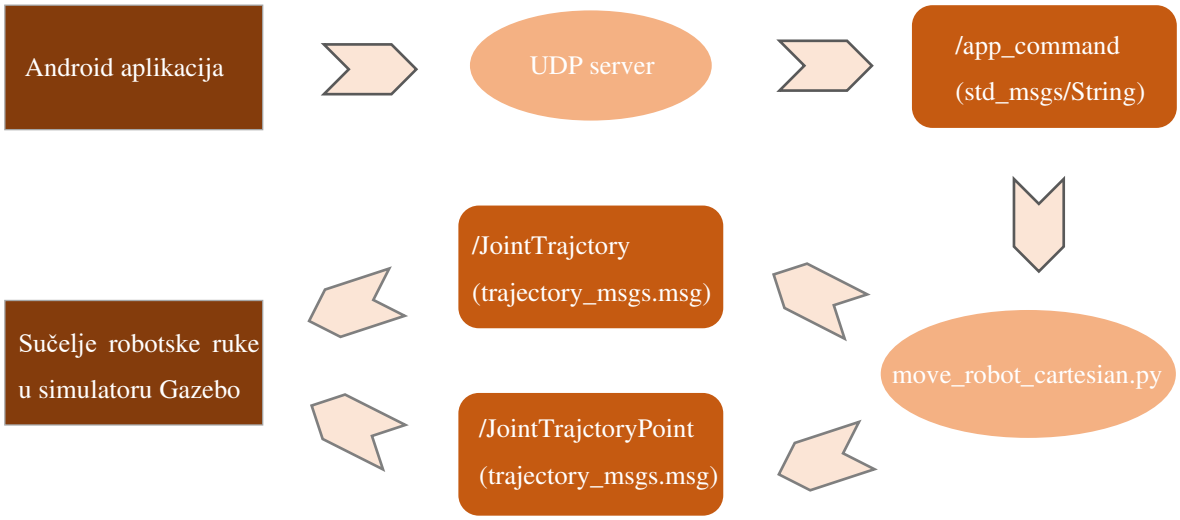


Sl 2.4 Izgled aktivnosti kontrolera u Kartezijevom koordinatnom sustavu

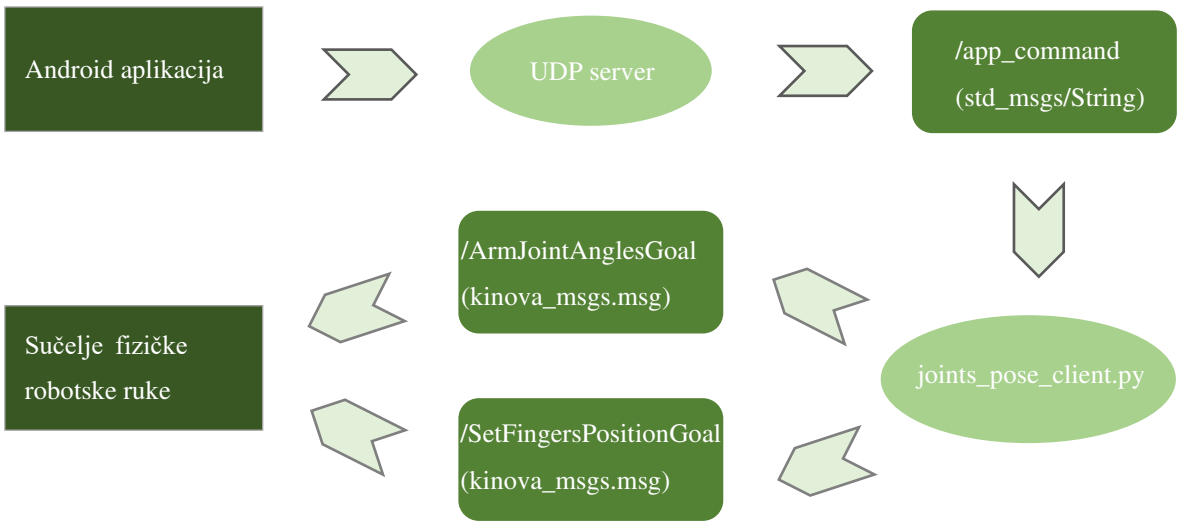
2.3. Komunikacija s robotom

Komunikacija Android aplikacije s robotom odvija se preko UDP servera. UDP (User Datagram Protocol) ([2]) je transportni protokol koji pruža uslugu prijenosa informacije uz najmanje moguće kašnjenje te se primjenjuje kad je važnija vremenska transparentnost nego semantička jer je riječ o nespojenoj i nepouzdanoj usluzi. Aplikacija se preko UDP servera povezuje na lokalnu mrežu računala na kojem imamo instaliran ROS i preko ROS čvora šaljemo upute za pokretanje robotskoj ruci. Podaci koji putuju preko UDP servera su oblika UDP datagrama i oni se šalju preko servera do ROS čvora svaki puta kad se korisnik u aplikaciji pritisne bilo koji od gumbova. UDP datagram je definiran Internet protokolom i to je blok podataka koji se šalje na mrežu kao jedna poruka. Poruke se primaju na server tako dugo dok je aplikacija povezana sa njim, kad aplikacija prekine vezu, server prestaje primati

njezine poruke i sluša na svome portu kad će doći novi zahtjevi klijenata. U ovom projektu u komunikacijski kanal spadaju Android aplikacija i dva ROS čvora pisana u programskom jeziku Python, prvi je UDP_server.py, a za upravljanje robotskom rukom imamo dvije varijante ROS čvora ovisno o tome radi li se o upravljanju rukom u Gazebo simulatoru (move_robot_cartesian.py) ili o upravljanju fizičkom rukom u laboratorijskom okruženju (joints_pose_client.py) pa iz toga razloga postoje dva dijagrama komunikacijskog kanala.



Sl 2.5 Dijagram komunikacijskog kanala za teleoperaciju u simulatoru Gazebo



Sl 2.6 Dijagram komunikacijskog kanala za teleoperaciju fizičke ruke

3. Implementacijski detalji

3.1. Korištene tehnologije

3.1.1. ROS

ROS ([8]) je meta-operacijski sustav otvorenog koda za programiranje robota koji pruža usluge poput:

- implementacije često korištenih funkcija
- apstrakcije hardvera
- prijenosa poruka između procesa
- kontrole uređaja niske razine
- upravljanja paketima

U nekim aspektima sličan je robotskim okvirima poput ([8]) Microsoft Robotics Studija, Playera i Yarpa. Također sadrži biblioteke i alate za pisanje, izgradnju, projektiranje i dobivanje koda na više računala. Implementira nekoliko različitih stilova komunikacije te uključuje komunikaciju preko usluga. Preko tema pruža asinkroni protok podataka, a podatke pohranjuje na poslužitelju parametara.

Primarni cilj ([8]) koji ROS želi postići jest podržati ponovnu upotrebu koda u razvoju i istraživanju robotike. On je distribuirani okvir procesa koji omogućuje tijekom izvođenja oslabljeno povezivanje i pojedinačno dizajniranje izvršnih datoteka. Ti procesi mogu se grupirati u skupove i pakete koji se kasnije mogu jednostavno distribuirati i dijeliti. ROS također podržava sustav podijele koda u repozitorije koji omogućuju distribuciju te takav dizajn omogućuje nezavisne odluke o implementaciji i razvoju, od razine datoteka do razine zajednice, ali konačno, ROS infrastrukturnim alatima, sve se može spojiti. Postoje još neki ciljevi ROS okvira koji idu u prilog primarno cilju dijeljenja i suradnje, a neki od njih su:

- ROS – agnostic knjižnice: preferirani razvojni model za pisanje istih biblioteka s čistim funkcionalnim sučeljem
- jezična neovisnost – lako implementiranje u bilo koji moderni programski jezik
- jednostavno testiranje – ugrađeni testni okvir „rostest“ koji olakšava pokretanje testnih uređaja
- skalabilnost – prikladan za velike razvojne procese i velike runtime sustave

- tankost – dizajniran da bude što tanji, kod pisan u ROS-u može se koristiti i lako integrirati s drugim za robote dizajniranim softverskim okvirima

Trenutno ROS radi samo na Unix platformama i prvenstveno je testiran na Ubuntu i Mac OS X sustavima, iako dalje podršku i za neke Linux platforme i drugo.

3.1.2. Gazebo

Gazebo ([5]) je simulacijsko okruženje s kompletnim paketom alata, razvojnih datoteka i usluga u oblaku koje omogućuju lakšu simulaciju. Neke korisne značajke koje Gazebo posjeduje jesu:

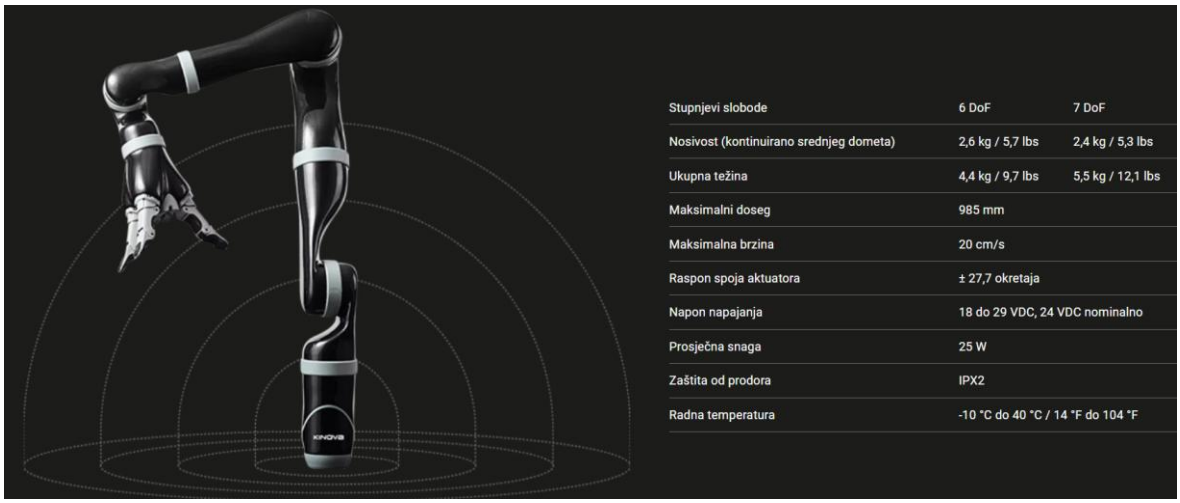
- Dinamičko učitavanje sredstava simulacije
    - Gazebo može automatski učitavati i uklanjati sredstva simulacije koristeći prostorne informacije kako bi se poboljšala izvedba
- Distribuirana simulacija
    - kako bi poboljšao performanse Gazebo podržava korištenje više poslužitelja
- Podesiva izvedba
    - omogućuje kontroliranje simulacije rada u stvarnom vremenu, hoće li biti brže od ili sporije od stvarnog vremena

Gazebo omogućuje simulacije robota u složenim vanjskim i unutarnjim okruženjima koje su vrlo učinkovite i dosta precizne. Također nudi mogućnost dizajniranja robota, brzo testiranje algoritama, izvođenje regresijskih ispitivanja te treniranje sustava umjetne inteligencije korištenjem realnih scenarija. Njegova upotreba je velika jer je besplatan i uz to nudi prikladno programsko i grafičko sučelje, fizički je robusnog mehanizma te pruža visokokvalitetnu grafiku.

3.2. Jaco robotska ruka

Robotska ruka odabrana za korištenje u ovom diplomskom radu je Jaco robotska ruka tvrtke Kinova, koja je razvila više generacija robota, a odabrana ruka spada u generaciju robota Gen 2. Generacija Gen 2 ([6]) robota predstavlja jedinstvene, ultra-lagane i spretne robote s ugrađenim kontrolerom koji su jednostavni za korištenje i imaju sposobnost prilagodbe za sva radna okruženja i sve vrste scenarija. Robot Gen 2 generacije može manipulirati objektima u nekom trodimenzionalnom prostoru koristeći vrlo malo energije, siguran je za rad s ljudima, prenosiv i vrlo lagan, sve to omogućuje mu da korisnicima ponudi

performanse, jednostavnost i fleksibilnost u jednom. Tehnički podaci i izgled same ruke prikazani su na slici (Sl 3.1)



Stupnjevi slobode	6 DoF	7 DoF
Nosivost (kontinuirano srednjeg dometa)	2,6 kg / 5,7 lbs	2,4 kg / 5,3 lbs
Ukupna težina	4,4 kg / 9,7 lbs	5,5 kg / 12,1 lbs
Maksimalni doseg	985 mm	
Maksimalna brzina	20 cm/s	
Raspon spoja aktuatora	± 27,7 okretaja	
Napon napajanja	18 do 29 VDC, 24 VDC nominalno	
Prosječna snaga	25 W	
Zaštita od prodora	IPX2	
Radna temperatura	-10 °C do 40 °C / 14 °F do 104 °F	

Sl 3.1 Izgled i tehnički podaci Kinova Gen2 robota [6]

Robotska ruka Gen2 može imati 6 ili 7 DOF (stupnjeva slobode) obično ili sferno. Broj stupnjeva slobode označuje broj zglobova ruke, ako ruka ima 6 DOF znači da je rukom moguće upravljati uzduž 6 osi koje predstavljaju rameni, laktni i ručni zglob, a ako ima 7 DOF upravljiva je uzduž 7 osi. Usporedba nekih specifikacija za ruku sa 6 ili 7 DOF dana je na slici (Sl 3.1). Robotska ruka ima dvije varijante izgleda šake, sadrži 2 ili 3 prsta. U ovom projektu odabrana je robotska ruka sa 6 DOF i 3 prsta.

Takvo upravljanje rukom koje omogućuje brojne mogućnosti pokreta oponaša svestranost i glatkoću ljudske ruke pa joj je s toga glavni cilj poboljšati kvalitetu života za korisnike s fizičkim nedostacima kako bi ostvarili svoj puni potencijal i poboljšali kvalitetu svog života. Dizajnirana je tako da se može integrirati u potpunosti u svakodnevni život, npr. može se montirati na motorizirana invalidska kolica.

3.3. Način upravljanja rukom

Upravljanje rukom u simulatoru Gazebo vrši move_robot_cartesian.py ROS čvor na način da prima naredbe tipa std_msgs/String i sadrži callback funkciju kojom se mijenja parametar predanog zgloba ili u Kartezijevom koordinatnom sustavu pomak po određenoj

osi za +/- 0.1 radijan dokle god je strelica pritisnuta. Naredbe koje taj ROS čvor može primiti od Android aplikacije su:

1. Aktivnost u koordinatnom sustavu robota:

- „l1-l6“ – predani zglob zakreće u smjeru kazaljke na satu ili ga pomiče gore ovisno o kojem se zglobu radi
- „r1-r6“ – predani zglob zakreće u smjeru kazaljke na satu ili ga pomiče gore ovisno o kojem se zglobu radi
- „SetJoints“ – uz to u naredbi se nalaze i pozicije za svaki od zglobova
  - svaki zglob se postavlja na vrijednost koja je zaprimljena za njegovu poziciju
- „xyz“ otvara aktivnost kontrolera u Kartezijevom koordinatnom sustavu

2. Aktivnost u Kartezijevom koordinatnom sustavu

- „x_pos“, „y_pos“, „z_pos“ – pomiču robotsku ruku u pozitivnom smjeru po zadanoj osi tako dugo dok je strelica pritisnuta
- „x_neg“, „y_neg“, „z_neg“ – pomiču robotsku ruku u negativnom smjeru po zadanoj osi tako dugo dok je strelica pritisnuta
- „SetPose“ – uz to u naredbi se nalaze i vrijednosti x, y, z koordinata
  - pomoću inverzne kinematike izračunava se vrijednost na koju treba postaviti svaki od zglobova da bi zaprimljena x, y, z pozicija bila zadovoljena
- „joints“ – otvara aktivnost kontrolera u koordinatnom sustavu robota za pomicanje pojedinih zglobova

Zajedničke naredbe:

- „grab“ – zatvara šaku odnosno skuplja robotske ruke
- „release“ – otvara šaku odnosno rastvara prste robotske ruke

Upravljanje rukom fizičkom rukom u laboratoriju vrši joints_pose_client.py ROS čvor na način da prima naredbe tipa std_msgs/String i sadrži callback funkciju kojom se mijenja parametar predanog zgloba za +/- 11.0 stupnjeva, a za Kartezijev koordinatni sustavu pomak od 0.2 radijana u određenom smjeru dokle god je strelica pritisnuta. Naredbe koje ROS čvor može primiti od Android aplikacije su iste kao kod simulatora

3.4. Instalacija i pokretanje projekta

3.4.1. Pokretanje projekta za simulaciju u Gazebo simulacijskom okruženju

Kako bismo mogli pokrenuti simulaciju u Gazebo simulacijskom okruženju, najprije je potrebno preuzeti Gazebo pomoću naredbe

```
$ sudo apt-get install ros-indigo-gazebo-ros*.
```

Za omogućavanje upravljanja u Gazebo simulatoru potrebno je instalirati ROS-ov repozitorij upravljača pomoću naredaba:

```
$ sudo apt-get install ros-<distro>-gazebo-ros-control
$ sudo apt-get install ros-<distro>-ros-controllers*
```

Kad smo uspješno instalirali Gazebo simulator, potrebno je kreirati catkin workspace za pohranu kinova-ros paketa. Za kreiranje catkin workspace-a pozivaju se sljedeće naredbe:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

Nakon što se catkin workspace uspješno kreirao, sa githuba je potrebno skinuti kinova-ros paket koristeći sljedeću naredbu

```
$ git clone https://github.com/Kinovarobotics/kinova-ros
```

Na posljetku, aplikaciju za upravljanje i potrebne Python kodove napravljene u ovom diplomskom projektu preuzeti sa githuba koristeći naredbu

```
$ git clone
https://github.com/EvelynFurdi/JacoArmTeleoperation.git
```

Pokretanje simulacije i upravljanje robotom izbodi se kroz nekoliko terminala. U prvom terminalu potrebno je pokrenuti naredbu `$ roscore` kako bi se omogućila komunikacija između ROS čvorova. U drugom terminalu pokreće se UDP_server.py iz preuzetih kodova u mapi JacoArmTeleoperation kako bi se aplikacija povezala na računalo na kojem pokrećemo simulaciju. UDP_server se pokreće naredbom

```
$ python3 ./UDP_server.py.
```

Kad je server uspješno pokrenuti Android aplikacija se povezuje na njega koristeći ispravne vrijednosti za IP adresu i port. Nakon uspješnog povezivanje aplikacije preko servera, u 3 terminalu pokrećemo simulaciju robotske ruke Jaco j2n6s300 u Gazebo simulacijskom okruženju. Za simulaciju robotske ruke u Gazebo-u trebamo se najprije pozicionirati unutar catkin workspace-a za pristup kinova-ros paketu, za to se koristi naredba

```
$ cd catkin_ws/src/kinova-ros
```

i onda pokrećemo simulaciju ruke korištenjem naredbe

```
$ roslaunch kinova_gazebo robot_launch.launch
 kinova_robotType:=j2n6s300.
```

Za upravljanjem robotskom rukom preko aplikacije potrebno se pozicionirati u direktorij s preuzetim kodovima

```
$ cd ~/…/JacoArmTeleoperation/pythonFiles
```

(„ … “ treba zamijeniti sa putanjom gdje se direktorij JacoArmTeleoperation nalazi)

i tu pokrenuti naredbu `$ python3 move_robot_cartesian.py j2n6s300.`

Sve je spremno i pritiskom bilo koje tipke u aplikaciji u simulaciji se izvodi odgovarajuća naredba, robot se pokreće i to je vidljivo na simulaciji.

3.4.2. Pokretanje projekta za upravljanje fizičkom rukom u laboratoriju

Kako bismo pokrenuli fizičku ruku u laboratoriju, ruku treba preko USB kabla uključiti u računalo na kojem ćemo pokretati programe za upravljanje rukom. Za pristup ruci putem USB-a treba kopirati datoteku pravila udev 10-kinova-arm.rules iz ~/catkin_ws/src/kinova-ros/kinova_driver/udev u /etc/udev/rules.d/ korištenjem naredbe:

```
$ sudo cp kinova_driver/udev/10-kinova-arm.rules
 /etc/udev/rules.d/
```

Nakon toga catkin workspace kreira se jednako kao i kod simulacije u Gazebu i u njega se sprema kinova-ros pakete. Za kreiranje catkin workspacea pozivaju se sljedeće naredabe:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
```

```
$ catkin_make
```

Kao i kod simulacije potrebno je sa githuba je potrebno skinuti kinova-ros paket koristeći sljedeću naredbu

```
$ git clone https://github.com/Kinovarobotics/kinova-ros
```

Na posljetku, aplikaciju za upravljanje i potrebne python kodove napravljene u ovom diplomskom projektu preuzeti sa githuba koristeći naredbu

```
$ git clone
https://github.com/EvelynFurdi/JacoArmTeleoperation.git
```

Pokretanje i upravljanje fizičkom robotskom rukom također se izvodi kroz nekoliko terminala, gdje se samo jedna naredba razlikuje od pokretanja simulacije.

Prvi terminal: `$ roscore`

Drugi terminal:

```
cd ~/…/JacoArmTeleoperation/pythonFiles
```

(„ … “ treba zamijeniti sa putanjom gdje se direktorij JacoArmTeleoperation nalazi)

```
$ python3 ./UDP_server.py.
```

Treći terminal: `$ cd catkin_ws/src/kinova-ros`

i onda pokrećemo fizičku ruku korištenjem naredbe

```
$ roslaunch kinova_bringup kinova_robot.launch
 kinova_robotType:=j2n6s300.
```

Četvrti terminal :

```
$ cd ~/…/JacoArmTeleoperation/pythonFiles
```

(„ … “ treba zamijeniti sa putanjom gdje se direktorij JacoArmTeleoperation nalazi)

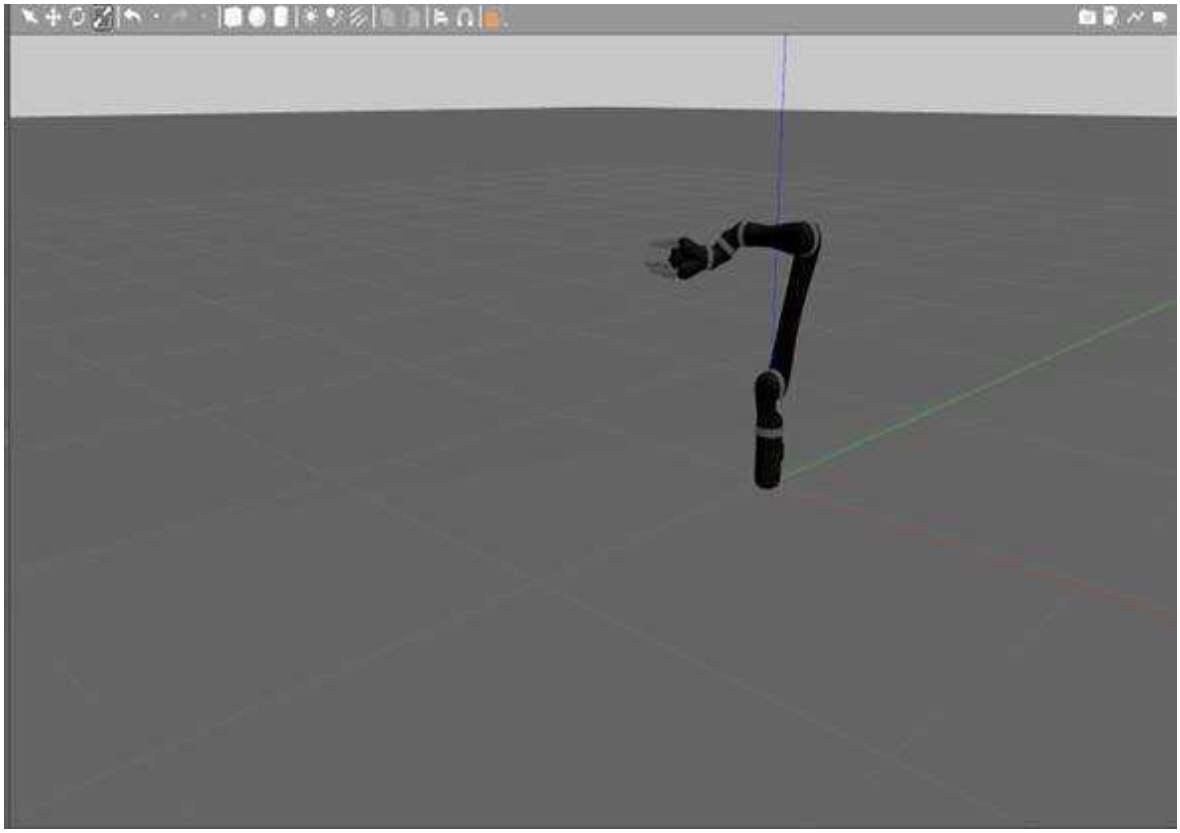
```
$ python3 joints_pose_client.py j2n6s300.
```

Sve je spremno i pritiskom bilo koje tipke u aplikaciji u simulaciji se izvodi odgovarajuća naredba, fizička ruka se pokreće.

4. Eksperimentalni rezultati

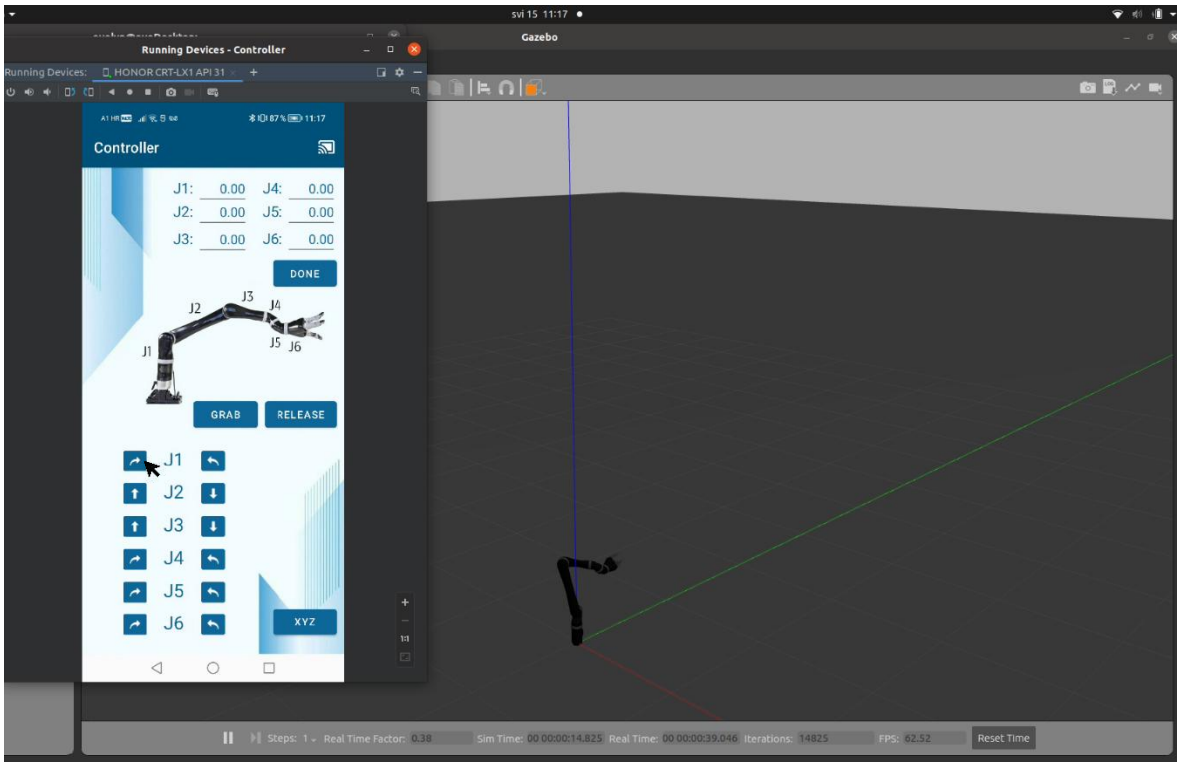
4.1. Teleoperacija u simulatoru Gazebo

Prilikom pokretanja simulacije robotske ruke u Gazebo ruka se nalazi u svojoj početnoj poziciji (Sl 4.1). U početnoj poziciji vrijednosti zglobova su {0.0, 2.9, 1.3, 4.2, 1.4, 0.0}

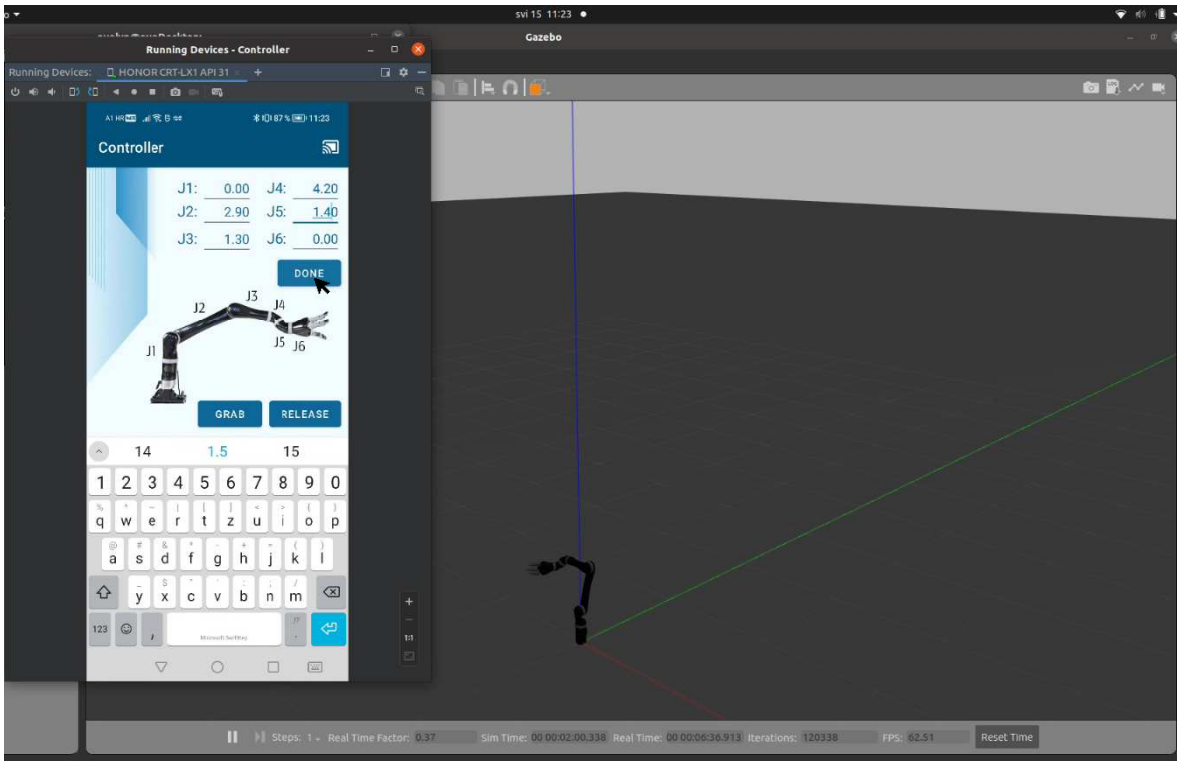


Sl 4.1 Jaco ruka u simulatoru Gazebo u svojoj početnoj poziciji

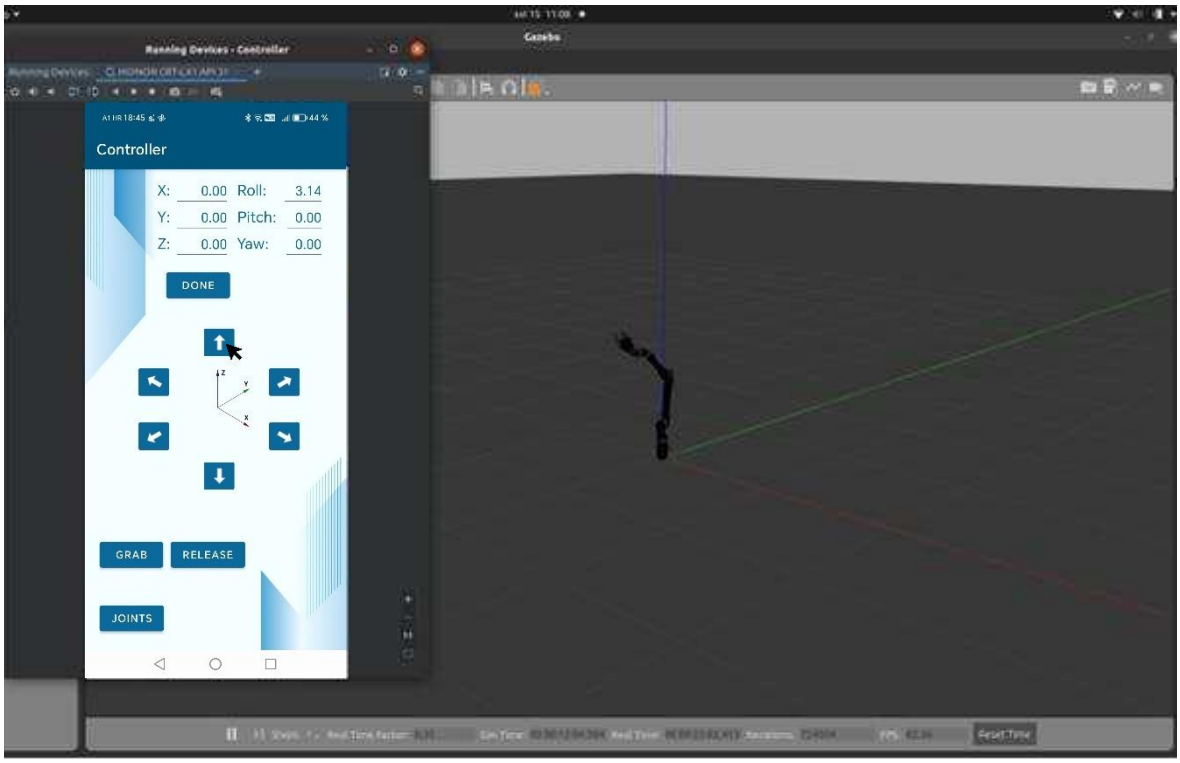
Slijedi prikaz rezultata simulacije pokretanja ruke u Gazebo simulacijskom okruženju u vidu niza slika (Sl 4.2 - Sl 4.6) kroz koje će biti prezentirana funkcionalnost aplikacije. Svaka od slika prikazuje ekran računala s prikazom zaslona aplikacije te što je u aplikaciji pritisnuto te rezultat pomaka ruke u Gazebo-u nakon toga.



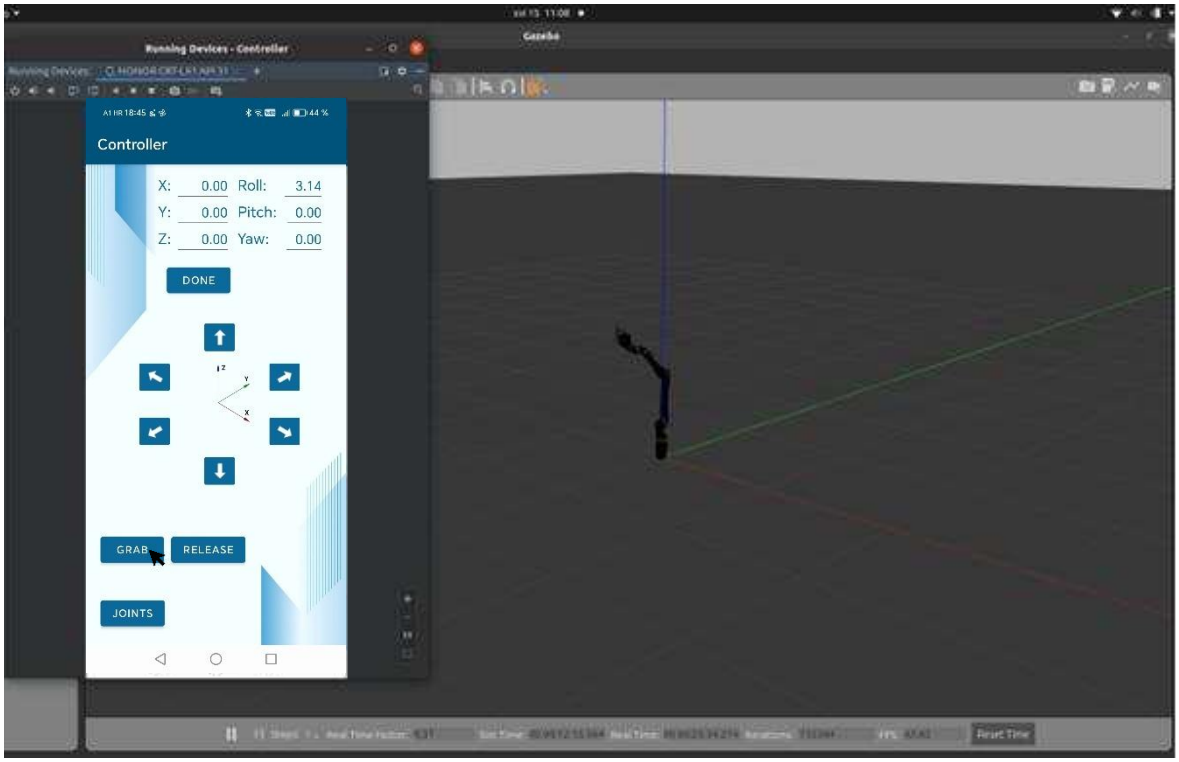
Sl 4.2 Prikaz položaja ruke nakon pritiska i otpuštanja tipke za zakretanje zgloba J1 u smjeru kazaljke na satu



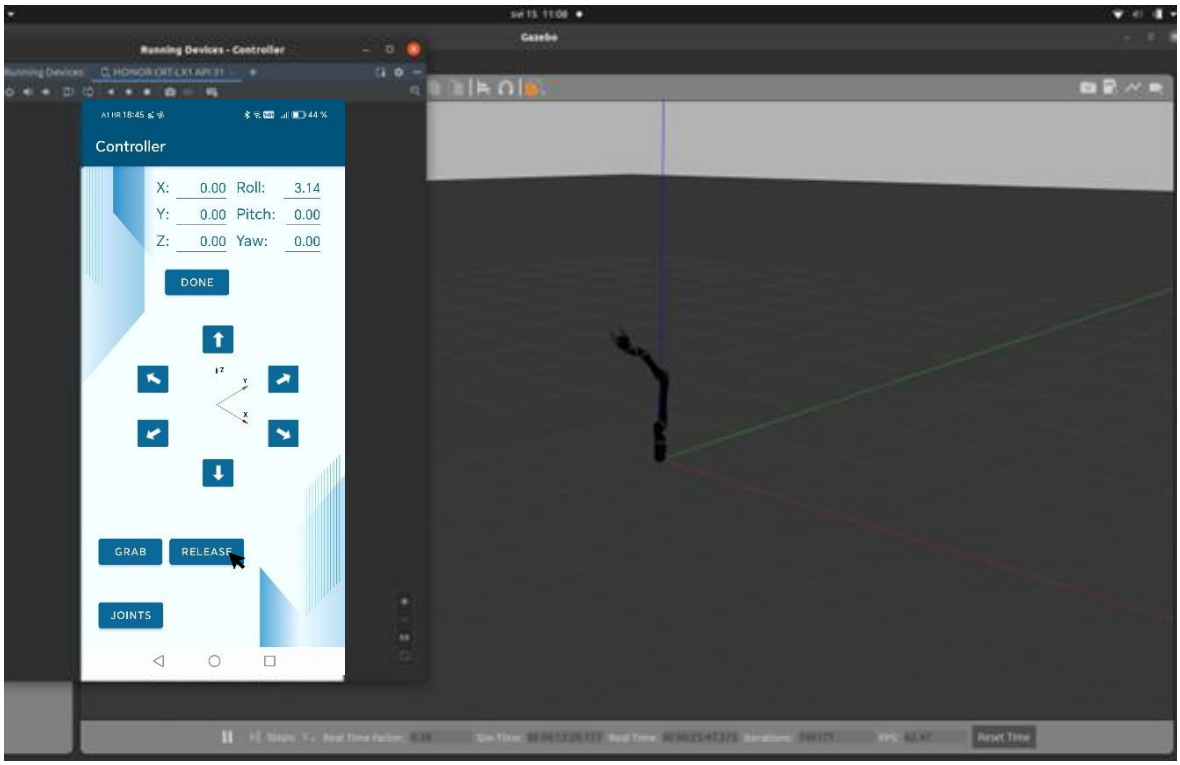
Sl 4.3 Prikaz položaja ruke nakon pritiska na gumb za postavljanje svih vrijednosti jointova (vrijednosti postavljene na home poziciju)



Sl 4.4 Prikaz položaja ruke nakon pritiska i otpuštanja strelice za pomicanje robotske ruke prema gore (u pozitivnom smjeru z osi)



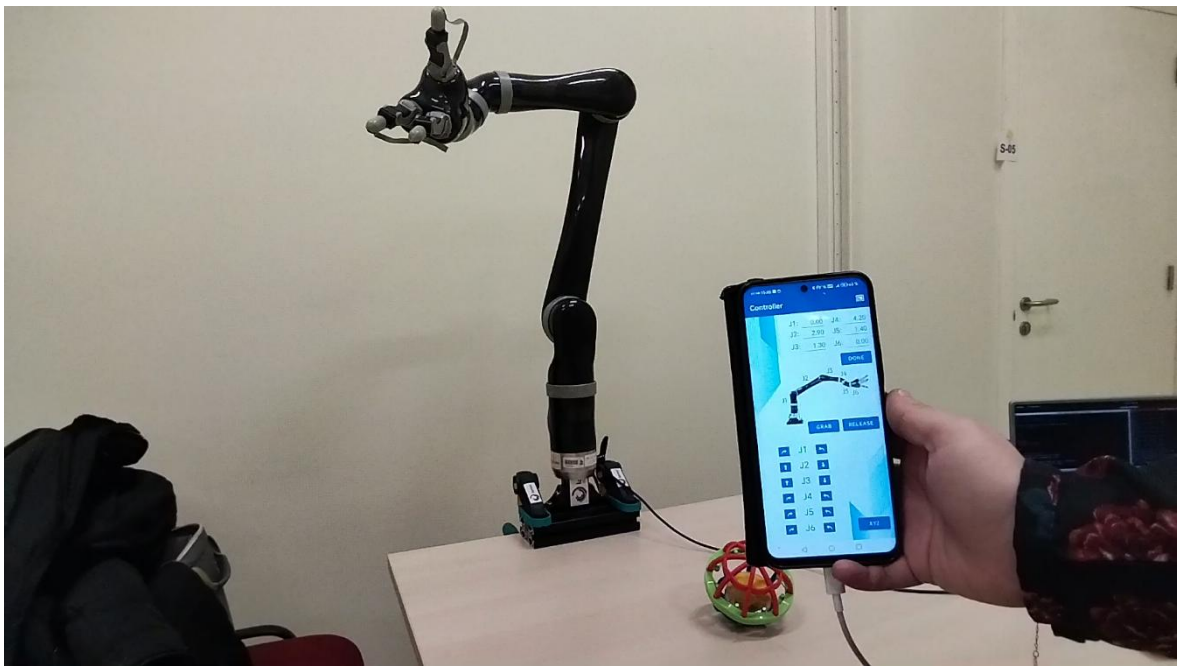
Sl 4.5 Prikaz položaja ruke nakon pritiska gumba grab (zatvori šaku)



Sl 4.6 Prikaz položaja ruke nakon pritiska na gumb release (otvaranje šake)

4.2. Teleoperacija u laboratorijskom okruženju

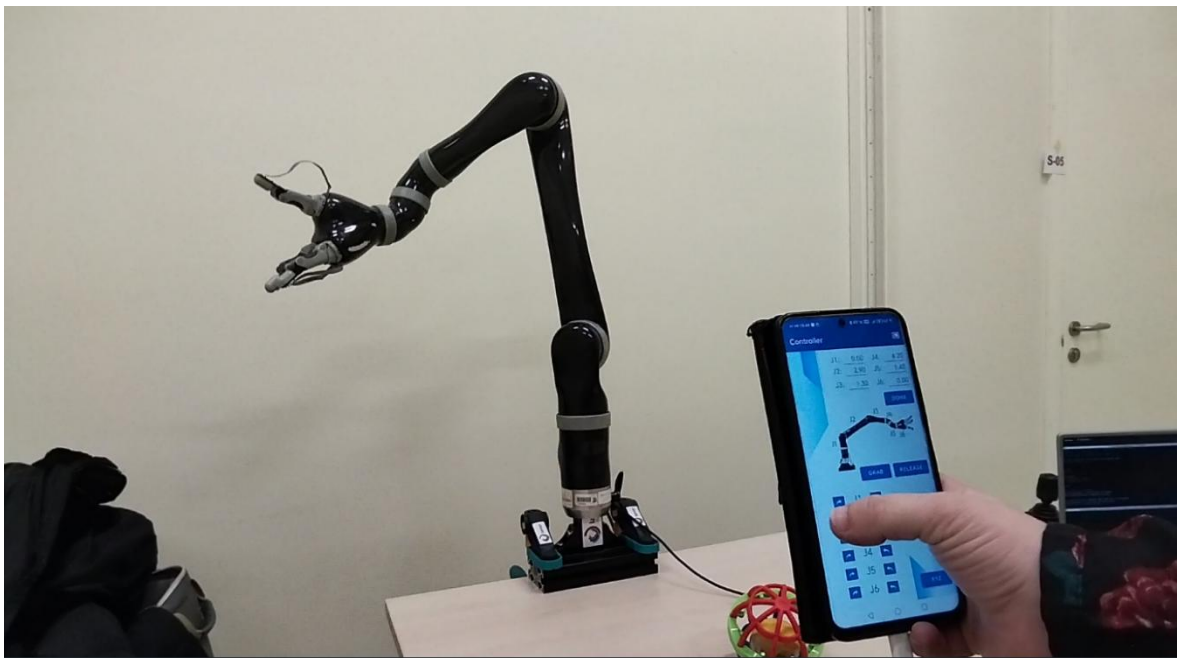
Prikaz fizičke robotske ruke u laboratorijskom okruženju u početnoj poziciji (Sl 4.7) gdje su vrijednosti zglobova su {0.0, 2.9, 1.3, 4.2, 1.4, 0.0}



Sl 4.7 Prikaz fizičke ruke u početnoj poziciji

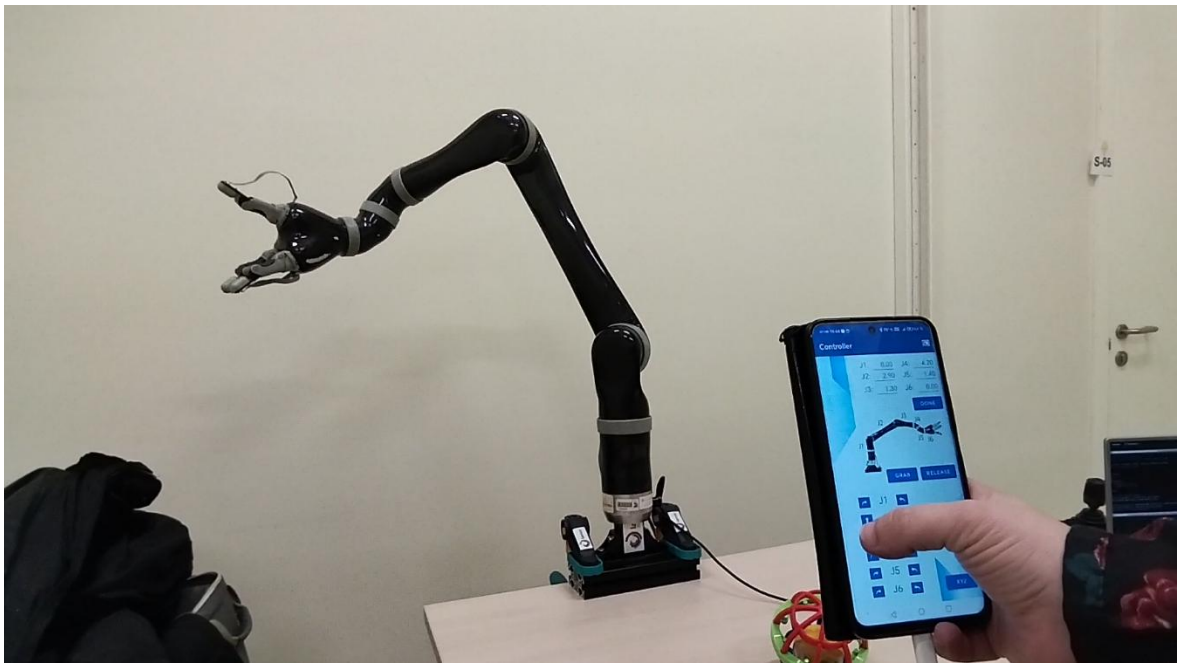
Slijedi prikaz rezultata upravljanja fizičkom rukom laboratorijskom okruženju u vidu niza slika (Sl 4.8) kroz koje će biti prezentirana funkcionalnost aplikacije. Svaka od slika prikazuje odabir na zaslonu aplikacije te rezultat pomaka fizičke ruke nakon toga.

Kad iz početne pozicije počnemo upravljati rukom na način da pritisnemo lijevu strelicu uz j2 (okrenutu prema gore), vidljivo je da se zglob j2 fizičke ruke u odnosu na početni položaj (Sl 4.7) počinje uzdizati, vrh šake se odmiče u lijevo što je vidljivo na slici (Sl 4.8).



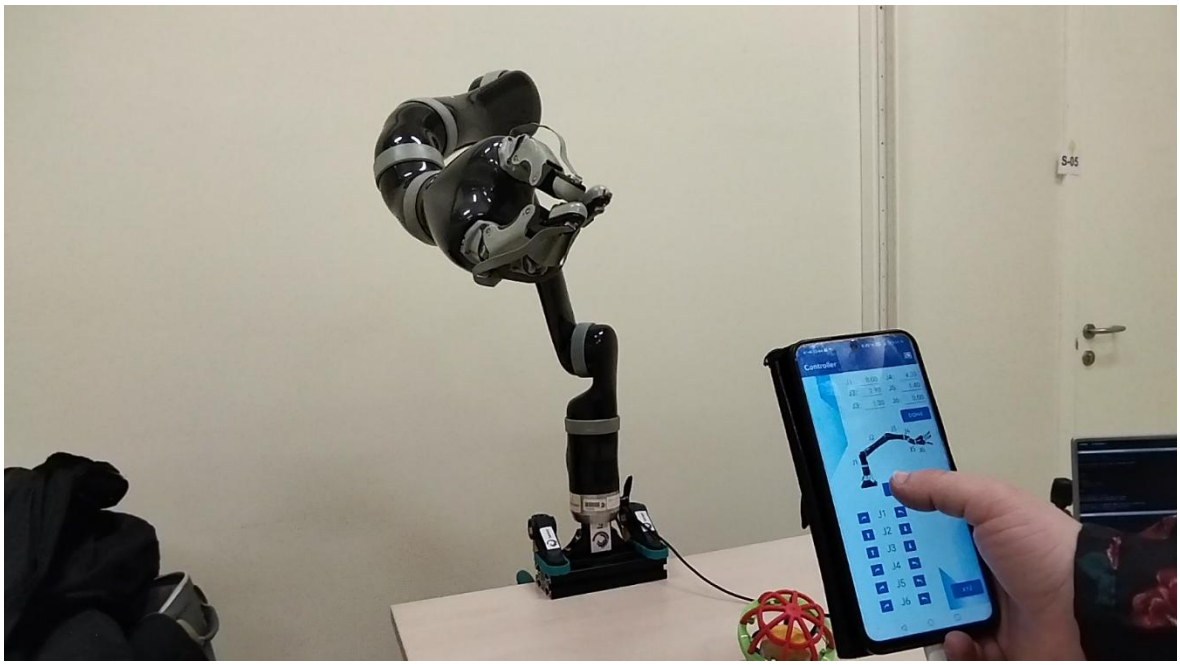
Sl 4.8 Prikaz položaja ruke prilikom pritiska na strelicu "l2" koja pokreće zglob j2 za (+ delta)

Ako nakon toga pritisnemo lijevu strelicu uz j3 (okrenutu prema gore), zglob j3 se počinje podizati i vrh šake se odmiče u lijevo što je prikazano na slici (Sl 4.9). Kad se ruka nalazi u tom položaju i u aplikaciji odlučimo pritisnuti desnu strelicu uz j1 zglob, cijela ruka rotira se suprotno od kazaljke na satu (udesno). Na slici (Sl 4.10) možemo vidjeti da je to rezultiralo okretanjem ruke prema nama. U ovom položaju šaka je dobro vidljiva pa se u toj poziciji mogu lijepo vidjeti rezultati pritiska na gumbove grab i release koji rezultiraju otvaranjem i zatvaranjem šake što je prikazano na slikama (Sl 4.11, Sl 4.12) koje slijede.

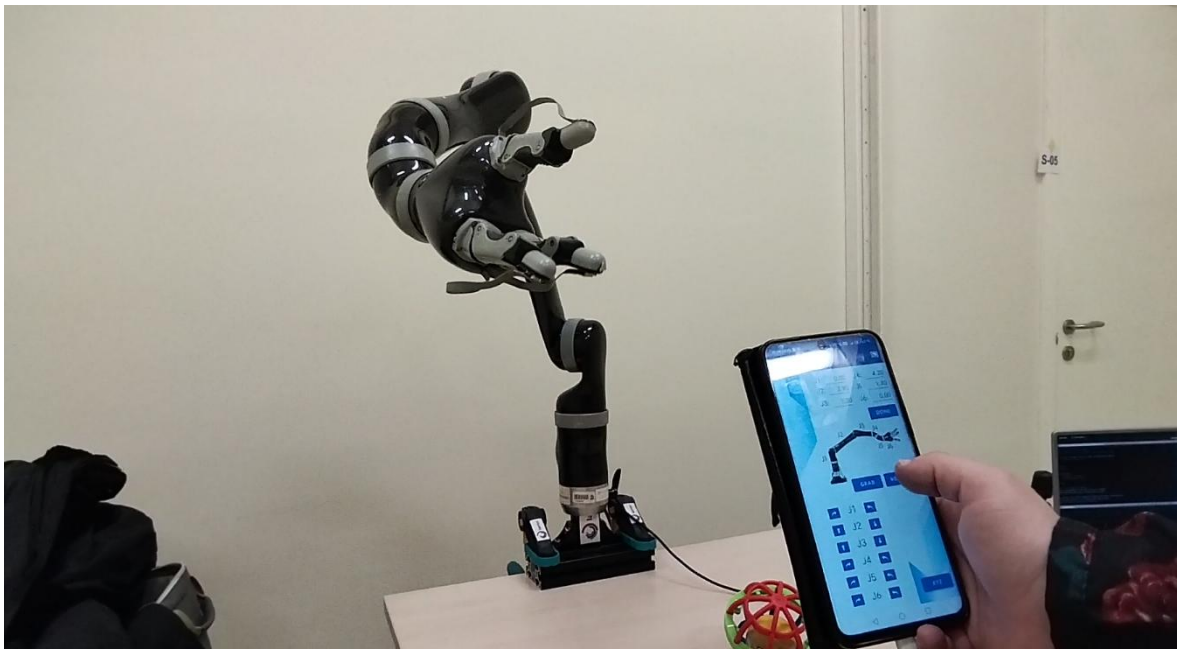


Sl 4.9 Prikaz položaja ruke prilikom pritiska na strelicu "l3" koja pokreće zglob j3 za (+ delta)

Sl 4.10 Prikaz položaja ruke prilikom pritiska na strelicu "r1" koja zakreće zglob j1 za (- delta)

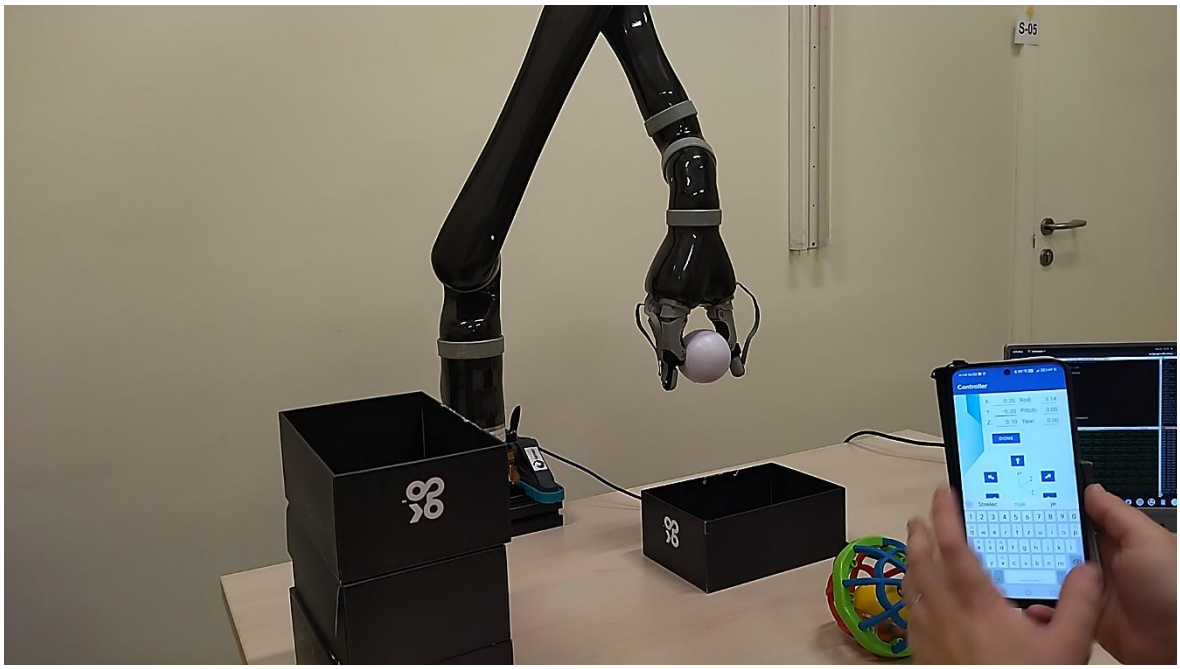


Sl 4.11 Prikaz zatvaranja šake pritiskom na gumb grab



Sl 4.12 Prikaz otvaranja šake pritiskom na gumb release

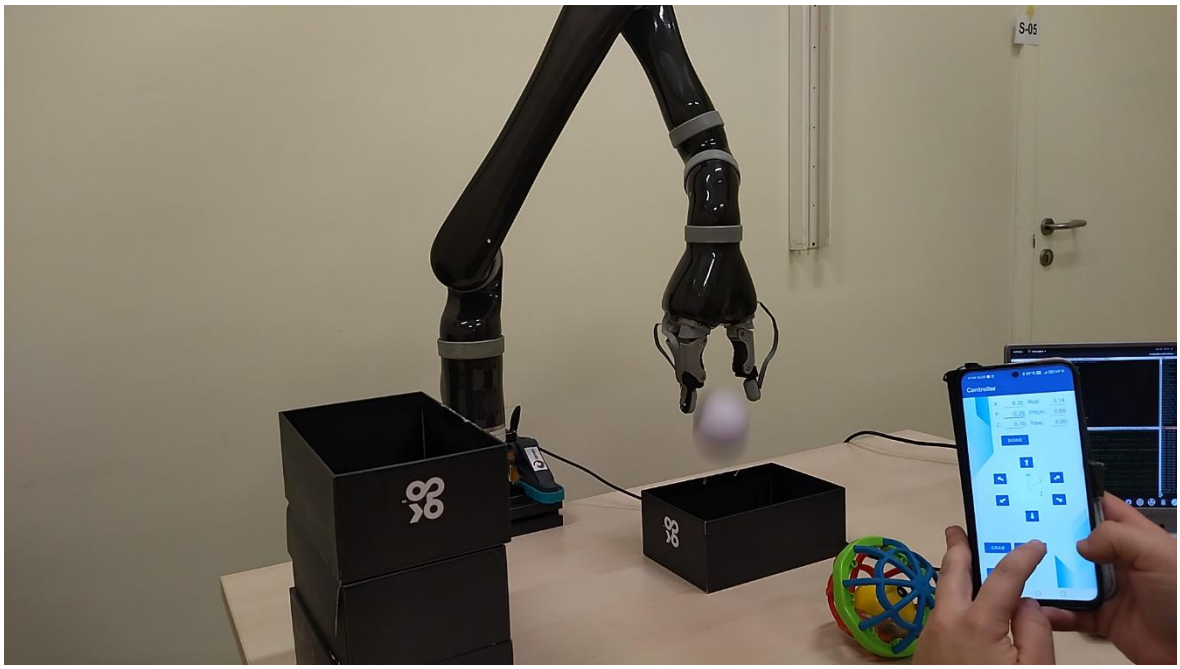
U nastavku biti će kroz niz slika () prikazane funkcionalnost kontrolera u Kartezijevom koordinatnom sustavu i kakvim to pomicanjem fizičke ruke rezultira. Na slici (Sl 4.13) prikazan je ishodišni položaj ruke i u aplikaciji su upisane vrijednosti u radijanima x = 0.2, y= -0.2, z = 0.10 te je roll = 3.14 te se na slici (Sl 4.14) vidi rezultat pomaka ruke pritiskom na gumb „done“ ispod polja s upisanim vrijednostima.



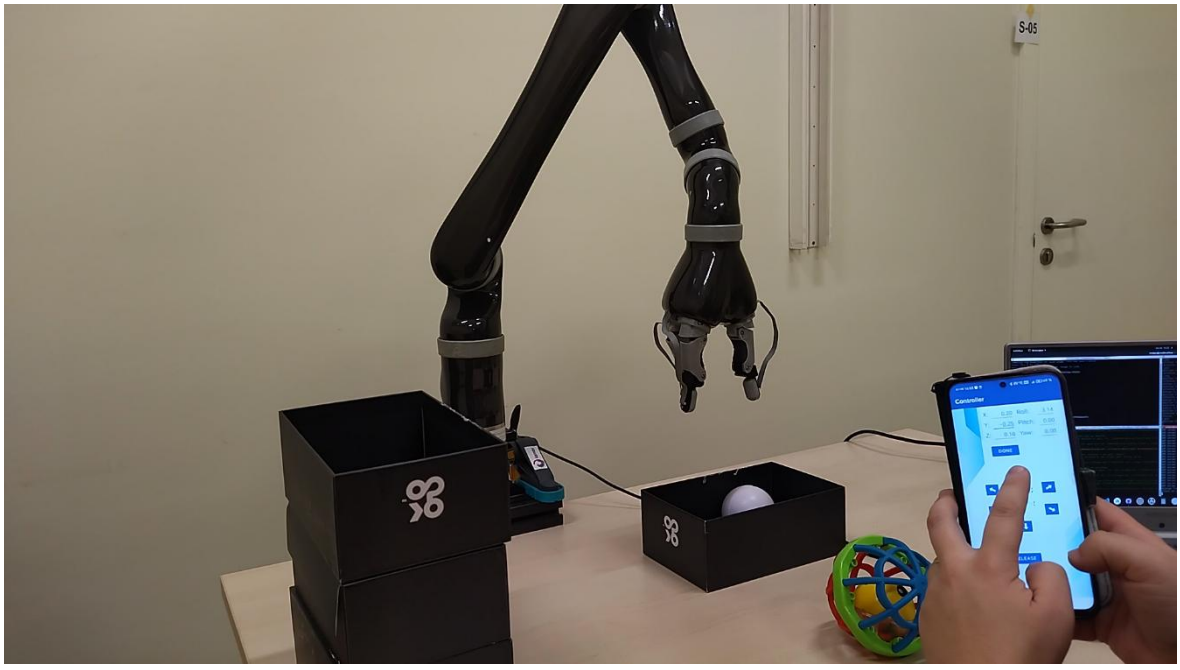
Sl 4.13 Prikaz ishodišnog položaja ruke prije pritiska gumba „done“ ispod unesenih vrijednosti

Sl 4.14 Prikaz položaja ruke nakon pritiska na gumb „done“ s unesenim vrijednostima

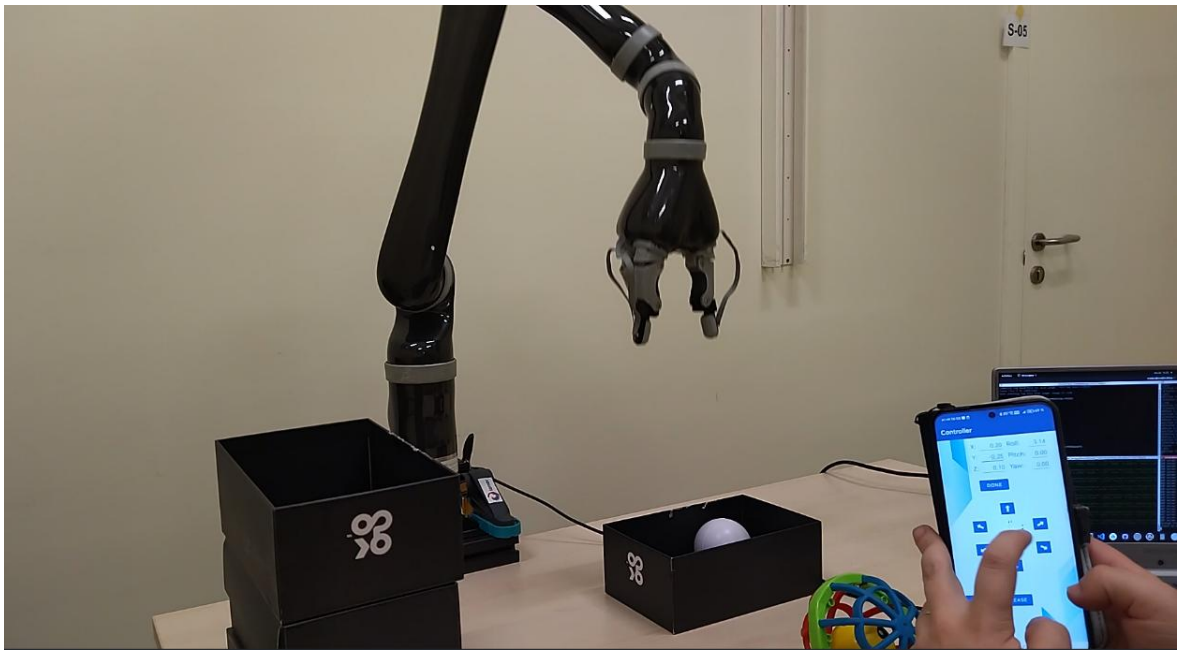
Kada bismo se nakon ovih radnji, uz još neke radnje zadane kroz aplikaciju, pozicionirali iznad kutije, mogli bismo iskoristiti gumb release da ubacimo lopticu u kutiju. To je prikazano na slici (Sl 4.15). Kad je loptica uspješno ispuštena, pritiskom na strelicu prema gore („z_pos“) na slici (Sl 4.16), ruka se uspješno odmiče gore (Sl 4.17).



Sl 4.15 Prikaz ispuštanja loptice prilikom pritiska na gumb release



Sl 4.16 Pritisak na strelicu prema gore za pomicanje ruke u pozitivnom smjeru z-osi



Sl 4.17 Prikaz pomaka ruke prema gore nakon otpuštanja strelice prema gore

Zaključak

Razvijeni sustav za teleoperaciju robotske ruke uspješno integrira Android aplikaciju, ROS čvorove i kinematičke algoritme za upravljanje rukom.

Simulacija u Gazebo okruženju omogućila je učinkovito testiranje funkcionalnosti, dok je eksperimentalno upravljanje fizičkom rukom bilo nešto zahtjevnije i zahtijevalo preinake u kodu radi različitog ponašanja fizičke ruke i ruke u simulatoru. Uspješno upravljanje rukom u laboratoriju potvrdilo je primjenu ostvarenog algoritma u stvarnosti.

Implementacija Kartezijevog koordinatnog sustava olakšava korištenje aplikacije za teleoperaciju korisnicima bez prethodnog iskustva u robotici, dok modularnost sustava omogućuje daljnji razvoj i prilagodbu u različitim radnim okruženjima.

Buduće nadogradnje mogu uključivati optimizaciju algoritama, primjenu umjetne inteligencije za autonomno planiranje gibanja te podršku za dodatne robotske platforme.

Literatura

[1] Kucuk, S., & Bingul, Z. *Robot kinematics: Forward and inverse kinematics* (pp. 117-148). London, UK: INTECH Open Access Publisher, 2006.

[2] Lovrek,I., Matijašević M., Ježić G., Jevtić D. Komunikacijske mreže (radna inačica udžbenika)

[3] Pavlic, E. Teleoperacija robotskom rukom korištenjem Android mobilnog uređaja, završni rad, Fakultet elektrotehnike i računalstva, 2021.

[4] Android Developers. Poveznica: https://developer.android.com/studio/intro;

[5] Gazebo. Poveznica: http://gazebosim.org/; pristupljeno 17. prosinca 2024.g

[6] Kinova, Gen 2 robots. Poveznica: https://www.kinovarobotics.com/product/gen2-robots; pristupljeno 4. siječnja 2025.g
pristupljeno 14. prosinca 2024.g

[7] Robotics-explained, Jacobian. Poveznica: https://robotics-explained.com/jacobian; pristupljeno 4. siječnja 2025.g

[8] Ros Wiki Documentation http://wiki.ros.org/; pristupljeno 17. prosinca 2024.g

Sažetak

Ovim diplomskim radom opisan je razvoj sustava za teleoperaciju robotske ruke Jaco tvrtke Kinova, generacije 2, korištenjem Android mobilne aplikacije.

U radu je objašnjeno povezivanje aplikacije s ROS-om na računalu koje koristi ROS čvorove za upravljanje robotskom rukom te su prikazani rezultati upravljanja ruke kroz razvijenu aplikaciju u Gazebo simulacijskom okruženju i laboratorijskim uvjetima.

Kroz poglavlja su detaljno objašnjeni ključni pojmovi korišteni u radu, uključujući virtualna i simulacijska okruženja poput Android Studija i Gazebo simulatora te operacijske sustave Android i ROS. Također, pružen je uvid u robotsku ruku Jaco Gen 2, njezine tehničke karakteristike i funkcionalnosti.

Glavni dio rada fokusira se na implementaciju algoritama za upravljanje rukom u Kartezijevom koordinatnom sustavu, funkcionalnosti aplikacije te vizualni pregled korisničkog sučelja. Objašnjen proces komunikacije između aplikacije i ROS-a putem UDP servera, koji omogućuje precizno upravljanje rukom u simulatoru ili fizičkom rukom u laboratoriju.

Na kraju su prikazani eksperimentalni rezultati kroz slike iz simulacijskog i laboratorijskog okruženja, potvrđujući funkcionalnost i pouzdanost razvijenog sustava.

Summary

This thesis describes the development of a teleoperation system for the Jaco robotic arm by Kinova, Generation 2, using an Android mobile application.

The main focus of the thesis is the implementation of inverse kinematics for controlling the arm in the Cartesian coordinate system. The application's connection to ROS on a computer running ROS nodes for robotic arm control is explained, along with the results of controlling the arm through the developed application in both the Gazebo simulation environment and laboratory conditions.

The chapters provide a detailed explanation of the key concepts used in the thesis, including the principles of inverse kinematics, virtual and simulation environments such as Android Studio and the Gazebo simulator, as well as operating systems like Android and ROS. Additionally, an overview of the Jaco Gen 2 robotic arm and its technical characteristics is included.

The main part of the thesis focuses on the implementation of algorithms for controlling the arm in the Cartesian coordinate system, the application's functionalities, and a visual review of the user interface. The process of communication between the application and ROS via a UDP server is explained, enabling precise control of the robotic arm in the simulator or the physical arm in the laboratory.

Finally, experimental results are presented through images from the simulation and laboratory environments, confirming the functionality and reliability of the developed system.