KUKA-ROBOTIN OHJELMOINTI

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Riihimäki, Kone- ja tuotantotekniikka

Kevät, 2017

Marcos Herrero

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

TIIVISTELMÄ

Koulutus
Kampus

Tekijä	Marcos Herrero	**Vuosi** 2017
Työn nimi	KUKA-robotin ohjelmointi	
Työn ohjaaja	Raimo Ponkkonen	

TIIVISTELMÄ

Opinnäytetyön tavoitteena oli perehtyä KUKA System Softwaren (KSS) ja KUKA Robot Language -kielen (KRL) käyttöön ohjelmoitaessa KUKA-nivelvarsirobottia. Työn tilaajana oli Hämeen ammattikorkeakoulun Riihimäen yksikön konetekniikan koulutusohjelma.

Hämeen teknillisen ammattikorkeakoulun Riihimäen yksikön konetekniikan laboratorion KUKA KR 140 L100 -nivelvarsirobotti on opetuskäytössä mm. robotiikan kursseilla. Robotin ohjelmointi on tapahtunut pääosin online-ohjelmointina, paikoituspisteet opettamalla. Robotin ohjelmoinnissa on hyödynnetty KUKA System Softwaren käyttöliittymän valmiita käskyvalikoita.

Opinnäytetyössä on tutkittu KRL:n käyttöä robotin liikkeiden ohjelmoinnissa ja käytettäessä erilaisia työkaluja. Tilaajan toiveiden mukaisesti opinnäytetyössä tarkasteltiin erityisesti taulukkomuotoisten muuttujien käyttöä, aliohjelmarakenteita sekä robotin ulkopuolisen työkalun käyttöä, kun robotilla liikutetaan työkappaletta.

Opinnäytetyötä ja sen aikana toteutettuja case-esimerkkiohjelmia voidaan käyttää jatkossa konetekniikan koulutusohjelman robotiikan opetuksessa opiskelijoiden tuki- ja kurssimateriaalina.

Avainsanat KUKA, robotiikka, ohjelmointi, KRL

Sivut 79 sivua, joista liitteitä 15 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Name of degree programme
Campus

Author Marcos Herrero **Year** 2017

Subject KUKA robot programming

Supervisor Raimo Ponkkonen

ABSTRACT

Aim of the thesis has been to familiarize oneself with and study the KUKA System Software (KSS) and KUKA Robot Language (KRL) when programming the KUKA joint axis robot. This thesis has been commissioned by Mechanical Engineering program of the Häme University of Applied Sciences.

The KUKA KR 140 L100 joint axis robot in the Machine Lab of the Mechanical Engineering program of the Häme University of Applied sciences is used in teaching and training on Robotics courses. Programming of the robot is implemented mainly by teaching the coordinates, using the command menus found in KUKA System Software's UI.

Use of KRL in programming the robot movement and interaction with various tools has been studied in this thesis. By request of the commissioner, special interest has been given in studying the use of array variables, sub program structures and fixed tools situated outside the robot.

The thesis and the case example programs can be used as reference and study material on future Robotics courses.

Keywords KUKA, robotics, programming, KRL

Pages 79 pages including appendices 15 pages

SISÄLLYS

Liitteet

1 LYHENTEET JA SELITYKSET

CP (Continous Path)
Liike, jonka kulku on joko suoraviivaista ja kaarevaa ja joka kulkee ennalta määrättyä reittiä pitkin

DSL (Domain Specific Language)
Täsmäkieli, joka on tarkoitettu jollekin tietylle tai erityiselle sovellusalueelle

KCP (KUKA Control Panel)
KUKA-robotin ohjaamiseen ja ohjelmointiin käytettävä ohjain

KRC (KUKA Robot Controller)
KUKA-robotin robotinohjain

KRL (KUKA Robot Language)
KUKA-robotin ohjelmointiin tarkoitettu omisteinen täsmäkieli

KSS (KUKA System Software)
KUKA Robot Controllerin järjestelmäohjelma

KUKA.HMI (KUKA Human-Machine-Interface)
KUKA-robotin ohjaamisen käytettävän KCP:n käyttöliittymä

Omisteinen kieli
Ohjelmointikieli, jonka käyttöoikeuslisenssi sallii ohjelmiston käyttämisen tietyin ehdoin, mutta jota ei ilman erillistä lupaa saa muokata tai levittää edelleen. Se perustuu suljettuun lähdekoodiin, joka käsitetään ohjelmistoyrityksen liikesalaisuudeksi

2 JOHDANTO

Opinnäytetyö pyrkii esittelemään KUKA Robot Language -kielen (version 5.3.) käyttöä KUKA-nivelvarsirobotin ohjelmoinnissa. Kielen tärkeimpien käskyjen syntaksi ja toiminta on pyritty käsittelemään yleisesti ja esimerkkien kautta.

Osana opinnäytetyötä on toteutettu hieman laajempia, erilaisia käskyjä ja ohjelmarakenteita yhdisteleviä robottiohjelmia. Toteutetut ohjelmat on esitelty case-tyyppisesti luvussa 14. Ohjelmien kokonaiset ohjelmalistaukset löytyvät opinnäytetyön liitteistä.

3 KUKA YLEISESTI

3.1 KUKA-robotin esittely

KUKA KR 140 L 100 on Kuka Roboter GmbH:n valmistama nivelvarsirobotti, joka peruskokoonpanossaan koostuu kuusiakselisesta nivelvarsirobotista, robottiohjaimesta (Kuka Robot Controller, KRC) kytkentäkaapeleineen sekä robottiohjaimeen kytketystä hallintalaitteesta (Kuka Control Panel, KCP). Robotin työkalupäähän on asennettu erillinen sovitin, johon erilaiset robotin liikuttamat työkalut voidaan kiinnittää.

Robottiohjaimen ohjelmiston versionumero on 5.3. ja sen julkaisuvuosi on 2006. Opinnäytetyö on tehty robottiohjaimen käyttämän ohjelmistoversion (KUKA System Software, KSS) tukemaa Kuka Robot Language (KRL) -versiota käyttäen, joten sen soveltuminen uudempaa ohjelmistoversiota käyttäviin robottikokoonpanoihin on syytä tarkistaa.

4 KRL – SYNTAKSI JA RAKENNE

4.1 Robottiohjelmoinnista

Robottien ohjauksen ohjelmointi voidaan jakaa kahteen lähestymistapaan: paikan päällä tehtävään ja etänä tehtävään. Paikan päällä tehtävä eli ns. online-ohjelmointi toteutetaan joko automaattisesti eli opettamalla tai käsin koodaten. Online-ohjelmoinnin aikana robotilla ei voida suorittaa muita ohjelmia vaan se joudutaan pitämään pois muusta käytöstä. Offline-ohjelmointi sen sijaan voidaan suorittaa tietokoneella, joka ei ole liitettynä robottiin. Ohjelmaa voidaan testata simuloidussa ympäristössä ja siirtää vasta sen jälkeen robotin järjestelmään.

Useimmilla robotinvalmistajilla on omiin robottijärjestelmiinsä tarkoitetut täsmäkielet eli DSL:t (Mühe;Angerer;Hoffmann;& Reif, 2017). KUKA Robot Language eli KRL on Pascalia muistuttava omisteinen kieli, joka on tarkoitettu KUKA robotin ohjaamiseen (Braumann & Brell-Cokcan, 2017).

4.2 KRL:n tiedostorakenne

4.2.1 .SRC ja .DAT

Perusmuotoinen ohjelma koostuu kahdesta erillisestä tiedostosta. Ne ovat nimiosaltaan samanlaisia, mutta poikkeavat tiedostopäätteeltään. SRC-loppuinen tiedosto sisältää kaikki ohjelman käskylauseet. Myös muuttujien tiedot voivat sijaita SRC-tiedostossa. DAT-tiedosto eli Data List voi pitää sisällään ainoastaan muuttujien esittelyt sekä niiden arvojen asetukset.

Sekä SRC että DAT tiedosto alkavat aina sanalla DEF ja päättyvät sanaan END.

4.2.2 Esittelyt ja lauseet

Ennen kuin muuttujalle voidaan asettaa arvo, se tulee esitellä ohjelmakoodin alussa. KRL:ssä kaikki esittelyt tulee tehdä ennen muita lauseita ns. esittelyosiossa. Esittelyosiossa ei muuttujille voi vielä asettaa arvoja vaan se tulee tehdä lauseosiossa. Ainoastaan Data List -tiedostossa muuttujalle voidaan asettaa arvo jo esittelyn yhteydessä. Käytettäessä KRL:n ennalta määriteltyjä tietotyyppejä, voidaan DECL-sana jättää pois muuttujan esittelystä.

```
DECL INT A          ;muuttuja A on tyyppiä INT
DECL REAL B         ;muuttuja B on tyyppiä REAL
INT C               ;muuttuja C on tyyppiä INT

;--- muuttujien esittelyt ennen lauseita ---
INI

A=7                 ;muuttuja A saa arvon 7
B=5.12              ;muuttuja B saa arvon 5.12
C=4                 ;muuttuja C saa arvon 4
```

4.2.3 FOLD

Halutut kohdat ohjelmakoodissa voidaan piilottaa näkyvistä. Piilotetut kohdat eivät näy peruskäyttäjän näkymässä, mutta Expert-tason käyttäjä voi halutessaan palauttaa piilotetut kohdat näkyviin. Muista käskyistä poiketen FOLD aloitetaan ja lopetetaan käskyä edeltävän puolipisteen kanssa.

```
;FOLD ALUSTUS
.. ;piilotettu osa
.. ;piilotettu osa
;ENDFOLD
```

Kun osio on piilotettu, jää näkyviin ainoastaan teksti ”ALUSTUS”.

4.3 **Tietotyypit**

Maarit Harsun kirjassaan käyttämää jaottelua käyttäen, tietotyypit voidaan KRL:ssä jakaa kahteen ryhmään: (2005) Primitiivisiin tietotyyppeihin sekä niistä johdettuihin rakenteellisiin tietotyyppeihin.
Primitiivisiä eli skalaarisia tietotyyppejä ovat mm. kokonais- ja reaalilukutyypit. Rakenteellisia tietotyyppejä ovat esim. taulukkotyypit.

4.3.1 Primitiiviset tietotyypit

KRL:n primitiiviset tietotyypit (Taulukko 1) ovat kokonaislukutyyppi, reaalilukutyyppi, looginen tietotyyppi sekä yksittäisestä ASCII-merkistä koostuva tietotyyppi.

Tietotyyppi	**Kokonais-luku**	**Reaaliluku**	**Looginen**	**Merkki**
Avainsana	INT	REAL	BOOL	CHAR
Arvo	-2^{11} … $2^{31}-1$	±1.1E-38 … ±3.4E+38	TRUE tai FALSE	ASCII merkki

Taulukko 1.

4.3.2 Rakenteelliset tietotyypit

Taulukkomuotoinen tietotyyppi voi pitää sisällään usean primitiivisen tietotyypin arvot. Tietotyyppien tulee kuitenkin olla keskenään samanlaiset.

```
DECL INT KAPPALEET[3] ;kokonaislukutaulukko

KAPPALEET[1]=4 ;taulukon ensimmäinen solu
KAPPALEET[2]=5 ;taulukon toinen solu
KAPPALEET[3]=2 ;taulukon kolmas solu
```

Huom! Monista muista ohjelmointikielistä poiketen KRL vaatii, että taulukon solujen numerointi alkaa numerosta 1 eikä 0.

Taulukot voivat olla myös 2- tai 3-ulotteisia

```
INT RIVI
INT SARAKE
INT KERTOTAULU[3,10]

FOR SARAKE=1 TO 10
   FOR RIVI=1 TO 3
      KERTOTAULU[SARAKE,RIVI]=SARAKE*RIVI
   ENDFOR
ENDFOR
```

Taulukkomuuttuja KERTOTAULU pitää sisällään 30 solua. Solujen arvot näkyvät alla (Taulukko 2).

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30

Taulukko 2.

Kolmiulotteinen taulukko luodaan kuten kaksiulotteinen taulukko:

```
INT TASO
INT RIVI
INT SARAKE
INT AJOSUORITUKSET[2,3,5]

FOR TASO=1 TO 2
   FOR SARAKE=1 TO 5
      FOR RIVI=1 TO 3
         AJOSUORITUKSET[TASO,SARAKE,RIVI]=TRUE
      ENDFOR
   ENDFOR
ENDFOR
```

TASO 1				
true	true	true	true	true
true	true	true	true	true
true	true	true	true	true

TASO 2				
true	true	true	true	true
true	true	true	true	true
true	true	true	true	true

Joistain muista ohjelmointikielistä poiketen, merkkijonoja ei KRL:ssä tallenneta erilliseen STRING-tyyppiseen muuttujaan. Merkkijono tallennetaan CHAR-tyyppiseen yksiulotteiseen taulukkoon.

```
CHAR NIMI[3]
CHAR TOINEN_NIMI[6]
CHAR KOLMAS_NIMI[7]

NIMI[1]="T"  ;yksi merkki kerrallaan
NIMI[2]="I"
NIMI[3]="M"

TOINEN_NIMI[4]="W"

KOLMAS_NIMI[]="LIISA" ;merkkijono kerralla
```

Taulukon NIMI sisältö:

T	I	M

Taulukon TOINEN_NIMI sisältö:

			W		

Taulukon KOLMAS_NIMI sisältö:

L	I	I	S	A		

STRUC

Taulukkomuotoinen tietotyyppi vaatii, että taulukon solujen arvot tulee olla keskenään samaa tietotyyppiä. INT-tyyppinen taulukko voi siten sisältää vain INT-tyyppisiä arvoja.

KRL mahdollistaa erilaisia tietotyyppejä sisältävät taulukot, kun käytetään määritystä STRUC-tietotyypin esittelyssä.

```
STRUC POS REAL X,Y,Z,A,B,C INT S,T
;-- määritellään POS-niminen tietotyyppi -
;-- tietotyyppi koostuu kuudesta
;-- reaalilukutyyppisestä tietueesta
;-- ja kahdesta kokonaislukutyyppisestä
;-- tietueesta

STRUC OMATYPE REAL X, INT Y, BOOL B

DECL POS SIJAINTI
; esitellään SIJAINTI-niminen muuttuja
; joka on tietotyypin POS mukainen

DECL OMATYPE TIETOKASSI
;esitellään TIETOKASSI -niminen muuttuja
; joka on tietotyypin OMATYPE mukainen

;asetetaan arvot muuttujalle SIJAINTI

SIJAINTI.X=34.4
SIJAINTI.Y=-23.2
SIJAINTI.Z=100.0
SIJAINTI.A=90
SIJAINTI.B=29.5
SIJAINTI.C=3.5
SIJAINTI.S=2
SIJAINTI.T=6

;arvot voidaan asettaa myös näin:
TIETOKASSI={X 34.4,Y=20,B=TRUE}
```

KRL:ssä voidaan määritellä myös lueteltuja tietotyyppejä. Luetellut tietotyypit voivat saada vain jonkin ennalta määritellyn arvon.

```
ENUM VALIKKO A1, B2, C3
;määritellään lueteltu tietotyyppi VALIKKO
;tietotyypin mahdolliset arvot A1, B2 ja C3

DECL VALIKKO VALINTA
;esitellään VALINTA niminen muuttuja
;joka on tietotyypin VALIKKO mukainen

VALINTA="B2"
;asetetaan VALINTA-muuttujan arvoksi B2
```

4.3.3 Järjestelmään määritellyt rakenteiset tietotyypit

Järjestelmään on valmiiksi määritelty rakenteisia tietotyyppejä. Näitä tyyppejä ei määrittää STRUC-käskyllä eikä niitä esitellessä tarvitse tietotyypin edessä sanaa DECL.

STRUC AXIS
AXIS-tietotyypin avulla voidaan määritellä piste nivelkoordinaatiston avulla. Tietotyyppi sisältää REAL-tyypit A1, A2, A3, A4, A5 ja A6. Jokainen REAL-tyypeistä vastaa yhtä nivelrobotin niveltä ja arvo ilmaisee nivelen astekääntymän absoluuttisena arvona.

```
..
AXIS OMAPISTE
OMAPISTE={A1 10,A2 0,A3 30,A4 0,A5 10,A6 0}
..
```

STRUC FRAME
FRAME-tietotyypin avulla voidaan määritellä piste suorakulmaisen koordinaatiston avulla. Tietotyyppi sisältää REAL-tyypit X, Y, Z, A, B ja C. Näistä kolme ensimmäistä; X, Y ja Z; ilmaisevat koordinaatteja käytettävän työkoordinaatiston origon suhteen. Kolme jälkimmäistä; A, B ja C; ilmaisevat astemuutosta käytettävän työkoordinaatiston suhteen. ABC kuvattu tarkemmin luvussa 5.2.

```
..
FRAME UUSIPISTE
UUSIPISTE={X 150,Y 250,Z 0,A 0,B 0,C 0}
..
```

STRUC POS

POS-tietotyypin avulla voidaan määritellä piste suorakulmaisen koordinaatiston avulla. Tietotyyppi sisältää samat REAL-tyypit kuin FRAME, mutta niiden lisäksi se sisältää myös INT-tyypit S ja T. INT-tyypit S ja T ovat kokonaislukumuodossa annettavia binääriarvoja. Niiden määräytyminen on tarkemmin selitetty KUKA-järjestelmämanuaalin Expert Programming –osassa (KUKA Roboter GmbH, 2006).

```
..
POS P1,P2
P1={X 540,Y 630,Z 1500,A 0,B 90,C 0,S 2,T 35}
P2={X 40,Y 30,Z 15,A 0,B 0,C 0,S 'B010',T 35}
..
```

S ja T voidaan antaa myös binäärimuodossa kirjoittamalla arvot yksinkertaisten lainausmerkkien sisään ja lisäämällä alkuun B-kirjain.

S(=STATE) ja T(=TURN) käytetään POS-tietotyypissä ilmaisemaan yksiselitteistä asentoa ja asemaa tilanteessa, jossa PTP-liikekäskyn yhteydessä annettu paikkakoordinaatti on mahdollista saavuttaa useassa eri asennossa. On suositeltavaa, että robotin ensimmäinen liikekäsky (useimmiten BCO) annettaisiin POS-tyypin koordinaattien avulla. Mikäli myöhemmissä PTP-käskyissä ei paikkakoordinaateissa annetta toisia STATE ja TURN arvoja, käyttää robotti alkuperäistä STATE-arvoa ja valitsee TURN-arvoista sopivimman.

Muita järjestelmään määriteltyjä rakenteellisia tietotyyppejä ovat E6AXIS ja E6POS. Ne vastaavat AXIS- ja POS-tietotyyppejä, mutta mahdollistavat vielä 6 lisäakselin arvojen määrityksen (E1, E2, E3, E4, E5, E6).

4.4 Järjestelmämuuttujat

Järjestelmä sisältää valmiiksi luotuja Data List -tiedostoja, joissa on määriteltynä useita järjestelmämuuttujia. Järjestelmään luotuja Data List -tiedostoja ovat mm. $MACHINE.DAT, $CUSTOM.DAT ja $CONFIG.DAT

Järjestelmän Data List -tiedostoissa määritellyt muuttujat ovat kaikkien ohjelmien käytettävissä. Jotkut muuttujista ovat vain luettavissa, toiset ovat

myös arvoltaan muutettavissa. Tarkempi listaus ja kuvaus järjestelmämuuttujista löytyy KUKA robotin järjestelmädokumentaation osiosta System Variables (KUKA Roboter GmbH, 2005).

Järjestelmämuuttujan nimi alkaa aina $-merkillä. Tämän vuoksi merkin käyttöä tulisi välttää omien muuttujien nimissä.

4.4.1 $FLAG

KRC tukee yhteensä 1024 $FLAG järjestelmämuuttujaa ($FLAG[1] .. $FLAG[1024]) joita voidaan käyttää globaalisti yleisinä markkereina. $FLAG järjestelmämuuttujan arvo voi olla TRUE tai FALSE.

4.4.2 $CYCFLAG

KRC tukee yhteensä 32 $CYCFLAG järjestelmämuuttujaa. Nämä järjestelmämuuttujat ovat syklisiä, eli niiden tila tarkistetaan jatkuvasti ohjelman aikana, riippumatta ohjelmalaskurin kulloisestakin sijainnista. $CYCFLAG järjestelmämuuttujan arvo voi olla TRUE tai FALSE. Sen etuna on, että sen arvo voidaan asettaa millä tahansa Boolean-lauseen avulla. $CYCFLAG järjestelmämuuttujan arvo voidaan lukea esim. kahden erillisen INPUT-signaalin arvojen perusteella.

```
..
;-- INPUT lähde 1 saa nimeksi ANTURI_1
SIGNAL ANTURI_1 $IN[1]
;-- INPUT lähde 2 saa nimeksi ANTURI_2
SIGNAL ANTURI_2 $IN[2]

;-- Cyclical Flag 1 saa arvonsa ---
;-- AND lauseen tuloksena ---------

$CYCFLAG[1] = ANTURI_1 AND ANTURI_2

;-- $CYCFLAG[1] = TRUE -------------
;-- VAIN kun ANTURI_1 = TRUE -------
;-- JA ANTURI_2 = TRUE -------------

..
```

4.5 **Operaattorit**

Muuttujien arvoja voidaan käsitellä, tarkastella ja vertailla erilaisten operaattorien avulla. KRL:ssä operaattorit voidaan jakaa aritmeettisiin, geometrisiin, vertaileviin, loogisiin ja bittioperaattoreihin.

Aritmeettisista operaattoreista käytössä ovat perusmuotoiset operaattorit +, -, * ja /.

```
INT A
REAL B

A = 2 + 5          ;A=7
B = 2.5 + 2        ;B=4.5
B = 5 / 2          ;B=2.5
A = 5.0 * 2        ;A=10
```

Geometrinen operaattori **:** linkittää yhteen kaksi POS- tai FRAME-tyyppistä muuttujaa. Operaattori lisää vasemmanpuolisen muuttujan koordinaattiarvoihin oikeanpuolisen muuttujan koordinaattiarvot.

```
FRAME PAIKKA_1
FRAME PAIKKA_2
FRAME SIJAINTI

PAIKKA_1={X 450,Y 600,Z 800,A 0,B 0,C 0}
PAIKKA_2={X 80,Y 110,Z 55,A 0,B 0,C 0}

SIJAINTI=PAIKKA_1:PAIKKA_2
;SIJAINTI arvo on nyt vaihtoehtoisesti
;{X 530,Y 710,Z 855,A 0,B 0,C 0}
;tai
;{X 530,Y 710,Z 855,A -180,B -180,C -180}
```

Vertailuoperaattorien avulla voidaan suorittaa vertailuja muuttujien välillä. Vertailuoperaattorin tulos on aina BOOL-tyyppinen. Vertailu voidaan tehdä vain operaattorin sallimien tietotyyppien välillä.

Operaattori	Selitys	Sallitut tietotyypit
==	yhtä suuri kuin	INT, REAL, CHAR, ENUM, BOOL
<>	erisuuri kuin	INT, REAL, CHAR, ENUM, BOOL
>	suurempi kuin	INT, REAL, CHAR, ENUM
<	pienempi kuin	INT, REAL, CHAR, ENUM
>=	suurempi tai yhtä suuri	INT, REAL, CHAR, ENUM
<=	pienempi tai yhtä suuri	INT, REAL, CHAR, ENUM

```
BOOL A,B  ; esitellään kaksi
          ; BOOL-tyyppistä muuttujaa

B = 10 < 3                                  ;B=FALSE
A = 9/3 == 3                                ;A=TRUE
A = "F" < "Z"                               ;A=TRUE
B = ((B == A) <> (10.1 > 10)) == TRUE ;B=TRUE
```

Logiikkaoperaattorien AND, OR, XOR ja NOT avulla voidaan muodostaa erilaisia logiikkaoperaatioita lauseiden tai muuttujien välillä. Logiikkaoperaattorien tulos on aina BOOL-tyyppinen. Tuloksen arvo riippuu logiikkaoperaattoreista ja on oheisen logiikkataulun mukainen:

Operaattori		**NOT A**	**A AND B**	**A OR B**	**A EXOR B**
A = TRUE	B = TRUE	FALSE	TRUE	TRUE	FALSE
A = TRUE	B = FALSE	FALSE	FALSE	TRUE	TRUE
A = FALSE	B = TRUE	TRUE	FALSE	TRUE	TRUE
A = FALSE	B = FALSE	TRUE	FALSE	FALSE	FALSE

Bittioperaattorien avulla voidaan suorittaa loogisia operaatioita bittitasolla. Bittioperaattorit toimivat kuten logiikkaoperaattorit, mutta lukujen sijaan vertailu tehdäänkin niiden yksittäisten bittien välillä, joista luvut koostuvat. Bittioperaattorien toimintaa on selostettu tarkemmin myös KUKA dokumentaation Expert Programming -osiossa (KUKA Roboter GmbH, 2006).

5 KOORDINAATISTOJÄRJESTELMÄT

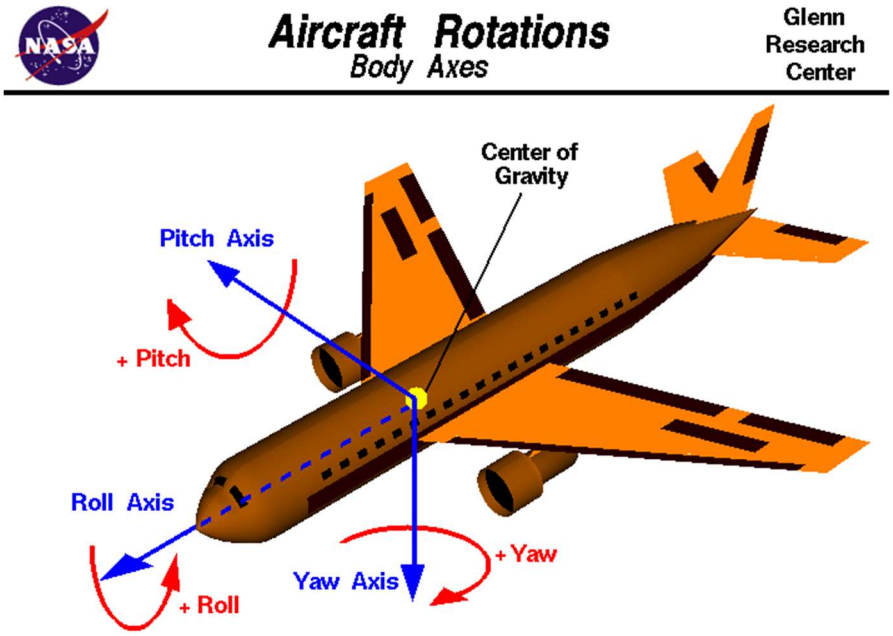
5.1 Nivelkoordinaatisto

Nivelkoordinaatistossa robotin paikoitus on ilmoitettu sen jokaisen nivelen kiertokulman avulla. KUKA C2 -nivelrobotissa on 6 kääntyvää niveltä ja jokainen asema voidaan ilmaista kuuden suureen (A1, A2, A3, A4, A5, A6) avulla. Robottia käyttävälle ja ohjelmoivalle ihmiselle on helpompaa hahmottaa sijainti suorakulmaisessa koordinaatistossa. Robottiohjain huolehtii koordinaatiston kääntämisestä nivelkoordinaatiston ja suorakulmaisen koordinaatiston välillä (Kuva 2).

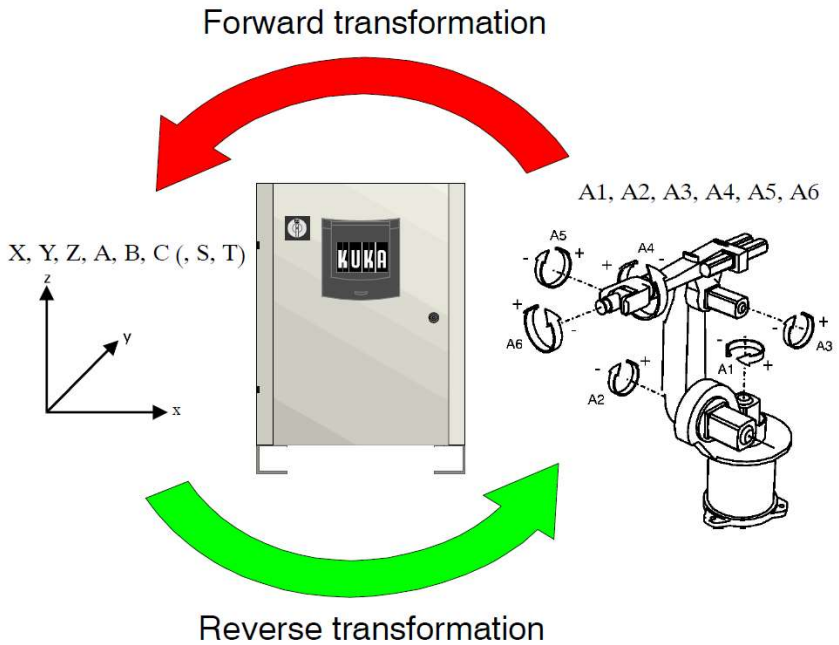
5.2 Suorakulmainen koordinaatisto

Suorakulmaisessa eli karteesisessa koordinaatistossa on käytössä kolme akselia (**X, Y, Z**) jotka ovat toisiaan vasten kohtisuorassa ja kohtaavat origossa. Paikkakoordinaatissa ilmoitetaan etäisyys origosta jokaiseen suuntaan.

Jotta robotti voidaan kohdistaa oikein koordinaatiston pisteeseen, tarvitaan lisäksi kolme kiertoa kuvaavaa koordinaatistoarvoa (**A, B, C**). Arvot A, B ja C kuvaavat lentokonedynamiikasta tuttuja *roll-pitch-yaw* –perusliikkeitä (Kuva 1). Arvo A on kääntyminen ja se kuvaa kiertoa pystyakselin ympäri (yaw). Arvo B on nyökkääminen ja se kuvaa kiertoa poikkiakselin ympäri (pitch). Arvo C on kallistuminen ja se kuvaa kiertoa pituusakselin ympäri (roll).

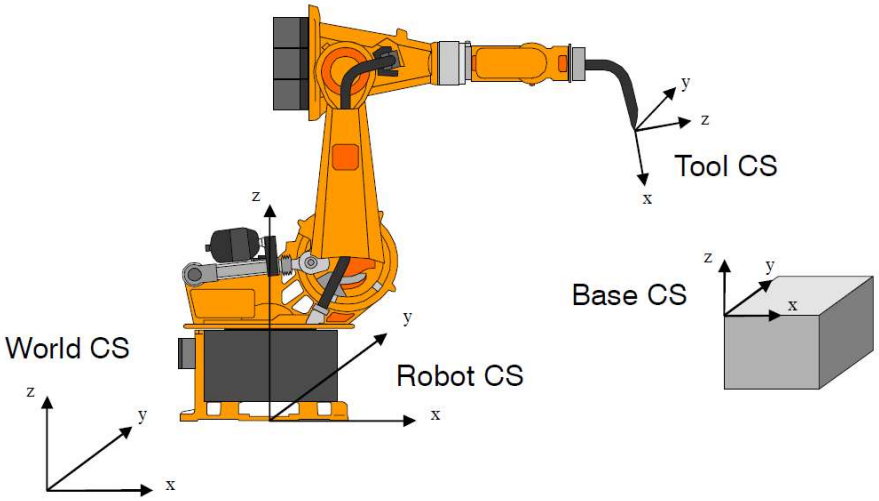


Kuva 1. Roll-Pitch-Yaw –liikkeet A,B,C parametreja annettaessa. (NASA Glenn Research Center, 2017)



Kuva 2. KRC:n suorittama koordinaatistojen kääntäminen (KUKA Roboter GmbH, 2006)

5.3 Koordinaatistot (WORLD / ROBOT / TOOL / BASE)



Kuva 3. KUKA-robotin koordinaatistot (KUKA Roboter GmbH, 2006)

Maailmankoordinaatisto, jota vastaa järjestelmämuuttuja **$WORLD** on kiinteäksi määritelty ja toimii referenssipisteenä kaikille muille määritellyillä koordinaatistoille (Kuva 3).

Robottikoordinaatisto, jota vastaa järjestelmämuuttuja **$ROBROOT** voi paikalleen asennetulla robotilla olla sama kuin maailmankoordinaatisto. Jos robotti on asennettu kiinteästi muualle kuin käytettävän maailmankoordinaatiston origoon, määritellään robottikoordinaatiston origon koordinaatit suhteessa maailmankoordinaatistoon. Mikäli robotti kykenee liikkumaan esim. omalla alustallaan, sen robottikoordinaatiston sijainti vaihtelee suhteessa maailmankoordinaatistoon.

Työkalukoordinaatistoa vastaa järjestelmämuuttuja **$TOOL** ja sen origo määritellään sijoittumaan esim. työkalun kärkipisteeseen. Tällöin työkalun sijaintia määritellessä referenssinä käytetään maailmankoordinaatistoa. Jos itse työstettävä kappale sijaitsee robotin päässä ja sitä liikutellaan kiinteästi sijoitettua työkalua vasten, viitataan työkalukoordinaatistolla robotin päähän sijoitettuun työkappaleeseen.

Työkoordinaatistoa vastaa järjestelmämuuttuja **$BASE**. Työkoordinaatiston avulla työkappaleelle (tai alueelle) määritellään sijainti suhteessa maailmankoordinaatistoon. Työliikkeet voidaan ohjelmoida työkoordinaatistossa, jonka origon sijainti suhteessa maailmankoordinaatistoon on määritelty. Työskenneltäessä useiden identtisten työkappaleiden kanssa, voidaan liikkeet toistaa samanlaisina työkoordinaatistoa siirtämällä. Tämä voidaan toteuttaa esim. järjestelmämuuttujaa $BASE muuttamalla. Kun

työkappaletta liikutetaan robotin avulla kiinteästi sijoitettua työkalua vasten, viitataan työkoordinaatistolla kiinteästi sijoitettuun työkaluun.

6 KÄYTÖN ALOITUS

6.1 Työkalupisteen asetus

Järjestelmään voidaan tallentaa 16 työkalupisteen tiedot. Työkaluun voidaan viitata muuttujanimellä TOOL_DATA[1]..TOOL_DATA[16] ja sen arvo voidaan lukea järjestelmämuuttujaan $TOOL.

Uusi työkalu määritellään käyttöön KSS:n TOOL-valikon avulla.

6.1.1 Työkalu robotin päässä

Valittavissa on yhteensä kuusi erilaista tapaa määritellä työkalupiste. Ohjelmaesimerkeissä käytetyt työkalut on määritelty käyttöön XYZ 4-Point -metodin avulla.

XYZ 4-Point -metodissa robotin päähän kiinnitetty työkalu paikoitetaan samaan pisteeseen neljältä eri suunnalta (Kuva 4).

Kuva 4. Työkalupisteen asetus

6.1.2 Työkalu kiinteästi robotin ulkopuolelle asennettuna

Valittavissa on kaksi erilaista tapaa määritellä työkalupiste, joka on robotin ulkopuolelle kiinteästi asennettuna. Ohjelmaesimerkeissä käytetyt ulkoiset työkalupisteet on määritelty käyttöön 5-D -metodin avulla.

5-D -metodissa robotin päähän kiinnitetty, jo aikaisemmin määritelty työkalu tuodaan uuden määriteltävän työkalun työkalupisteeseen (Kuva 5). Robotin työkalulaippaa käännettään tämän jälkeen siten, että työkalupiste suuntautuu kohtisuorasti työkalulaippaa vasten.

Kuva 5. Ulkopuolisen kiinteän työkalupisteen asetus

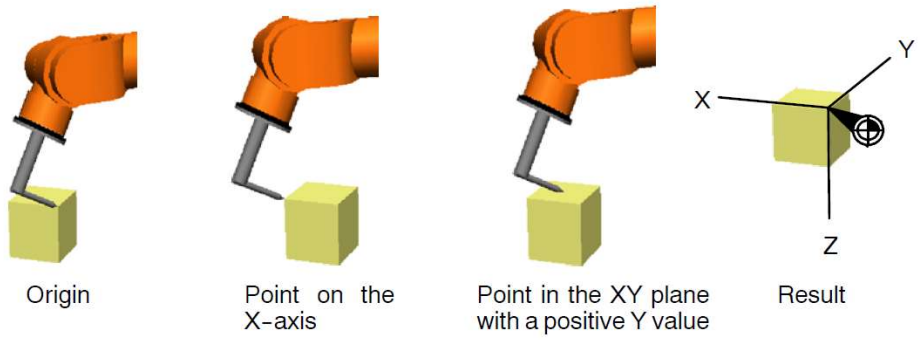
6.2 **Työkoordinaatiston asetus**

Järjestelmään voidaan tallentaa 16 työkoordinaatiston tiedot. Työkoordinaatistoon voidaan viitata muuttujanimellä BASE_DATA[1]..BASE_DATA[16] ja sen arvo voidaan lukea järjestelmämuuttujaan $BASE.

6.2.1 Työkappale robotin ulkopuolella

Valittavissa on kolme erilaista tapaa määritellä työkoordinaatisto kappaleelle, joka on robotin ulkopuolella. Ohjelmaesimerkeissä käytetyt työkoordinaatistot on määritelty käyttöön 3-Point -metodin avulla.

3-Point -metodissa robotin päässä oleva työkalu tuodaan työkappaleessa kohtaan, johon halutaan määritellä työkoordinaatiston origo. Pisteen tallennuksen jälkeen tallennetaan vielä yksi piste työkoordinaatiston positiiviselta X-akselilta ja yksi piste työkoordinaatiston positiiviselta XY-tasolta (Kuva 6).

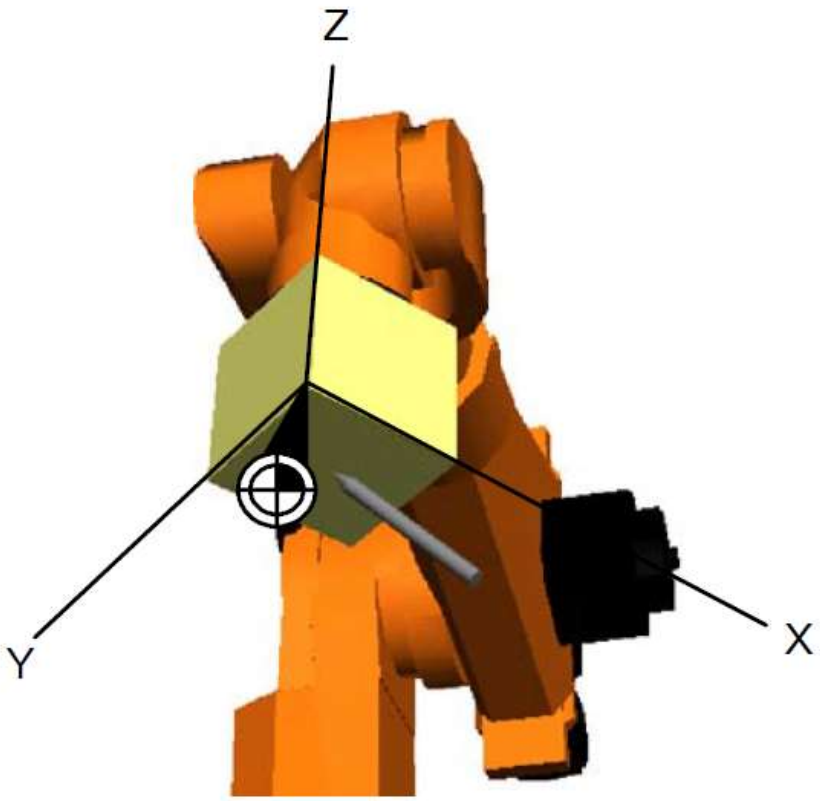


Kuva 6. Työkoordinaatiston asetus 3-Point -metodilla (KUKA Roboter GmbH, 2006)

6.2.2 Työkappale robotin päässä liikuteltavana

Valittavissa on kaksi erilaista tapaa määritellä työkoordinaatisto kappaleelle, joka on robotin ohjailema. Ohjelmaesimerkeissä käytetyt työkalupisteet on määritelty käyttöön Direct Measuring -metodin avulla.

Direct Measuring -metodissa robotin päässä liikuteltava työkappale tuodaan lähelle robotin ulkopuolelle asennettua työkalua ja paikoitetaan kohtaan, johon halutaan määritellä työkoordinaatiston origo. Pisteen tallennuksen jälkeen tallennetaan vielä yksi piste työkoordinaatiston positiiviselta X-akselilta ja yksi piste työkoordinaatiston positiiviselta XY-tasolta (Kuva 7).



Kuva 7. Robotin liikutteleman kappaleen työkoordinaatiston asetus (KUKA Roboter GmbH, 2006)

6.3 **Interpolaatio**

Järjestelmämuuttuja $TOOL viittaa aina robotin päähän kiinnittyneeseen koordinaatistoon, huolimatta siitä onko robotin päähän kiinnitettynä liikuteltava työkalu vai työkappale.

Järjestelmämuuttuja $BASE vastaavasti viittaa aina robotin ulkopuolella olevaan koordinaatistoon, huolimatta siitä onko kyseessä työkappale vai kiinteästi robotin ulkopuolelle asennettu työkalu.

Järjestelmämuuttuja $IPO_MODE pitää sisällään tiedon siitä, liikutetaanko robotin päässä työkalua vai työkappaletta. Järjestelmämuuttujan arvon perusteella robottiohjain tekee tarvittavat laskutoimitukset, jotta robotin liike on halutun kaltainen.

$IPO_MODE = #TCP
Robotti liikuttaa työkappaletta ja työkalu on robotin ulkopuolelle asennettu.

$IPO_MODE=#BASE
Robotti liikuttaa työkalua ja työkappale sijaitsee robotin ulkopuolella.

6.4 **Uuden ohjelman aloitus**

Uuden ohjelman luominen aloitetaan valitsemalla valikosta ’NEW’, minkä jälkeen voidaan valita mallipohja luotavalle ohjelmalle. Robottiohjelmiston mukana on toimitettu kuusi valmista mallipohjaa (templates) ohjelman luomiseen. Näistä käsiteltyinä alla kaksi: Module ja Expert. Tarkemmat kuvaukset mallipohjista löytyvät KSS:n dokumentaation Expert Programming -osiosta.

6.4.1 Mallipohjat (Module ja Expert)

Mallipohjan valinnan jälkeen annetaan luotavalle ohjelmalle nimi. Nimi voi olla korkeintaan 24 merkkiä pitkä. Järjestelmä luo annetun nimen mukaiset tiedostot sekä lisää automaattisesti tiedostopäätteet .SRC ja .DAT.

Expert-mallipohjaa käytettäessä järjestelmä luo automaattisesti SRC- ja DAT-tiedostot. Tiedostot sisältävät ainoastaan aloittavan DEF ja päättävän END rivin. Mallipohjaa käytettäessä tulee järjestelmämuuttujien arvot asettaa käsin.

Module-mallipohjaa käytettäessä järjestelmä luo automaattisesti SRC- ja DAT-tiedostot perusrivien kanssa. Nämä perusrivit sisältävät mm. järjestelmämuuttujien alustuksen sekä HOME-paikoitusrivin. Module-mallipohjan lisäämät rivit ovat oletuksena piilotettu FOLD-lohkon sisään. Rivit saadaan näkyviin valitsemalla ’OPEN ALL FOLDS’.

```
DEF ESIMERKKI ()

INI

PTP HOME Vel=100%

PTP HOME Vel=100%

END
```

Module-mallipohjaa käyttäen luotu uusi .SRC tiedosto.

```
;FOLD INI
  ;FOLD BASISTECH INI
    GLOBAL INTERRUPT DECL 3 WHEN
$STOPMESS==TRUE DO IR_STOPM ( )
    INTERRUPT ON 3
    BAS (#INITMOV,0 )
  ;ENDFOLD (BASISTECH INI)
  ;FOLD USER INI
    ;Make your modifications here

  ;ENDFOLD (USER INI)
;ENDFOLD (INI)
```

Module-mallipohjan lisäämä INI ei ole yksittäinen käsky vaan sarja käskyjä, jotka on piilotettu FOLDia käyttäen.

6.5 **Koordinaattipisteiden asetus**

Koordinaattipiste voidaan määritellä esim. POS- tai FRAME-tyyppisen muuttujan avulla.

```
..

FRAME PAIKKA_1
POS PAIKKA_2

PAIKKA_1={X 100,Y 100,Z 50,A 45,B 0,C 0}
PAIKKA_2={X 50,Y 200,Z 0,A 0,B 0,C 0,S 2,T 5}
..
```

Pisteen paikka voidaan määritellä myös nykyisen sijainnin perusteella. Nykyinen sijainti voidaan lukea järjestelmämuuttujasta $POS_ACT.

```
DECL POS SIJAINTI1 ;luodaan POS tyyppiset
DECL POS SIJAINTI1 ; muuttujat SIJAINTI1
                   ; ja SIJAINTI2
;-- asetetaan muuttujan arvoksi nykyinen
sijainti
SIJAINTI1 = $POS_ACT
;-- siirretään x-koordinaatistossa -100mm
SIJAINTI1.X = SIJAINTI1.X-100


SIJAINTI2 = $POS_ACT
SIJAINTI2.X = SIJAINTI2.X+250
SIJAINTI2.Y = SIJAINTI2.Y+100

PTP HOME

PTP SIJAINTI1 ;ajo pisteeseen SIJAINTI1
WAIT SEC 2    ;odotetaan 2 sekuntia
PTP SIJAINTI2 ;ajo pisteeseen SIJAINTI2

;uudelleenmääritellään SIJAINTI2
SIJAINTI2.X = SIJAINTI1.X
SIJAINTI2.Y = SIJAINTI1.Y
;SIJAINTI2 on nyt sama kuin SIJAINTI1

PTP SIJAINTI2

PTP HOME
```

7 LIIKEKÄSKYT

7.1 Point-to-point

Robotti suorittaa point-to-point -liikkeen mahdollisimman nopeasti. Kaikkien akselien liike mitoitetaan alkamaan ja päättymään samanaikaisesti. Liikkeestä johtuen liikerataa ei voida ohjelmoitaessa etukäteen tarkasti ennustaa vaan KRC laskee ja toteuttaa nopeimman mahdollisen.

7.1.1 PTP

PTP-liikekäskyn perässä annetut koordinaatit lasketaan valitun työkoordinaatiston origosta.

```
FRAME KOHDE_1

$BASE = BASE_DATA[2]
$TOOL = TOOL_DATA[10]
KOHDE_1={FRAME:X 250,Y 250,Z 100,A 0,B 0,C 0}

;--- ajo PTP -liikkeellä paikkaan KOHDE_1 -
PTP KOHDE_1

;-- ajo uuteen pisteeseen, jonka sijainti on
;-- määritetty suhteessa työkoordinaatistoon

PTP {X 100,Y 100,Z 0,A 0,B 0,C 0}

;-- uusi piste suhteessa työkoordinaatistoon
;-- kaikkia koordinaatteja ei tarvitse antaa

PTP {Y 400,Z 300}
```

7.1.2 PTP_REL

PTP_REL-liikekäskyn koordinaatit lasketaan robotin nykyisen sijainnin mukaan. Annetut koordinaatit (sekä taso- että nivelkoordinaatit) lisätään senhetkisen aseman koordinaatteihin tai vähennetään niistä.

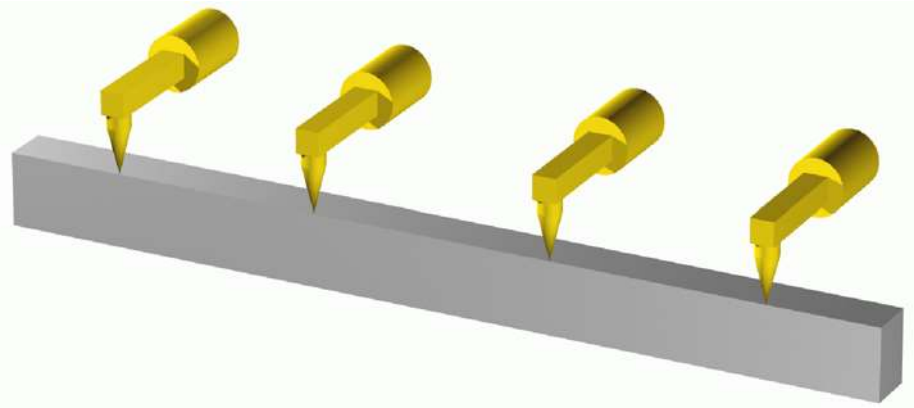
```
..

$BASE = BASE_DATA[6]
$TOOL = TOOL_DATA[2]

;-- paikoitus koordinaatteihin -
PTP {X 500,Y 500,Z 100,A 0,B 0,C 0}

;-- siirto nykyisestä asemasta
PTP_REL {X 100,Y 100,Z 50}
;-- uusi asema on nyt -----------
; {X 600,Y 600,Z 150}

PTP {X 100,Y 100}
;-- uusi asema on nyt ----------
; {X 100,Y 100, Z 0}
```

7.2 CP (=Continous Path)

Point-to-point -liikkeessä ainoastaan liikkeen alku- ja päätepiste ovat määriteltävissä. Liikkeen aikana KRC vastaa liikeratojen laskemisesta ja robotin akselien asennoista. CP- eli jatkuvan radan liikkeessä työkalun paikoitus ja asento voidaan määritellä koko liikkeen ajan.

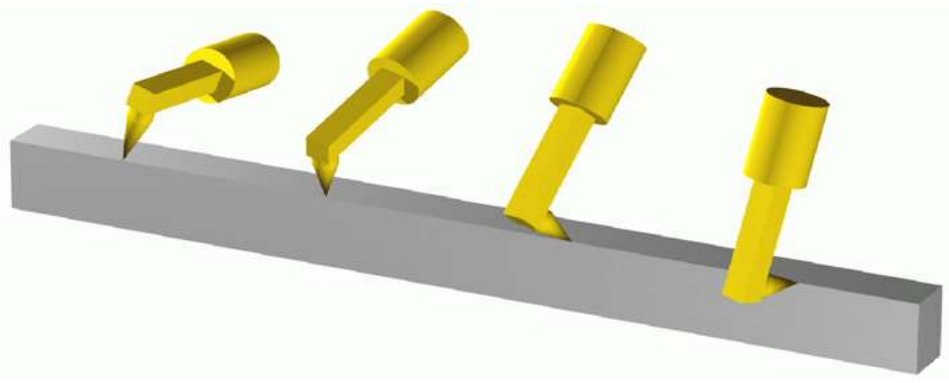
Työkalupisteen orientaatiota määriteltyyn pisteeseen nähden ohjataan järjestelmämuuttujalla $ORI_TYPE.

$ORI_TYPE = #CONSTANT
Työkalupisteen orientaatio säilyy muuttumattomana koko liikkeen ajan (Kuva 8).



Kuva 8. $ORI_TYPE=#CONSTANT (KUKA Roboter GmbH, 2006)

$ORI_TYPE=#VAR
Työkalupisteen orientaatio muuttuu liikkeen aikana (Kuva 9). Tämä on oletusasetus.

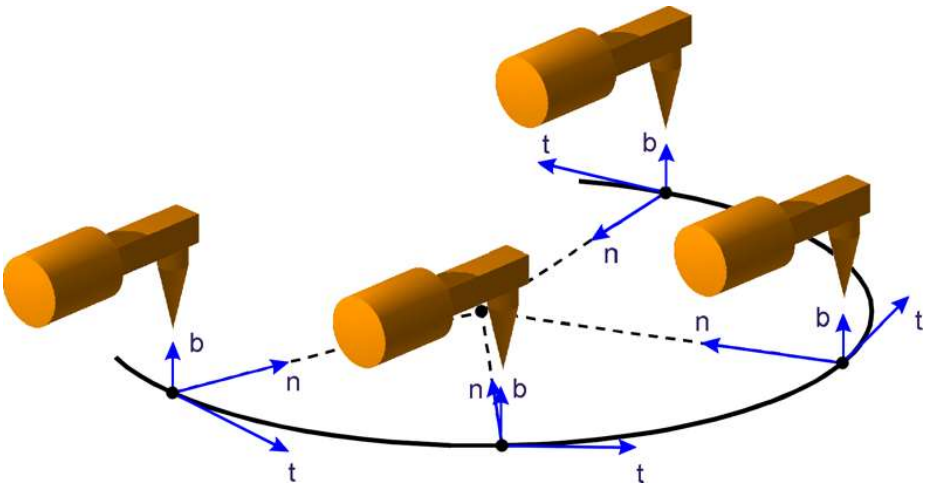


Kuva 9. $ORI_TYPE=#VAR
(KUKA Roboter GmbH, 2006)

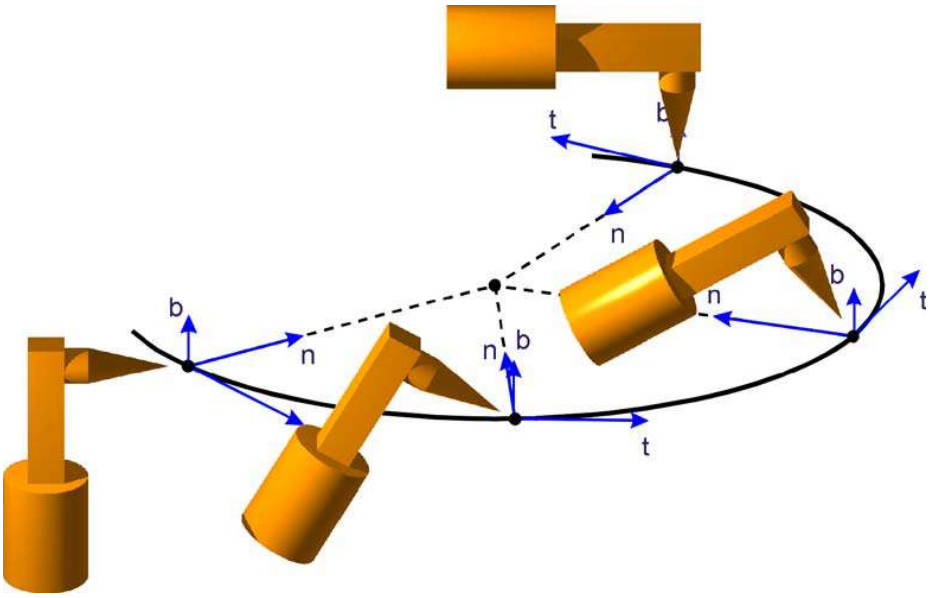
Ympyräliikkeen aikana orientaatiota voidaan ohjata lisäksi järjestelmämuuttujalla $CIRC_TYPE.

$CIRC_TYPE=#BASE
Työkalupisteen suuntautuminen säilyy samana läpi ympyräliikkeen. Tämä on oletusasetus.

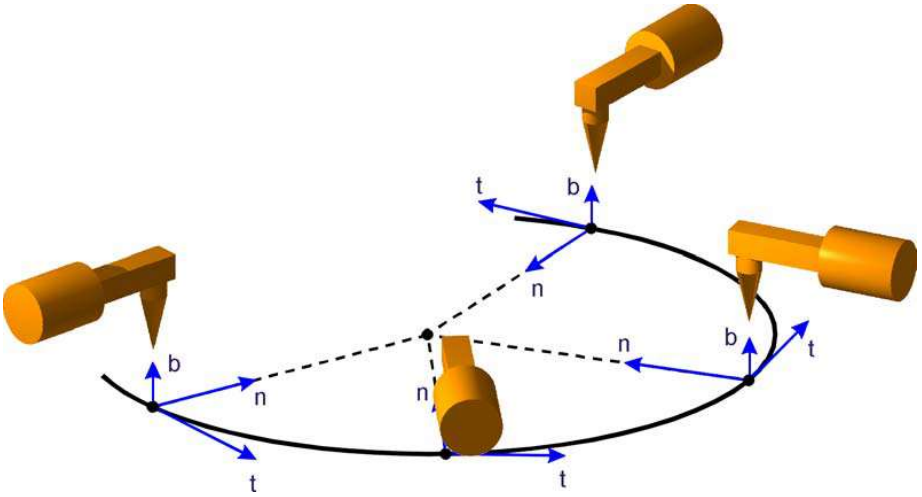
$CIRC_TYPE=#PATH
Työkalupiste säilyttää suuntautumisensa kohti ympyräkaaren keskipistettä läpi ympyräliikkeen.



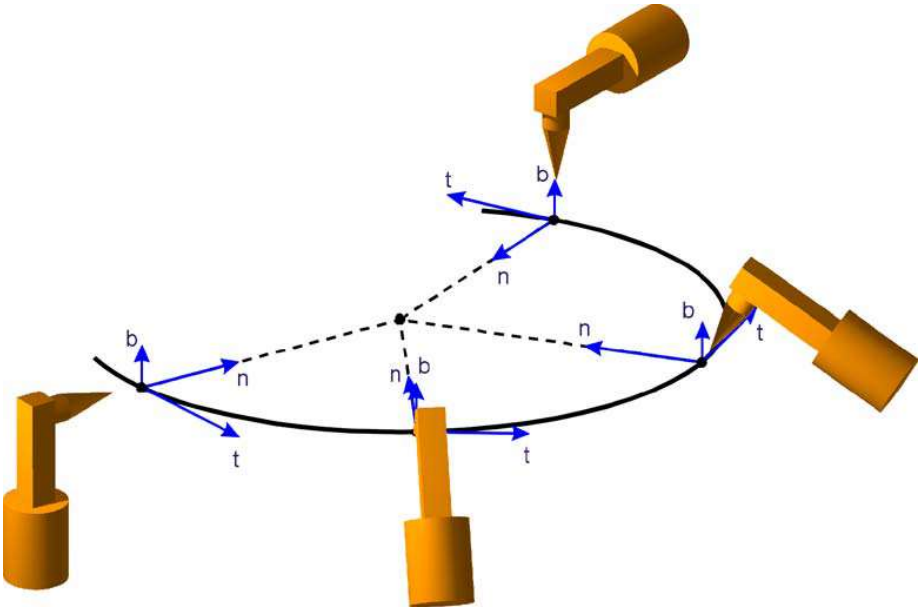
Kuva 10. $ORI_TYPE=#CONST - $CIRC_TYPE=#BASE (KUKA Roboter GmbH, 2006)



Kuva 11. $ORI_TYPE=#VAR - $CIRC_TYPE=#BASE (KUKA Roboter GmbH, 2006)



Kuva 12. $ORI_TYPE=#CONSTANT - $CIRC_TYPE=#PATH (KUKA Roboter GmbH, 2006)



Kuva 13. $ORI_TYPE=#VAR - $CIRC_TYPE=#PATH (KUKA Roboter GmbH, 2006)

7.2.1 LIN

Suoraviivaisen liikkeen koordinaatti annetaan aina suoraviivaisessa koordinaatistossa. Päätepisteen S ja T arvot eivät muutu, vaan ne määräytyvät alkupisteen perusteella.

```
..

;HOME piste määritelty yksiselitteisesti
;nivelkoordinaatiston arvoilla
HOME={AXIS: A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}

PTP HOME ;--- BCO ajo

;-- oletusarvona $ORI_TYPE=#VAR
;-- työkalupisteen orientaatio muuttuu
;-- liikkeen aikana ---
LIN {X 30,Y 50,Z 30,A 6,B 5,C 3}

;-- päätepisteen S ja T arvot eivät muutu
LIN {POS:X 30,Y 150,Z 30,A 6,B 5,C 3,S 2,T 35}

$ORI_TYPE=#CONSTANT
;-- työkalupisteen orientaatio ei muutu
;-- liikkeen aikana
LIN {FRAME:X 100,Y 300,Z 0,A 20,B 10,C 0}

..
```

7.2.2 LIN_REL

Myös LIN_REL-käskyllä suoraviivaisen liikkeen koordinaatti annetaan suoraviivaisessa koordinaatistossa. Koordinaattipiste ilmoitetaan ja lasketaan kuitenkin suhteessa senhetkiseen pisteeseen.

```
..
LIN {FRAME:X 100,Y 100,Z 50,A 0,B 0,C}

LIN_REL {FRAME:X 50}
;-- koordinaatit X 150, Y 100 . . .

LIN_REL {Y: 100}
;-- koordinaatit nyt X 150 Y 200 . . .
```

7.2.3 CIRC

Kaariliikkeen alkupisteenä toimii aina senhetkinen sijainti. Liikkeen toteuttamiseen käytetään CIRC- tai CIRC_REL-käskyä. Käskyn yhteydessä annetaan kaksi koordinaattia, jotka ilmaisevat paikkaa työpistekoordinaatistossa. Ensimmäinen annettu koordinaatti määrittelee ns. välipisteen (Auxiliary Point). Toinen annettu koordinaatti määrittelee kaaren päätepisteen. Välipisteen osalta huomioidaan ainoastaan X-, Y- ja Z-koordinaatit.

```
..
FRAME MIDDLEPOINT
FRAME ENDPOINT

PTP HOME ;-- BCO

;-- työkoordinaatisto
$BASE = BASE_DATA[2]
;-- työkalukoordinaatisto
$TOOL = TOOL_DATA[4]

MIDDLEPOINT={X 100,Y 100,Z 0}
ENDPOINT={X 150,Y 50,Z 0,A 30,B 10,C 25}

;-- yksiselitteisesti määritelty asema
PTP {POS: X 50,Y 50,Z 0,A 0,B 0,C 0,S 2,T 10}

CIRC MIDDLEPOINT,ENDPOINT

..
```

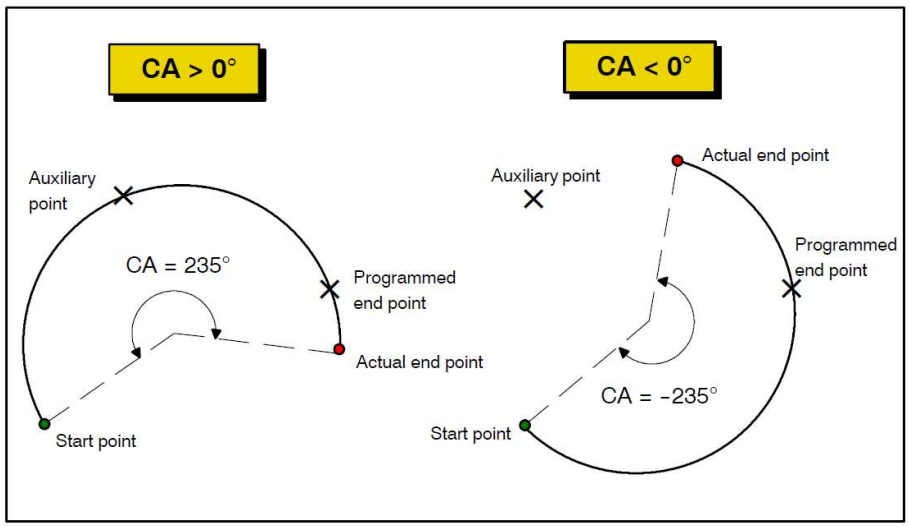
7.2.4 CIRC_REL

CIRC_REL-käsky toimii kuten CIRC-käsky. Erona on se, että CIRC_REL-käskyä käytettäessä koordinaattipisteet ilmaisevat paikkaa suhteessa senhetkiseen sijaintiin.

```
..

;-- yksiselitteisesti määritelty asema
PTP {POS: X 50,Y 50,Z 0,A 0,B 0,C 0,S 2,T 10}

CIRC_REL {X 50,Y 50},{X 100,Y 0}

..
```

7.2.5 CA (=Circular Angle)

Kaariliikkeen loppukoordinaattien perässä voidaan käyttää CA-valintaa. Kaaren välipiste ja päätepiste annetaan kuten normaalisti, mutta kaaren todellinen päätepisteen todellinen paikka määräytyy annetun keskuskulman (Circular Angle) perusteella (Kuva 14). KSS-dokumentaation Expert Programming -osan kuva selventää pisteen sijoittumista käytettäessä CA-valintaa.



Kuva 14. CA-valinnan vaikutus kaariliikkeen päätepisteeseen (KUKA Roboter GmbH, 2006)

7.3 Advance Run

Advance Run -ominaisuuden avulla liikeradat lasketaan ja suunnitellaan jo ennen kuin ohjelmalaskuri siirtyy liikekomennon kohdalle. Ominaisuus mahdollistaa yhdistetyt liikkeet, joissa liikettä ei pysäytetä sen loppupisteeseen vaan sen sijaan se sulautuu osaksi seuraavaa liikettä. Advanced Run -ominaisuuden avulla toteutetut liikkeet mahdollistavat nopeammat usean liikkeen sarjat, kun robotin akseleita ei tarvitse kiihdyttää ja hidastaa tarkkaan pisteeseen paikoituksen vuoksi.
(Mühe;Angerer;Hoffmann;& Reif, 2017)

Advance Run hyödyntää omaa erillistä Advance Run -ohjelmalaskuria, jonka puskurin kokoa voidaan säätää $ADVANCE-järjestelmämuuttujan avulla (Kuva 15). $ADVANCE-järjestelmämuuttujan arvo voi olla välillä 1-5 ja sen oletusarvo on 3.

```
14   $ADVANCE=1
15
16   LIN  {X 1620,Y 0,Z 1910,A 0,B 90,C 0}
17
18   STROM={STROM*1.2}/0.5
19   FOR I=1 TO  6
20       $VEL_AXIS[I]=60
21       $ACC_AXIS[I]=35
22   ENDFOR
23
24   PTP  PUNKT6
25
26   SPANNUNG=110
27
28   PTP  PUNKT7
```

Kuva 15. Advance Run käytössä (KUKA Roboter GmbH, 2006)

Kuvassa 15 ohjelmalaskuri (Main Run Pointer) on rivin 16 kohdalla. Järjestelmämuuttuja $ADVANCE on asetettu arvoon 1. Tällä asetuksella ohjelmalaskuria seuraava lohko (rivit 18–22) on käsitelty samanaikaisesti rivin 16 liikekäskyn kanssa ja Advance Run -laskuri on rivillä 24 valmistelemassa seuraavaa käskylohkoa.

7.4 **Likimääräinen paikoitus**

Likimääräisen paikoituksen (Approximate Positioning) toteuttamiseksi Advance Run on oltava aktiivinen ja $ADVANCE-järjestelmämuuttujan arvon on oltava vähintään 1.

7.4.1 PTP-PTP

Järjestelmämuuttuja $APO_DIS_PTP voidaan määritellä jokaiselle robotin akselille erikseen. Arvona käytetään astelukua joka ilmaisee akselin kääntökulmaa. Järjestelmämuuttujan $APO.PTP avulla määritellään osuus maksimikulmasta, jonka saavuttamisen jälkeen käytetään likimääräistä paikoitusta liikeratojen laskemisessa. Kun PTP-liikekäsky halutaan toteuttaa likimääräistä paikoitusta hyödyntäen, käskyrivin loppuun lisätään C_PTP.

```
..
INT I

;-- kaikkien kuuden akselin
;-- maksimikulman asetus
;-- 90 asteeseen

FOR I=1 TO 6
   $APO_DIS_PTP[I]=90
ENDFOR

;-- likimääräinen paikoitus käyttöön
;-- kun akselin kulma saavuttaa 50%
;-- maksimiarvosta eli 45%

$APO.CPTP = 50

;-- PTP liike pisteeseen POINT3
;-- käyttäen likimääräistä paikoitusta

PTP POINT3 C_PTP
```

7.4.2 LIN-LIN

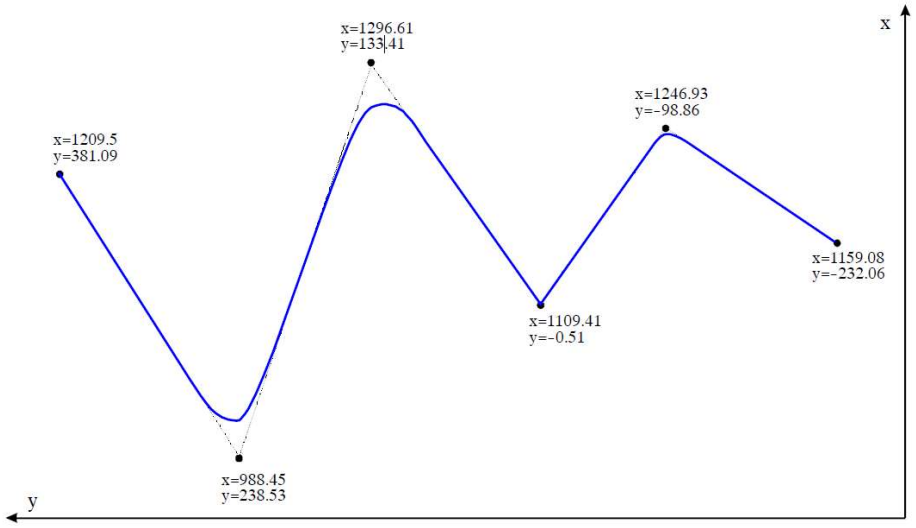
Kun halutaan hyödyntää likimääräistä paikoitusta kahden suoraviivaisen liikkeen välillä, muokataan jotain kolmesta järjestelmämuuttujasta.

$APO_CDIS-järjestelmämuuttujan avulla voidaan määrittää etäisyys loppupisteestä, jonka saavuttamisen jälkeen käytetään likimääräistä paikoitusta. $APO_DIS-järjestelmämuuttujan arvona käytetään millimetrejä. Kun LIN liikekäsky halutaan toteuttaa etäisyysperusteista likimääräistä paikoitusta hyödyntäen, käskyrivin loppuun lisätään **C_DIS**.

$APO_CORI-järjestelmämuuttujan avulla voidaan määrittää kulma, jonka saavuttamisen jälkeen käytetään likimääräistä paikoitusta. Kun pääsuuntaakseli saavuttaa halutun kulma-aseman loppupisteeseen nähden, käytetään likimääräistä paikoitusta. $APO_CORI-järjestelmämuuttujan arvona käytetään astelukua. Kun LIN liikekäsky halutaan toteuttaa kulmaperusteista likimääräistä paikoitusta hyödyntäen, käskyrivin loppuun lisätään **C_ORI**.

$APO_CVEL-järjestelmämuuttujan avulla määritellään nopeusarvo, jonka saavuttamisen jälkeen käytetään likimääräistä paikoitusta. $APO_CVEL-järjestelmämuuttujan arvona käytetään prosenttiosuutta $VEL-järjestelmämuuttujassa määritellystä akselin maksiminopeudesta. Kun LIN-liikekäsky halutaan toteuttaa nopeusperusteista likimääräistä paikoitusta hyödyntäen, käskyrivin loppuun lisätään **C_VEL**.

```
..
;-- tarkka paikoitus koordinaatteihin
PTP {POS: X 1159.08,Y -232.06,Z 716.38,A
171.85,B 67.32,C 162.65,S 2,T 10}

;-- etäisyysperusteinen
;-- likimääräinen paikoitus
LIN {X 1246.93,Y -98.86,Z 715,A 125.1,B 56.75,C
111.66} C_DIS

;tarkka paikoitus
LIN {X 1109.41,Y -0.51,Z 715,A 95.44,B 73.45,C
70.95}

;kulmaperusteinen
LIN {X 1296.61,Y 133.41,Z 714.99,A 150.32,B
55.07,C 130.23} C_ORI

;nopeusperusteinen
LIN {X 988.45,Y 238.53,Z 714.99,A 114.65,B
50.46,C 84.62} C_VEL

;tarkka paikoitus
LIN {X 1209.5,Y 381.09,Z 715,A -141.91,B
82.41,C -159.41}

..
```



Kuva 16. Likimääräinen paikoitus (KUKA Roboter GmbH, 2006)

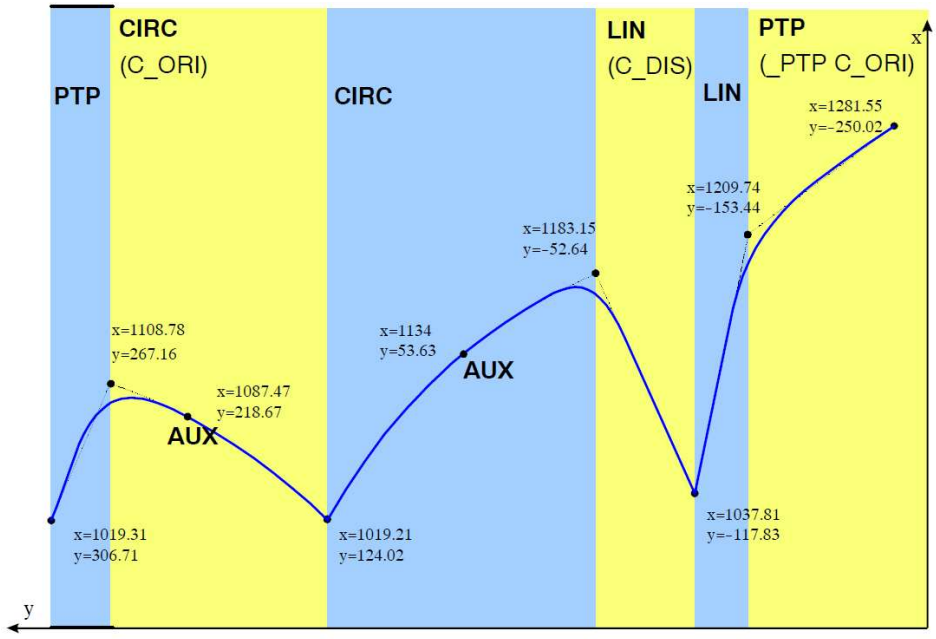
7.4.3 CIRC-CIRC ja CIRC-LIN

Järjestelmämuuttujat $APO_CDIS, $APO_CORI ja $APO_CVEL ovat käytettävissä myös, kun halutaan määritellä likimääräisen paikoituksen parametreja kaarevien liikkeiden välillä (CIRC–LIN) tai kaariliikkeen ja suoraviivaisen liikkeen välillä.

7.4.4 PTP-CP ja CP-PTP

Kun yhdistetään Point-to-point -liike (PTP) sekä Continous Path -liike (CP; LIN tai CIRC) on käytössä samat komennot. PTP-liikekomennon, jonka halutaan käyttävän likimääräistä paikoitusta, perään kirjoitetaan C_PTP. Jos seuraava liike CP-muotoinen, voidaan lopuksi lisätä myös sen komentona C_DIS, C_ORI tai C_VAL. Mikäli CP-liikkeen perään kirjoitetaan vain C_PTP-määre, käytetään PTP-liikettä seuraavan CP-liikkeen ohjauksessa oletuksena C_DIS määritystä.

```
;-- tarkka paikoitus
PTP {POS: X 1281.55,Y -250.02,Z 716,A 79.11,B
68.13,C 79.73,S
6,T 50}

;PTP-liikkeen likimääräinen paikoitus
PTP {POS: X 1209.74,Y -153.44,Z 716,A 79.11,B
68.13,C 79.73,S
6,T 50} C_PTP C_ORI

;LIN-liikkeen päätepisteen tarkka paikoitus
LIN {X 1037.81,Y -117.83,Z 716,A 79.11,B
68.13,C 79.73}

;-- likimääräisen paikoituksen etäisyyden
;-- uudelleenmäärittely
$APO.CDIS=25

;-- suoraviivaisen liikkeen
;-- etäisyysperusteinen
;-- likimääräinen paikoitus
LIN {X 1183.15,Y -52.64,Z 716,A 79.11,B 68.13,C
79.73} C_DIS

;-- kaaren tarkka määritys
CIRC {POS: X 1134,Y 53.63,Z 716},{X 1019.21,Y
124.02,Z 716,A
79.11,B 68.12,C 79.73}

;-- kaaren likimääräinen määritys
CIRC {POS: X 1087.47,Y 218.67,Z 716},{X
1108.78,Y 267.16,Z 716,A
79.11,B 68.12,C 79.73} C_ORI

;-- PTP liikkeen tarkka määritys
PTP {POS: X 1019.31,Y 306.71,Z 716,A 80.8,B
68,C 81.74,S 6,T
59}
```



Kuva 17. PTP-CP ja CP-PTP (KUKA Roboter GmbH, 2006)

8 TYÖKAPPALEET KOORDINAATISTOSSA

8.1 Työkoordinaatiston vaihto

Järjestelmämuuttuja $BASE pitää sisällään valitun työkoordinaatiston. KUKA Control Panelin KUKA.HMI:n avulla voidaan käyttöön määritellä yhteensä 16 erilaista työkoordinaatistoa. Etukäteen määritelty työkalukoordinaatisto voidaan ottaa käyttöön viittaamalla siihen muodossa BASE_DATA[xx], jossa xx on valitun työkoordinaatiston numero välillä 1-16.

Työkoordinaatisto voidaan määrittää myös ohjelman sisällä koordinaattien avulla. Käsiteltävien kappaleiden sijainteja voi silloin olla enemmän kuin KUKA.HMI:n avulla muistiin määriteltävien työkoordinaatistojen enimmäislukumäärä.

Työkoordinaatiston siirron jälkeen määritellyn pisteen paikka työkoordinaatistossa pysyy muuttumattomana (suhteessa työkoordinaatiston origoon), mutta pisteen sijainti maailmankoordinaatistossa muuttuu.

```
..
;--- pisteen MAALI määritys työkoordinaatistossa
FRAME MAALI={X 15,Y 15,Z 0,A 0,B 0,C 0}

;--- työkoordinaatiston valinta
$BASE=BASE_DATA[6]

PTP MAALI ;siirtyminen pisteeseen MAALI

;--- työkoordinaatiston siirto
$BASE.X = $BASE.X+15 ;siirto 15mm X-suuntaan
$BASE.Y = $BASE.Y+15 ;siirto 15mm Y-suuntaan
;-- määritellyn työkoordinaatiston
;-- BASE_DATA[6] arvo ei muutu

PTP MAALI ;siirtyminen pisteeseen MAALI

;--- työkoordinaatiston palautus alkuperäiseen
$BASE = BASE_DATA[6]

PTP MAALI
..
```

8.2 **Työkalun vaihto**

Järjestelmämuuttuja $TOOL pitää sisällään valitun työkalukoordinaatiston. KCP:n KUKA.HMI:n avulla voidaan käyttöön määritellä yhteensä 16 erilaista työkalukoordinaatistoa. Etukäteen määritelty työkalukoordinaatisto voidaan ottaa käyttöön viittaamalla siihen muodossa TOOL_DATA[xx], jossa xx on valitun työkalukoordinaatiston numero välillä 1-16.

```
..
;--- valitaan tyokoordinaatisto
$BASE = BASE_DATA[6]

;--- valitaan työkalukoordinaatisto
$TOOL=TOOL_DATA[14]

PTP xp1  ;ajo pisteeseen xp1
;-- paikoitus valitun työkalupisteen mukaan

;--- työkalukoordinaatiston vaihto
$TOOL=TOOL_DATA[15]

PTP xp1  ;ajo pisteeseen xp1
;-- paikoitus valitun työkalupisteen mukaan
..
```

9 OHJELMAN SUORITUKSEN OHJAUS

9.1 Lausetyypit

Robotin toiminnot kuvataan ohjelmassa lauseilla. Lauseita on erilaisia ja kirjassaan Maarit Harsu jakaa ne kolmeen ryhmään.

Primitiivisiä lauseita ovat sijoituslause, aliohjelman kutsu ja tyhjä lause. Lauseet voivat olla myös monimutkaisempia, jolloin usein puhutaan kontrollirakenteista. Kontrollirakenteita eli lauseita, jotka vaikuttavat kontrollin kulkuun, ovat esimerkiksi valintalause ja toistolause. Yhdeksi lausetyypiksi voidaan lisäksi laskea kompositio (eli koottu lause), joka sisältää listan peräkkäin suoritettavia lauseita.
(Harsu, 2005)

9.2 GOTO

GOTO on ohjelman suoritusta ohjaavista kontrollikäskyistä yksinkertaisempia. Käsky suoritetaan ilman ehtoja ja ohjelmassa siirrytään käskyn perässä määriteltyyn paikkaan ohjelmassa. Siirtymäpaikka on nimettävä ja nimeämisessä kannattaa käyttää jotain selkeää nimeä helpottamaan ohjelman myöhempää tarkastelua.

```
PTP xp1            ;ajo pisteeseen
PTP xp2            ;ajo pisteeseen

GOTO PALLETOINTI
PTP xp1            ;ei suoriteta, vaan ohitetaan
PALLETOINTI:       ;siirtymäpiste
..
```

9.3 IF

IF on ohjelman suoritusta ohjaava valintalause. Se on niin sanottu kaksisuuntainen valintalause, koska se ohjaa suoritusta kahteen vaihtoehtoiseen suuntaan. Käskyn rakenne on muotoa IF – THEN – ELSE. IF käskyä seuraa varsinainen ehto. KRL vaatii, että toteutuneen ehdon tapauksen haara on aloitetta sanalla THEN. Jos ehto ei toteudu, vaihtoehtoinen haara määritetään sanan ELSE perään.

```
INT A=5              ;muuttujan A arvo on 5

IF A>4 THEN          ;jos A on suurempi kuin 4
PTP xp1              ;ajo pisteeseen xp1
ELSE                 ;jos ehto ei toteudu
PTP xp2              ;ajo pisteeseen xp2
ENDIF
```

ELSE voidaan haluttaessa jättää kokonaan pois. Tällaisessa tapauksessa toteutumattoman ehdon perässä olevat käskyt ohitetaan ilman vaihtoehtoista suuntaa ja ohjelman suoritus jatkuu normaalisti. IF-lause päättyy aina sanaan ENDIF. IF-lauseita voidaan käyttää myös sisäkkäisinä.

```
INT A,B

IF A==0 THEN
   PTP xp1           ;jos A arvo on 0
ELSE
   IF A>B THEN
      PTP xp2        ;jos A on suurempi kuin B
   ENDIF
   PTP xp3           ;jos A on pienempi kuin B
ENDIF
```

9.4 **SWITCH**

Jos IF-muotoiset ehtolauseet ovat sisäkkäisiä, tarkastetaan sisempi ehto vain siinä tapauksessa, että ulompi ehto toteutuu. Jos taas IF-muotoiset ehtolauseet ovat rinnakkaisia (kirjoitettu peräkkäin), arvioidaan ne kaikki, ellei ehtolause itsessään muuta ohjelman suoritusta.

Kun halutaan ohjata ohjelman suoritusta useammalla vaihtoehdolla niin, että voimassa voi olla vain yksi ehto kerrallaan, on perusteltua käyttää SWITCH-monivalintalausetta. Käskyn perään kirjoitetaan valintaehto. Vaihtoehtoiset haarat kirjoitetaan CASE-sanan perään. CASE-lohkojen perässä voi olla DEFAULT-lohko joka suoritetaan, mikäli mikään CASE-ehdoista ei täyty. SWITCH-lause päätetään käskyllä ENDSWITCH.

```
; muuttuja AJOSUUNTA saa arvonsa
; sen mukaan kumpi (1 tai 2) tulosignaali
; on voimassa

SIGNAL AJOSUUNTA $IN[1] TO $IN[2]

SWITCH AJOSUUNTA
  CASE 'B01'                ;AJOSUUNTA on 1
     AJO_ETEEN()
  CASE 'B10'                ;AJOSUUNTA on 2
     AJO_TAAKSE()
  DEFAULT          ;signaalivirhe tai muu
     VIRHERUTIINI()
ENDSWITCH
```

KRL asettaa rajoituksia ehdossa käytettävän muuttujan tietotyypin suhteen. Ehtomuuttuja voi olla ainoastaan INT, CHAR tai ENUM -tyyppinen.

9.5 **FOR**

FOR on ns. määrätty toistolause. Määrätyssä toistolauseessa silmukka toistuu, kunnes ennalta määrätty lukumäärä toistojen suhteen on saavutettu. FOR-rakenne on kätevä esimerkiksi taulukkomuotoisen tiedon läpikäyntiin.

FOR-käskyä seuraa alku- ja loppuarvojen asetus. STEP käsky on vaihtoehtoinen ja sitä voidaan käyttää FOR-käskyn perässä, jos halutaan jättää välejä silmukkalaskurissa (esim. käsitellä vain joka toinen tai kolmas rivi taulukosta).

```
INT I
INT J
INT KERTOTAULU[10,10] ;10x10 kokoinen taulukko

FOR I=1 TO 10
   FOR J=1 TO 10
      KERTOTAULU[I,J] = J*I
   ENDFOR
ENDFOR
```

Toistolause luo 10x10 taulukkoon kertotaulun:

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50

6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

9.6 **WHILE JA REPEAT**

Määräämättömässä toistolauseessa lopetusehto voi sijaita silmukan alussa (WHILE), jolloin silmukka toistetaan nolla tai useampia kertoja. Jos lopetusehto sijaitsee lopussa (REPEAT), silmukka suoritetaan vähintään kerran. (Harsu, 2005).

```
SIGNAL NAPPI1 $IN[4] ;napin 1 painamisen arvo
SIGNAL NAPPI2 $IN[5] ;napin 2 painamisen arvo

;--- WHILE TOISTOLAUSE ------------------
WHILE NAPPI1==TRUE ;suoritetaan niin kauan
                   ;kuin nappi 1 on painettuna
  PTP xp1
  PTP xp2
ENDWHILE

;--- REPEAT TOISTOLAUSE -----------------

REPEAT
  PTP xp3
  PTP xp4
UNTIL NAPPI2==TRUE ;suoritetaan kunnes
                    ;nappia 2 on painettu
```

9.7 **LOOP**

Toistolauseke voidaan ohjelmoida myös loputtomaksi käyttäen LOOP-rakennetta. LOOP-muotoinen toistolause voidaan lopettaa EXIT komennolla, kun sen sisäinen ehto täyttyy.

```
PTP HOME
LOOP                ;silmukka toistetaan
   PTP xp1
   PTP xp2
   IF $IN[2]==TRUE THEN ;kunnes $IN[2] asetettu
      EXIT
   ENDIF
ENDLOOP
PTP HOME
```

9.8 **WAIT**

WAIT-käskyllä ohjelma määrätään odottamaan joko ennalta määrätty aika tai se määrätään odottamaan jonkun ehdon täyttymistä.

WAIT FOR -käskyn perässä oleva ehto voidaan lukea järjestelmämuuttujista $IN, $OUT, $CYCFLAG, $TIMER_FLAG ja $FLAG.

```
..
PTP HOME ;-- paikoitus alkuasemaan

WAIT SEC 2 ;-- odotetaan 2 sekunnin ajan

LIN xP1

;-- odotetaan kunnes ehto on tosi
WAIT FOR $CYCFLAG[1]==TRUE
PTP xP2

..
```

10 **TRIGGER**

10.1 **Komennon suoritus liikkeen aikana**

Kun halutaan esim. käynnistää tai sulkea työkalu tietyssä vaiheessa suoritettavaa liikerataa, voidaan käyttää TRIGGER-käskyä. Haluttu toiminto voidaan asettaa liikkeen alku- (DISTANCE=0) tai loppupisteen (DISTANCE=1) perusteella, tai se voidaan asettaa tietyn liikerataa pitkin kuljetun matkan (PATH) päässä olevan pisteen perusteella. Suoritettavaa käskyä voidaan joko aikaistaa tai viivyttää millisekunneissa mitaten, käyttämällä DELAY-tarkenninta. TRIGGER-lauseen loppuun määritellään suoritettavaksi haluttava toiminto. Toiminto voi olla aliohjelman kutsu, muuttujan arvon asetus tai OUTPUT-asetus. TRIGGER-lause sijoitetaan aina ennen liikekäskyä, johon se määräytyy.

10.2 **Asetus alku- tai loppupisteen perusteella**

```
..
;TRIGGER lause
;-- paikka määräytyy alkupisteen perusteella
;-- suoritetaan, kun ollaan alkupisteessä
;-- käskyn alkamista viivytetään 5 ms ajan
;-- käskynä asetetaan output 2 päälle

TRIGGER WHEN DISTANCE=0 DELAY=5 DO
$OUT[2]=TRUE

;-- liikekäsky pisteeseen P1
LIN P1

;-- paikka määräytyy loppupisteen perusteella
;-- suoritetaan kun loppupiste saavutettu
;-- käskyn alkamista aikaistetaan 10 ms

TRIGGER WHEN DISTANCE=1 DELAY=-10 DO
$OUT[3]=TRUE

;-- liikekäsky pisteeseen P2
PTP P2

..
```

10.3 **Asetus kuljetun matkan päässä olevan pisteen perusteella**

```
..
;-- suoritetaan kun ollaan kuljettu 15mm matka
;-- käskyä viivytetään 10 ms ajan

TRIGGER WHEN PATH=15 DELAY=10 DO $OUT[3]=TRUE

LIN P4

;-- suoritetaan kun ollaan kuljettu 40mm matka
;-- käskyä aikaistetaan 20 ms
;-- käskynä kutsutaan aliohjelmaa SP1

TRIGGER WHEN PATH=40 DELAY=-20 DO SP1()

PTP P5

..
```

11 INTERRUPT

11.1 Keskeytyksenhallinta

Ohjelmaa ajettaessa on usein tarpeen pystyä reagoimaan sen suorituksen aikana ilmeneviin tilanteisiin. Tilanteet voivat olla ennakoituja, kuten esim. odotettavissa oleva tila, tai ennakoimattomia, kuten virhetila joka aiheutuu robotin ylittäessä määritellyn työalueen rajat.

INTERRUPT-käsky tarjoaa mahdollisuuden ohjelman keskeytyksenhallintaan. Käskyn avulla voidaan määritellä tiloja, jotka aiheuttavat ohjelman keskeytyksen, sekä toimet, jotka ohjelman on määrä suorittaa keskeytyksen johdosta. Keskeytyksen aiheuttavat tilat arvioidaan suorituksesta erillään, joten ne ovat riippumattomia ohjelmalaskurin senhetkisestä sijainnista.

Keskeytyksille annetaan niitä määriteltäessä prioriteetti, joka käyttäjän määrittelemissä keskeytyksissä voi olla arvoltaan joko väliltä 1-39 tai 81-128. Keskeytykset käsitellään prioriteetin osoittamassa järjestyksessä.

Keskeytyksille määritellään ehto, jonka toteutuminen aiheuttaa keskeytyksen. Ehtolause Boolean-tyyppiä ja se voidaan määritellä joko BOOL-muuttujan, kokonaisen Boolean-lauseen, signaalimuuttujan tai vertailulauseen avulla.

INTERRUPT-käskyn keskeytysehdon perään kirjoitetaan sen aliohjelman nimi, jota kutsutaan keskeytysehdon ollessa tosi. Kun INTERRUPT-käskyn kutsuma aliohjelma on suoritettu loppuun, jatkuu ohjelman normaali suoritus (ellei aliohjelma ole määritelty esim. suoritettavan pääohjelman ajon lopettamiseen).

```
..
;-- määritellään keskeytys ---------------
;-- prioriteetti 5 -----------------------
;-- keskeytys aiheutuu signaalista 3 -----
;-- keskeytyksen aiheutuessa -------------
;-- kutsutaan aliohjelmaa SP1()

INTERRUPT DECL 5 WHEN $IN[3]==TRUE DO SP1()

;-- Kytketään päälle kaikki määritellyt -
;-- keskeytykset

INTERRUPT ON
..
```

11.2 **Järjestelmämuuttujat keskeytyksen yhteydessä**

Keskeytyksen aiheutuessa voi olla tarpeen tietää robotin asema. Järjestelmämuuttujat tarjoavat mahdollisuuden lukea ja tallentaa robotin asematietoja. Kaikki tässä yhteydessä esiteltävät järjestelmämuuttujat ovat E6POS-rakenteellisia tietotyyppejä. $POS_INT-järjestelmämuuttujaa lukuun ottamatta niitä voidaan käyttää myös muualla kuin INTERRUPT-käskyn kutsumissa aliohjelmissa.

```
..
;aliohjelma SP1

DEF SP1()

;FRAME tietotyyppinen muuttuja VIRHEPAIKKA
;on esitelty pääohjelman DATA LIST –tiedostossa
;ja on siksi myös aliohjelman käytettävissä

VIRHEPAIKKA.X = $POS_INT.X
VIRHEPAIKKA.Y = $POS_INT.Y
VIRHEPAIKKA.Z = $POS_INT.Z
VIRHEPAIKKA.A = $POS_INT.A
VIRHEPAIKKA.B = $POS_INT.B
VIRHEPAIKKA.C = $POS_INT.C

;-- E6POS tietotyyppiset muuttujat
;-- ALKUPAIKKA ja LOPPUPAIKKA
;-- ovat samoin esitelty
;-- pääohjelman DATA LIST -tiedostossa

;-- luetaan muistiin keskeytyneen liikkeen
;-- alkupiste

ALKUPAIKKA = $POS_BACK

;-- luetaan muistiin keskeytyneen liikkeen
;-- loppupiste

LOPPUPAIKKA = $POS_FOR


..
```

Järjestelmämuuttuja $_POS_INT sisältää tiedot robotin sijainnista keskeytyksen aiheutuessa.

Järjestelmämuuttuja $POS_ACT sisältää tiedot robotin nykyisestä sijainnista.

Järjestelmämuuttuja $POS_BACK sisältää tiedot liikkeen alkupisteen sijainnista.

Järjestelmämuuttuja $POS_FOR sisältää tiedot liikkeelle määritellyn loppupisteen sijainnista.

11.3 **Suoritettavan liikkeen keskeytys**

Mikäli keskeytyksen kutsumassa aliohjelmassa halutaan pysäyttää suoritettava liike, se tehdään BRAKE-käskyllä.

BRAKE käskyllä aliohjelmassa pysäytetty liike jatkuu uudelleen, kun palataan keskeytyksen kutsuneesta aliohjelmasta takaisin.

```
..

;-- BRAKE käskyn perään lisätty F
;-- pysäyttää liikkeen
;-- rajummin kuin pelkkä BRAKE

BRAKE F

..
```

11.4 **Keskeytysohjelmasta palaaminen ennenaikaisesti**

Mikäli keskeytysohjelmasta halutaan palata takaisin jo ennen kuin se on kokonaan suoritettu loppuun, käytetään käskyä RESUME. Käsky lopettaa keskeytyksen kutsuneen aliohjelman ja palauttaa ohjelman suorituksen ennen keskeytystä vallinneeseen tilaan.

```
;-- aliohjelma jota keskeytys kutsuu –

DEF KESKEYTYS()
;-- aliohjelman ulkopuolella määritellyn
;-- muuttujan I arvon kasvatus yhdellä

I=I+1

;robotin sijainti keskeytyksen aiheutuessa
POSITION[I] = $POS_INT

;-- jos luettuja sijainteja on jo 3
;-- pysäytetään liike
;-- ja palataan takaisin pääohjelmaan

IF I==3 THEN
   BRAKE
   RESUME
ENDIF

;-- aliohjelma jatkuu ---
..
```

12 INPUT/OUTPUT

12.1 $IN, $OUT, $OUT_C

KRC voi tunnistaa ja käsitellä yhteensä 1026 sisään tulevaa INPUT-signaalia ja 1024 uloslähtevää OUTPUT-signaalia. Valmiita liitäntöjä on seuraavasti:

INPUT 1-16 ($IN[1] .. $IN[16])
OUTPUT 1-16 (max. 100mA), ($OUT[1] .. $OUT[16])
OUTPUT 17-20 (max. 2A), ($OUT[17] .. $OUT[20])

Järjestelmämuuttujien $IN[x] ja $OUT[x] arvoja voidaan lukea ja kirjoittaa.

$OUT-järjestelmämuuttujan kutsuminen katkaisee Advance Run –toiminnon. Jos halutaan käyttää $OUTPUT järjestelmämuuttujaa signaalin antamiseen esim. loppuun viedyn liikekäskyn jälkeen, voidaan käyttää $OUT_C-järjestelmämuuttujaa $OUT_C[x]. $OUT_C-järjestelmämuuttujan kutsuminen ei katkaise Advance Run -toimintoa, mutta $OUT_C-järjestelmämuuttujalle voidaan ainoastaan kirjoittaa arvo, ei lukea sitä.

Yhteensä 8 samanaikaista $OUT_C-järjestelmämuuttujaa voi olla käytössä ja niillä voidaan viitata mihin tahansa käsiteltävissä olevaan 1024 OUTPUT -signaaliin.

$OUT_C-järjestelmämuuttujan arvo voi olla mikä tahansa Boolean-lause, joten pelkän käsin asetetun TRUE- tai FALSE-arvon sijaan se voi saada arvonsa myös kokonaisen Boolean-lauseen tuloksena.

12.2 **Digitaaliset signaalit**

Tulo- ja lähtösignaalille voidaan määritellä nimi. Mikäli signaalille määritellään vain yksi tulo/lähtö, signaalia käsitellään bittimuotoisena eli se on joko päällä tai pois.

Useampi tulo/lähtö voidaan liittää ryhmäksi, jonka avulla voidaan käsitellä digitaalisia signaaleja. Ryhmittämällä 2 tuloa ryhmäksi, voidaan digitaalisena signaalina vastaanottaa arvoja 0,1,2 ja 3 (binääriluvut 00, 01, 10 ja 11).

```
SIGNAL VINKKARI_VASEN = $OUT_C[1]
SIGNAL VINKKARI_OIKEA = $OUT_C[2]
SIGNAL VIRHEVALO = $OUT[3]
SIGNAL NAPPIOHJAUS = $IN[1] TO $IN[2]
BOOL VIRHE

VIRHE = FALSE

VINKKARI_VASEN = FALSE
VINKKARI_OIKEA = FALSE
VIRHEVALO = FALSE

$BASE = BASE_DATA[1]
$TOOL = TOOL_DATA[1]

PTP HOME

$VASEN

WHILE VIRHE==FALSE
SWITCH NAPPIOHJAUS
   CASE 'B01' ;jos painettu nappia 1
      VINKKARI_OIKEA = TRUE
      VINKKARI_VASEN = FALSE
      WAIT SEC 1
      LIN_REL{X 100}
   CASE 'B10' ;jos painettu nappia 2
      VINKKARI_VASEN = TRUE
      VINKKARI_OIKEA = FALSE
      WAIT SEC 1
      LIN_REL{X -100}
   CASE 'B11' ;molempien nappien painallus
              ;keskeyttää ohjelman
      VIRHEVALO = TRUE
      VIRHE=TRUE
ENDSWITCH
ENDWHILE
```

12.3 **PULSE**

Ulostulosignaali voidaan halutessa asettaa päälle määrätyksi ajaksi.

```
SIGNAL MERKKI_1 $OUT[1]
SIGNAL MERKKI_2 $OUT[2]

MERKKI_1 = FALSE
MERKKI_2 = FALSE

;-- MERKKI_1 päälle 2,5 sekunnin ajaksi
PULSE(MERKKI_1,TRUE,2.5)

WAIT SEC 2

;-- MERKKI_2 päälle 1,2 sekunnin ajaksi
PULSE(MERKKI_2,TRUE,1.2)
```

13 **ALIOHJELMAT JA FUNTIOT**

13.1 **Aliohjelmat ja funktiot ohjelmakoodissa**

Kun ohjelma sisältää osia, joita toistetaan useaan otteeseen, on perusteltua sijoittaa ne omiksi aliohjelmiksi tai funktioiksi. Aliohjelmia ja funktioita voidaan myöhemmin kutsua suoritettaviksi tarpeen mukaan. Aliohjelmien ja funktioiden käyttö myös nopeuttaa uuden ohjelman kirjoittamista, mikäli aikaisemmin luotuja toimintoja voidaan hyödyntää uuden ohjelman luomisessa.

Aliohjelma on kuin mikä tahansa muu suoritettava ohjelma. Kun ohjelma kutsuu toista ohjelma, kutsuttavasta ohjelmasta tulee silloin kutsuvan ohjelman aliohjelma. Myös aliohjelma voi kutsua toisia ohjelmia, jolloin niistä tulee sen aliohjelmia.

```
;-- pääohjelma PROG1 ---------

DEF PROG1()
  ;-- pääohjelma kutsuu aliohjelmaa 1---

  SUB_PROG(1)
  ..

  ;-- pääohjelma kutsuu aliohjelmaa 2 ---

  SUB_PROG(2)
  ..
END

;-- paikallinen aliohjelma SUB_PROG1 ----

DEF SUB_PROG1()
  ..
  ;-- aliohjelma 1 kutsuu -----
  ;-- aliohjelmaa 2 ----

SUB_PROG2()
  ..
END

;-- paikallinen aliohjelma SUB_PROG2 ----

DEF SUB_PROG2()
  ..
END
```

Funktiot eroavat aliohjelmista siinä, että ne palauttavat suorituksensa jälkeen arvon sitä kutsuneelle ohjelmalle. Muuttujatyyppi, jonka funktio palauttaa sitä kutsuneelle ohjelmalle määritellään funktion alkurivillä.

```
DEFFCT INT OMAFUNKTIO()
  DECL INT PALAUTUS_LUKU

  PALAUTUS_LUKU = 5

  RETURN PALAUTUSLUKU
ENDFCT
```

13.2 **Aliohjelmien ja funktioiden sijoittuminen**

Aliohjelmat ja funktiot määritellään alkamaan sanalla DEF (aliohjelma) tai DEFFCT ja ne päätetään vastaavasti sanalla END tai ENDFCT. Aliohjelmat ja funktiot voivat sijaita samassa .SRC-tiedostossa pääohjelman kanssa, jolloin ne sijoitetaan pääohjelman perään omina lohkoinaan. Kutsuttava

aliohjelma voi sijaita myös toisessa .SCR-tiedostossa. Tällöin se on määriteltävä käyttöön sitä kutsuvan pää- tai aliohjelman alussa sanalla EXT.

Jos jokin aliohjelma tai funktio halutaan luoda sellaiseksi, että se on myös muiden ohjelmien käytettävissä, tulee niiden sijaita omassa erillisessä .SRC-tiedostossa. Vaihtoehtoisesti, jos muiden ohjelmien käyttöön jaettavaksi haluttu aliohjelma tai funktio sijaitsee alkuperäisen pääohjelman kanssa samassa .SRC-tiedostossa, voidaan sen nimen eteen kirjoittaa sana GLOBAL. Tällä tavoin nimetty aliohjelma ja funktio voidaan kutsua toisen ohjelman toimesta käyttöön kutsumalla sitä ohjelman alussa sanalla EXT.

13.3 **Muuttujat aliohjelmissa ja funktioissa**

Usein muuttujien arvoa joudutaan käsittelemään sekä pääohjelmassa, kutsutuissa aliohjelmissa ja mahdollisesti myös funktioissa. Tämän vuoksi on tärkeää ymmärtää muuttujien näkyvyysrajoitukset liikuttaessa ohjelmien ja funktioiden välillä.

Muuttujat, jotka on esitelty pääohjelman yhteydessä .SRC-tiedostossa, eivät ole aliohjelmien ja funktioiden käytettävissä.

Muuttujat, jotka on esitelty pääohjelman erillisessä Data List (.DAT) -tiedostossa, ovat paikallisten (LOCAL) aliohjelmien ja funktioiden käytettävissä. Sen sijaan, jos käyttöön on kutsuttu myös ulkopuolisia (GLOBAL) aliohjelmia tai funktioita, muuttujat eivät sellaisenaan ole niiden käytettävissä.

Aliohjelmissa ja funktioissa määritellyt muuttujat eivät ole pääohjelman käytettävissä.

13.4 **Parametrilistat muuttujien käsittelyn apuna**

Muuttujia voidaan käsitellä ohjelmien ja funktioiden välillä riippumatta niiden sijoittelusta, kun käytetään apuna ns. parametrilistoja. Käsiteltävät muuttujat voidaan välittää aliohjelma- tai funktiokutsun yhteydessä joko arvoina tai viitteinä. Niiden käsittely ja vaikutus poikkeavat toisistaan, joten on tärkeää ymmärtää näiden kahden tavan erot.

```
DEF LASKURI
INT X
INT Y
INT Z

X=5 ;muuttujan X arvo on 5

;-- muuttuja Y saa arvonsa funktiolta NELIO_A

Y=NELIO_A(X)
;-- Y = 25 ---
;-- X = 25 ---

;-- muuttuja Z saa arvonsa funktiolta NELIO_B

Z=NELIO_B(X)
;-- Z = 625 ---
;-- X = 25 ---

END

DEFFCT INT NELIO_A(LUKU:OUT)
  ;-- LUKU viittaus X:n arvoon, X=5 --
  INT LUKU
  ;-- LUKU toiseen potenssiin korotettuna --
  LUKU=LUKU*LUKU
  ;-- palautetaan LUKU, jonka arvo on nyt 25
  ;-- viittauksen johdosta myös X:n arvo on 25
  RETURN LUKU
END

DEFFCT INT NELIO_B(LUKU:IN)
  ;-- LUKU on X:n arvo eli 25 ---
  INT LUKU
  ;-- LUKU toiseen potenssiin korotettuna --
  LUKU=LUKU*LUKU
  ;-- palautetaan LUKU, jonka arvo on nyt 625
  ;-- välitetyn X:n arvo pysyy samana
  RETURN LUKU
END
```

14 ESIMERKKEJÄ, CASEJA

14.1 Advance Run ja likimääräinen paikoitus robotin liikkeissä

Määritellään pisteitä työkoordinaatistoon ja ohjelmoidaan robotti kulkemaan reitti kaikkien pisteiden kautta. Liikekäskyt toteutetaan sekä tarkkaa paikoitusta että likimääräistä paikoitusta käyttäen.

Määritetään työkoordinaatistoon aloituspiste sekä kuusi muuta pistettä, joiden kautta robotin liikerata halutaan kulkevan.

```
..
FRAME ALOITUS
FRAME A
FRAME B
FRAME C
FRAME D
FRAME E
FRAME F

..

ALOITUS={X 25,Y 25,Z 27}
A={X 15,Y 270}
B={X 70,Y 350}
C={X 100,Y 290}
D={X 150,Y 360}
E={X 200,Y 100}
F={X 250,Y 100}
..
```

Robotin liikeradan ohjelmointiin käytetään LIN-liikekäskyä. Robotti etenee suoraviivaisesti pisteestä toiseen.

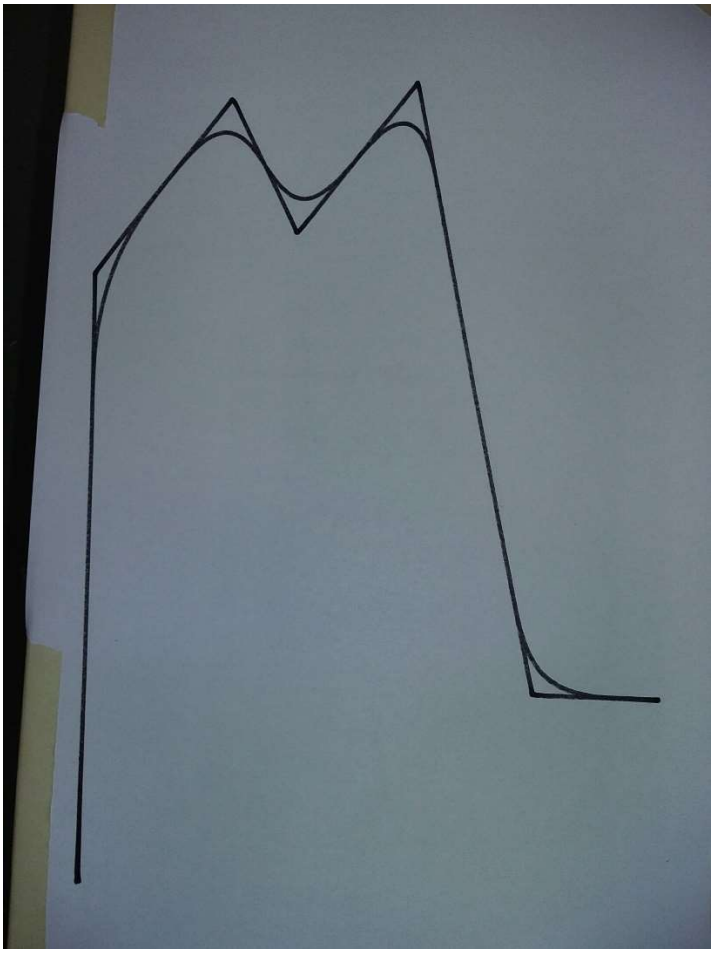
```
..
LIN_REL{z -30}

LIN A
LIN B
LIN C
LIN D
LIN E
LIN F

LIN_REL{z 30}
..
```

Toisen liikeradan ohjelmointiin käytetään myös LIN-liikekäskyä. Ennen liikekäskyä määritetään käyttöön Advance Run -toimintoa hyödyntävä likimääräinen paikoitus asettamalla $APO.DIS järjestelmämuuttujan arvoksi 40.

Likimääräinen paikoitus LIN-liikekäskyn aikana määritetään käyttöön lisäämällä liikekäskyn perään C_DIS.

```
..

$APO.CDIS=40

LIN_REL{z -30}

; -- pisteet A-E likimääräisesti
; -- paikoitettuja
LIN A C_DIS
LIN B C_DIS
LIN C C_DIS
LIN D C_DIS
LIN E C_DIS

;-- piste F tarkasti paikoitettu
LIN F

LIN_REL{z 30}

..
```

Ohjelmaa ajettaessa robotti kulkee ensimmäisen liikeradan tarkasti pisteisiin paikoittuen. Toisen liikeradan robotti kulkee paikoittuen tarkasti ainoastaan lähtö- ja päätepisteeseen (Kuva 18).



Kuva 18. Ohjelman kulkemat kaksi liikerataa

Ohjelmalistaus kokonaisuudessaan löytyy liitteestä 1.

14.2 **Kuvion piirto eri kappaleisiin**

Halutaan piirtää sama ympyräkuvio eri työkappaleisiin. Koska kuvio toistuu muuttumattomana kaikissa työkappaleissa, voidaan kuvion piirtäminen sijoittaa toistettavan silmukan sisään. Siirtyminen työkappaleesta toiseen toteutetaan $BASE-järjestelmämuuttujan arvoa muuttamalla.

Paikoituspiste kuvion piirrolle annetta FRAME-tyyppisessä muuttujassa ALOITUS. Työkappaleiden sijainnit voitaisiin myös antaa FRAME-muotoisessa muuttujassa, mutta esimerkin vuoksi tässä tapauksessa työkappaleiden sijaintia varten luodaan oma STRUC-muoto KAPPALETYPE. Sen rakenne on samanlainen kuin FRAME-muuttujan, eli sisältää koordinaatit reaalilukumuodossa.

Koordinaatit annettaan KAPPALETYPE-tyypin vaatimassa muodossa taulukkomuuttujaan SIJAINTI.

```
; -- määritellään FRAME muotoinen muuttuja
; -- ALOITUS
DECL FRAME ALOITUS

; -- määritellään uusi muuttujatyyppi
; -- KAPPALETYPE
STRUC KAPPALETYPE REAL X, REAL Y, REAL Z, REAL
A, REAL B, REAL C

; -- määritellaan taulukkomuuttuja SIJAINTI,
; -- joka on tyyppiä KAPPALETYPE
DECL KAPPALETYPE SIJAINTI[3]

SIJAINTI[1]={X 0,Y 0,Z 0,A 0,B 0,C 0}
SIJAINTI[2]={X 140,Y -350,Z 0,A 31,B 0,C 0}
SIJAINTI[3]={X 140,Y -350,Z 0,A -59,B 0,C 0}

; -- aloituspisteen koordinaatit
ALOITUS={x 10,y 10,z 28}
```

Muodostetaan silmukkarakenne, joka käy läpi kaikki kolme määriteltyä kappaleen sijaintia. Työkappaleen vaihto toteutetaan työkoordinaatistoa siirtämällä, joten kuvion piirtäminen voidaan suorittaa samoin paikoituskäskyin kaikissa kappaleissa. On huomattava, että koska työkoordinaatiston vaihto tehdään $BASE-järjestelmämuuttujan arvoa muuttamalla, on sen arvo alustettava uudelleen jokaisen kappaleen kohdalla silmukan alussa. Jokaisen kappaleen sijainti on määritelty suhteessa alkuperäiseen BASE_DATA[6] työkoordinaatistoon.

```
; -- käsitellään kaikki SIJAINTI-taulukkoon
; -- määritellyt kappaleet
FOR I=1 TO 3
    ;-- asetetaan vertailukohta
    $BASE = BASE_DATA[6]

    ;-- asetetaan kappaleen sijainti
    $BASE.X = $BASE.X+SIJAINTI[I].X
    $BASE.Y = $BASE.Y+SIJAINTI[I].Y
    $BASE.Z = $BASE.Z+SIJAINTI[I].Z
    $BASE.A = $BASE.A+SIJAINTI[I].A
    $BASE.B = $BASE.B+SIJAINTI[I].B
    $BASE.C = $BASE.C+SIJAINTI[I].C

   ;-- siirytään aloituspisteeseen
   ;-- työkappaleessa
   PTP ALOITUS

   ;-- piirretään ympyräkuvio
   LIN_REL{x 0,y 0,z -31}
   CIRC_REL{x 10,y -10},{x 10,y 10},CA 360
   LIN_REL{x 0,y 0,z 31}

ENDFOR
```

Ohjelmalistaus kokonaisuudessaan löytyy liitteestä 2.

14.3 **Kuvioryhmä eri kappaleissa**

Kuvioryhmä koostuu useasta toistettavasta samanlaisesta kuviosta. Kuviot halutaan toistaa eri kappaleissa siten, että kappaleiden kohdalla voidaan erikseen määritellä, kuinka monta riviä ja kuinka monta kuviota toistetaan.

Laajennetaan jos aikaisemmassa ohjelmassa käytössä ollutta STRUC-rakenteista muuttujatyyppiä. Lisätään rakenteiseen tyyppiin kaksi muuttujaa parametreina.

```
;--- maaritellaan uusi muuttujatyyppi
;--- KAPPALETYPE
STRUC KAPPALETYPE REAL X, REAL Y, REAL Z, REAL
A, REAL B, REAL C, INT ROWS, INT COLS

DECL KAPPALETYPE SIJAINTI[3]
```

Luodulle taulukkomuuttujalle SIJAINTI voidaan nyt antaa kolmen eri työkappaleen sijainnit. Lisäksi voidaan erikseen määritellä, kuinka monta kuviota, ja riviä halutaan piirtää.

```
; -- kappale 1 -------------
; -- 1 rivi, 3 kuviota
SIJAINTI[1]={X 0,Y 0,Z 0,A 0,B 0,C 0,ROWS
1,COLS 3}

; -- kappale 2
; -- 2 rivia, 4 kuviota
SIJAINTI[2]={X 140,Y -350,Z 0,A 31,B 0,C 0,ROWS
2,COLS 4}

; -- kappale 3
; -- 4 rivia, 2 kuviota

SIJAINTI[3]={X 140,Y -350,Z 0,A -59,B 0,C
0,ROWS 4,COLS 2}
```

Käsitellään kaikki taulukkomuuttujan SIJAINTI tiedot ja välitetään ne aliohjelmalle, joka huolehtii kuvioiden piirtämisestä. Aliohjelmaa kutsuttaessa sille välitetään arvoina myös aloituskohdan, sekä työkappalekohtaisesti määritellyt tiedot piirrettävien kuvioiden määrästä.

```
FOR I=1 TO 3
    $BASE = BASE_DATA[6]
    ;-- asetetaan kappaleen sijainti
    $BASE.X = $BASE.X+SIJAINTI[I].X
    $BASE.Y = $BASE.Y+SIJAINTI[I].Y
    $BASE.Z = $BASE.Z+SIJAINTI[I].Z
    $BASE.A = $BASE.A+SIJAINTI[I].A
    $BASE.B = $BASE.B+SIJAINTI[I].B
    $BASE.C = $BASE.C+SIJAINTI[I].C

    ;--- luetaan muistiin piirrettavien rivien
    ;--- ja kuvioiden maara
    ROWS=SIJAINTI[I].ROWS
    COLS=SIJAINTI[I].COLS

    ;---- kutsutaan aliohjelmaa---
    KUVIOPIIRTO(ALOITUS,ROWS,COLS)
ENDFOR
```

Muodostetaan kuvion piirtävistä liikekäskyistä oma aliohjelma, jota kutsutaan. Aliohjelma luodaan samaan tiedostoon pääohjelman kanssa.

```
DEF KUVIOPIIRTO(ALOITUS :IN,ROWS :IN,COLS :IN)

;muuttujien deklaraatiot aliohjelmassa
DECL FRAME ALOITUS

INT I
INT J
INT ROWS
INT COLS

..
END
```

Aliohjelmaan määritellään silmukkarakenne, joka toistaa halutun määrän kuvioita ja rivejä.

```
;-- toistetaan rivien lukumäärän mukaisesti
FOR J=1 TO ROWS

    PTP ALOITUS ;-- aloituspaikoitus

    ; -- toistetaan kuvioiden lukumäärän
    ; -- mukaisesti
    FOR I=1 TO COLS
    LIN_REL{x 0,y 0,z -31}
    CIRC_REL{x 10,y -10},{x 10,y 10},CA 360
    LIN_REL{x 0,y 30,z 31}
    ENDFOR

    ;-- jokainen rivi alkaa
    ;-- 30mm päästä edellisestä
   ALOITUS.X=ALOITUS.X+30

ENDFOR
```

Ohjelmalistaus kokonaisuudessaan löytyy liitteestä 3.

14.4 **Työkappaleiden siirto ja pinoaminen**

Halutaan luoda ohjelma, jossa robotti siirtää paineilmatarttujan avulla pinon kappaleita paikasta A paikkaan B. Siirrettävien kappaleiden lukumäärän ja koon halutaan olevan muunneltavissa. Samoin nouto- ja luovutuspaikan halutaan olevan muunneltavissa.

Määritellään taulukkomuuttujaan nouto- ja luovutuspisteiden koordinaatit. Nouto- ja luovutuspisteen paikka määritellään siten, että itse piste säilyy suhteessa työkoordinaatiston origoon aina samana. Siirtyminen nouto- ja luovutuspaikan välillä tapahtuu työkoordinaatistoa vaihtamalla.

Lisäksi määritellään omat muuttujat siirreltävien kappaleiden lukumäärän sekä kappaleen paksuuden antamista varten. Myös noudettujen ja luovutettujen kappaleiden lukumäärän seuraamiseksi määritellään omat muuttujat.

```
..

;--- maaritellaan uusi muuttujatyyppi
KAPPALETYPE
STRUC KAPPALETYPE REAL X, REAL Y, REAL Z, REAL
A, REAL B, REAL C

;--- maaritellaan taulukkomuuttuja SIJAINTI,
;--- joka on tyyppia KAPPALETYPE
DECL KAPPALETYPE SIJAINTI[2]

INT KAPPALE_LKM
INT KAPPALE_PAKSUUS
INT KAPPALE_NOUTO
INT KAPPALE_LUOVUTUS
INT I

..

ALOITUS={x 150,y 150,z 100}
; -- aloituspiste työkoordinaatiston
; -- origosta mitattuna

; -- työstettävien kappaleiden sijainti
; -- työkoordinaatistossa ---
; -- sijannit annetaan STRUC muuttujan
; -- KAPPALETYPE määräämässä muodossa
; -- muuttujalle SIJAINTI

SIJAINTI[1]={X 0,Y 0,Z 0,A 0,B 0,C 0}
SIJAINTI[2]={X 50,Y -500,Z 0,A 0,B 0,C 0}

; -- liikuteltavien kappaleiden
; -- lukumäärä ja paksuus

KAPPALE_LKM = 5
KAPPALE_PAKSUUS = 10

..
```

Siirrettävien kappaleiden läpikäynti tapahtuu silmukassa.

```
..
; -- nouto- ja luovutuslaskurin alustus

KAPPALE_NOUTO=KAPPALE_LKM
KAPPALE_LUOVUTUS=0

; -- kappaleet käsittelevä silmukka

FOR I=1 TO KAPPALE_LKM

..

ENDFOR
..
```

Koska kyseessä on pinottavat kappaleet, on noutokorkeuden muututtava sen mukaan, kuinka monta kappaletta on noudettavassa pinossa. Samoin luovutuskorkeuden on muututtava vastaavasti. Kappaleiden käsittely ja siirtäminen sijoitetaan silmukkarakenteen sisään.

```
..
; -- noutokorkeuden säätö
$BASE.Z =
$BASE.Z+KAPPALE_NOUTO*KAPPALE_PAKSUUS

;ajo noutopisteeseen
PTP ALOITUS

; -- tarttujan lasku alas
LIN_REL{z -100}

; -- tarttujan imu päälle
$OUT[10] = TRUE

; -- tarttujan nosto ylös
LIN_REL{z 100}

; -- noutolaskurin arvon muutos
KAPPALE_NOUTO=KAPPALE_NOUTO-1

; -- työkoordinaatiston vaihto
$BASE = BASE_DATA[6]
$BASE.X = $BASE.X+SIJAINTI[2].X
$BASE.Y = $BASE.Y+SIJAINTI[2].Y

; -- luovutuskorkeuden säätö
$BASE.Z =
$BASE.Z+10+KAPPALE_LUOVUTUS*KAPPALE_PAKSUUS

; -- ajo luovutuspaikkaan
PTP ALOITUS

LIN_REL{z -100}

; -- luovutuslaskurin arvon muutos
KAPPALE_LUOVUTUS = KAPPALE_LUOVUTUS+1

;tarrain pois
$OUT[10] = FALSE

WAIT SEC 0.5

LIN_REL{z 100}

;paluu alkupisteeseen
$BASE = BASE_DATA[6]
..
```

Ohjelman silmukka käsitellään niin monta kertaa kuin kappaleiden lukumäärän sisältävä muuttuja määrittelee. Kappaleet noudetaan yksitellen pinosta ja ne pinotaan luovutuspaikalle.

Ohjelmalistaus kokonaisuudessaan löytyy liitteestä 4.

14.5 **Kappaleiden liikuttelu robotin avulla (kiinteä työkalu)**

Luodaan ohjelma, jossa robotti ohjaa ensin robotin varteen kiinnitettyä työkalua, esim. paineilmatarttuja. Tämän jälkeen robotti ohjaa robotin varteen kiinnitettyä työkappaletta (kappale paineilmatarttujassa) ulkoista kiinteää työkalua (esim. pora) vasten.

Määritellään käyttöön kiinteästi paikalleen asetettu paineilmapora uudeksi työkaluksi. KRC käsittelee kaikkia robotin ulkopuolella sijaitsevia kappaleita työkoordinaatistoina (BASE), huolimatta siitä, onko kyseessä työkappale vai työkalu.

Kun uusi työkalupiste on määritelty ja kalibroitu se robotin päässä olevan työkappaleen kanssa, voidaan työkalua vaihtaa kesken ohjelman suorituksen.
Muokataan edellisen esimerkin ohjelmalistausta siten, että kappaleen noudon ja luovutuksen välillä kuljetetaan kerätty kappale paineilmaporan luo ja porataan jokaiseen kappaleeseen neljä reikää.

Muodostetaan oma aliohjelma, joka huolehtii kappaleen kuljetuksesta ja paikoituksesta paineilmaporan luo. Koska kappaletta porattaessa robotti ohjaa kappaletta eikä työkalua, vaihdetaan järjestelmämuuttujan $IPO_MODE arvoksi #TCP. Tällä käskyllä KRC interpoloi liikkeet ja ohjelmoidessa voidaan paikoittaa työkalu aivan kuin sitä liikuteltaisiin paikallaan olevaa kappaletta varten.

```
DEF KULJETUS()

;-- esitellään neljä FRAME-muuttujaa

FRAME REIKA1
FRAME REIKA2
FRAME REIKA3
FRAME REIKA4

;-- nostetaan ylöspäin 200mm matka
LIN_REL{Z 200}

;-- AJO PORAUKSEEN ------------------
;-- vaihdetaan työkaluksi paineilmapora
$BASE = BASE_DATA[14]

;-- vaihdetaan työkoordinaatistoksi
;-- tarttujan päässä oleva työkappale
$TOOL = TOOL_DATA[8]

;-- ROBOTTI OHJAA TYOKAPPALETTA
$IPO_MODE=#TCP

;-- PORAN PAIKOITUS KAPPALEESEEN
PTP {X 150,Y 150,Z -50}

;-- Määritetään 4 porattavan
;-- reiän koordinaatit
;-- $POS_ACT = nykyinen sijainti

REIKA1.X=$POS_ACT.X+20
REIKA1.Y=$POS_ACT.Y
REIKA2.X=$POS_ACT.X
REIKA2.Y=$POS_ACT.Y-20
REIKA3.X=$POS_ACT.X-20
REIKA3.Y=$POS_ACT.Y
REIKA4.X=$POS_ACT.X
REIKA4.Y=$POS_ACT.Y+20

;-- paikoitetaan pisteisiin ja porataan
LIN REIKA1
PORAUS()
LIN REIKA2
PORAUS()
LIN REIKA3
PORAUS()
LIN REIKA4
PORAUS()
END
```

Kappaleen kuljetuksesta ja paikoituksesta huolehtivan aliohjelman sisällä kutsutaan vielä toista erillistä aliohjelmaa PORAUS. Sen tehtävänä on huolehtia varsinaisen porausliikkeen suorittamisesta. Porausliike halutaan suorittaa hieman pienemmällä nopeudella, joten liikenopeus säädetään alemmalle tasolle $VEL.CP-järjestelmämuuttujan arvoa muuttamalla. Paineilmaporaa ohjataan digitaaliseen Output 2 -liitäntään liitetyn ohjausventtiilin avulla.

```
DEF PORAUS()

;-- MUUTETAAN LIIKENOPEUS PORAUKSEN AIKANA

$VEL.CP=0.02

;-- PORA KAYNTIIN

$OUT[2]=TRUE

;-- LIIKE ALAS

LIN_REL {Z 20}

WAIT SEC 0.5
;-- LIIKE YLOS

LIN_REL {Z -20}

;-- PORA POIS PAALTA

$OUT[2]=FALSE

;-- LIIKENOPEUDEN PALAUTUS

$VEL.CP=0.5

END
```

Ohjelmalistaus kokonaisuudessaan löytyy liitteestä 5.

15 YHTEENVETO JA AJATUKSIA

Opinnäytetyön tavoitteena oli perehtyä KUKA-robotin ohjelmointiin KRL-kieltä käyttäen, laajentaa nykyisessä opetuskäytössä olevaa käskyvalikoimaa sekä tuottaa näitä hyödyntäen muutamia malliohjelmia case-esimerkkeinä käytettäväksi myöhemmässä opetuskäytössä.

Opinnäytetyön ensimmäinen osa oli tutustuminen KUKA robotin järjestelmäohjelmiston käyttöön sekä ohjelmoinnissa käytettävän KRL-kielen syntaksiin ja rakenteeseen. Opinnäytetyössä käsitellyt esimerkit pyrittiin, jos mahdollista, valitsemaan siten, että ne tukivat myöhemmissä case-esimerkeissä esiteltyjä rakenteita.

Opinnäytetyön jälkimmäinen osa oli robotin ääressä tapahtunut ohjelmointityö sekä tuotettujen ohjelmarakenteiden dokumentointi. KUKA-robotin sijainti konetekniikan laboratorion lukitussa tilassa rajoitti hieman pääsyä työskentelemään robotin äärellä. Tästä huolimatta onnistuin töiden jakamisen avulla työskentelemään intensiivisemmin robotin äärellä silloin kun siihen tarjoutui tilaisuus.

Opinnäytetyön tuloksena syntyi melko kattava kuvaus KUKA-robotin ohjelmointiin käytettävän KRL-kielen ominaisuuksista sekä viisi erilaista case-esimerkkiä robotin käytöstä ja ohjauksesta. Opinnäytetyön tekijänä koin antoisimpana juuri mahdollisuuden työskennellä robotin äärellä ohjelmoiden.

Oman haasteensa opinnäytetyön toteutukselle toi melko kireä aikataulutus. Opinnäytetyön aloituksen ja valmiin työn toimituksen väliin jäi ainoastaan reilu kuukausi aikaa. Onnistunut ja riittävän tarkka opinnäytetyön määrittely sen suunnitteluvaiheessa auttoi kuitenkin osaltaan fokuksen pitämisessä ja varmisti, että lopputulos vastasi myös tilaajan toiveita.

LÄHTEET

Braumann, J.;& Brell-Cokcan, S. (20. 4 2017). *Parametric Robot Control, Integrated CAD/CAM for Architectural Design.* Noudettu osoitteesta http://www.robotsinarchitecture.org/wp-content/uploads/2012/04/ACADIA2011_robarch.pdf

Harsu, M. (2005). Ohjelmointikielet. Periaatteet, käsitteet, valintaperusteet. Talentum.

KUKA Roboter GmbH. (2005). System Variables. *KUKA System Software (KSS)*.

KUKA Roboter GmbH. (2006). Expert Programming. *KUKA System Software (KSS) Release 5.2*.

KUKA Roboter GmbH. (2006). KR C2 / KR C3 Start-up. *KUKA System Software (KSS) Relase 5.2*.

KUKA Roboter GmbH. (2006). Operator Control. *KUKA System Software (KSS) Release 5.2*.

Mühe, H.;Angerer, A.;Hoffmann, A.;& Reif, W. (26. 4 2017). *On reverse-engineering the KUKA Robot Language.* Noudettu osoitteesta https://arxiv.org/abs/1009.5004

NASA Glenn Research Center. (8. 5 2017). *Aircraft Rotations, Body Axes.* Noudettu osoitteesta NASA: https://www.grc.nasa.gov/WWW/K-12/airplane/rotations.html

Liite 1

LIKIMÄÄRÄINEN PAIKOITUS

```
DEF viivakoe( )

; -- FRAME muuttujien esittelyt
FRAME ALOITUS
FRAME A
FRAME B
FRAME C
FRAME D
FRAME E
FRAME F

INI

; -- määritellään pisteet

ALOITUS={X 25,Y 25,Z 27}
A={X 15,Y 270}
B={X 70,Y 350}
C={X 100,Y 290}
D={X 150,Y 360}
E={X 200,Y 100}
F={X 250,Y 100}

PTP HOME ;-- BCO run

;-- työkalun ja työkoordinaatiston määrittely

$BASE=BASE_DATA[6]
$TOOL=TOOL_DATA[16]

;-- tarkka paikoitus aloituspisteeseen

PTP ALOITUS

;-- työkalu suoraan alas

LIN_REL{z -30}

; -- liikesarja määriteltyjen
; -- pisteiden kautta kulkien

LIN A
LIN B
LIN C
LIN D
LIN E
LIN F

;-- työkalu suoaan ylös
```

```
LIN_REL{z 30}

; -- suoritetaan toinen liikesarja -----

; -- tarkka paikoitus aloistupisteeseen

PTP ALOITUS

; -- määritetään $APO.CDIS
; -- järjestelmämuuttujan arvo

$APO.CDIS=40

; -- työkalu suoraan alas
LIN_REL{z -30}

; -- likimääräinen paikoitus
; -- pisteisiin A-E

LIN A C_DIS
LIN B C_DIS
LIN C C_DIS
LIN D C_DIS
LIN E C_DIS

; -- tarkka paikoitus
; -- pisteeseen F

LIN F

; -- työkalu suoraan ylös

LIN_REL{z 30}

PTP HOME

END
```

Liite 2

KUVIO ERI TYÖKAPPALEISSA

```
DEF paikoitus( )

;--- suoritetaan sarja toimintoja -----
;--- kaikissa taulukkoon maaritellyissa tyokoordinaatis-
toissa ---

;--- maaritellaan FRAME muotoinen muuttuja ALOITUS
DECL FRAME ALOITUS

;--- maaritellaan uusi muuttujatyyppi KAPPALETYPE
STRUC KAPPALETYPE REAL X, REAL Y, REAL Z, REAL A, REAL B,
REAL C

;--- maaritellaan taulukkomuuttuja SIJAINTI,
;--- joka on tyyppia KAPPALETYPE
DECL KAPPALETYPE SIJAINTI[3]
INT I

INI

ALOITUS={x 10,y 10,z 28}
; -- aloituspiste
; -- tyokoordinaatiston origosta mitattuna
; -- A,B,C arvot luetaan tyokoordinaatistosta
; -- koska niita ei ole ylikirjoitettu

;--- tyostettavien kappaleiden sijainti
;--- tyokoordinaatistossa ---
;--- sijannit annettaan STRUC muuttujan KAPPALETYPE
;--- maaraamassa muodossa muuttujalle SIJAINTI
;--- SIJAINTI itse on 3x1 kokoinen taulukko

SIJAINTI[1]={X 0,Y 0,Z 0,A 0,B 0,C 0}
SIJAINTI[2]={X 140,Y -350,Z 0,A 31,B 0,C 0}
SIJAINTI[3]={X 140,Y -350,Z 0,A -59,B 0,C 0}

PTP HOME ;-- BCO Run

$TOOL = TOOL_DATA[16] ; -- kaytetaan tyokalua #16
$BASE = BASE_DATA[6] ; -- kaytetaan tyokoordinaatistoa #6

;---- kasitellaan kaikki SIJAINTI-taulukkoon
;---- maaritellyt kappaleet

FOR I=1 TO 3
    $BASE = BASE_DATA[6] ; -- asetetaan vertailukohta

    ;-- asetetaan kappaleen sijainti
    $BASE.X = $BASE.X+SIJAINTI[I].X
```

```
    $BASE.Y = $BASE.Y+SIJAINTI[I].Y
    $BASE.Z = $BASE.Z+SIJAINTI[I].Z
    $BASE.A = $BASE.A+SIJAINTI[I].A
    $BASE.B = $BASE.B+SIJAINTI[I].B
    $BASE.C = $BASE.C+SIJAINTI[I].C

    PTP ALOITUS ; -- aloituspaikoitus

    LIN_REL{x 0,y 0,z -31} ;pystysuora liike alas
    CIRC_REL{x 10,y -10},{x 10,y 10},CA 360 ;kuvion piirto
    LIN_REL{x 0,y 0,z 31} ;nouseva liike

ENDFOR

PTP HOME

END
```

Liite 3

KUVIORYHMÄ ERI TYÖKAPPALEISSA

```
DEF kuvioryhma( )

;--- suoritetaan sarja toimintoja -----
;--- kaikissa taulukkoon maaritellyissa tyokoordinaatis-
toissa ---
;--- maaritellaan FRAME muotoinen muuttuja ALOITUS
DECL FRAME ALOITUS

;--- maaritellaan uusi muuttujatyyppi KAPPALETYPE
STRUC KAPPALETYPE REAL X, REAL Y, REAL Z, REAL A, REAL B,
REAL C, INT ROWS, INT COLS

;--- maaritellaan taulukkomuuttuja SIJAINTI,
;--- joka on tyyppia KAPPALETYPE
DECL KAPPALETYPE SIJAINTI[3]

INT I
INT ROWS
INT COLS

INI
; -- aloituspiste
; -- tyokoordinaatiston origosta mitattuna
; -- A,B,C arvot luetaan tyokoordinaatistosta
; -- koska niita ei ole ylikirjoitettu

ALOITUS={x 10,y 10,z 28}

; -- tyostettavien kappaleiden sijainti
; -- tyokoordinaatistossa ---
; -- sijannit annettaan STRUC muuttujan KAPPALETYPE
; -- maaraamassa muodossa muuttujalle SIJAINTI
;--- SIJAINTI itse on 3x1 kokoinen taulukko

SIJAINTI[1]={X 0,Y 0,Z 0,A 0,B 0,C 0,ROWS 1,COLS 4}
SIJAINTI[2]={X 140,Y -350,Z 0,A 31,B 0,C 0,ROWS 2,COLS 4}
SIJAINTI[3]={X 140,Y -350,Z 0,A -59,B 0,C 0,ROWS 3,COLS 4}

PTP HOME ; -- BCO Run

$TOOL = TOOL_DATA[16] ;kaytetaan tyokalua #16


;---- kasitellaan kaikki SIJAINTI-taulukkoon
;---- maaritellyt kappaleet

FOR I=1 TO 3
```

```
    $BASE = BASE_DATA[6] ;asetetaan vertailukohta

    ;-- asetetaan kappaleen sijainti
    $BASE.X = $BASE.X+SIJAINTI[I].X
    $BASE.Y = $BASE.Y+SIJAINTI[I].Y
    $BASE.Z = $BASE.Z+SIJAINTI[I].Z
    $BASE.A = $BASE.A+SIJAINTI[I].A
    $BASE.B = $BASE.B+SIJAINTI[I].B
    $BASE.C = $BASE.C+SIJAINTI[I].C

    ;--- luetaan muistiin piirrettavien rivien
    ;--- ja kuvioiden maara
    ROWS=SIJAINTI[I].ROWS
    COLS=SIJAINTI[I].COLS


    ;---- suoritetaan aliohjelma---
    KUVIOPIIRTO(ALOITUS,ROWS,COLS)

    ; -- aliohjelmalla ei paasya ---------------
    ; -- paaohjelman muuttujiin ----------------
    ; -- aloituspisteen ja kuvioiden tiedot on
    ; -- siksi valitettava aliohjelmakutsun mukana ---
ENDFOR

PTP HOME
END


; ---------- ALIOHJELMA ------------------
DEF KUVIOPIIRTO(ALOITUS :IN,ROWS :IN,COLS :IN)

;muuttujien deklaraatiot aliohjelmassa
DECL FRAME ALOITUS
INT I
INT J
INT ROWS
INT COLS

FOR J=1 TO ROWS ;rivien lkm

    PTP ALOITUS ;aloituspaikoitus

    FOR I=1 TO COLS ;kuvioiden lkm
        ;-- pystysuora like alas
        LIN_REL{x 0,y 0,z -31}
        ;-- ympyrän piirto
        CIRC_REL{x 10,y -10},{x 10,y 10},CA 360
        ;-- viistoon ylos nouseva like
        LIN_REL{x 0,y 30,z 31}
    ENDFOR
   ;-- jokainen rivi alkaa 30mm paasta edellisesta
```

```
   ;-- ALOITUS muuttuja on arvona
   ;-- aliohjelmakutsun yhteydessä toimitettu
   ;-- sen arvo ei siksi muutu paaohjelmassa

ALOITUS.X=ALOITUS.X+30

ENDFOR
END
; -----------------------------------------------
```

Liite 4

KAPPALESIIRTO

```
DEF kappalesiirto( )

;--- määritellään uusi muuttujatyyppi KAPPALETYPE

STRUC KAPPALETYPE REAL X, REAL Y, REAL Z, REAL A, REAL B,
REAL C

;--- määritellään taulukkomuuttuja SIJAINTI,
;--- joka on tyyppiä KAPPALETYPE

DECL KAPPALETYPE SIJAINTI[2]

;--- muuttujien esittelyt

INT KAPPALE_LKM
INT KAPPALE_PAKSUUS
INT KAPPALE_NOUTO
INT KAPPALE_LUOVUTUS
INT I

$TOOL = TOOL_DATA[14] ;käytetään työkalua #14
$BASE = BASE_DATA[6] ;käytetään työkoordinaatistoa #6

; aloituspiste työkoordinaatiston origosta mitattuna
; A,B,C arvot luetaan työkoordinaatistosta
; koska niitä ei ole ylikirjoitettu

ALOITUS={x 150,y 150,z 100}

;--- työstettavien kappaleiden sijainnit -------------
;--- suhteessa kaytettävään työkoordinaatistoon ------
;--- sijannit annettaan STRUC muuttujan KAPPALETYPE --
;--- määräämässä muodossa muuttujalle SIJAINTI -------
;--- SIJAINTI itse on 3x1 kokoinen taulukko ----------

SIJAINTI[1]={X 0,Y 0,Z 0,A 0,B 0,C 0}
SIJAINTI[2]={X 50,Y -500,Z 0,A 0,B 0,C 0}

;--- työstettävien kappaleiden lukumäärä

KAPPALE_LKM = 5

;--- työstettävien kappaleiden paksuus

KAPPALE_PAKSUUS = 10

INI
```

```
;-- jos imu jäänyt päälle aloittaessa
;-- suljetaan se

IF $OUT[10]==TRUE THEN
        $OUT[10] = FALSE
ENDIF

PTP HOME ;-- BCO Run

;-- käytettävä työkoordinaatisto

$BASE = BASE_DATA[6];

;-- käytettävä työkalu

$TOOL = TOOL_DATA[14];

;-- alustetaan kappaleiden
;-- nouto- ja luovutuslaskurit

KAPPALE_NOUTO=KAPPALE_LKM
KAPPALE_LUOVUTUS=0

; -- käsitellään kaikki kappaleet

FOR I=1 TO KAPPALE_LKM

    ; -- kappaleen nouto ------------------

    ; -- pinon korkeus asettuu oikealle tasolle
    ; -- noudettavien kappaleiden määrän perusteella

    $BASE.Z = $BASE.Z+KAPPALE_NOUTO*KAPPALE_PAKSUUS

    PTP ALOITUS ;ajo noutopisteeseen
    LIN_REL{z -100} ;lasketaan tarrain alas
    $OUT[10] = TRUE ;tarrain päälle
    LIN_REL{z 100} ;nostetaan tarrain ylös

    ;noutolaskurin uuden arvon asetus

    KAPPALE_NOUTO=KAPPALE_NOUTO-1

    ; -- muutetaan työkoordinaatiston paikkaa -----

    $BASE = BASE_DATA[6]
    $BASE.X = $BASE.X+SIJAINTI[2].X
    $BASE.Y = $BASE.Y+SIJAINTI[2].Y

    ; -- luovutuspinon korkeus asettuu oikealle tasolle
    ; -- jo luovutettujen kappaleiden määrän perusteella
```

```
    $BASE.Z = $BASE.Z+10+KAPPALE_LUOVUTUS*KAPPALE_PAKSUUS
    ;Z-koordinaatin pitää kasvaa kappaleen paksuuden verran
    ;jokaista toimitettua kappaletta kohti

    ; -- kappaleen luovutus ---------------

    PTP ALOITUS ;ajetaan luovutuspisteeseen
    LIN_REL{z -100} ;lasketaan tarrain alas

    ;-- Luotuvuslaskurin arvon kasvatus

    KAPPALE_LUOVUTUS = KAPPALE_LUOVUTUS+1

    $OUT[10] = FALSE ;tarrain pois päältä

    ; -- 0.5 sekunnin VIIVE ENNEN POISAJOA
    WAIT SEC 0.5

    LIN_REL{z 100} ;nostetaan tarrain ylos

    ; --- palautetaan työkoordinaatiston paikka ----
    ; -- alkuperäiseen asemaan

    $BASE = BASE_DATA[6]

ENDFOR

PTP HOME ;kun kaikki palat siirretty, palataan alkuun
END
```

Liite 5

KIINTEÄ TYÖKALU LIIKUTELTAVIEN KAPPALEIDEN KANSSA

```
DEF POROUSKOE( )

;--- määritellään uusi muuttujatyyppi KAPPALETYPE
STRUC KAPPALETYPE REAL X, REAL Y, REAL Z, REAL A, REAL B,
REAL C

;--- maaritellaan taulukkomuuttuja SIJAINTI,
;--- joka on tyyppia KAPPALETYPE

DECL KAPPALETYPE SIJAINTI[2]
INT KAPPALE_LKM
INT KAPPALE_PAKSUUS
INT KAPPALE_NOUTO
INT KAPPALE_LUOVUTUS
INT I

; -- CP-liikkeen likimääräisessä paikoituksessa käytettävän
; -- järjestelmämuuttujan arvon asetus

$APO.CDIS=40

$TOOL = TOOL_DATA[14] ;kaytetaan tyokalua #14
$BASE = BASE_DATA[6] ;kaytetaan tyokoordinaatistoa #6
ALOITUS={x 150,y 150,z 100}
; -- aloituspiste
; -- tyokoordinaatiston origosta mitattuna
; -- A,B,C arvot luetaan tyokoordinaatistosta
; -- koska niita ei ole ylikirjoitettu

; -- tyostettavien kappaleiden sijainti
; -- tyokoordinaatistossa ---
; -- sijannit annetaan STRUC muuttujan KAPPALETYPE
; -- maaraamassa muodossa muuttujalle SIJAINTI
; -- SIJAINTI itse on 2x1 kokoinen taulukko

SIJAINTI[1]={X 0,Y 0,Z 0,A 0,B 0,C 0}
SIJAINTI[2]={X 50,Y -500,Z 0,A 0,B 0,C 0}

KAPPALE_LKM = 5
KAPPALE_PAKSUUS = 10

INI

; -- jos paineilmatarraimeen on jäänyt imu päälle -
; -- suljetaan se ----------
IF $OUT[10]==TRUE THEN
    $OUT[10] = FALSE
ENDIF
```

```
; -- jos paineilmapora on jäänyt käyntiin ---
; -- sammutetaan se ---------
IF $OUT[2]==TRUE
    $OUT[2]=FALSE
ENDIF

PTP HOME ;-- BCO Run


; -- käytettävä työkalu ------
$TOOL = TOOL_DATA[14];

; -- alustetaan kappaleiden
; -- nouto- ja luovutuslaskurit

KAPPALE_NOUTO=KAPPALE_LKM
KAPPALE_LUOVUTUS=0

FOR I=1 TO KAPPALE_LKM
    ; -- referenssinä käytettävä työkoordinaatisto ---
    $BASE = BASE_DATA[6];

    ; -- noutopaikan työkoordinaatiston asetus
    $BASE.X = $BASE.X+SIJAINTI[1].X
    $BASE.Y = $BASE.Y+SIJAINTI[1].Y

    ; -- pinon korkeus asettuu oikealle tasolle
    ; -- noudettavien kappaleiden määrän perusteella

    $BASE.Z =
$BASE.Z+SIJAINTI[1].Z+KAPPALE_NOUTO*KAPPALE_PAKSUUS

    ;ajetaan noutopisteeseen
    PTP ALOITUS

    ;lasketaan tarvittava määrä alaspain
    LIN_REL{z -100}

    ;tarrain päälle
    $OUT[10] = TRUE

    ;siirto ylös
    LIN_REL{z 100}

    ;noutolaskurin uuden arvon asetus

    KAPPALE_NOUTO=KAPPALE_NOUTO-1


    KULJETUS()

    ;ajo jättöpisteeseen
```

```
    $BASE = BASE_DATA[6]
    $TOOL = TOOL_DATA[14]

    ;ROBOTTI OHJAA TYÖKALUA
    $IPO_MODE=#BASE

    $BASE.X = $BASE.X+SIJAINTI[2].X
    $BASE.Y = $BASE.Y+SIJAINTI[2].Y
    $BASE.Z =
$BASE.Z+10+KAPPALE_LUOVUTUS*KAPPALE_PAKSUUS

    ;Z-sijainnin pitaa kasvaa kappaleen paksuuden verran
    ;jokaista toimitettua kappaletta kohti


    ;ajetaan luovutuspisteeseen
    PTP ALOITUS

    ;lasketaan tarvittava maara alaspain
    LIN_REL{z -100}

    ;Kappelelukua pienennetaan
    KAPPALE_LUOVUTUS = KAPPALE_LUOVUTUS+1

    ;tarrain pois
    $OUT[10] = FALSE

    ;VIIVE ENNEN POISAJOA
    WAIT SEC 0.5

    ;siirto ylos
    ; -- käytetään likimääräistä paikoistusta
    LIN_REL{z 100} C_DIS

ENDFOR

PTP HOME

END

; -- KULJETUS ALIOHJELMA ------------
DEF KULJETUS()

FRAME REIKA1
FRAME REIKA2
FRAME REIKA3
FRAME REIKA4


;NOSTETAAN YLOSPAIN 200MM
;-- käytetään likimääräistä paikoitusta
LIN_REL{Z 200} C_DIS
```

```
;AJO PORAUKSEEN

; -- vaihdetaan työkoordinaatisto
; -- käytetään robotin ulkopuolista paineilmaporaa
$BASE = BASE_DATA[14]

; -- vaihdetaan työkalu
; -- liikutetaan kappaletta tarttujalla
$TOOL = TOOL_DATA[8]

;-- ROBOTTI OHJAA TYOKAPPALETTA

$IPO_MODE=#TCP

;-- PORAN PAIKOITUS KAPPALEESEEN

PTP {X 150,Y 150,Z -50}

;TEHDAAN YMPYRAN KAARELLE NELJA REIKAA

REIKA1.X=$POS_ACT.X+20
REIKA1.Y=$POS_ACT.Y

REIKA2.X=$POS_ACT.X
REIKA2.Y=$POS_ACT.Y-20

REIKA3.X=$POS_ACT.X-20
REIKA3.Y=$POS_ACT.Y

REIKA4.X=$POS_ACT.X
REIKA4.Y=$POS_ACT.Y+20

; --VIEDAAN ALKUPISTEISIIN
; -- ja kutsutaan porauksen suorittavaa aliohjelmaa

LIN REIKA1
PORAUS()
LIN REIKA2
PORAUS()
LIN REIKA3
PORAUS()
LIN REIKA4
PORAUS()
END

; -- KULJETUS ALIOHJELMAN LOPPU ------------------

; -- PORAUS ALIOHJELMA -----------------
DEF PORAUS()

;MUUTETAAN LIIKENOPEUS PORAUKSEN AIKANA
```

```
$VEL.CP=0.1

;PORA KAYNTIIN

$OUT[2]=TRUE

;LIIKE ALAS

LIN_REL {Z 20}

WAIT SEC 0.5

;LIIKE YLOS

LIN_REL {Z -20}

;PORA POIS PAALTA

$OUT[2]=FALSE

;LIIKENOPEUDEN PALAUTUS

$VEL.CP=0.5

END
; -- PORAUS ALIOHJELMAN LOPPU ----------------------
```