M Ű E G Y E T E M 1 7 8 2

Intelligens robotok és járművek laboratórium

I1 - Robotprogramozás

Készítette:

Drexler Dániel András

Adjunktus

BME IIT

2016

Tartalomjegyzék

Ábrák jegyzéke

1.

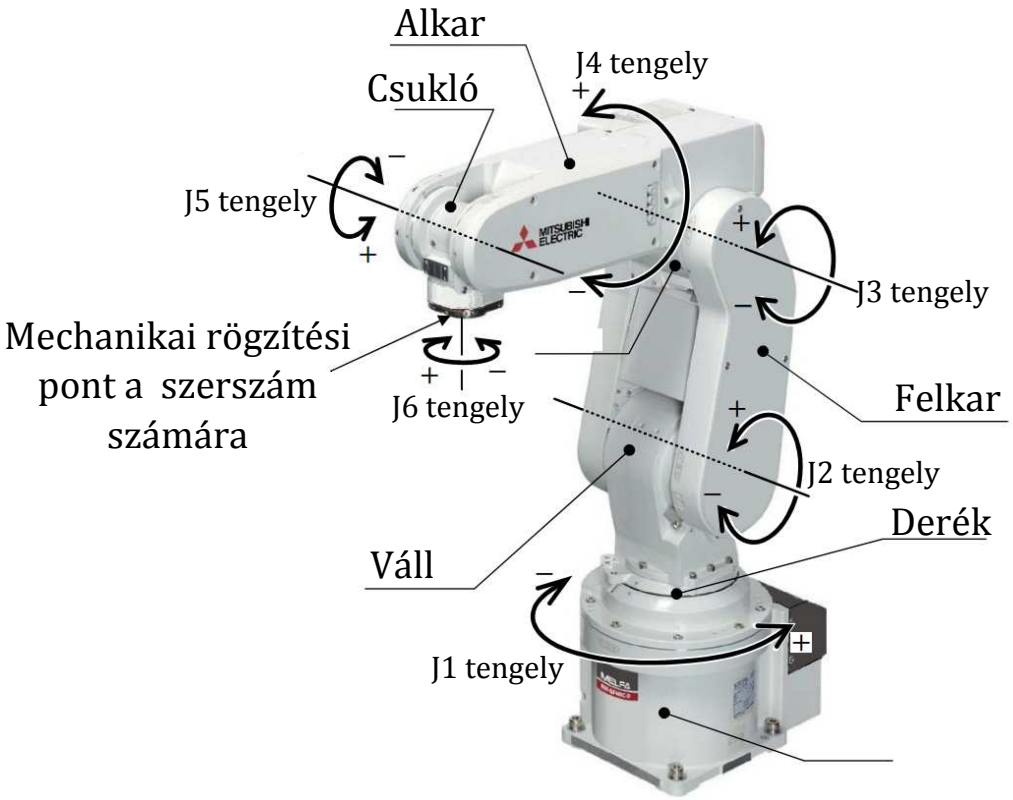
Bevezetés

A modern gyártástechnológiában elterjedten használnak ipari robotokat bizonyos könnyen automatizálható feladatok ellátására. A leggyakrabban előforduló alkalmazás az anyagmozgatás (*pick and place*), azaz bizonyos tárgyakat egyik helyről a másikra teszünk. Az anyagmozgatást bonyolíthatjuk azzal, hogy az objektumokat egy bizonyos speciális geometriájú tárolóból (pl. sörösrekesz) pakolunk ki (depalettázás - *depalettize*), illetve ilyen tárolóba pakolunk be (palettázás - *palettize*). Amennyiben szenzorokat is alkalmazunk a robotkar mellett, akkor lehetővé válik a mozgatott anyagok bizonyos, a szenzorok által mért jellemzők szerinti szétválogatása (*sort*). Ezek a feladatok általában jól reprodukálhatók (az objektumok "ugyanonnan" jönnek, "ugyanoda" mennek), könnyen algoritmizálhatók (egyszerű nyelven lekódolhatók) és monotonok (a feladatok nem változnak). A `https://www.youtube.com/watch?v=IfojHo9cVOk` linken található videó az ipari robotok néhány tipikus alkalmazását mutatja be.

Bár a leggyakoribb alkalmazás az anyagmozgatás, meg kell említeni még a bizonyos szempontból bonyolultabb alkalmazásokat, mint például a hegesztés, festés, esztergálás, stb. A hegesztésre és anyagmozgatási alkalmazásra láthatunk példát a `https://www.youtube.com/watch?v=sjAZGUcjrP8` linken található videón, ami egy autó gyártósor egy részletét mutatja be.

A mérésen egy anyagmozgató, válogató, palettázó feladatot kell megoldani. Egy egyrekeszes tárolóból érkező munkadarabokat (34 mm átmérőjú, 10 mm magas hengereket) anyagfajtájuk (bronz, alumínium, fa) és színük szerint kell szétválogatni, és egy 2×3 méretű tárolórekesz megfelelő helyére elhelyezni.

A feladatot egy Mitsubishi RV-2F-Q ipari robottal (1.1. ábra) kell megoldani [1]. A robotkar vezérlését a CR750-Q robotvezérlő végzi (1.2.



1.1. ábra. Mitsubishi RV-2F-Q ipari robotkar a csuklók és szegmensek szokásos elnevezésével [1]

ábra bal oldal). A robotvezérlő tartalmazza a teljesítményelektronikát, egy egyszerű ember-gép interfészt, ellátja a biztonsági funkciókat, kapcsolódik a betanító pulthoz (1.2. ábra jobb oldal), amivel a robot mozgatható, illetve kapcsolódik a PLC-hez. A PLC egy "hagyományos PLC" (tápegység, CPU modul, digitális ki- és bemeneti modul) kiegészítve egy robotvezérlő modullal, amin a pályatervezési és egyéb robotikai algoritmusok futnak. A mérés során használt szenzorok a PLC digitális bemeneti moduljára csatlakoznak, a szenzorok által mért adatokat a PLC CPU kártyáján futó (már elkészült) program minden ciklusban beolvassa, és egy olyan memóriaterületre helyezi, ami a robotvezérlő modullal meg van osztva, így a mért szenzoradatok a robot programból beolvashatók.

A robot program megírható a betanítópulton, azonban kényelmesebb a fejlesztést személyi számítógépen, a MELSOFT Navigator fejlesztőkörnyezetben elvégezni [2]. A PC Etherneten kommunikál a PLC-vel, így a robotrendszer PC-ről programozható. A MELSOFT Navigator-

1.2. ábra. A CR750-Q robotvezérlő és a betanítópult

ban kell beállítani a hardverkonfigurációt (PLC és a robotvezérlő konfigurálása, már elkészült), majd a robotvezérlő modulra kattintva belépünk az RT ToolBox2 környezetbe [3], ahol a robot programot megírhatjuk (a mérésen ezt a környezetet fogjuk használni). A robot programot MELFA Basic nyelven kell megírni [4], ez egy egyszerű Basic nyelv.

A mérés során elvégzendő feladat tehát pontok betanítása, a program megírása, és végül a program tesztelése. Ezekről bővebben a 2. fejezetben lesz szó. A 3. fejezet tartalmazza a mérési feladat részletes leírását, míg a 4. fejezetben a mérésre való felkészülést segítő kérdések vannak.

2.

Robotprogramozás

A robotok programozásánál alapvető probléma, hogy valahogyan meg kell adni a robot mozgását, illetve a mozgás végső célját. Azt az utasítást, hogy "menj oda", illetve "mozogj arra", valahogy le kell fordítani a robot nyelvére. Az első lépés a koordinátarendszerek bevezetése, amiben pontokat, illetve irányokat adhatunk meg. Bizonyos koordinátarendszerek a robotrendszerben alapértelmezetten definiálva vannak, ezek módosíthatók, de erre a mérésen nem lesz szükség.

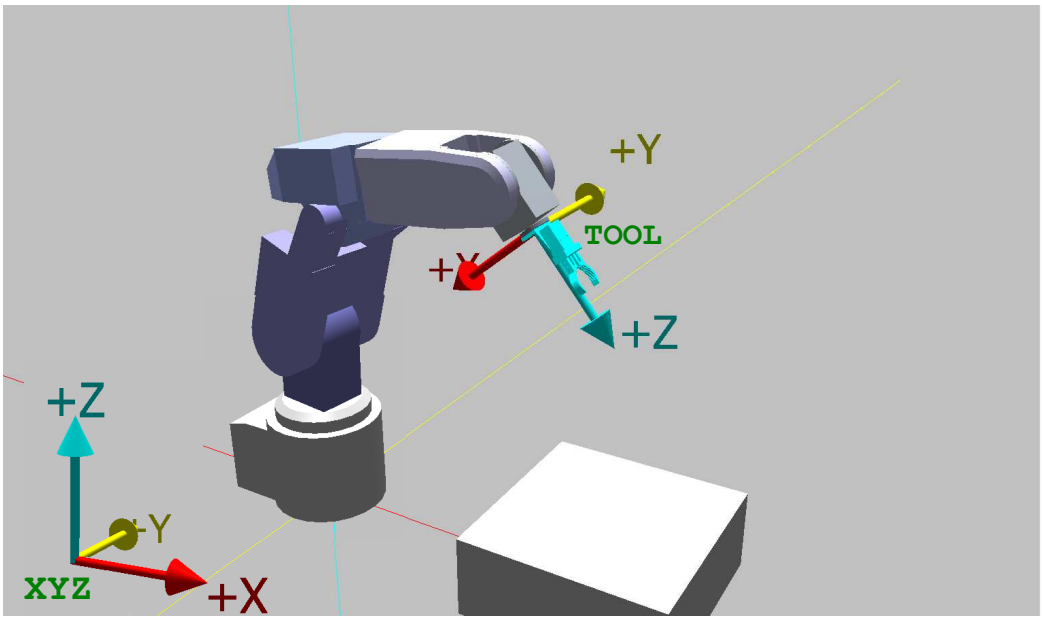
A következő lépés a robot "céljának" megadása, azaz, hogy hová menjen. Ehhez meg kell adni a robot végberendezésének (ami egy pneumatikus megfogó lesz) a pozícióját (hol van?) és orientációját (merre néz, hogy áll?). A leggyakrabban ezt betanítással adják meg (*teaching by showing*), a mérés során is így fogunk eljárni.

A pont, illetve pontok betanítása után megírjuk a robot programot, ami definiálja a robot által elvégzendő feladatot. Végezetül teszteléssel ellenőrizzük, hogy a megírt program helyes-e.

2.1. Koordinátarendszerek

A robot mozgatását különböző koordinátarendszerekben végezhetjük. A legtriviálisabb koordináták az úgynevezett csuklókoordináták, amik a robot csuklóinak a szögei. A robotnak hat darab rotációs csuklója van, tehát a csuklókoordinátáknak hat koordinátája van. A csuklókoordináták egy konkrét értékét szokás csuklókonfigurációnak nevezni (a robotrendszerben a csuklókoordinátákat a *Joint* címke jelzi).

A világ koordinátrendszer (vagy bázis koordinátarendszer, az angol szaknyelvben elterjedt a *spatial coordinate frame* kifejezés is) egy derékszögő koordinátarendszer (2.1. ábrán az *XYZ* koordinátarendszer),



2.1. ábra. A világkoordinátarendszer (*XYZ*) és a szerszám koordinátarendszer (*TOOL*)

ami a robot alapjához van rögzítve, és a csuklókonfigurációktól független, azaz időben nem változik. A világkoordinátarendszer z tengelye általában a gravitációs gyorsulásvektorral ellentétes irányba mutat (ahogy a 2.1. ábrán és a mérésen használt robotnál is), bár ez függ a robot rögzítésének a módjától. A világkoordinátarendszert a robotrendszerben az *XYZ* címke jelöli. Példaként a 2.2. ábrán az *XYZ* keret (a robotikában szokás a koordinátarendszer helyett a keret kifejezést is használni) z tengelye menti elmozdulást láthatunk.

A szerszámkoordinátarendszer vagy test koordinátarendszer a robot utolsó szegmenséhez rögzített derékszögű koordinátarendszer, ami a robottal együtt mozog (2.1. ábrán a *TOOL* koordinátarendszer). Az angol szaknyelvben elterjedt a *tool* és a *body coordinate frame* kifejezés is. A test (*body*) keret általában arra utal, hogy a testtel együtt mozog a koordinátarendszer. A robotrendszerben a test koordinátarendszert a *TOOL* címke jelöli. A *TOOL* koordinátarendszer origója alapértelmezetten a szerszám rögzítési pontjának a közepén van, és a z tengelye a rögzítési felületre merőleges, és a robottól távolodó irányba mutat. A koordinátarendszer origója és a tengelyei is eltranszformálhatók, erre szerszámcsere esetén lehet szükség. A mérésen az alapértelmezett beállítást fogjuk használni. Példaként a 2.3. ábrán a *TOOL* keret z tengelye menti elmozdulást láthatunk.

2.2. Betanítás, robot mozgatás

A robot programozásakor többször meg kell adni, hogy a robot hova menjen. Ezt a későbbiekben pozíciónak, vagy pontnak fogjuk nevezni (itt most a pozíció alatt a szerszámkeret pozíciójának és orientációjának egy konkrét értéket értjük). A robot pozíciója leírható például az *XYZ* keretben, vagy a *Joint* keretben. A két megadás között jelentős különbség van. Amennyiben a pozíciót csuklókoordinátákban adjuk meg (azaz a *Joint* keretben), akkor ez a megadás egyértelmű, pontosan elő van írva, hogy a robot csuklóinak milyen szögben kell állni abban a pozícióban. Amennyiben a robot pozícióját világkoordinátarendszerben (azaz az *XYZ* keretben) adjuk meg, akkor ha a robotot egy megfelelő utasítással a megadott pozícióba küldjük, különféle csuklókonfigurációkkal is elérheti a megadott pozíciót. Ennek az az oka, hogy egy előírt szerszám pozíció és orientáció érték több különböző csuklókonfigurációval elérhető (az antropomorfikus robotkaroknál legfeljebb nyolc különböző megoldás lehet, ennél a konkrét robotnál legfeljebb négy különböző csuklókonfiguráció tartozhat egy rögzített szerszám pozícióhoz és orientációhoz).

A programozási nyelvben a pozíciókat tároló úgynevezett pozícióváltozók kitüntetett változók, és a nevük mindig p betűvel kezdődik. A pozíció változók letárolhatók *Joint* és *XYZ* keretben is, illetve az is beállítható, hogy az előző bekezdésben említett négy megoldás közül melyiket válassza a rendszer. A robot bizonyos csuklói képesek többször is körbefordulni, ezért a pozícióváltozónál az is beállítható, hogy az egyes csuklók hányszor forduljanak körbe.

A pozíciók megadásakor hat koordinátát kell előírnunk. *Joint* keret használata esetén a hat koordináta a robot hat csuklójának a szöge fokban, *XYZ* keret esetén pedig az első három koordináta a szerszámkeret origójának pozíciója az *XYZ* keretben megadva (a rendszer ezeket a koordinátákat rendre X, Y és Z betűkkel jelöli), az utolsó három koordináta pedig a szerszámkeret és a világkeret tengelyei között forgatást írja le három számmal, amik rendre a világkeret x, y, és z tengelyei körüli forgatások fokban (a rendszer ezeket a szögeket rendre A, B és C betűkkel jelöli).

A koordináták kézzel történő megadása sokszor körülményes, ezért ehelyett a robotot a betanító pult segítségével a kívánt pozícióba mozgatjuk, majd ezt a pozíciót kiolvassuk. A pozíció kiolvasását az RT Toolbox támogatja. A kiolvasott pozíciót ezek után pozíció változóként el tudjuk menteni. Ezt a műveletet nevezik betanításnak.

A betanítás egy időigényes feladat, és nagy körültekintést igényel,

például el kell kerülni, hogy a robot ütközzön a környezetében lévő tárgyakkal. A robot programot célszerű úgy elkészíteni, hogy a lehető legkevesebb pozíciót kelljen betanítani. A robot program megírásához nincs szükség a betanított pozíciókra, a programban szereplő pozícióváltozókat a program megírása után is betaníthatjuk. Természetesen a helyes működéshez szükség van az elkészült programra és a pozícióváltozók betanítására is.

A gyakorlatban karbantartás után a robot munkakörnyezete (kis mértékben) megváltozhat, ilyenkor szükséges lehet a pozíciók újratanítására. A programot célszerű úgy megírni, hogy a munkacella kis mértékű változtatása esetén elég legyen a pozíciókat újra tanítani. A karbantartási idő lerövidítése érdekében is célszerű a feladatot olyan programmal megvalósítani, amihez a lehető legkevesebb pozíciót kell betanítani.

A robot mozgatása előtt bizonyos biztonsági lépéseket el kell végezni:

1. A robotvezérlőn a kulcsos kapcsolót *manual* állásba kell állítani.

2. A betanítópult oldalán lévő gombot (*dead man button*) félig be kell nyomni (a pult mindkét oldalán van ilyen gomb, elegendő az egyiket benyomni) és végig nyomva kell tartani. Amennyiben a gombot felengedjük,vagy túl erősen nyomjuk, a rendszer leállítja a robotot.

3. A betanítópulton a *servo* nyomógomb (bal oldal) megnyomásával elindítjuk a robotvezérlő teljesítményelektronikáját és kioldjuk a fékeket, ezek után lehet a robotot mozgatni.

4. A betanítópult *JOG* gombjának megnyomása után előjön az a menü, ahol a robot pozíciójának koordinátáit változtathatjuk. Ezt nevezzük *JOG* üzemmódnak.

JOG üzemmódban a robotot több koordinátarendszerben mozgathatjuk. A koordinátarendszert a betanítópulton kiválaszthatjuk (érintőképernyő tetején), a három gyarkan használt koordinátarendszer a *Joint*, *XYZ* és *TOOL* keretek. *JOG* üzemmódban a betanítópult jobb oldalán egy oszlopban megjelenik hat koordináta, a koordináták mellett pedig + és - feliratú nyomógombok vannak. A nyomógombok segítségével tudjuk a koordináták értékét növelni (+) illetve csökkenteni (-).

Amennyiben a kiválasztott keret a *Joint* keret, akkor a koordináták a robot csuklóinak a szögei, tehát ilyenkor a robotot csuklónként tudjuk mozgatni.

Amennyiben a kiválasztott keret az *XYZ* keret, a koordináták az X,Y,Z,A,B,C értékek, amik a szerszámhoz rendelt keret origójának a pozíciói a világkeretben megadva (X,Y,Z értékek mm-ben), illetve a szerszámhoz rendelt keret orientációja, a világkeret x, y, és z tengelye körüli elforgatásként megadva (A,B,C szögek fokban). Ilyenkor a robot végberendezését egyenes mentén tudjuk elmozdítani a világkeret tengelyei mentén (az első három koordináta változtatásával), illetve a világkeret tengelyei körül tudjuk a szerszámot forgatni (az utolsó három koordináta változtatásával). A 2.2. ábrán egy egyenes menti elmozdulást látunk, a világkeret z tengelye mentén, pozitív irányba.

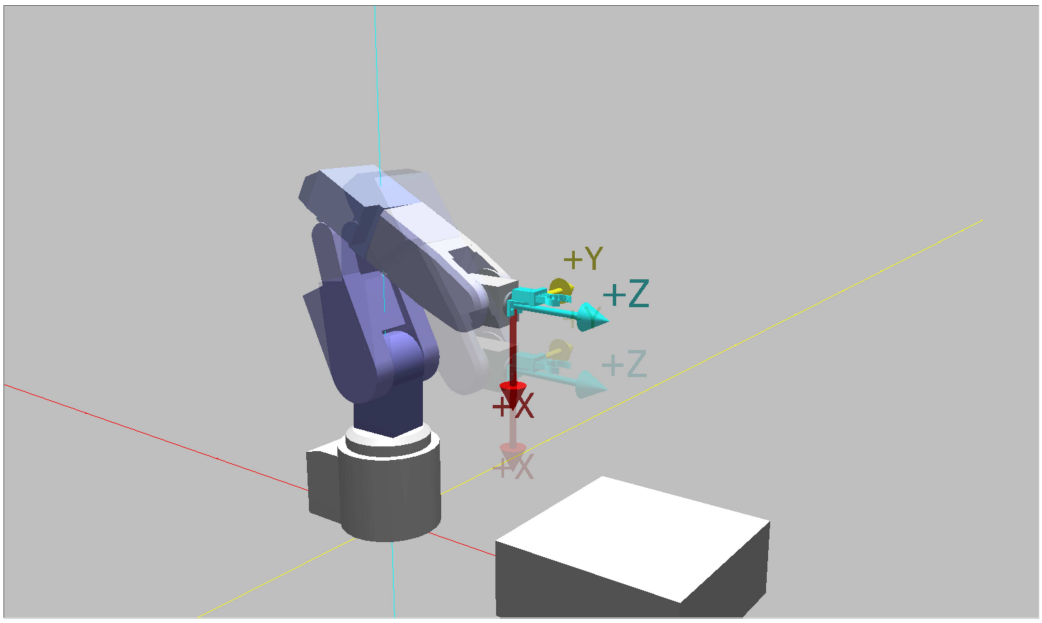
Amennyiben a kiválasztott keret a *TOOL* keret, a koordináták az X,Y,Z,A,B,C értékek, amik a szerszámhoz rendelt keret origójának a pozíciói a világkeretben megadva (X,Y,Z értékek mm-ben), illetve a szerszámhoz rendelt keret orientációja, a világkeret x, y, és z tengelye körüli elforgatásként megadva (A,B,C szögek fokban), vagyis ugyanazok, mint *XYZ* keret esetén (ennek az az oka, hogy a szerszámhoz rendelt koordinátarendszer origójának pozícióját és a koordinátarendszer tengelyeit nem lenne értelme saját magában megadni). A különbség az *XYZ* kerethez képest mégis az, hogy ilyenkor a robot végberendezését szintén egyenes mentén tudjuk elmozdítani, azonban most a *TOOL* keret tengelyei mentén (az első három koordináta változtatásával), illetve a *TOOL* tengelyei körül tudjuk a szerszámot forgatni (az utolsó három koordináta változtatásával). A 2.3. ábrán egy egyenes menti elmozdulást látunk, a *TOOL* keret z tengelye mentén, pozitív irányba.

JOG üzemmódban a robot sebességét állíthatjuk, a betanítópult kijelzőjének a bal alsó sarkában található legördülő menü, illetve a bal alsó sarokban lévő felfelé és lefelé mutató nyilakkal felcímkézett nyomógombok segítségével. Biztonsági okokból a maximálisan beállítható sebesség *JOG* üzemmódban a robot maximális sebességének a 10 százaléka.

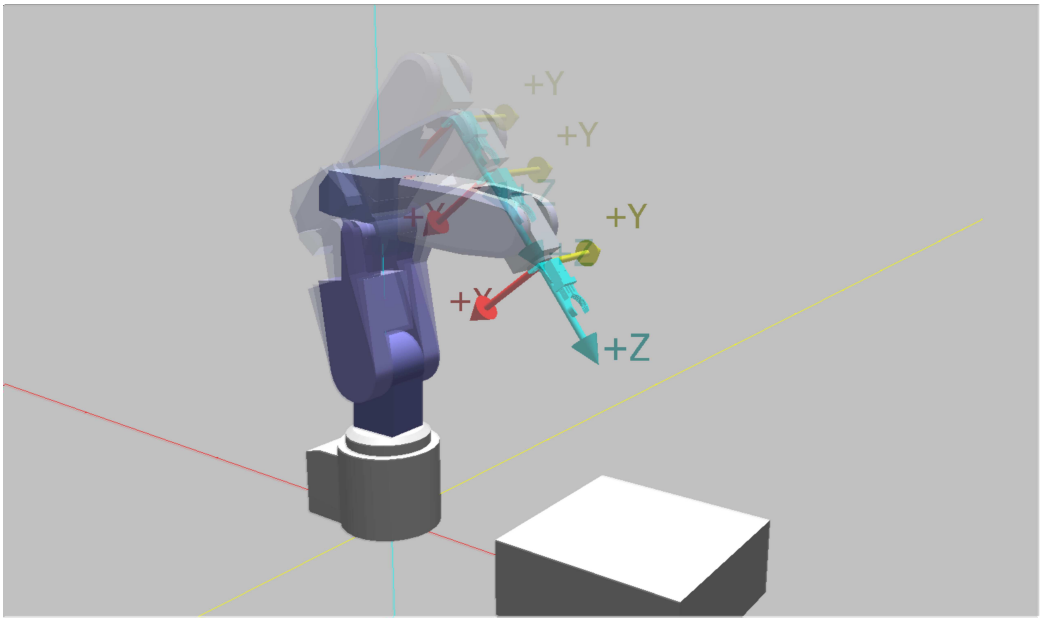
2.3. Programozás MELFA BASIC nyelven

A robot programot MELFA BASIC V nyelven kell megírni. A nyelv egy basic nyelv, a végrehajtás soronként történik, a sorokat címkével el lehet látni (a címkék csillaggal kezdődnek, pl. **cimke*), és a címkével ellátott sorokra a program végrehajtás során át lehet ugrani *goto* utasítás segítségével (pl. *goto *cimke*). A programnyelv nem case sensitive, azaz a kis és nagybetű nincs megkülönböztetve.

A programnyelvben kétfajta változó használható, az egyik a pozí-



2.2. ábra. Elmozdulás az *XYZ* keret z tengelye mentén pozitív irányba



2.3. ábra. Elmozdulás a *TOOL* keret z tengelye mentén pozitív irányba

cióváltozó, ami egy hatdimenziós vektor, a másik pedig a skalárváltozó. A pozícióváltozók neve mindig p betűvel kezdődik, a skalárváltozók neve nem kezdődhet p illetve c betűvel. A változók neve legyen lehetőleg rövid és ne tartalmazzon különleges karatereket (_ karatert sem). A változók típusát deklarálni nem kell, a deklaráció az első értékadáskor (vagy betanításkor) automatikusan történik. A változókkal lehet operálni különböző aritmetikai utasítások (+-*/) segítségével. Pozícióváltozók esetén a +, - és * műveletek vannak értelmezve, a + és - műveletek koordinátánkénti összeadást, illetve kivonást jelölnek, a * művelet pedig relatív koordinátákat (itt nem részletezzük, a mérésen nem használjuk).

A programnyelv hardverhez kötődő alapvető utasítása a *servo* utasítás, amit az on és off paraméterekkel tudunk hívni. A *servo on* utasítás bekapcsolja a robot teljesítményelektronikáját, és kikapcsolja a fékeket (célszerű az utasítást a program elejére írni), míg a *servo off* utasítás kikapcsolja a teljesítményelektronikát, és bekapcsolja a fékeket (a robot bizonyos csuklóiban fékek vannak annak érdekében, hogy a robot ne "essen" össze kikapcsolt állapotban).

Az *ovrd* utasítással beállíthatjuk a robot maximális sebességét. Az utasítás paramétere egy egész szám 0 és 100 között, ami azt mondja meg, hogy a robot legfeljebb az elérhető legnagyobb sebesség hány százalékával mozoghasson. Például az *ovrd 50* sor után következő mozgató utasítások a robotot legfeljebb az elérhető maximális sebesség 50 százalékával mozgatják.

A robot mozgatásához a mérésen a *mov* és *mvs* utasításokat fogjuk használni. A két utasítás között a különbség, hogy a *mov* utasítás az aktuális és a paraméterként előírt pozíció között az interpolációt csuklóváltozókban végzi, míg az *mvs* utasítás Descartes koordinátákban interpolál. A Descartes koordinátákban történő interpoláció alatt azt értjük, hogy a mozgató utasítás a robot megfogóját egy egyenes mentén mozgatja. Ennek a mozgatási módnak a hátránya, hogy a robot komplex munkatere miatt gyakran előfordulhat, hogy az aktuális és az előírt pozíció között nem létezik ilyen pálya, az előnye viszont az, hogy könnyen jósolható, hogy a robot végberendezése milyen pályán fog mozogni. Mivel a csuklókoordináták és a végberendezés pozíciója és orientációja közötti összefüggés nemlineáris, ezért ha az interpoláció (ami a szakasz kezdeti és végső részétől eltekintve lineáris) csuklóváltozókban történik, akkor a végberendezés nem egyenes pályán fog mozogni. Épp ezért a csuklókoordinátákban való mozgás kiszámíthatatlanabb, nehezen látjuk előre, hogy a robot hogyan fog eljutni a kívánt célpozícióba. Ennek a módnak az előnye viszont, hogy csuklókoordinátákban

való interpolációval tetszőleges pozícióból eljuthatunk egy másik pozícióba.

Mindkét utasítás paramétere a célpozíció, ami egy pozícióváltozó. Opcionálisan megadható egy másik paraméter, amivel az eredeti pozíciót eltranszformálhatjuk a *TOOL* keret z tengelye mentén. Tehát, például a *mov p1,50* utasítás azt jelenti, hogy mozogjunk abba a pontba csuklóváltozókban interpolált pályán, amit úgy kapunk, ha a már betanított *p1* pontot eltoljuk a *TOOL* keret z tengelye mentén 50 mm-el. Ennek a paraméternek az az előnye, hogy gyakran szükség lehet arra, hogy egy pontba egy bizonyos irányból jussunk el (a paramétert akkor tudjuk hasznosítani, ha ez az irány a *TOOL* keret z tengelyével egybeesik). Ezt megtehetjünk úgy, hogy a szerszámot a betanított pont közelébe mozgatjuk (pl. *mov p1,-50*), egyenes vonalú pályán elmegyünk a betanított pontba (pl. *mvs p1*), elvégezzük, amit akartunk, majd visszamozgunk egyenes mentén (pl. *mvs p1,-50*).

A mozgatóutasításoknak pozícióváltozóként egy kifejezést is meglehet adni, pl. a *p1+(0,0,100,0,0,0)* kifejezés egy olyan pont, aminek a z koordinátája a világkoordinátarendszerben 100 mm-el több, mint a betanított *p1* pontnak, azaz a *mov p1+(0,0,100,0,0,0)* utasítás hatására a robot a *p1* pont fölé megy 100 mm-el (feltéve, hogy a világkoordinátarendszer z tengelye a gravitációval ellentétes irányba mutat, a mérésen használt elrendezésnél ez teljesül). Mivel értékadás nem történik, ezért a *p1* pont értéke ettől az utasítástól nem változik.

A robotvezérlő a mozgásokat összeköti, azaz ha elérünk egy pontba, a robot azonnal indul a következő pontba. Ez nem mindig előnyös, például nincs elegendő idő a tárgy megfogásához, a mérés elvégzéséhez, vagy a tárgy elengedéséhez (az ejtésből könnyen hajítás lehet). Ezeknek a problémáknak a megoldására szolgál a *dly* utasítás, ami arra kényszeríti a robotot, hogy a paraméterként megadott ideig várakozzon, és csak az idő letelte után hajtsa végre a következő utasítást. Például a *dly 0.5* utasítás hatására a robot fél másodpercet várakozik mielőtt végrehajtaná a következő utasítást.

A robotra szerelt megfogó pneumatikus, amit a *hopen* és *hclose* utasításokkal tudunk nyitni illetve zárni. A robotra négy pneumatikus csatorna van szerelve, ezért az utasításoknak paraméterként a csatorna számát is meg kell adni. A mérésen használt megfogót az 1-es pneumatikus csatornán tudjuk vezérelni, tehát a *hopen 1* utasítással a megfogót kinyitjuk, a *hclose 1* utasítással pedig bezárjuk.

A szenzorok által mért adatokat az *m_in()* utasítás segítségével tudjuk beolvasni. Az utasítás bemeneti változója az a memóriacím, ahonnan olvasni akarunk. Az érzékelők kimenete bináris. A mérésen há-

rom szenzort használunk:

- Az első szenzor egy kapacitív közelségérzékelő, aminek a kimenete 1, ha valamit érzékel, és 0, ha nem. A mért értéket a PLC-ben futó (már elkészült) program minden PLC ciklusban a robotvezérlő 10160 című memóriaterületére másolja, tehát az *m_in(10160)* utasítás beolvassa a kapacitív közelségérzékelő által mért értéket.
- A második szenzor egy induktív közelségérzékelő, aminek a kimenete 0, ha fémet érzékel, különben 1. A mért értéket a robotvezérlő 10161 címen érhetjük el, tehát az *m_in(10161)* utasítással tudjuk beolvasni, hogy a szenzor mit mér.
- A harmadik szenzor egy színérzékelő, aminek három bináris kimenete van. Az egyes kimenetek akkor veszik fel az 1 értéket, ha a szenzor közelébe egy előre megadott színű obektum érkezik. A kimenetekhez rendelt színek a szenzoron programozhatók. A szenzor kimenetein megjelenő értékek a 10162,10163,10164 című memóriaterületekre érkeznek.

A programban megadhatók logikai kifejezések és elágazó utasítások is. A feltételes elágazást az *if,then,else* utasítások segítségével tudjuk megvalósítani. Az *if* utasítás feltételrésze csak egy logikai kifejezést tartalmazhat. A *then* utasításnak egy sorban kell szerepelnie az *if* utasítással. Az *else* ág elhagyható. A feltételes elágazás háromféleképpen használható:

- *if* <feltétel> *then* < egy utasítás> .
- *if* <feltétel> *then* < egy utasítás> *else* <egy utasítás>.
- *if* <feltétel> *then*
  <több sornyi utasítás>
  else
  <több sornyi utasítás>
  endif.

A következő egyszerű példaprogramban a robot megközelít egy tárolót (*p1* betanított pont) *TOOL* z irányból, megfog egy objektumot, elviszi egy szenzor fölé (*p2* betanított pont), a szenzorhoz viszi, beolvassa a mért értéket, és a mért értéktől függően a *p3* vagy *p4* betanított pontnál lévő rekeszbe teszi az objektumot. Miután a feladatot elvégezte, kikapcsolja a hajtásokat és bekapcsolja a fékeket. A ' karakter utáni részek a nyelvben kommentnek számítanak.

```
servo on ’hajtás bekapcsolás
ovrd 50 ’maximális sebesség 50%-ára korlátozunk
hopen 1 ’megfogyó nyitása (hátha előzőleg zárva maradt)
mov p1,-50 ’a tároló megközelítése
mvs p1 ’a megfogó benyúl a tárolóba
hclose 1 ’megfogja az objektumot
dly 0.3 ’várakozik, hogy legyen idő megfogni
mvs p1,-50 ’kinyúl a tárolóból
mov p2+(0,0,50,0,0,0) ’a szenzor fölé megy 50 mm-el
mvs p2 ’a szenzorhoz megy
dly 0.3 ’vár, hogy legyen ideje a szenzornak mérni
mert = m_in(10160) ’beolvassa a mért értéket
’a 10160-as memóriacímről
dly 0.1 ’vár, hogy ne akkor mozogjon el amikor még mér
mvs p2,(0,0,50,0,0,0) ’a szenzor fölé megy 50 mm-el
’ha a mérés egyes, akkor a p4-be tesszük az objektumot
if mert=1 then goto *negyes
’ha a mérés 0, akkor a p3-ba tesszük az objektumot
mov p3+(0,0,100,0,0,0) ’a p3 tároló fölé megy
mvs p3 ’a p3 tárolóhoz mozog
hopen 1 ’elengedi az objektumot
dly 0.2 ’vár, hogy az elengedésből ne hajítás legyen
mov p3+(0,0,100,0,0,0) ’a p3 fölé megy
goto *vege ’a program végére ugrik
*negyes ’a p4 tárolóba helyezés
mov p4+(0,0,100,0,0,0) ’a p4 fölé megy
mvs p4 ’a p4 rekeszhez megy
hopen 1 ’elengedi az objektumot
dly 0.2 ’vár
mov p4+(0,0,100,0,0,0) ’a p4 rekesz fölé mozog
*vege ’a program vége
servo off ’hajtás kikapcsolása
end ’befejezi a programot
```

3.

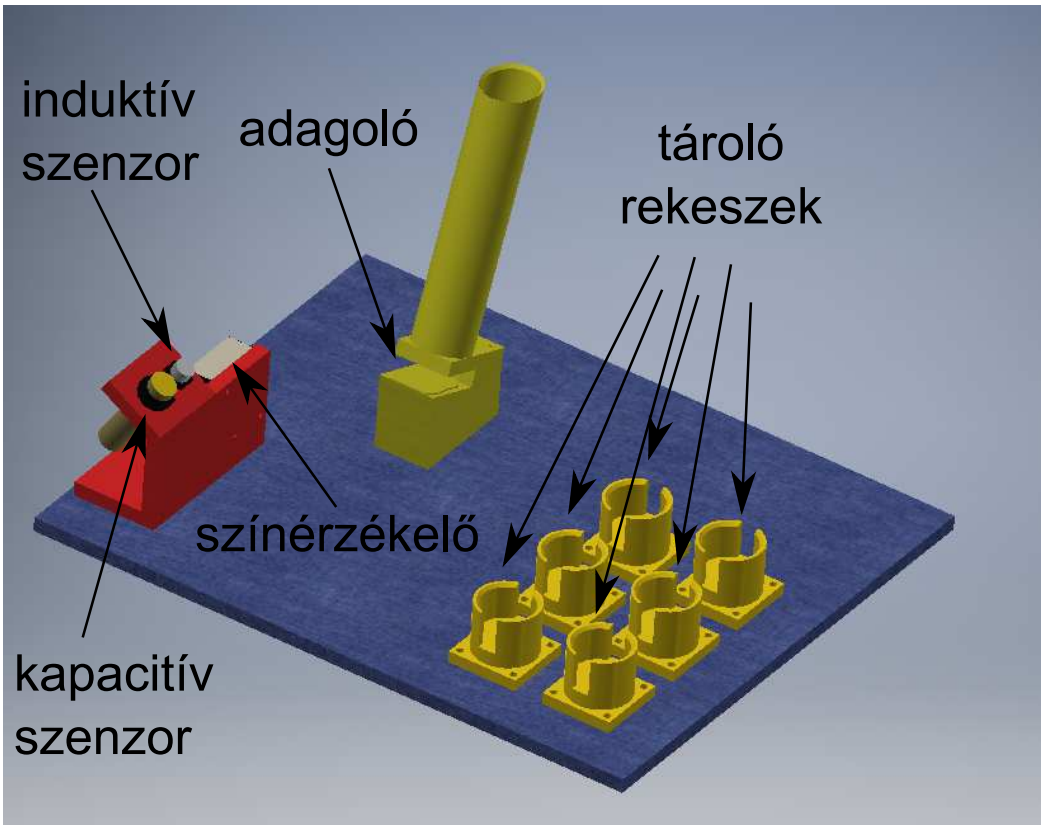
A mérési feladat

A mérés során 12 darab, 10 mm magas, 34 mm átmérőjű hengert kell egy adagolóból kiszedni, a szenzorokkal beazonosítani, majd a megfelelő rekeszbe helyezni. A 12 darab henger közül 4 darab fából, 4 darab alumíniumból és 4 darab bronzból készült. A mérési elrendezés háromdimenziós modellje a 3.1. ábrán, míg a felülnézeti vázlata a 3.2. ábrán látható.

Az első lépés az, hogy **az objektumot megfogjuk**. Az objektumok az adagolóból érkeznek (3.1. ábra), az adagoló nyílása 20°-os szöget zár be a talajjal, ezért a TOOL z irányból kell megközelíteni (a pontot úgy kell betanítani, hogy a TOOL z irány irányvektora is 20°-ot zárjon be a talajjal). Az adagolóból kivett objektumot először a kapacitív közelségérzékelővel kell megmérni. Amennyiben a közelségérzékelő nem érzékel semmit, a megfogási művelet nem sikerült, próbáljuk újra (kivéve, ha már az összes objektumot a helyére tettük, ilyenkor a program álljon le).

Amennyiben a megfogás sikeres, tehát a kapacitív szenzor jelez, akkor a következő lépés, hogy **beazonosítjuk a henger anyagát**. Megmérjük az induktív szenzorral, ami alapján eldönthetjük, hogy az objektum fém vagy fa. A fémből készült hengerekről a színérzékelő segítségével döntjük el, hogy alumíniumból, vagy bronzból vannak-e.

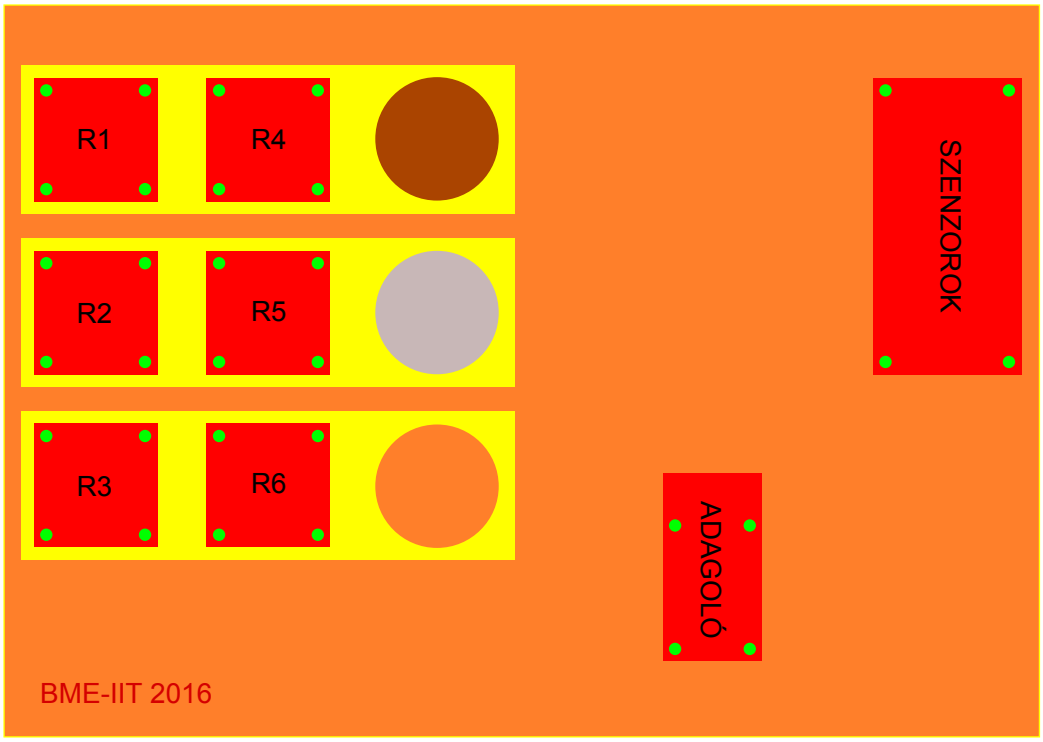
Miután beazonosítottuk a henger anyagát, **elhelyezzük a megfelelő tárolóba**. A fa hengereket az R1 és R4 címkéjű rekeszekbe, az alumínium hengereket az R2 és R5 címkéjű rekeszekbe, a bronz hengereket pedig az R3 és R6 címkéjű rekeszekbe helyezzük (3.2. ábra). Mindegyik rekeszbe kettő henger fér, és anyagfajtánként négy darab hengert kell eltárolni. Miután mind **a 12 hengert eltettük, a művelet véget ért**, a hajtást kikapcsoljuk, és a program leáll. **Amennyiben nem fogyott még el** az összes hengert, **a műveletsort újra kezdjük**



3.1. ábra. A mérési elrendezés

a következő objektum megfogásával.

A feladat megvalósításához a robotnak pontokat kell betanítani, és meg kell írni a megfelelő programot MELFA BASIC V nyelven. A pontok betanítása és a program megírása, illetve tesztelése a mérésvezető felügyelete mellett történik.



3.2. ábra. A mérési elrendezés vázlata felülnézetből

4.

Ellenörző kérdések

1. Mi a különbség a *mov* és az *mvs* mozgató utasítások között?
2. Mit nevezünk pozícióváltozónak? Milyen betűvel kezdődik a pozícióváltozó neve MELFA BASIC nyelvben?
3. Mire lehet használni a címkéket MELFA BASIC nyelvben? Milyen karakterrel kezdődnek a címkék?
4. Hogyan lehet elkerülni, hogy a robot a betanított pontba érkezés után egyből tovább induljon?
5. Miért van szükség pontok betanítására?
6. Mi a különbség a *Joint* és az *XYZ* keret között?
7. Mi a különbség az *XYZ* és a *TOOL* keret között?

Irodalomjegyzék

[1] „Mitsubishi industrial robot, cr750-q/cr751-q controller, rv-2f-q series standard specifications manual.” http://dl.mitsubishielectric.com/dl/fa/document/manual/robot/bfp-a8902/bfp-a8902v.pdf, 2016.

[2] „Melsoft iq works, beginner’s manual.” http://dl.mitsubishielectric.com/dl/fa/document/manual/melsoft/sh080902eng/sh080902engi.pdf, 2016.

[3] „Rt toolbox2 / rt toolbox2 mini user’s manual.” http://meltrade.hu/letoltes/rt-toolbox2-users-manual-bfp-a8618-t-01-14-pdf/, 2016.

[4] „Mitsubishi industrial robot - cr750/cr751 series controller instruction manual - detailed explanations of functions and operations.” http://www.allied-automation.com/wp-content/uploads/2015/02/MITSUBISHI_CR750CR751-Controller-Instruction-Manual-Detailed-Explanations-of-Functions-and-Operations.pdf, 2016.