1. Intelligens robot/kéz rendszer irányítórendszerének felépítése. Sorolja fel a tárgy rekonfigurálásánál felhasznált modelleket. Hogyan határozható meg a kontaktus pont következ_ helye a tárgy és az ujj között el_írt relatív sebesség esetén? Milyen típusú matematikai feladat megoldásaként számíthatók a kontaktuspontban a mozgáshoz szükséges er_k?

Az irányítási rendszer alrendszerekre osztható [19]. Az első alrendszer a robot irányító rendszer, amely a host PCből, annak kiterjesztő rackjéből és a tengelyhajtások teljesítmény elektronikájából (analóg PI szabályozók + PWM) áll. A kiterjesztő rack 3 speciális ARC kártyát tartalmaz. Minden kártya tartalmaz egy i386EX processzort, egy DSP-t (TMS320C31) a robotirányítási algoritmusok realizációjához, egy tacho-processzort 4 tengely encoder jeleinek fogadására és DAC-ket a referencia jelek (id_ben változó alapjelek) kiadásához az alacsony szintű áram (nyomaték) szabályozási körök számára [9,10]. A kártyák a host PC-hez dual port RAMokon keresztül csatlakoznak. A kommunikáció a kártyák között a dual port RAM-okon keresztül, vagy közvetlenül CAN-buszon történhet.
A robot irányító szoftvere több processzoron fut egyidejűleg, ami a task szétosztást közöttük egy lényeges kérdéssé teszi.
A TMS processzorok realizálják a szabályozási hurkokat A robotirányító rendszer úgy lett megtervezve, hogy képes legyen különböző robotirányítási algoritmusok megvalósítására.

A Geometriai modell
B Kinematikai modell (sebességek és szögsebességek)
C Statikus erők
D Kontaktus modellek
F Felület modellek

A kontaktuspont mozgása két merev test között meghatározható a $[t,t+\delta t]$ intervallumban kis δt mellett a Montana egyenletek felhasználásával, mivel *Vr* ismert.
Montana-egyenletek:

$$\dot{\alpha}_f = M_f^{-1}\hat{R}_\psi\left(K_f + \tilde{K}_o\right)^{-1}\left[\begin{pmatrix} {}^l\omega_y \\ -{}^l\omega_x \end{pmatrix} + \tilde{K}_o\begin{pmatrix} {}^l v_x \\ {}^l v_y \end{pmatrix}\right]$$

$$\dot{\alpha}_o = M_o^{-1}\hat{R}_\psi\left(K_f + \tilde{K}_o\right)^{-1}\left[\begin{pmatrix} {}^l\omega_y \\ -{}^l\omega_x \end{pmatrix} - K_f\begin{pmatrix} {}^l v_x \\ {}^l v_y \end{pmatrix}\right]$$

$$\dot{\psi} = T_o M_o \dot{\alpha}_o + T_f M_f \dot{\alpha}_f + \omega_z$$

$$0 = v_z$$

Lineáris programozási feladat megoldásaként adódnak a mozgáshoz szükséges erők.

2. Intelligens robot/kéz rendszer irányítórendszerében az egyensúlyi kontaktuser_k meghatározása, a Peshkin/Sanderson-féle minimális teljesítmény elv, a kontaktuserők meghatározása lineáris programozással.

J Az egyensúlyi kontaktus erők meghatározása

Feltételezvén, hogy a pillanatnyi megfogási konfiguráció már meghatározásra került, a kontaktus erők is meghatározhatók a Peshkin és Sanderson által javasolt minimális teljesítmény elv alapján [28]:

"Egy kvázi-statikus rendszer azt a mozgást választja a korlátozásokat kielégítő összes lehetséges mozgás közül, amelyik minimalizálja a pillanatnyi teljesítményt".

$$min\, P_v = -<V_o, F_{o,ext} + J_o^T \tilde{F}_c>$$

korlátozások	(3)	egyensúly
	(4)	tárgy tolása
és	(6)	gördülő kontaktus
vagy	(8)	csúszó kontaktus.

Itt $\tilde{F}_c$ egyenlő F_c -vel, amelyben a z komponens 0-val lett helyettesítve. A kontaktus erőket a fenti lineáris programozási feladat megoldása adja.

3. Intelligens robot/kéz rendszer irányítórendszerében a relatív segességek generálása, a hibák kezelése, globális és lokális mozgástervez_.

K A relatív sebességek generálása

A pillanatnyi inverz ujj mozgási feladat megoldása igényli a V_r relatív sebességet. Tiszta gördülő és tiszta csúszó kontaktus esetén V_r nemnulla komponenseinek száma $2k$. A keresés komplexitásának csökkentése érdekében diszkretizáljuk a relatív sebességet tiszta gördülő és tiszta csúszó kontaktus esetén rendre

$$\left(\omega_{ix}, \omega_{iy}\right) \in \left\{\omega_x^{\min}, 0, \omega_x^{\max}\right\} \times \left\{\omega_y^{\min}, 0, \omega_y^{\max}\right\}$$

$$\left(v_{ix}, v_{iy}\right) \in \left\{v_x^{\min}, 0, v_x^{\max}\right\} \times \left\{v_y^{\min}, 0, v_y^{\max}\right\}$$

alapján (csak 3 érték lehetséges minden nemnulla komponens számára, amely 3^{2k} lehetséges értéket enged meg V_r számára). Ha szükség van új V_r értékre, akkor véletlen generálással állítjuk elő ebből a halmazból.

L A hibák kezelése

Tegyük fel, hogy a *t1* id_pillanatban az inverz ujj mozgás megoldó hibát detektált. Három hiba típust különböztetünk meg, és mindegyikük esetén egy új *Vr* relatív sebességet generálunk az inverz ujj mozgás megoldó számára a következ_ elv alapján:

i) Ha a hiba oka az ütközésmentességi vagy az elérhet_ségi feltétel megsértése, akkor új *Vr* részvektort generálunk minden, a feltételt nem kielégít_ ujjhegy számára.

ii) Ha a surlódási vagy az egyensúlyi feltétel nem elégíthet_ ki a lineáris programozási feladat számára (a kontaktus er_k generálásakor), akkor új 2k dimenziós *Vr* vektort generálunk.

iii) Ha az *Fi* ujjhegy félgömb-alakú és gördül_ kontaktus volt érvényben, hiba keletkezhet, amikor a kontaktus pont kívülre kerül a gömbszer_ részen és így nem folytatható a gördül_ mozgás. Ebben az esetben csúszó mozgásra térünk rá a mozgás folytatásakor, és új véletlen *Vr* értéket generálunk.

Miután választottunk egy új *Vr* értéket, újra indítjuk az utolsó megfelel_ *TBo(t1)* konfigurációból az ujjak pillanatnyi mozgásának meghatározását. Ezt a folyamatot addig ismételjük, amíg nem keletkezik az összes feltételt kielégít_ ujj inverz mozgás megoldás és kontaktus er_ megoldás.

A mozgástervező algoritmusa

A jelölés egyszerűsítése érdekében vezessük be rendre a $T_o := T_{Bo}$ és $T = (T_{Bo}, T_{B,l1}, \ldots, T_{B,lk})$ rövidített alakokat a tárgy és az egész rendszer számára. Továbbá utaljon rendre az előző "previous" szituációra p és az aktuális "current" szituációra c. A mozgás tervező egy globális tervezőből (*global planner*) és egy lokális tervezőből (*local planner*) áll. A globális tervező $\Delta T >> \delta t$ idő lépésközt használ és meghatározza az új rész-célokat a lokális tervező számára a T_{start} és $T_{o,goal}$ közötti úton. A globális tervező nem tekinti a korlátozásokat.

A *lokális tervező* fogadja a $(T_{o,p}, T_{o,c}, V_{o,c})$ előírásokat a globális tervezőtől és kezdi meghatározni a *T(t)* pályát és az *F(t)* kontaktus erőket, amelyek szükségesek a tárgy mozgásához $T_{o,p}$ és $T_{o,c}$ között a korlátozások kielégítése mellett. A lokális tervező δt idő lépésközt használ, generálja a relatív sebességeket, megoldja az inverz ujj mozgási feladatot, meghatározza a kontaktus erőket és lekezeli az esetleges hibákat a korábban leírt módon. Visszatér a globális tervezőhöz, ha a elérte a $T_{o,c}$ rész-célt.

A *globális tervező* elindul a tárgy $T_{o,start}$ helyzetéből, kiszámít a tárgy számára egy ezt követő véges rész-cél halmazt csak tiszta transzlációs vagy tiszta orientációs mozgást választva V_o sebességgel ΔT ideig. Nem vizsgálja a korlátozásokat. Mivel az orientáció 3 skalárral jellemezhető, T_o beágyazható az R^6, T pedig az $R^{6(k+1)}$ térbe. Ha egy rész-cél generálásra került, meghívódik a lokális tervező. Ha a lokális tervező el tudja érni a rész-célt, akkor generálja a megfelelő *T(t)* trajektóriát és az optimális *F(t)* kontaktus erőket, amelyek elvezetnek a $T_{o,c}$ helyzetbe, ellenkező esetben a globális tervező tiltottnak tekinti a $T_{o,c}$ helyzetet. A összekötöttség a rész-célok között egy inkrementális *G* gráfban tárolódik, amelynek csúcspontjai R^6 részhalmazát alkotják. Ezek 6-dimenziós szomszédos cellákkal reprezentálhatók. Azonban a cellák halmazát nem építjük fel a priori módon, hanem csak a meglátogatott cellákat generáljuk.

4. Intelligens robot/kéz rendszer virtuális valóság rendszerének (virtual reality system) feladatai. A gyors ütközésdetektálás hierarchikus felépítése. Ismertesse a három javasolt ütközésdetektáló algoritmus elvét. Mi a kalibrált virtuális valóság és hogyan valósítható meg?

B Virtuális valóság rendszer

A virtuális valóság arra használható, hogy vizuálisan megjelenítse a robot akcióit a humán operátor számára, és ezt rendszerint valósid_ben kell elvégezni. Egy elvárás az is, hogy meghatározható legyen, vajon a robot el tudja-e végezni az el_írt akciót úgy, hogy magával és/vagy az _t körülvev_ környezettel nem ütközik. Ezért egy hatékony többszintű ütközésdetektáló algoritmus került implementálásra [38,39].

Az *ütközésdetektáló algoritmus* hierarchikus. Először alacsony-precizitású teszteket hajt végre, majd a további lépésekben egyre pontosabbakat. A számítási költsége a későbbi lépéseknek magasabb, mint a megel_z_ké, de ezeket már csak kisebb mennyiség_ adaton kell elvégezni.

Az ütközésdetektálás szintjei a következ_k: (1) tengelyekkel párhuzamos befoglaló dobozok, (2) tetsz_leges befoglaló dobozok, (3) tetsz_leges orientációjú háromszögek.

A *tengely-párhuzamos befoglaló dobozok* esetén minden tárgyat körülveszünk egy olyan (téglatest alakú) dobozzal, amelynek élei párhuzamosak annak a koordinátarendszernek a tengelyeivel, amelyet “világ” (world) koordinátarendszernek tekintünk, és ezeket teszteljük ütközés szempontjából (3. ábra). Ezek a tesztek csak összehasonlítás-típusú operációkat igényelnek. Ez a módszer nem túl pontos, különösen ha tetsz_leges orientációjú hosszúkás-alakú tárgyakat tesztelünk. Ennek számítási költsége *O(N2)*, ahol *N* a tárgyak száma.

A sokkal alkalmasabb (téglatest alakú) *tetszőleges befoglaló dobozok* esetén a lokális befoglaló dobozok rendszerint sokkal keskenyebbek, mint a tengelypárhuzamosak (és emellett alakjuk konstans). Ezeket a dobozokat transzformáljuk a világ-koordinátarendszerbe és teszteljük ütközésre.

A legáltalánosabb *tetsz_leges orientációjú háromszögek* módszere akkor alkalmazandó, ha az els_ két módszer nem ad egzakt eredményt. Két háromszög tesztelése kétszint_ folyamat: (1) Ha az els_ háromszög minden pontja a másik síkjának egyik oldalán van, akkor diszjunktak. (2) Ha az egyik éle metszi a másik háromszöget, akkor ütköznek.

Annak érdekében, hogy a virtuális világot összehozzuk a valódi környezettel, a rendszer *kalibrációja* kívánatos. Mivel a sztereo rész nem használ kalibrált kamerát, a grafikus részben implementáltuk a kamerakalibrációt, hogy a grafikus szimuláció eredménye megjeleníthet_ legyen a valódi kamera képpel együtt. Ez azt jelenti, hogy a kamera paramétereit, pl. a pozíciót, orientációt, fókusz távolságot stb. identifikálni kellett a valódi kameráról nyert képekb_l. Az identifikáció eredménye felhasználásra került a megjelenítési fázisban. Pinhole kamera modellt feltételezve a kalibrálás Faugeras módszerével [13] történt.