



GÁBOR DÉNES FŐISKOLA

Molnár Zsolt, Zátroch Zoltán,
Kovács János:

**IPARI ROBOT 3D MODELLJÉNEK
SZINKRONIZÁLT VEZÉRLÉSE**

Konzulens:

Dr. Kopácsi Sándor

Tudományos **Diákköri Konferencia**

BUDAPEST

2012.

Tartalomjegyzék

1.Bevezetés	3
2.Az előző TDK munkánk összefoglalása	4
3.Hasonló kutatások bemutatása	5
3.1 Robologix	5
3.2 Simbad.....	5
3.3 Webots	6
3.4 RoboWorks	6
3.5 Easy-Rob	6
3.6 VirCA rendszer ismertetése.....	7
3.7 Hasonló kutatások összehasonlítása(összevetése)	8
4.A modellek továbbfejlesztése	9
4.1Rubik kockát kirakó alkalmazás	10
4.2Az ISF alkalmazás	10
4.3Az új modellek	11
5A soros kommunikáció alkalmazása a robot és a modell szinkronizálására.....	15
5.1 A robotvezérlő	15
5.2 A KAREL program bemutatása.....	18
6. VirCA komponensek.....	21
6.1 A serial communicator VirCa komponens	21
6.2 A Fanuc robot VirCAkomponens	24
6.3 A VirCA komponens	28
6.4 Komponensek közötti kapcsolat	29
7. Egyéb beviteli eszközök alkalmazása	31
7.1 A kinect szenzor.....	31
7.2 A Wii kontrollér	31
8. Eddig elért eredmények	32
9. Továbbfejlesztési lehetőségek.....	33
10. Irodalomjegyzék.....	34
11. Ábrajegyzék	35
12. Rövidítés-jegyzék.....	36

1.Bevezetés

Munkánkat 2011. szeptemberében kezdtük el háromdimenziós robotmodellezés témában, Dr. Kopácsi Sándor konzulensünk segítségével. TDK munkánk nagy részét a Magyar Tudományos Akadémia Számítástechnikai és Automatizálási Kutatóintézetében (MTA-SZTAKI) nemrég kialakított robotlaboratóriumban végeztük.

Célunk e dolgozat megírásával az, hogy ismertessük a TDK munkánk során elvégzett feladatainkat és a megvalósításhoz használt eszközöket. A dolgozatban bemutatjuk a VirCA nevű virtuális kollaborációs rendszert, melyet a munkánk során használtunk, és amellyel képesek vagyunk egy adott robotot akár földrészekkel távolabbról irányítani, mindezt úgy, mintha az valójában a közelünkben lenne. Ezt követően összehasonlítjuk az általunk megismert hasonló kutatási eredményeket ezen a területen és bemutatjuk a VirCA előnyeit ezekkel a rendszerekkel szemben. A dolgozat további részében bemutatjuk az eddig elért eredményeinket ebben a témában.

Feladatunk az volt, hogy elkészítsük egy Fanuc S 430i F típusú ipari robot és környezetének háromdimenziós modelljét, majd ezt a modellt a VirCA rendszer segítségével három dimenzióban megjelenítve szinkronizáltan vezéreljük a valódi robottal. Ezzel a kutatással azt kívántuk megvizsgálni, hogy hogyan kapcsolható össze egy valós robot annak virtuális modelljével, és hogy mennyire alkalmas egy háromdimenziós modell arra, hogy annak segítségével vezéreljünk egy valós robotot. A modell elkészítés lépéseiről bővebben az előző TDK [1] dolgozatunkban számoltunk be, ebben a dolgozatban elsősorban a modell és a valódi robot szinkronizált vezérléséről lesz szó.

TDK munkánk során olyan új eszközökkel és módszerekkel ismerkedtünk meg, majd alkalmaztunk, amelyek segítségével egymástól fizikailag távol levő gépeket összekapcsolhatunk, virtuális valóság alapú háromdimenziós szimulációt és vezérlést valósíthatunk meg az interneten keresztül. Weblapunkon [2] megtekinthetők az elért eredmények, azok dokumentációja, ismertető a TDK munkánkról, valamint videók a robotról, környezetéről, a VirCA-ról és annak előkészítő alkalmazásáról.

2. Az előző TDK munkánk összefoglalása

Előző TDK munkánk során [1] a robotlabor alaprajzának segítségével, laborban végzett mérések nélkül elkészítettük a labor háromdimenziós modelljét. A robotlaborban végzett méréseket követően a laborban található bútorokat és a Fanuc robotot modelleztük a Google SketchUp 8, valamint a Solid Works programmal. A robotot részekre bontottuk a későbbi mozgítás céljából, valamint a koordináta-rendszerek origójához pozicionáltuk a robot megfelelő darabjait, illetve a többi modellezett tárgyat is, hogy a későbbi elhelyezés egyszerűbb legyen. Ezek után a méretarányos modellt átkonvertáltuk a megfelelő „ogre” [3] formátumra, amelyet a VirCA rendszer a megjelenítéshez használ.

A konvertálás után a szoba berendezése következett, amelyet a „scene” fájlban végeztük el, amelynek felépítése az XML szintaktikájához hasonló. A node-ok létrehozásával és pozicionálásával a tárgyak a megfelelő helyre kerültek a szobában. Ezt a robot elhelyezése követte.

A robotot a fanuc_test program segítségével állítottuk össze. A darabokat, hasonlóan a szoba összerakásához, pozicionáltuk. Egyes részeit hierarchikus objektumként regisztrálva illesztettük egymásra, hogy a későbbi mozgítás során ezek együtt tudjanak mozogni. A robot mozgítása ugyanígy a fanuc_test program segítségével történt, amit egy sokvezérlési ciklus segítségével oldottunk meg. A robot a vele szemben lévő keret és a tőle balra elhelyezkedő asztal között mozog.

Végül mindezek életre keltéséhez az elkészült modellt a VirCA rendszer segítségével összeraktuk a fanuc_test programmal és a VirCA kötetőjével együtt, amelyet a 3.6-os fejezetben bemutatunk. Ezt követően az elkészült modellt az MTA-SZTAKI 3D barlangjában jelenítettük meg, ahol képesek voltunk az általunk modellezett robotlabort három dimenzióban bejárni, látva a robot működését.

3. Hasonló kutatások bemutatása

Mielőtt munkánkat elkezdtük volna, utánanéztünk, hogy milyen hasonló kutatások zajlanak a világban, és megvizsgáltuk ezeket, majd ezt követően eldöntöttük, hogy számunkra melyik lenne a legmegfelelőbb célunk megvalósításához. Több alkalmazást is megvizsgáltunk, a következőkben ezeket mutatjuk be részletesebben.

3.1 Robologix

A robologix [4] egy szimulációs alkalmazás, amelyet arra fejlesztettek ki, hogy a számítógépünkön szimulálhassunk egy robotot és annak környezetét. A robologix-szal könnyedén tesztelhetünk, futtathatunk és ellenőrizhetünk olyan általunk írt programokat, melyeket hat szabadságfokú ipari robotokra írtunk meg. Az alkalmazás tartalmaz olyan funkciókat, mint például a hegesztés, festés, tárgyak megfogása és áthelyezése, amelyeket testre a szabott környezetben végezhetünk el. A program tartalmaz egy szimulált tanító konzolt, amely segítségével képesek vagyunk irányítani a robotot, illetve módosíthatjuk az általunk írt programokat, valamint képesek vagyunk programokat feltölteni erre a szimulált eszközre. Ez a program egy felhasználóbarát háromdimenziós környezetet biztosít számunkra, könnyedén összehasonlíthatjuk a robotra írt programjainkat, így optimalizálhatjuk a ciklusidőt, ami nagyon fontos lehet egy gyártási folyamat során.

3.2 Simbad

A Simbad [5] egy ingyen letölthető és használható, java alapú, nyílt forráskódú háromdimenziós robotszimulátor, amelyet kutatási és oktatási célokra fejlesztettek ki. Elsősorban olyan kutatóknak és programozóknak, akik mesterséges intelligenciával, gépi tanulással és más MI algoritmusokkal foglalkoznak a robot-automatizálás területén. Ebben az alkalmazásban lehetőségünk van arra is, hogy saját robotvezérlőt írjunk, módosítsuk a környezetet és különböző szenzorokat használjunk. Ezen kívül az alkalmazás képes háromdimenziós megjelenítésre, egy vagy több robot szimulációjára, alkalmazhatunk a szenzorok közül távolságérzékelő szenzorokat (sonar és infravörös) kontaktszenzorokat, és vizuális szenzorokat (szín monoszkópikus kamera) Mindezeket egy grafikus felületen keresztül kezelhetjük.

3.3 Webots

A Webots [6] egy olyan fejlesztői környezet, amelyet arra használhatunk, hogy modellezünk, programozzunk és szimuláljunk mobil robotokat. Lehetőségünk van egy általunk kialakított környezet létrehozására, amelyben az elhelyezett objektumoknak nem csak a megjelenését állíthatjuk be, hanem például azoknak tömegét, súrlódását is. Nagy mennyiségben állnak rendelkezésre a különböző szimulált szenzorok és aktuátorok, amelyekkel felszerelhetjük bármelyik robotunkat. A robotvezérlő egy beépített integrált fejlesztői környezeten (IDE) keresztül programozható. A robot viselkedése tesztelhető az általunk létrehozott fizikailag realisztikus környezetben. Ezt az alkalmazást elsősorban egyetemek és kutatóközpontok használják a világ különböző pontjain.

3.4 RoboWorks

A RoboWorks [7] egy oktatási célra kifejlesztett 3D robot szimulátor, amelyben nagy mennyiségű előre elkészített ipari robot modellje megtalálható. Lehetőség van a modellek animálására és elkészítésére az alkalmazáson belül, és mozgathatjuk a robotot a virtuális térben. Erre a program a RoboTalk-ot egy nyílt forráskódú szoftvert használ, mellyel akár TCP/IP protokollon keresztül is képes kommunikálni a RoboWorks-el. Ennek előnye, hogyha a vezérlőprogram például egy másik gépen Linux rendszeren fut, akkor is tudjuk irányítani a RoboWorksben megjelenített modellt.

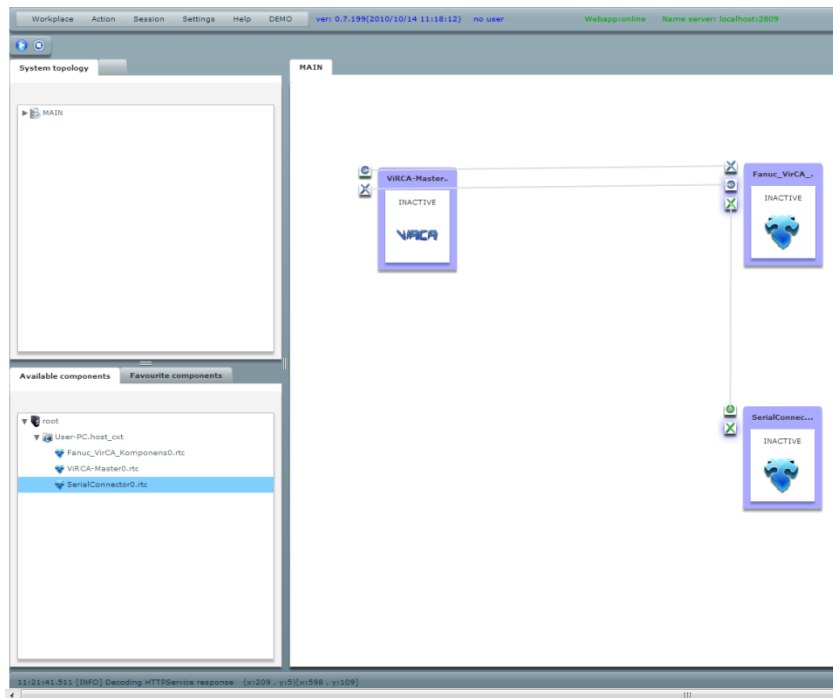
3.5 Easy-Rob

Az Easy-Rob [8] „Easy way to robotics” segítségével megtervezhetjük és kipróbálhatjuk az általunk létrehozott robotcellákat, amelyen azt az elkerített területet értjük, ahol a robot munkát végez. Az ütközések, mozgási tartományok, ciklusidők tesztelésére és optimalizálására is lehetőséget nyújt számunkra a program, amelyeknek fontos szerepük van egy gyártási folyamat során. A program segítségével növelhetjük a tervezési szakasz hatékonyságát, ezzel csökkenthetjük az indítási időt. A folyamatot követhetjük az elkészített robot és környezetének megjelenítésével. Az Easy-Rob egy Microsoft Windows alkalmazás, melynek demo verziója ingyenesen letölthető és kipróbálható.

3.6 VirCA rendszer ismertetése

A TDK munkánk során lehetőségünk volt az MTA-SZTAKI által kifejlesztett interaktív virtuális valóságkezelő rendszerével megismerkedni, amelynek a neve Virtual Collaboration Arena [9]. A VirCA egy háromdimenziós, internet- és komponensalapú elosztott rendszer, amelyben lehetőségünk van a virtuális és a valós környezet összekapcsolására és megjelenítésére, ami térben és időben egymástól független lehet. Továbbá lehetőségünk van a virtuális és a valós eszközök közötti együttműködésére és manipulálására is.

Ahhoz, hogy a komponensek valóban egymással tudjanak kommunikálni, a VirCA úgynevezett kötöztető moduljájának segítségével van szükség (lásd az 1. ábrán). A kép bal oldalán találhatóak a komponensek, amiket felhúzhatunk a jobb oldali fehér felületre, majd a megfelelő portokat egymással összekötve létrejön a komponensek közötti kapcsolat.



1. ábra VirCA kötöztető

3.7 Hasonló kutatások összevetése

Az általunk elemzett programok közül mindegyik lehetőséget ad robotok és azok környezete modelljének megjelenítésére. Lehetőséget nyújtanak arra, hogy különböző mozgásokat tanítsunk, teszteljünk, valamint szenzorokat is elhelyezhetünk a virtuális térben. Ezek az alkalmazások nagyszerűen használhatóak szimulációra, tesztelésre, és oktatásra, ám az összes ezek közül csupán szimuláció. A bemutatott programok esetében legfeljebb arra van lehetőség, hogy a modellt vezéreljük. Az ismertetett programok legtöbbje ingyenesen letölthető, nyílt forráskódú, ám akad közöttük olyan is, amelyből ingyenesen csupán a próbaverzió tölthető le. A munkánk során az is szempont volt, hogy lehetőleg minél kisebb anyagi ráfordítással érjük el kitűzött céljainkat. Az MTA-SZTAKI által kifejlesztett és általunk használt VirCA rendszer szintén ingyenesen letölthető alkalmazás. Azért választottuk a VirCA-t, mert itt a szimuláción kívül lehetőségünk van valós vezérlésre is. Ez azért volt fontos, mert a modellek elkészítése után mi nem csak a modellezett robotot szeretnénk volna vezérelni, hanem ezzel szinkronizáltan a SZTAKI robot-laboratóriumában található valódi robotot is. A VirCA segítségével erre is képesek vagyunk, ám az előbb felsorolt hasonló programokkal erre nincs lehetőségünk, csupán annyi, hogy szimulálunk egy környezetet, valamint vezéreljük a modellt. Az általunk kiválasztott VirCA rendszerben megtehetjük azt is, hogy a háromdimenziós modellt és a valódi robot szinkronizáltan működtetjük, illetve a VirCA-ból kiadott utasításokat nem csak a modell robot, hanem ezzel párhuzamosan a valódi robot is végrehajtja.

4. A modellek továbbfejlesztése

Az előző TDK dolgozatunkban [2] már bővebben ismertettük a modellezés menetét. A dolgozat megtekinthető a robotmodell3d.hu weboldalon. Ebben a fejezetben röviden összefoglaljuk a modellezés lépéseit, és hogy az általunk elkészített két alkalmazáshoz milyen új modelleket kellett készíteni és beépíteni a meglévő modellek közé. A modellezés a Google Sketchup programmal történik, amellyel könnyedén és gyorsan modellezhetjük a 3 dimenzióban megjeleníteni kívánt testeket, majd miután ez megtörtént, az elkészített modellt egy a Google Sketchup-ba beépülő ogre konverterrel átalakítjuk ogre [3] formátumba. Eerre azért van szükség, mert a VirCA 3D motorját az Ogre 3D engine szolgáltatja, így ennek megfelelő formátumba kell alakítanunk a modellfájlokat, hogy azok a VirCA számára megjeleníthetők legyenek. A modellek elkészítése és átkonvertálása után azokat elhelyezzük a VirCA megfelelő könyvtáraiban, és innentől kezdve azok megjeleníthetővé válnak. Ennek a megjelenítésnek két módját különíthetjük el. Az első ezek közül, amikor az elkészített modellt a VirCA-ban egy szoba részeként szeretnénk megjeleníteni, ekkor létre kell hoznunk egy könyvtárat, ami megegyezik a szoba nevével és egy ezzel megegyező nevű, „scene” kiterjesztésű fájlt, amelyben három koordinátával leírhatjuk, hogy a szobában melyik tárgy hová kerüljön, ezen kívül lehetőségünk van fényforrások elhelyezésére is a virtuális térben. A másik módszer az, amikor ebben az általunk létrehozott szobában szeretnénk megjeleníteni egy gépet vagy robotot. Ekkor ezt egy általunk megírt komponensen keresztül jelenítjük meg, amelyet a továbbiakban részletesebben is kifejtünk majd.

A modellek továbbfejlesztésére szükségünk azért volt szükség, hogy a két alkalmazást, a Rubik-kocka kirakását és az inkrementális lemezalakítást valós időben a VirCA-n keresztül is követni, illetve magát a robotot modellen keresztül vezérelni tudjuk. Az előző munkánk során elkészítettük a szobát és a robot modelljét, azonban több dolog is hiányzott még ahhoz, hogy ezek az alkalmazások pontosan úgy nézzenek ki a virtuális térben, ahogyan a valóságban is. Több modellel is ki kellett egészítenünk az előzőleg elkészült munkánkat, ezek a következők voltak: a Rubik-kocka, a kockához tartozó állvány, egy megfogó a robothoz, amellyel az képes felemelni és forgatni azt, továbbá egy másik megfogó eszköz a robothoz, amely az inkrementális lemezalakításhoz szükséges,

valamint az ehhez tartozó a lemezdarab és azok különböző állapotai, amelyekkel látható lesz, hogy a megmunkálás éppen milyen fázisban van.

Ahhoz azonban, hogy megértsük, miért volt mindezek elkészítésére szükségünk, és hogy mi is történik valójában a kockakirakás és a lemezalakítás folyamán, szükség van arra, hogy röviden elmagyarázzuk ezek folyamatát a valóságban.

4.1 Rubik kockát kirakó alkalmazás

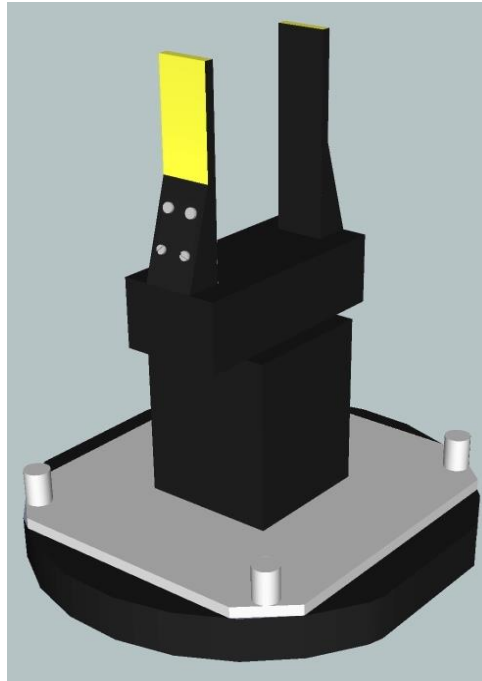
Ebben az alkalmazásban a robot kirakja a Rubik-kockát egy külön erre a célra kialakított megfogó eszközzel. A folyamat első lépéseként a robot megmutatja egy, a kockatartó állványban elhelyezett kamerának a Rubik-kocka mind a hat oldalát. Ez az állványban elhelyezett kamera összeköttetésben áll egy színfelismerő programmal, amely - miután a robot megfelelő sorrendben megmutatta a kocka összes oldalát - felismeri a kocka alapállapotát, majd ezt az állapotot eltárolja egy „log” fájlban. A színek felismerése után a program a Singmaster jelölőrendszert használja és kiszámolja, hogy milyen lépéseket kell tenni a kocka kirakásához, majd ezeket a lépéseket szintén eltárolja ugyanazon „log” fájl másik soraiban. Ez egy fontos lépése a folyamatnak, mert ebből a log fájlból fogunk információkat nyerni, amit a 6.2 fejezetben, majd részletesen is ismertetünk. Ezt követően a program a kiszámolt lépéseket elküldi a robotnak, amely ezek alapján előre leprogramozott mozgásokkal sorban végrehajtja azokat és kirakja a kockát.

4.2 Az ISF alkalmazás

Az inkrementális lemezalakítás [10] során egy külön erre a célra kialakított nyomófejet használ a robot a síklemez megformálására. Ezen a nyomófejen található egy erőmérő berendezés, amelynek a végén egy fémgolyó, amely érintkezik a lemezfelülettel. A síklemez darab egy erre a célra kialakított keretben van rögzítve. Lemezalakításkor a robot, különböző alakzatokat képes megformálni a keretbe befogott síklemez darabokból olyan módon, hogy az erre a célra kialakított nyomófejjel nyomja a felületet egy előre beprogramozott útvonal mentén haladva. Ez az eljárás akkor különösen hasznos, ha csak egyedi és kis mennyiségű munkadarab legyártására van szükség, ugyanis ez a folyamat hosszabb, mint például egy préseléssel készült munkadarab elkészítése, azonban itt nincs szükség a forma legyártására.

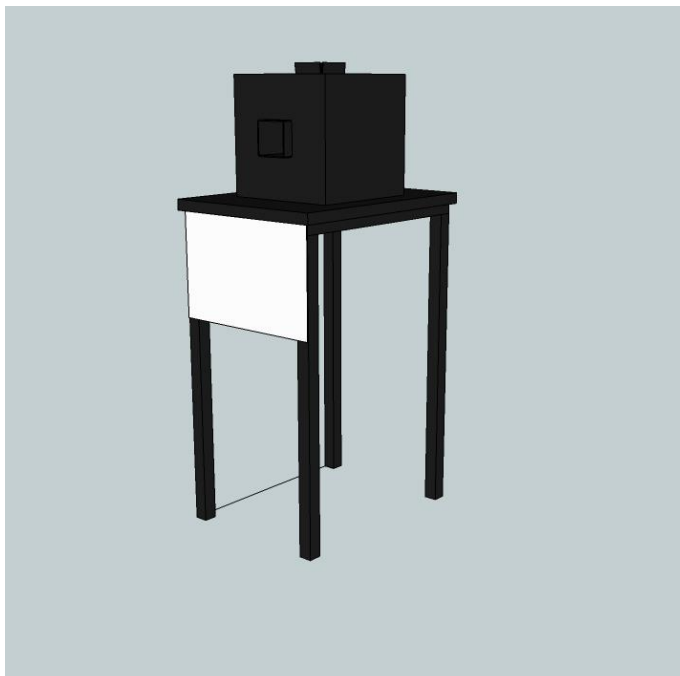
4.3 Az új modellek

Feladatunkat a kockamegfogó elkészítésével kezdtük el. Erre a már az előzőekben is említett Google Sketchup programot használtuk. A pontos méretek levétele után elkészítettük a megfogó modelljét. Ebben az esetben két modellt is csináltunk a megfogó két állapotáról. Az egyik állapot, amely során a megfogó nyitva van (lásd 2. ábra) a másik állapot pedig, amely során az eszköz „megfog” állapotban van.



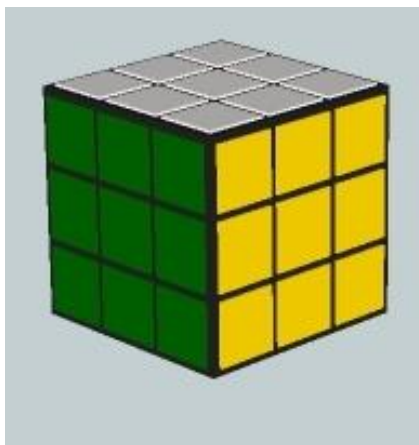
2. ábra Kockamegfogó eszköz

Ezt követően elkészítettük a kockatartó állvány modelljét (lásd 3. ábra), amelyben a szín felismeréshez szükséges kamera helyezkedik el. Erről szintén méreteket vettünk, majd létrehozuk a modellt és átkonvertáltuk, ahogyan az előző dolgozatban [1] ezt leírtuk.



3. ábra Kockatartó állvány

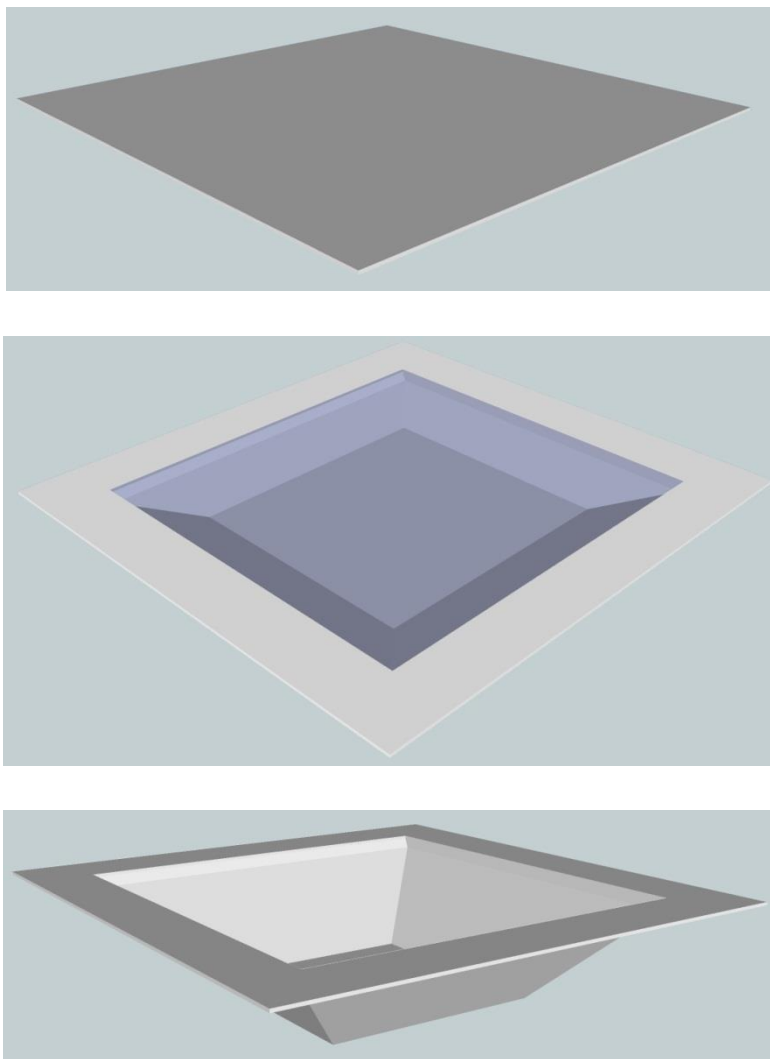
Ezt követte a kocka modelljének elkészítése (lásd 4. ábra). Ez már nem volt olyan egyszerű, mint az előző néhány modell elkészítése, mert nem volt elég méreteket venni a kockáról és elkészíteni a modellt, hanem a modellezés előtt végig kellett gondolni azt, hogyan kell felépítenünk a kockát ahhoz, hogy annak állapotváltozásai követhetők legyenek, és meg tudjuk jeleníteni a VirCA-ban. A modellt úgy készítettük el, hogy létrehozunk egy tömör alapkockát, majd erre elkészítettük hat darab, különböző színű kis lapot.



4. ábra Rubik-kocka modellje

A kockakirakás vizualizációjához szükséges modellek elkészítése után az inkrementális lemezalakításhoz tartozó testek modellezése következett. Ehhez az alkalmazáshoz el kellett készítenünk a nyomófejet, a síklemez munkadarabot, valamint annak különböző állapotait azért, hogy a lemezalakítás szintén követhető legyen a VirCA-n keresztül.

Először itt is a nyomófejet modelleztük, ahogyan eddig is, a Google Sketchup programmal. Miután méreteket vettünk a nyomófejről, elkészítettük annak modelljét. Ezt követően pedig a lemezdarabot modelleztük és annak a megmunkálás során kialakuló állapotait (lásd 5. ábra).



5. ábra Lemezállapotok

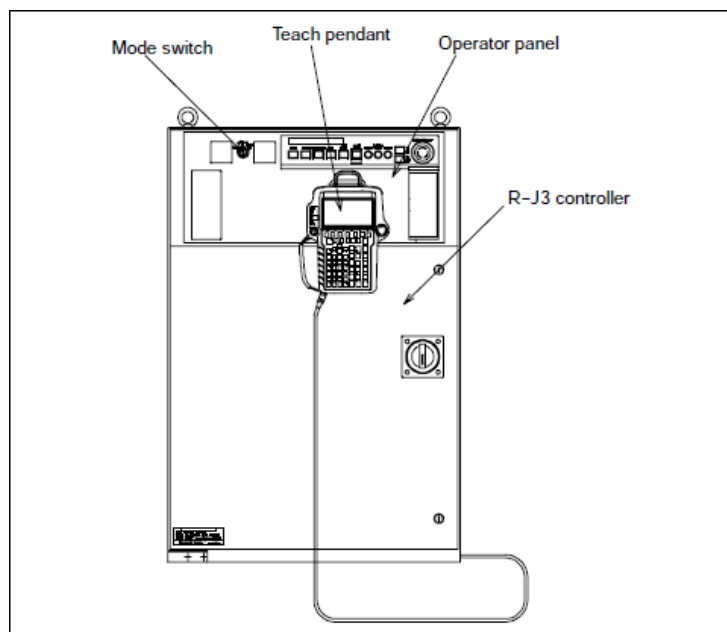
A keret elkészítésére, amely a valóságban a lemez befogására szolgál, már nem volt szükség, mert azt az előző munkánk során már ezt már megtettük és meg is jelenítettük a VirCA-ban. Ezek után következhetett a VirCA komponensek megírása, amelynek részleteit a 7. fejezetben bővebben is ismertetni fogjuk.

5 A soros kommunikáció alkalmazása a robot és a modell szinkronizálására

Munkánk a robotvezérlő és a VirCA közötti soros kommunikáción alapul. Soros porton keresztül történik a csuklókoordináták lekérdezése, amelyet egy komponens végez el, majd ezeket az adatokat egy adat porton keresztül továbbítja egy másik komponens felé, amely a kapott csuklókoordinátákat beállítja a modellen is, majd ezt a VirCA megjeleníti számunkra. Soros porton történik a visszirányú kommunikáció is, amikor mi vezéreljük a robotot a VirCA rendszeren keresztül. Ekkor a VirCA-ban megadott utasítást megkapja a Fanuc robot VirCA komponense, ami ezt továbbítja a SerialCommunicator VirCA komponens felé, ez a komponens pedig elküldi az utasítást a robotvezérlőnek a soros porton keresztül. Ezt követően a robotvezérlőn futó KAREL nyelven megírt program reagál az utasításra, és elindít egy általunk a VirCA-ból meghívott programot, illetve megmozdítja a robotot.

5.1 A robotvezérlő

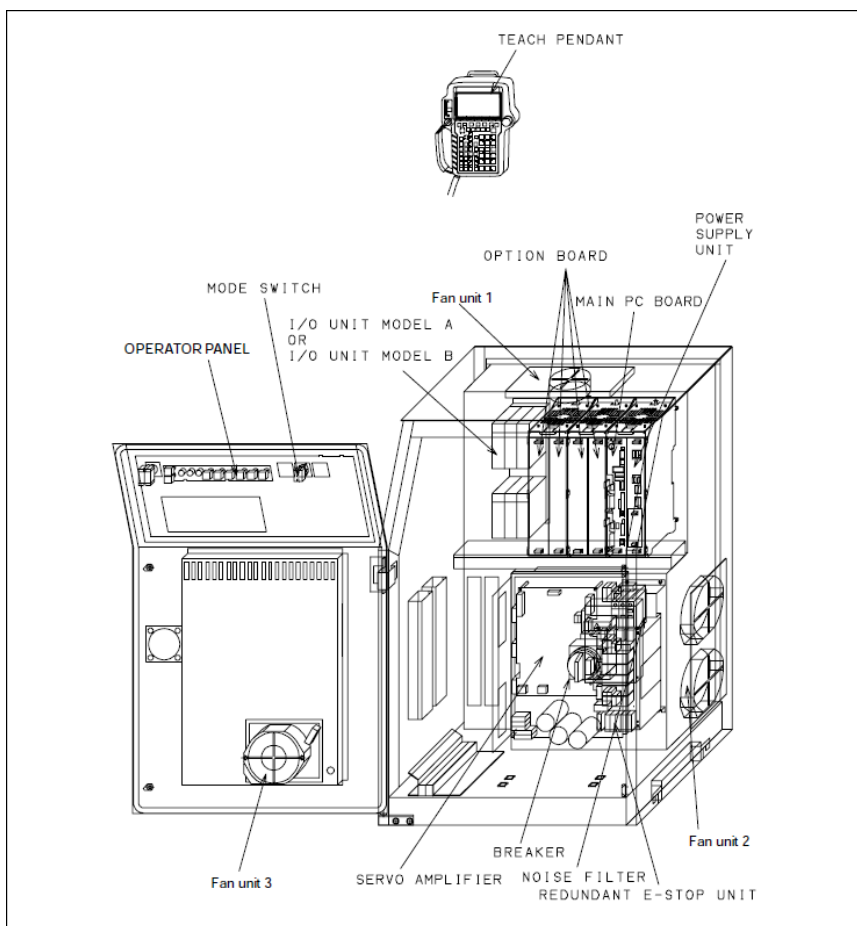
Az általunk modellezett Fanuc robothoz tartozik egy robotvezérlő egység is (lásd 6. ábra), amely a robot vezérléséért felelős. Ez két fő egységből áll: az R-J3-as típusú robotvezérlő egységből, és az ehhez tartozó tanító konzolból, amellyel programozhatjuk, illetve közvetlenül mozgathatjuk a robotot.



6. ábra Előlnézeti kép a robotvezérlőről

A fent látható 6. ábra jól szemlélteti a robotvezérlő legfontosabb részeit. A mód kapcsolóval „Mode switch” három mód közül választhatunk. „T1”, ha a robotot mozgatjuk: ilyenkor teljes sebességgel (2000 mm/s) történik a mozgás. „T2” állásban tudjuk könnyedén megtanítani a robotnak a különböző mozgásokat, ekkor a maximális sebesség 250 mm/s-ra korlátozódik. A kapcsoló harmadik lehetséges állása pedig az AUTO mód, amelyre akkor van szükség, ha a robotvezérlőn egy KAREL programot szeretnénk lefuttatni. A mi esetünkben ennek az állapotnak lesz fontos szerepe, hiszen mi egy, a robotvezérlőre megírt KAREL program segítségével fogjuk azt vezérelni. A tanító konzol a vezérlő második legfontosabb eleme. Ennek segítségével vagyunk képesek mozgatni a robotot, ugyancsak ezzel az eszközzel tudjuk programozni is, amely során megtaníthatjuk a különböző mozgásokat, majd az elmentett programjainkat bármikor meghívhatjuk ezen az eszközön keresztül. A programunk tesztelése alatt sokszor volt szükségünk ennek az eszköznek a használatára is, hiszen az általunk megírt KAREL program robotvezérlőre való feltöltésekor is szükséges az eszköz használata, valamint ezzel tudjuk kiválasztani és futtatni a robotvezérlőn tárolt programokat. Az 6 ábrán jelölve látható az „Operator panel”, amelyre szintén szükség van a robot vezérlése illetve programok futtatása esetén. Ezen a panelen különböző állapotjelző LED-eket és kapcsolókat találhatunk, információkat kaphatunk arról, hogy a robot milyen állapotban van. Ha egy KAREL programot

szeretnénk elindítani, akkor ahhoz nem elég azt a tanító konzolon kiválasztani, hanem a „Cycle start” gombot benyomva tehetjük meg, figyelve arra, hogy a „Mode switch” ekkor AUTO állapotba legyen állítva. Az operator panelen található a PCMCIA kártya csatlakozó felülete, amely elengedhetetlen, ha programokat szeretnénk feltölteni a vezérlőre. Magán a robotvezérlőn található még egy nagy főkapcsoló, mellyel áram alá tudjuk helyezni a berendezést, valamint lefelé fordítva e kapcsolót, kinyithatjuk a robotvezérlőt és hozzáférhetünk a belsejéhez. A kapcsoló úgy van kiképezve, hogy semmilyen esetben sem lehet kinyitni a vezérlőt úgy, hogy az áram alá van helyezve.



7. ábra A robotvezérlő belülről

Ha kinyitjuk a robotvezérlőt, akkor láthatjuk annak belső felépítését, hozzáférhetünk a különböző modulokhoz, az alaplaphoz, IO egységekhez. A mi esetünkben is szükség volt arra, hogy hozzáférjünk a robotvezérlő belsejéhez, hiszen kapcsolatot szerettünk volna létesíteni a vezérlő és egy másik számítógép között, amelyet egy RS-232 porton keresztül tettünk meg. Erre azért volt szükség, mivel ez a szinkronizált vezérlésnek

alapeleme. A port kivezetése az „Operator panel” hátulján található meg. Ide kötöttük be a soros kábelt, majd a vezérlő oldalán kiveztük azt és csatlakoztattuk a komponenst futtató számítógépbe. Ezt követően hozzáfektünk a KAREL program megírásához, hogy tesztelheessük: működik-e a soros kommunikáció.

5.2 A KAREL program bemutatása

A KAREL programozási nyelv egy robotprogramozási nyelv, amely nevét Karel Čapek-ről, az egyik legnagyobb hatású cseh íróról kapta, mégpedig azért, mert ő használta először a nemzetközileg is gyakran használt robot szót, az 1921-ben megjelent, R.U.R. című játékában. Ezt a nyelvet elsősorban Fanuc robotok programozására használják.

A mi esetünkben azért esett erre a nyelvre a választás, mivel egy tanító konzol programmal nem lenne lehetőségünk csak arra, hogy különböző mozgásokat megtanítsunk a robot számára, itt azonban szükség volt egy olyan programra, amely folyamatosan fut a robotvezérlőn, és soros porton kommunikál a számítógépen futó SerialCommunicator komponenssel. A program megírását egyszerű szöveges fájlban tehetjük meg olyan módon, hogy a kiterjesztésnek a „.kl”-t adjuk meg. Ezt követően egy egyszerű notepad alkalmazással - illetve a mi esetünkben a notepad++ alkalmazást használva - írhatjuk meg a kívánt programot. Ennek egy nagy hátránya, hogy nincsen egy olyan integrált fejlesztői környezet, amely szintaktikai ellenőrzést végezne az általunk írt kódon, így ha ilyen hiba történik, az nem derül ki azonnal, csak a fordítás folyamán.

A fordítás egy Microsoft Windows rendszerekre telepíthető WinOLPC fordítóprogrammal történik, amely az általunk látható és értelmezhető „.kl” kiterjesztésű fájlt átalakítja egy a robotvezérlő által értelmezhető „.pc” kiterjesztésű fájlra. Ha a kód írása során szintaktikai hibát vétettünk, amit nem vettünk észre, akkor azt a program fordításkor fogja jelezni nekünk. Miután a fordítás megtörtént, a fájlt feltölthetjük a robotvezérlőre. Ezt egy PCMCIA kártyán keresztül tesszük meg, amelyre felmásoljuk a lefordított programot. A kártyát a robotvezérlő operátor paneljén elhelyezkedő foglalatba kell illeszteni, majd a vezérlő indítása után a tanító konzol segítségével felmásoljuk a fájlt a robotvezérlőre, ezt követően pedig kiválasztjuk és futtatjuk azt.

Az általunk megírt KAREL programnak több feladatot is el kellett látnia. Egyik ezek közül az, hogy a robot éppen aktuális csuklókoordinátáit folyamatosan küldi a soros portra. Ez a feladat egy függvénnyel valósítható, meg amelynek neve CURJPOS. Több függvény is létezik a robot pozíciójának lekérésére, mi azonban azért ezt választottuk, mert a Fanuc robot VirCA komponenssel való együttműködése így a legegyszerűbb, hiszen az összeállított modellt így csak egyszerűen a csuklókoordináták behelyettesítésével tudjuk mozgatni. A Fanuc robot VirCA komponens leírásánál ezt a részt részletesen tárgyaljuk. A program ezzel azonban még nem volt készen, mert bár a robot kézzel való mozgatásánál nem volt probléma, a pozíciók folyamatos küldésével viszont esetünkben arra is szükség volt, hogy az aktuális pozíciókat egy programozott mozgás végrehajtásakor is küldje a soros portra. Először azt szeretnénk volna megvalósítani, hogy multitasking segítségével a KAREL program fut és folyamatosan küldi a csuklókoordinátákat, miközben a vezérlőn kiválasztunk és elindítunk egy operator panel segítségével egy programot, amely a robotot irányítja. Ám az operator panel írt program futásakor KAREL program háttérben való futtatására nem volt lehetőség, ezért egy másik megoldást választottunk.

A KAREL nyelvben létezik olyan függvény, amely segítségével egy megírt operator panel programból képesek vagyunk pozíció-adatokat nyerni. Ha ezt elértük, a függvény segítségével kinyert pozíciókból lehetőségünk van arra, hogy a KAREL program segítségével beállítsuk a robot csuklóit a tanító konzolra írt program soraiból kinyert adatokkal. Így tehát bármilyen - a robotnak betanított - mozgás futtatható lesz KAREL programon keresztül, és az aktuális pozíció így mindig kiküldésre kerül a soros porton. Ennek a megoldásnak még egy hatalmas előnye van, ugyanis lehetőségünk lenne arra is, hogy a mozgásokat offline módban előre leprogramozzuk KAREL nyelven. Az offline programozás azonban körülményesebb, mintha tanító konzolon keresztül tanítanánk meg a robotnak a különböző mozgásokat. Ez utóbbinál mi magunk mozgatjuk a robotot a különböző pozíciókba, és így veszünk fel a mozgáshoz szükséges más-más végpontokat, amelyet aztán a program összeköt.

Ezekkel a megoldásokkal a programunk már képes volt arra, hogy egy, a vezérlőn elindított program futása közben küldje a koordinátákat a soros portra. Meg kellett még azonban oldani azt is, hogy abban az esetben, ha a vezérlő a VirCA-ból kap utasítást, akkor

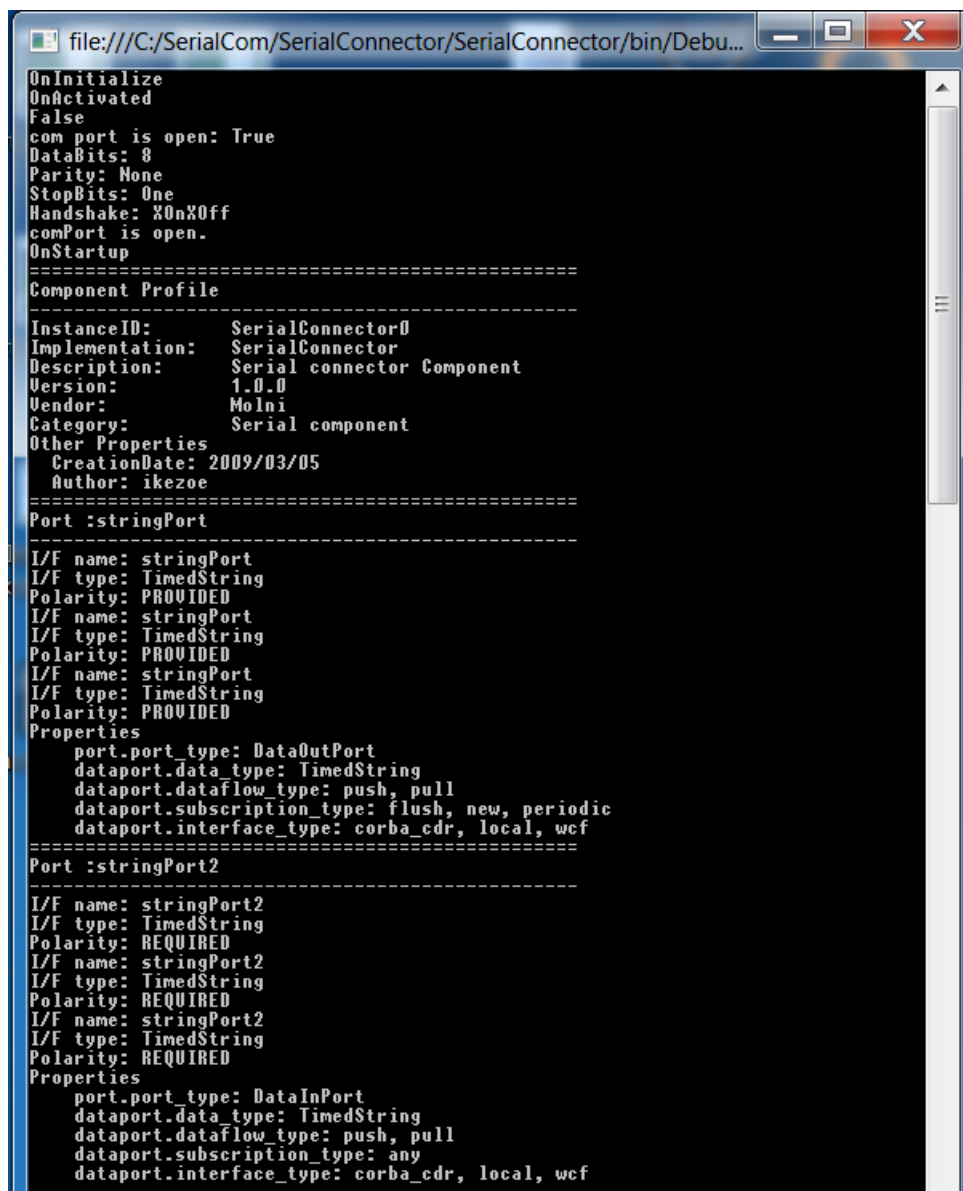
annak megfelelően indítson el egy programot, illetve mozgassa meg a robot valamelyik csuklóját. Ezek a következőképpen történnek: attól függően, hogy milyen programot szeretnénk elindítani, a VirCA menüjében kiválasztjuk a megfelelő programot, majd a VirCA visszaküld egy kulcsszót a SerialCommunicator komponens felé. Ezt követően, ha a SerialCommunicator megkapta ezt a kulcsszót, amely jelen esetben az indítani kívánt program neve, tudatni fogja a robotvezérlőn futó KAREL programmal, hogy egy programindítást szeretnénk végrehajtani. Ezt úgy teszi meg, hogy küld egy „PRG” kulcsszót a KAREL programnak, amely ennek hatására tudni fogja, hogy el kell indítania egy programot. Válaszul erre a KAREL program küld egy „RDY” üzenetet, hogy tudassa a SerialCommunicator komponenssel, hogy készen áll a program indítására és várja az indítani kívánt program nevét. Válaszul a SerialCommunicator elküldi a Fanuc robot komponensből kapott program nevét a KAREL programnak, amely ezt követően elindítja a kiválasztott programot. A program futásával párhuzamosan a soros porton kiküldött csuklókoordinátákat a SerialCommunicator komponensen keresztül a Fanuc robot komponensnek eljuttattuk, amely a kapott adatok alapján beállította a robotmodell csuklóit a valós robotnak megfelelő állapotba. Arról, hogy ez hogyan történik, bővebben a következő fejezetben beszélünk.

6. VirCA komponensek

Ebben a fejezetben részletesen ismertetjük az általunk elkészített főbb komponensek feladatát és az ezek között létrejövő kapcsolatokat. Az általunk elkészített komponenseket két programozási nyelven írtuk meg. A SerialCommunicator VirCA komponenst C#-ban, míg a Fanuc robot VirCA komponenst C++ -ban programoztuk. Ezek a komponensek úgynevezett Robotics Technology (RT) komponensek. Az OpenRTM egy nyílt forráskódú robot „middleware” technológia, amelyet a Japan's National Institute of Advanced Industrial Science and Technology fejlesztett ki. Ennek lényege, hogy különböző gépeket, robotokat, eszközöket kapcsolhatunk össze számítógépes hálózatokon keresztül. Ezeket a komponenseket kötjük össze a VirCA kötőgetőjén keresztül is. Maga a VirCA is egy RT komponens, ám ez jóval összetettebb, mint az általunk megírt komponensek. Feladata az, hogy az általunk létrehozott robotokat, gépeket három dimenzióban megjelenítse, valamint ezen felül képes ezeket vezérelni is. Egy RT komponens leegyszerűsítve különböző függvényekből áll, amelyek más-más események bekövetkezésekor kerülnek meghívásra. Ilyenek például az OnInitialize, OnStartup, OnActivated, OnExecute függvények. Mi a feladatunk elkészítése során ezeket használtuk.

6.1 A serial communicator VirCa komponens

Az első komponens, amelyet elkészítettünk a SerialCommunicator VirCA komponens volt. Feladata nagyon egyszerű, továbbítania kell a robotvezérlőtől soros porton kapott adatokat a Fanuc robot VirCA komponens felé, illetve az onnan érkező adatokat ennek a komponensnek kell elküldenie soros porton keresztül a robotvezérlő számára. A komponens rendelkezik egy be-és kimeneti adatporttal, illetve soros porton kommunikál a robotvezérlővel.



```
file:///C:/SerialCom/SerialConnector/SerialConnector/bin/Debu...
OnInitialize
OnActivated
False
com port is open: True
DataBits: 8
Parity: None
StopBits: One
Handshake: XOnXOff
comPort is open.
OnStartup
=====
Component Profile
-----
InstanceID:      SerialConnector0
Implementation:  SerialConnector
Description:     Serial connector Component
Version:         1.0.0
Vendor:         Molni
Category:        Serial component
Other Properties
  CreationDate:  2009/03/05
  Author:        ikezoe
=====
Port :stringPort
-----
I/F name: stringPort
I/F type: TimedString
Polarity: PROVIDED
I/F name: stringPort
I/F type: TimedString
Polarity: PROVIDED
I/F name: stringPort
I/F type: TimedString
Polarity: PROVIDED
Properties
  port.port_type: DataOutPort
  dataport.data_type: TimedString
  dataport.dataflow_type: push, pull
  dataport.subscription_type: flush, new, periodic
  dataport.interface_type: corba_cdr, local, wcf
=====
Port :stringPort2
-----
I/F name: stringPort2
I/F type: TimedString
Polarity: REQUIRED
I/F name: stringPort2
I/F type: TimedString
Polarity: REQUIRED
I/F name: stringPort2
I/F type: TimedString
Polarity: REQUIRED
Properties
  port.port_type: DataInPort
  dataport.data_type: TimedString
  dataport.dataflow_type: push, pull
  dataport.subscription_type: any
  dataport.interface_type: corba_cdr, local, wcf
```

8. ábra A SerialCommunicator VirCA komponens

Minden komponenst, így ezt is a VirCA köztöztetőjén keresztül tudjuk megjeleníteni illetve aktiválni. Az OnInitialize függvény hívása történik először, amikor a komponenst elindítjuk. Ezt a soros port megnyitása és a pufferek kiürítése követi.

Mielőtt ez megtörténik, egyeztetni kell a port beállításait, hogy mindkét oldalon megegyezzenek, így itt azokat a beállításokat használtuk, amelyeket a KAREL programban is szintén ugyanígy beállítottunk. Ezek a beállítások nagyon fontosak, hiszen ha nem egyeznek meg mindkét oldalon, akkor megeshet, hogy nem sikerül kommunikálni a másik

oldallal, illetve ha van is kapcsolat, akkor az elküldött adatok a másik oldalon hibásan jelennek meg. A mi esetünkben a következő beállításokat használtuk:

- „Baud rate: „9600”, amely a másodpercenként átvitt bitek számát határozza meg
- „Data bit: „8”, amellyel az adatbitek számát határozhatjuk meg
- „Parity: „None”, vagyis nem állítottunk be hibakeresésre szolgáló paritást egyik oldalon sem
- „Stop bits: „1”: a stop biteket jelenti, amelyeket minden karakter végén kiküldünk, így a másik oldal érzékeli, hol lesz a karakter vége

Valamint szükséges még a Data Terminal-t Ready állapotba helyezni, amely jelzi, hogy ez az oldal is kész az adatfogadásra. Ha ezt nem tennénk meg, akkor a másik oldalról érkező információkat nem lennének képesek kiolvasni a soros portról, illetve nem érkezne meg az adat.

Ezt követően lefut az OnStartup függvény és megtörténik minden, amely ebben a függvényben van leírva. Az OnStartup függvényben szöveges információkat írunk ki a konzolra, ezek a soros port beállításai és egyéb információk a komponensről. Amikor ez megtörténik akkor a komponens inaktív módban láthatóvá válik a VirCA köztöztetőjében. Ezt követően lehetőségünk van aktiválni a komponensünket, ilyenkor az OnActivated függvényben leírt események kerülnek végrehajtásra.

A programot úgy készítettük el, hogy az aktiváláskor megnyitott soros porton keresztül küld egy üzenetet a robotvezérlő felé, hogy készen áll az adatok fogadására. Ez az üzenet a „Ready”. Aktiváláskor a portot ezen beállítások szerint megnyitjuk, és ezt követően a komponens készen áll az adatok fogadására. Ahhoz, hogy ezek a koordináták feldolgozhatók legyenek, tudnunk kellett azt, hogy milyen formában kapjuk majd meg a csuklókoordinátákat. Ezt a már előzőekben említett KAREL program megírásakor állítottuk be. A program a port megnyitása után átlép az OnExecute függvénybe, ahol egy külső „rtc.config” fájl segítségével be tudjuk állítani, hogy másodpercenként hányszor fusson le. Amíg a komponensünk aktiválva van, addig ez a függvény folyamatosan az „rtc.config” fájl beállításai szerint fut le, akár másodpercenként többször is. Ebben a

függvényben történik meg a soros portról érkezett adatok továbbítása az adat portra. A portról kiolvasott koordinátákat itt a program egy string típusú változóban tárolja el úgy, hogy minden csuklókoordináta egy szóközzel van elválasztva egymástól.

Az OnExecute függvényen kívül a program tartalmaz még egy eseménykezelőt is, amelynek az a feladata, hogy folyamatosan figyeli a soros port bemeneti pufferét, ha pedig erre adat érkezik, akkor azt beleírja a már előzőekben említett string típusú változóba, ezáltal az OnExecute függvény folyamatosan a legfrissebb aktuális pozíciót fogja beleírni az adatport változójába. A komponensek közötti kapcsolat működéséről bővebben a 8.3-as fejezetben beszélünk.

6.2 A Fanuc robot VirCA komponens

Ez a komponens felelős a robotmodell megfelelő összeállításáért, a kockakirakó illetve az ISF alkalmazás háromdimenziós modelljének felépítéséért. Lehetőségünk lett volna arra is, hogy a kocka összeállítását valamint a lemezdarabok állapotváltozásainak követését külön-külön komponensekkel oldjuk meg. Mi azonban az egyszerűség kedvéért azt a megoldást választottuk, hogy ezeket egy komponensben elhelyezve állítjuk össze. A robot felépítéséről az előző TDK dolgozatunkban már bővebben [1] beszéltünk, így erről most csak néhány szót ejtünk.

A robot pozíciójának beállítását úgy oldottuk meg, hogy a porton érkező aktuális koordinátákat a komponens eltárolja egy string típusú változóban. Ebből a változóból aztán egy stringstream segítségével külön változókba szedjük a kapott csuklókoordinátákat. Miután ez megtörtént, át kell alakítanunk a szöveggént tárolt értékeket long típusúvá. Erre azért van szükség, mert az összeállított robot minden egyes csuklójához külön szögértéket tudunk hozzárendelni, így megváltoztathatjuk annak orientációját. Ezt a szögértéket egy long típusú változó behelyettesítésével adjuk meg. Ezekből a változókból hat darab van, hiszen hat szabadságfokú a robotunk. Ezek a változók név szerint: joint_0 joint_1, joint_2, joint_3, joint_4, joint_5, amelyeket hozzárendelünk a robot minden csuklójához. Az értékek hozzárendelését pedig az OnActivated függvényben tesszük meg, így a komponens aktiválása után, ha csuklókoordinátákat kapunk a robotvezérlőtől, akkor a robot orientációja a robot állásának megfelelően azonnal meg fog változni.

```

//fanuc_robot mozgatása bejövő adatok alapján.

string joint_0str=fanuc[3];//3
string joint_1str=fanuc[4];//4
string joint_2str=fanuc[5];//5
string joint_3str=fanuc[0];//0
string joint_4str=fanuc[1];//1
string joint_5str=fanuc[2];//2

//bejövő adatok konvertálása

joint_1=atof(joint_0str.c_str());
joint_2=atof(joint_1str.c_str());
joint_3=atof(joint_2str.c_str());
joint_4=atof(joint_3str.c_str());
joint_5=atof(joint_4str.c_str());
joint_6=atof(joint_5str.c_str());

cout<<joint_1<<" "<<joint_2<<" "<<joint_3<<" "<<joint_4<<" "<<joint_5<<" "<<joint_6<<endl;

// Az csuklók szögértékeinek beállítása

fanuc_1->setOrientation(Ogre::Quaternion(Ogre::Degree(joint_1),Ogre::Vector3::UNIT_Y));
fanuc_2->setOrientation(Ogre::Quaternion(Ogre::Degree((-1)*(joint_2)),Ogre::Vector3::UNIT_Z));
fanuc_3->setOrientation(Ogre::Quaternion(Ogre::Degree(joint_4),Ogre::Vector3::UNIT_Z));
fanuc_4->setOrientation(Ogre::Quaternion(Ogre::Degree(-1*joint_3),Ogre::Vector3::UNIT_X));
fanuc_5->setOrientation(Ogre::Quaternion(Ogre::Degree(joint_5),Ogre::Vector3::UNIT_Z));
fanuc_6->setOrientation(Ogre::Quaternion(Ogre::Degree(joint_6),Ogre::Vector3::UNIT_X));
mpAPI->refreshParts();
sleep(0.07);

```

9. ábra Kódrészlet: a csuklókoordináták beállítása

Ezt követően a csuklókoordináták beállítását el kell végeznünk az OnExecute függvényben is, így a robot mozgása már folyamatosan nyomon követhető lesz a VirCA-ban. A mozgás folytonosságát az execution rate paraméter beállításával tudjuk befolyásolni, mellyel a már említett OnExecute függvény meghívásának gyakoriságát állíthatjuk be. Minél magasabb értéket állítunk be, a robot mozgása annál folyamatosabb lesz.

A robot felépítése és csuklókoordinátáinak lekérdezése után a következő feladatunk az volt, hogy összeállítsuk a Rubik-kockát. A kockát a robothoz hasonló módon hierarchikus objektumként építettük fel, melynek alapjául a cube_base elnevezésű elem szolgált. Miután ezt az elemet megfelelő módon pozícionáltuk a kockatartón, következhetett a színek regisztrálása a kockára. A kocka színeinek a modellezés során létrehozott hat különböző színű lapot használtuk fel.

```

//cube front view
cube_Pos[0][0] =Ogre::Vector3(5.9,4.9,4.9);
cube_Pos[0][1] =Ogre::Vector3(5.9,4.9,2.9);
cube_Pos[0][2]= Ogre::Vector3(5.9,4.9,0.9);
cube_Pos[0][3]= Ogre::Vector3(5.9,2.9,4.9);
cube_Pos[0][4]= Ogre::Vector3(5.9,2.9,2.9);
cube_Pos[0][5]= Ogre::Vector3(5.9,2.9,0.9);
cube_Pos[0][6]= Ogre::Vector3(5.9,0.9,4.9);
cube_Pos[0][7]= Ogre::Vector3(5.9,0.9,2.9);
cube_Pos[0][8]= Ogre::Vector3(5.9,0.9,0.9);
//cube back view
cube_Pos[1][0]= Ogre::Vector3(-0.1,0.9,4.9);
cube_Pos[1][1]= Ogre::Vector3(-0.1,0.9,2.9);
cube_Pos[1][2]= Ogre::Vector3(-0.1,0.9,0.9);
cube_Pos[1][3]= Ogre::Vector3(-0.1,2.9,4.9);
cube_Pos[1][4]= Ogre::Vector3(-0.1,2.9,2.9);
cube_Pos[1][5]= Ogre::Vector3(-0.1,2.9,0.9);
cube_Pos[1][6]= Ogre::Vector3(-0.1,4.9,4.9);
cube_Pos[1][7]= Ogre::Vector3(-0.1,4.9,2.9);
cube_Pos[1][8]= Ogre::Vector3(-0.1,4.9,0.9);

cube_Orient[0]=Ogre::Quaternion(Ogre::Degree(0), Ogre::Vector3::ZERO);
cube_Orient[1]=Ogre::Quaternion(Ogre::Degree(0), Ogre::Vector3::ZERO);
cube_Orient[2]=Ogre::Quaternion(Ogre::Degree(90),Ogre::Vector3::UNIT_Y);
cube_Orient[3]=Ogre::Quaternion(Ogre::Degree(90), Ogre::Vector3::UNIT_Y);
cube_Orient[4]=Ogre::Quaternion(Ogre::Degree(90), Ogre::Vector3::UNIT_Z);
cube_Orient[5]=Ogre::Quaternion(Ogre::Degree(90), Ogre::Vector3::UNIT_Z);

for(int i=0; i<6; i++)
{
    for (int j=0; j<9; j++)
    {
        ss.str("");
        st.str("");
        ss<<i;
        st<<j;
        name= " "+ss.str()+" "+st.str();

        cube[i][j] = new CyberDevice::CyberDevicePart(name, cube_Pos[i][j], cube_Orient[i], cube_base_Scale);
        mpAPI->registerHierarchicalObject(cube_color2[i][j], myPoi, cube_base, cube[i][j]);
    }
}

```

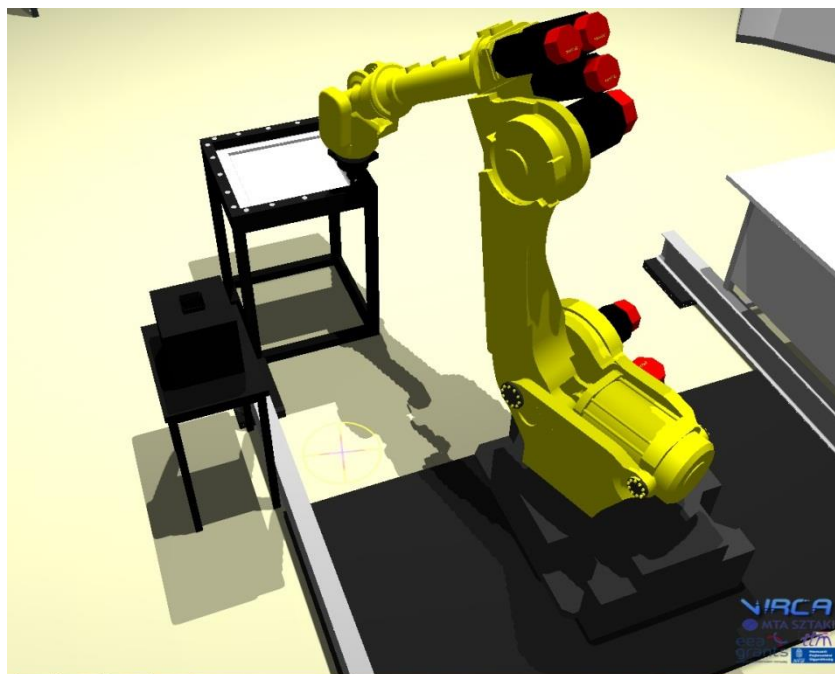
10. ábra A kocka összeállítása

Ezeknek a lapoknak a középpontját a megfelelő helyre pozicionálva hozzáregisztráltuk a cube_base nevű, kocka alapjául szolgáló elemhez. Oldalanként 9, összesen 54 darab színes lap regisztrálására volt szükség. Ezt követően a komponens indításakor a kocka kirakott állapotban jelenik meg a kockatartóban.

A kocka állapotváltozásának követésére a következő módszert találtuk ki. A komponens kiolvassa a kockakirakó szoftver által leírt kezdeti állapotot egy szöveges fájlból. Ez a szöveges fájl egy megegyezés szerinti körülményi irány alapján sorban leírja az egyes kis lapok színeit. A körülményi irányt és az adatokat kiolvassva a szöveges fájlból már képesek leszünk kiszínezni a kockát. Ez úgy történik, hogy a fájlból kiolvasott értékeket eltároljuk egy string típusú tömbbe elemenként. Miután ez megtörtént, kiregisztráljuk a kocka lapjait és magát a cube_base elemet, majd ezt követően visszaregisztráljuk a helyére

a cube_base elemet, viszont ekkor már a tömbben tárolt értékek alapján kerülnek fel a kockára a színes lapok, így a kocka pontosan olyan állapotban jelenik meg a modellben is, mint ahogyan a valóságban is.

Ezek után a robot elkezdi a kocka kirakását, amelynek mozgását a komponens követi a soros porton továbbított csuklókoordináták értékének folyamatos behelyettesítésével. Ahhoz, hogy tudjuk, a robot éppen megfogja-e a kockát vagy éppen elengedi azt, figyelniünk kell a megfogó állapotát. Erről információt a KAREL program küld a soros porton keresztül, amit a SerialCommunicator VirCA komponens egy a megfogónak létrehozott porton keresztül folyamatosan továbbít a Fanuc robot VirCA komponens számára. Ez a két állapot a „megfog” és az „elenged” lehet. Ha a komponens a porton keresztül egy „megfog” értéket kap, akkor ennek megfelelően a megfogó állapotának követésére létrehozott string típusú változó értékét „megfog”-ra állítja. Ekkor a Rubik-kocka egy általunk megírt „kocka-megfog” függvény segítségével kiregisztrálódik az éppen aktuális pozíciójából, és hozzáregisztrálódik a modell robot megfogójához. Így az folyamatosan a megfogóval együtt fog mozogni egészen addig, amíg újabb állapot nem érkezik az adat porton keresztül. Ha innen „elenged” értéket kap, akkor a „boolean”- típusú változó értéke false-ra változik, és ezt követően a kocka az éppen aktuális pozíciójában újraregisztrálódik a kockatartó állványhoz. Majd ha a robot forgat a kockán, akkor a megfogó portján keresztül egy „forgat”-érték érkezik. Ennek hatására történik meg a következő kocka-állapot kiolvasása a kockakirakó program által létrehozott fájlból. Ezután a kocka szintén újraregisztrálásra kerül a megfelelő színekkel. A kockakirakás követése során a komponens ezeket a lépéseket hajtja végre.



11. ábra ISF alkalmazás

A Fanuc robot VirCA komponens másik feladata az ISF alkalmazás megjelenítése. Abban az esetben, ha az ISF alkalmazást szeretnénk elindítani, akkor azt a megfelelő menüpont kiválasztásával a VirCA-ban tehetjük meg. Miután ez megtörtént és eldöntöttük, hogy az ISF alkalmazást szeretnénk elindítani, a csuklókoordináták folyamatos behelyettesítése mellett beregisztrálásra kerül az elkészített síklemez darab modellje a keretbe. Ezt követően folyamatosan zajlik a robot pozíciójának követése. Időközönként - a Rubik-kocka kirakásához hasonlóan - itt is kiregisztrálásra kerül a lemezdarab, majd ezt egy újabb lemezdarab beregisztrálása követi, ezáltal a modellben is nyomon követhetővé válik a lemez megmunkálásának folyamata.

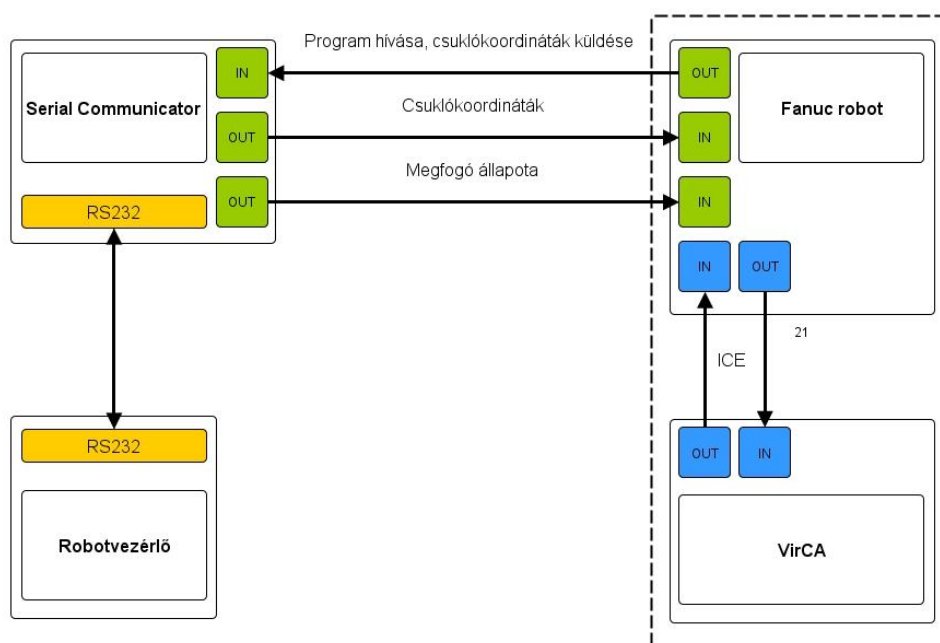
6.3 A VirCA komponens

Fontos megemlíteni a komponensek közül magát a VirCA-t is, hiszen ez is egy RT komponensként jelenik meg a VirCA kötőzetőjében. Ennek a komponensnek a feladata, hogy három dimenzióban megjelenítse a Fanuc robot VirCA komponens által összeállított modelleket, illetve annak környezetét, jelen esetben a szintén általunk elkészített robotlabor modelljét, amelyről előző TDK munkában [1] már részletesen beszámoltunk. A VirCA komponens rendelkezik két ICE porttal, egy bemeneti és egy kimeneti porttal. Ezen

keresztül kommunikál a rákapcsolt komponensekkel, jelen esetben a Fanuc robot VirCA komponenssel. Ehhez mindkét komponensnek rendelkeznie kell ki- és bemeneti ICE portokkal. Ha az ICE portok megfelelően vannak összekötve, akkor képesek vagyunk utasításokat küldeni a VirCA-ból bármelyik komponensünknek, amelyik rendelkezik VirCA porttal. Ezzel a módszerrel vagyunk képesek vezérelni a modellt illetve a valódi robotot is e komponens és a Fanuc robot VirCA komponens segítségével.

6.4 Komponensek közötti kapcsolat

Az előzőekben ismertetett komponensek különböző portokon keresztül kapcsolódnak egymáshoz. Ezeket a kapcsolatokat a következő (12.) ábra szemlélteti.



12. ábra A komponensek közötti kapcsolatok.

A 12. ábrán az előbb ismertetett komponensek közötti kapcsolat látható. A színes négyzetek jelentik a komponenseken elhelyezett portokat. A sárgával jelöltük a soros port, amelyek között a robotvezérlő és a SerialCommunicator komponensen kommunikál egymással. A SerialCommunicator komponensen helyezkedik el még három adatport is, amelyek közül egy bemeneti, kettő pedig kimeneti port. Ez utóbbiaknak különböző szerepük van: egyiken a komponens a csuklókoordinátákat továbbítja a Fanuc robot komponens felé, a másikon pedig a robot megfogójának állapotát küldi tovább. A bemeneti portra pedig a Fanuc robot komponens küld adatokat, amelyeket aztán a

SerialCommunicator továbbít a vezérlő felé, így elindíthat egy programot, illetve állíthat a robot csuklókoordinátáin. A Fanuc robot komponensen az előbb említett portoknak megfelelő bemeneti, illetve kimeneti portok találhatóak, valamint a komponens két ICE porttal is rendelkezik. A protok közül az egyik a bemeneti port, a másik pedig a kimeneti port melyeken keresztül a Fanuc robot komponens kommunikál a VirCA komponenssel. Tehát ha a VirCA-ból irányítjuk a modellt, illetve ha meghívunk egy programot, akkor az utasítások ezen a porton keresztül továbbítódnak majd a Fanuc robot komponens felé, illetve ha a robot orientációja megváltozik, akkor-e kapcsolaton keresztül azt a VirCA is észleli és megjeleníti.

A VirCA és Fanuc robot komponens az ábrán szaggatott vonallal van elkülönítve. Ennek az az oka, hogy e két komponensnek nem kell ugyanazon a számítógépen futnia, mint a SerialCommunicator komponensnek. Az internet és egy VPN kapcsolat segítségével ezek a komponensek akár földrészekkel távolabb lévő gépeken is futhatnak és összekapcsolhatjuk őket a VirCA kötöztetőjének segítségével.

7. Egyéb beviteli eszközök alkalmazása

7.1 A Kinect szenzor

Napjainkban a Kinect szenzor egy nagyon elterjedt eszköz, amelyet elsősorban Microsoft XBOX játékkonzolokhoz fejlesztettek ki, és azt a célt szolgálja, hogy a játékokat a mozgásunkkal vezérelhessük, joystick segítségével nélkül. Ezen kívül van egy másik felhasználási területe is a Kinect szenzornak, hiszen a Microsoft ingyenesen elérhetővé tette a Kinect SDK-t, így azt bárki letöltheti, és telepítheti a számítógépére, és ezáltal a számítógépéhez különböző programokat írhat rá. Így lehetőségünk van arra, hogy ezt az eszközt bekapcsoljuk a rendszerbe és egy komponensként tudjuk működtetni. A Kinect szenzor képes meghatározni az előtte álló ember távolságát, beazonosítja a végtagjait az ún. SkeletalTracking funkcióval, majd követi a beazonosított alak mozgását. A szenzor beazonosítja a fejet és követi annak mozgását, a testrész pozícióját pedig a komponens folyamatosan küldi a VirCA számára, ahol pedig a 3D pointer helyzete ennek megfelelően változik a virtuális térben, azaz ahogyan a Kinect előtt mozgunk, úgy változik a nézőpont is a VirCA-ban.

7.2 A Wii kontroller

A Kinect szenzorhoz hasonlóan a Wii kontrollerhez is letölthető SDK, így ennek programozására is van lehetőségünk. A Wii játékvezérlővel mi a robot modelljét mozgathatjuk. Terveink szerint a felhasználó a Wii kontrollert kezében tartva mozgathatja a robotot. A mozgatni kívánt szegmensét menüből lehet kiválasztani. A szegmens kiválasztása után biztonsági okokból a Wii kontrolleren megfelelő gombkombináció nyomva tartásával lehet indítani a robot mozgását. Ahogy a Wii kontrollert mozgathatjuk a kezünkben, úgy mozog a robot kiválasztott szegmense. A biztonság szem előtt tartása miatt a szabad mozgást szoftveres feltételekkel is korlátok közé kell szorítani, jóllehet a robotba és környezetébe be vannak építve a megfelelő hardveres védelmek.

8. Eddig elért eredmények

Előző TDK munkánk során [1] elkészítettük egy ipari robot, valamint környezetének háromdimenziós modelljét, majd ezt megjelenítettük a VirCA rendszerben. Ebben a dolgozatban ennek a munkának a folytatásaként az elkészült modellt és a valódi robotot szinkronizáltan vezéreltük különböző alkalmazásokban (Rubik-kocka kirakása, inkrementális lemezalakítás).

Munkánk során szükség volt néhány új test (Rubik-kocka, megfogók, kockatartó állvány) modellezésére, amelyeket először elkészítettünk, majd ezt követően megjelenítettük a modelltérben. Miután a szükséges modelleket elkészítettük az ISF és a Rubik alkalmazáshoz, el kellett készítenünk a komponenseket, valamint az ezekkel kapcsolatot tartó robotvezérlőn futó KAREL programot. Létrehoztuk a SerialCommunicator komponenst, amely soros porton keresztül kommunikál a robotvezérlővel, majd a Fanuc robot komponens következett, amely a robot, a Rubik-kocka kirakása és az inkrementális lemezalakítás modelljének megjelenítéséhez szükséges. Ezt követően pedig megírtuk azt a KAREL programot, amely elindítja a VirCA-ból kiválasztott programokat, valamint folyamatosan küldi az aktuális csuklókoordinátákat a komponensek számára.

Ezek után elkészült a szinkronizáltan vezérelhető modell, amely segítségével követhetjük a valódi robot mozgását, valamint képesek vagyunk a robotot a modellen keresztül vezérelni, akár abban az esetben is, ha földrészekkel távolabb vagyunk a valódi robottól.

Az eddig elért eredményeket publikáltuk CogInfoCom 2012 konferencián [11], emellett a Gyártástrend [12] újságban jelent meg munkánkról egy ismertető.

9. Továbbifejlesztési lehetőségek

Ebben a TDK-munkában bemutatott és megoldott feladatokkal munkánk még nem ért véget. További terveink közé tartozik, hogy a Solid Works-ben elkészített hajó modelljét közvetlenül a VirCA rendszerben jelenítsük meg. Miután a modell megjelent a virtuális térben, azaz a VirCA-ban, utána egy menüpont kiválasztásával lehet a megjelent modellt majd legyártani. A gyártási műveletet valódi robot végzi el, a modell elkészítéséhez egy sík lemezdarabot használ. Ebből fogja a robot elkészíteni a virtuális térben megjelent hajó modelljét. Az így legyártott modellt már is fel lehet használni a hajókészítés következő fázisában. Természetesen nem csak hajómodellek gyártásra alkalmas ez a project, hanem bármilyen Solid Works-ben elkészített alkatrészt legyártására is, valamint Kinect-szenzor más célú felhasználása, amelyben a felhasználó a szenzor elé áll, ezt követően az beszkenne annak arcát és testhálóját, amelyeket aztán a robot kialakítja egy sík lemezdarabból. Ez a folyamat pedig követhető és vezérelhető a VirCA rendszeren keresztül.

További terveink, hogy a robot vezérlését a biztonsági feltételeknek is eleget téve megtehessük, azaz a VirCA rendszeren keresztül Wii controller segítségével vezéreljük a modellt és valóságos robotot. A VirCA-án belül egy menü pont kiválasztásával lehet, majd ezt a funkciót elérni. A mozgatni kívánt szegmensét menüből lehet kiválasztani. A szegmens kiválasztása után biztonsági okokból a Wii kontrolleren megfelelő gombkombináció nyomva tartásával lehet indítani a robot mozgatását. Ahogy a Wii kontrollert mozgatjuk a kezünkben, úgy mozog a robot kiválasztott szegmense. Azért, hogy a robot szabad mozgatása biztonságosan megoldható legyen, ezt szoftveresen korlátoznunk kell a már robotba és környezetébe beépített hardveres védelmek mellett, melyek megakadályozzák, hogy a robot egy bizonyos magasság fölé emelkedjen, és ezzel kárt tegyen a környezetben.

Ez a fejlesztés azért hasznos, mert számítógépes modellen keresztül láthatjuk és vezérelhetjük a robot.

10. Irodalomjegyzék

- [1] Zátroch Zoltán - Tusz Bettina - Molnár Zsolt: *Háromdimenziós robotmodellezés*, Budapest, 2011.
- [2] Zátroch Zoltán - Molnár Zsolt: *Robotmodell3d*, <http://www.robotmodell3d.hu> letöltés dátuma: 2012 január
- [3] Ogre: <http://www.ogre3d.org> letöltés dátuma: 2011.október
- [4] Robologix: <http://www.robologix.com> letöltés dátuma: 2012 szeptember.
- [5] Simbad: <http://simbad.sourceforge.net> letöltés dátuma: 2012. január
- [6] Webots: <http://www.cyberbotics.com> letöltés dátuma: 2012. január
- [7] RobotWorks: <http://www.newtonium.com> letöltés dátuma: 2012.január
- [8] Easy-Rob: <http://www.easy-rob.de> letöltés dátuma: 2012.január
- [9] Virca: <http://www.virca.hu> letöltés dátuma: 2011.október.
- [10] János Nacs, Imre Paniti, Sándor Kopácsi: *Incremental Sheet Forming in Cyberspace - a Process Oriented Cognitive Robotics Application*. In: 2011 2nd International Conference on Cognitive Infocommunications, CogInfoCom 2011. Budapest, Magyarország, 2011.07.07-2011.07.09. pp. 1-5.(ISBN: 978-145771806-9)
- [11] Zátroch Zoltán - Molnár Zsolt: *Interactive robot modelling in 3D*, in CogInfoCom 2012 3rd IEEE International Conference Infocommunications, Kosice,Slovakia, December 2-5, 2012., pp. 641.
- [12] Mallász Judit (2012): 3D-s megjelenítés az iparban. Segítség a gyártáshoz, értékesítéshez, bemutatáshoz és oktatáshoz. gyartastrend.hu, július 5. (http://www.gyartastrend.hu/gepipar/cikk/3d_s_megjelenites_az_iparban) (utolsó megtekintés: 2012. november 27.).

11. Ábrajegyzék

1. ábra VirCA kötözető.....	7
2. ábra Kockamegfogó eszköz.....	11
3. ábra Kockatartó állvány.....	12
4. ábra A kocka modellje.....	12
5. ábra Lemez-állapotok.....	13
6. ábra Előlnézeti kép a robotvezérlőről.....	16
7. ábra A robotvezérlő belülről.....	17
8. ábra A SerialCommunicator VirCA komponens.....	22
9. ábra Kódrészlet a csuklókoordináták beállítása.....	25
10. ábra A kocka összeállítása.....	26
11. ábra ISF alkalmazás.....	28
12. ábra A komponensek közötti kapcsolatok.....	29

12. Rövidítés-jegyzék

TDK	Tudományos Diákköri Konferencia
MTA- SZTAKI	Magyar Tudományos Akadémia Számítástechnikai és Automatizálási Kutatóintézet
VirCA	Virtual Collaboration Arena
XML	Extensible Markup Language
3D	Három dimenzió
ISF	Inkrementális lemezalakítás
.kl	Karel program kiterjesztése
ICE	Szolgáltató port