TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky a mezioborových inženýrských studií

BAKALÁŘSKÁ PRÁCE

2008 **Radek Pšenička**

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: B 2612 Elektrotechnika a informatika
Studijní obor: Informatika a logistika

Název bakalářské práce

Vzájemná koordinace pohybu dvou robotů

Autor bakalářské práce: **Radek Pšenička**
Vedoucí bakalářské práce: **Doc. Mgr. Ing. Václav Záda, CSc.**
Konzultant: **Ing. Pavel Pirkl**

Rozsah práce a příloh

Počet stran textu: 58
Počet příloh: 1
Počet obrázků: 17
Počet tabulek: 3

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky a mezioborových inženýrských studií

Ústav mechatroniky a technické informatiky Akademický rok: 2007/2008

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jméno a příjmení: **Radek Pšenička**
studijní program: B 2612 - Elektrotechnika a informatika
obor: Informatika a logistika

Vedoucí ústavu Vám ve smyslu zákona o vysokých školách č.111/1998 Sb. určuje tuto bakalářskou práci:

Název tématu: **Vzájemná koordinace pohybu dvou robotů**

Zásady pro vypracování:

1. Seznamte se s programovacím jazykem Rapid a uživatelským prostředím ProgramMaker. Nastudujte možnosti generování trajektorií pohybu robota, včetně možnosti realizace vzájemné spolupráce dvou robotů.
2. V daném programu Rapid realizujte úlohu vzájemné koordinace pohybu dvou robotů. Koncový bod prvního robota se bude pohybovat po obecné trajektorii reprezentující například tvar formy. Druhý robot bude nahrazovat externí zařízení provádějící posun a natáčení formy.
3. Úlohu z předešlého bodu 2. ověřte na reálném systému, který bude dodán v průběhu zimního semestru.
4. Proveďte shrnutí dosažených výsledků realizované úlohy z bodů 2 a 3.

Rozsah grafických prací: dle potřeby dokumentace
Rozsah průvodní zprávy: cca 40 stran

Seznam odborné literatury:

[1]RobotStudio 3.1 User's Guide. Firemní dokumentace k programu RobotStudio, Švédsko 2004.

[2] Rapid Reference Manual. Firemní dokumentace k programu Rapid, Švédsko 2004.

Vedoucí bakalářské práce:	**Doc. Mgr. Ing. Václav Záda, CSc.**
Konzultant:	**Ing. Pavel Pirkl**

Zadání bakalářské práce:	**4. 10. 2007**
Termín odevzdání bakalářské práce:	**16. 5. 2008**

LS

....

Vedoucí ústavu Děkan

V Liberci dne

MÍSTOPŘÍSEŽNÉ PROHLÁŠENÍ:

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL. V tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

V Liberci dne 16.5. 2008

.....

Radek Pšenička

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Hálkova 6, 461 17 Liberec 1

Abstrakt

Cílem bakalářské práce bylo vytvořit několik úloh na vzájemnou koordinaci pohybů dvou robotů. Během koordinovaných pohybů se má provádět opracování obecného tvaru formy. Způsob opracování je typický pro průmyslové využití například pro nanášení lepidla, svařování a nebo obrušování.

Pro opracování byl zvolen plastový výlisek. Práce obsahuje čtyři různé způsoby řešení pro opracování, které jsou podrobně popsány. Pro práci jsou použiti dva roboti od firmy ABB s označením IRB140 a IRB1400. Koncový bod prvního robota se pohybuje po obecné trajektorii výlisku a simuluje tak jeho opracování. Druhý robot zde nahrazuje externí zařízení, provádějící posun a natáčení výlisku, který má připevněný místo nástroje. Při tvorbě úloh je využito vývojové prostředí RobotStudio 4.0, ProgramMaker a dále programovacího jazyka Rapid. Úlohy popsané v této práci byly úspěšně odzkoušeny v laboratoři inteligentních robotů.

Tato práce dále obsahuje postup pro kalibraci obou robotů. Obecnou metodu pro získání vzájemných pozic obou robotů. Použité kinematické vztahy související s touto prací. Dále jsou zde popsány vybrané instrukce jazyka Rapid, které jsou nezbytné pro řešení ukázkových úloh.

Klíčová slova:

- Robot
- ABB
- Rapid
- ProgramMaker
- Koordinovaný pohyb

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Hálkova 6, 461 17 Liberec 1

Abstract

The aim of this bachelor thesis was a creation of few exercises on relative coordination of movement for two robots. During the coordination of this movement the working of common shape of the form. The method of working the pressing is typical for industrial usage for example for application of glue, welding and grinding-off.

For working was elect molded plastic part. The thesis contains four different ways of solution for working, which are closely described. For a practical exercise were used two robots from a company ABB named IRB140 and IRB1400. The ending point of the first robot is moving over a common trajectory of pressing and simulates a working of the molded plastic part. The second robot substitutes external device which is making a movement and rotation of the molded plastic part, which is placed instead of a tool. For a creation of exercises is used the developing instrument RobotStudio 4.0, ProgramMaker and programming language Rapid. Practical exercises described in this thesis were successfully tested on a real control system in laboratory of the intelligent robots.

And next this thesis contains a calibrating process procedure of robots. Common method for acquisition each other positions of robots. Used kinematic relations related with this thesis. And next there are described chosen instructions of the Rapid language which are necessary for solving sample exercises.

Key words:

- Robot
- ABB
- Rapid
- ProgramMaker
- Coordination of movement

Obsah

Seznam obrázků

Seznam tabulek

Použité zkratky

BF	Base Frame
DI	Digital Input
DO	Digital Output
RS	RobotStudio 4.0
TCP	Tool Center Point
WObj	WorkObject

Úvod

Robotika je jedním z vědních oborů mechatroniky, která spojuje mechaniku, elektroniku a softwarové inženýrství. Robotika se zabývá vývojem a výzkumem robotů. Je jedním z předních světových průmyslových odvětví. Roboti jsou řízeni umělou inteligencí, kterou mu dodal člověk v podobě zdrojového kódu nazývající se program. Použití robotů nám přináší mnoho výhod jak při výrobních procesech, tak i při zvyšování pracovního výkonu. Výkon průmyslového podniku je důležitý pro jeho ekonomický chod. V dnešní době je „robotizace“, tedy proces, kdy nahrazujeme stroje za lidskou pracovní sílu, nezbytný. Tito neúnavní pracovníci dokážou podniku ušetřit nemalé peníze, které se po čase vrátí zpět za investici do této automatizace.

Téma vzájemné spolupráce robotů má pro praktické použití nepřeberné množství využití v průmyslových podnicích, například předávání pracovního objektu, dělba práce při opracování pracovního objektu, atd.

V této bakalářské práci se omezíme pouze na spolupráci dvou robotů. Oba roboti jsou umístěni v laboratoři inteligentních robotů v budově A, učebna S15. Oba roboti jsou od švédsko-švýcarské firmy ABB, která vyrábí roboty od roku 1972. Tato firma má nejrozšířenější základnu průmyslových robotů na celém světě. Solečnost ABB samozřejmě poskytuje také aplikační software pro své roboty, ve kterých lze jednoduše vytvářet aplikace. Tito roboti pomocí přídavných periférií dokážou vykonávat operace, jakou jsou například svařování, lakování, povrchová úprava, nebo také manipulace s materiály o hmotnosti do 500 kg. S roboty od firmy ABB se nejčastěji můžeme setkat například v automobilovém průmyslu, slévárenství, ale i také v potravinářském průmyslu atd.

Tato práce bude ukázkou průmyslového využití robotů, které je charakteristické například pro nanášení lepidla nebo sváření. Koncový bod prvního robota se bude pohybovat po obecné trajektorii reprezentující tvar formy. Tímto koncovým bodem je ocelová válcová tyč, připevněná k zápěstí robota. Druhý robot bude nahrazovat externí zařízení, provádějící posun a natáčení formy, kterou má připevněnou místo nástroje. Pro tuto práci je použito místo formy plastového výlisku. Výsledkem celé práce by měly být validní programy pro oba roboty, které budou díky nim provádět vzájemně koordinované pohyby při spolupráci. Tyto programy budou nahrány do řídících systémů obou robotů a budou prakticky demonstrovány v Laboratoři inte-

ligentních robotů v budově A, učebně S15 na konci celé bakalářské práce.

Pro práci je využit dle zadání ABB programovací jazyk Rapid a uživatelské prostředí ProgramMaker. Dále je použit softwarový produkt ABB RobotStudio 4.0, které má jako svou součást již zmíněné programovací prostředí ProgramMaker. V tomto aplikačním software lze jednoduše a rychle upravovat jednotlivé aplikace. RobotStudio 4.0 i ProgramMaker jsou software, které jsou běžnou aplikací pro operační systém Windows, je tedy možné použití na jakémkoliv počítači s ohledem na minimální požadavky tohoto software.

Pro seznámení s RobotStudiem a ProgramMakerem je využito:

- uživatelské příručky RobotStudio 4.0,
- dokumentace Stavitel Pyramid k pochopení základů programovacího jazyka Rapid a vytvoření jednoduchého nástroje,
- dokumentace ročníkového projektu Demonstrační aplikace v RobotStudiu.

Pro seznámení s reálným systémem a dále při práci je doporučeno využít následující dokumentace:

- dokumentaci elektrického zapojení robotů a jejich nástrojů,
- dokumentaci k oběma robotům pro jejich kalibraci.

Získané informace a zkušenosti jsou více rozvíjeny v této práci.

Práce je rozdělena do několika sekcí.

1. V první části se seznámíme s použitými roboty a nástroji, včetně popisu jejich připevnění k rameni robota. Dále ještě s možnosti komunikace mezi roboty.
2. Druhá část obsahuje postup při kalibraci robotů.
3. Ve třetí části se dozvíme, jak získat vzájemné pozice robotů v laboratoři.
4. Obsahem čtvrté části je popis a použití WorkObjectu.
5. V páté části jsou vybrané kinematické vztahy pro výpočet pozice plastového výlisku připevněného na robota.
6. Šestá část obsahuje popis použitých funkcí jazyka Rapid.
7. Závěrečná sedmá část obsahuje popis a řešení jednotlivých úloh pro opracování výlisku.

1 Použití roboti, pracovní nástroje a komunikace

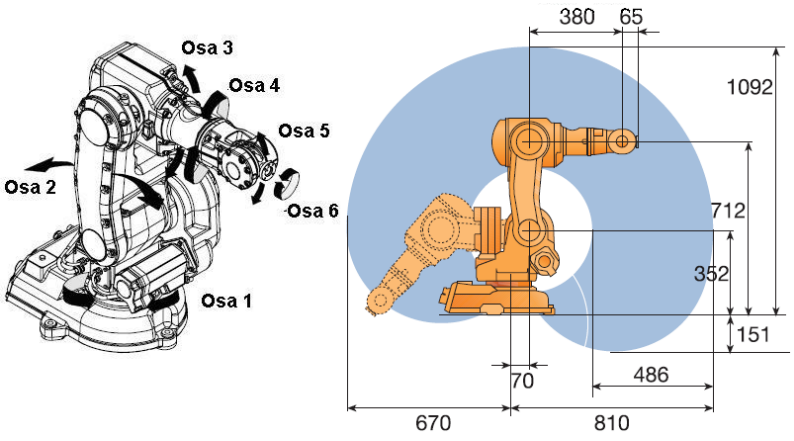
V této kapitole Vám budou přiblíženy oba typy použitých robotů a jejich technická specifikace. Dále u každého z robotů jsou popsány použité pracovní nastroje včetně označení připevnění k zápěstí robota. Možnosti komunikace mezi řídícími systémy obou robotů.

1.1 Robot s označením IRB140

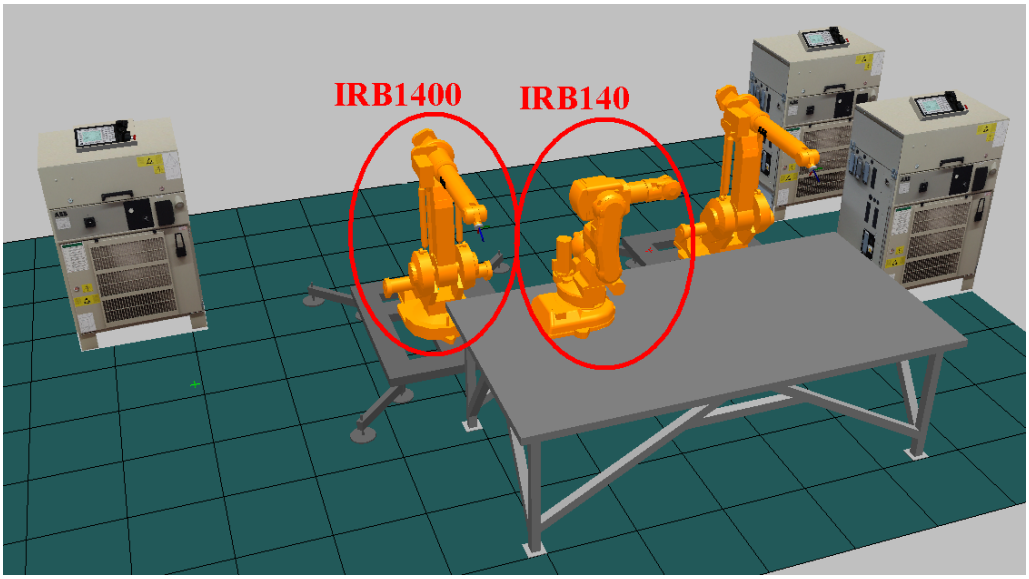
První robot bude používat jako pracovní nástroj „ocelovou tyčku“ (viz obrázek 3). Technická specifikace robota (viz tabulka 1) a dále označení rotačních os a rozsah pracovního prostoru (viz obrázek 4).

Hmotnost robota:	95 kg
Počet os volnosti:	6
Dosah ramene:	810 mm
Přesnost pohybu:	0.05 mm
Max. hmotnost pracovního nástroje:	5 kg
Max. rychlost pohybu:	2.1 m/s
Signálů na horním rameni:	12
Max. hlučnost řídícího systému:	70 dB

Tabulka 1: Tabulka technických parametrů IRB140



Obrázek 1: Označení rotačních os a rozsah pracovního rozsahu IRB140



Obrázek 2: Zvýraznění obou robotů využitých k práci v laboratoři inteligentních robotů, model vytvořený v RobotStudiu4.0

1.1.1 Pracovní nástroj (ocelová tyčka)

První nástroj je upevněn na robota IRB140. Je vyroben z oceli a má tvar válce s výškou 100 mm a průměrem 6 mm. Má na sobě vyřezaný závit, kterým ho lze snadno připevnit k zápěstí robota. Zde tento nástroj zastupuje například pistoli pro nanášení lepidla nebo svářecí hrot. Pro snazší simulaci v praxi je vhodné si vytvořit v RobotStudiu (RS) jednoduchý model tohoto nástroje se stejnými rozměry. Postup pro vytvoření nástroje v RS nalezneme v dokumentaci (Stavitel pyramid). Dole je obrázek nástroje včetně způsobu připevnění k rameni robota (viz obrázek 3).

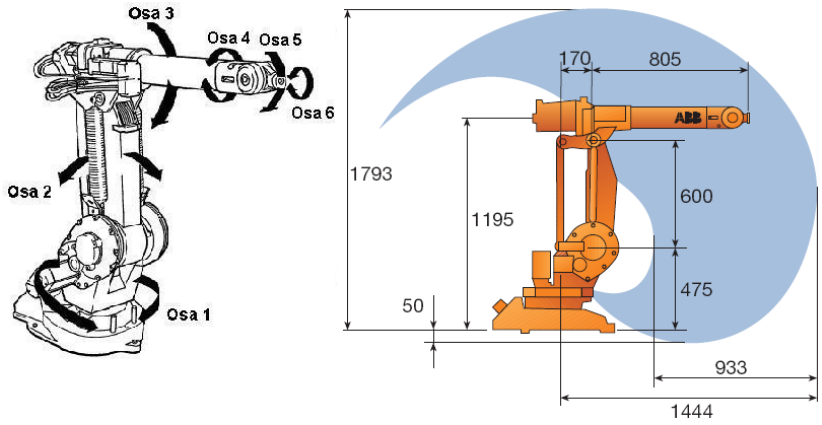
Obrázek 3: Nástroj ocelová tyčka a označení připevnění k zápěstí robota

1.2 Robot s označením IRB 1400

Druhý robot IRB1400 bude používat jako nástroj plastový výlisek, který bude posouvat a natáčet (viz obrázek 5). Technická specifikace robota (viz tabulka 2) a dále označení rotačních os a rozsah pracovního prostoru (viz obrázek 4).

Hmotnost robota:	225 kg
Počet os volnosti:	6
Dosah ramene:	1440 mm
Přesnost pohybu:	0.05 mm
Max. hmotnost pracovního nástroje:	5 kg
Max. rychlost pohybu:	2.1 m/s
Signálů na horním rameni:	12
Max. hlučnost řídícího systému:	70 dB

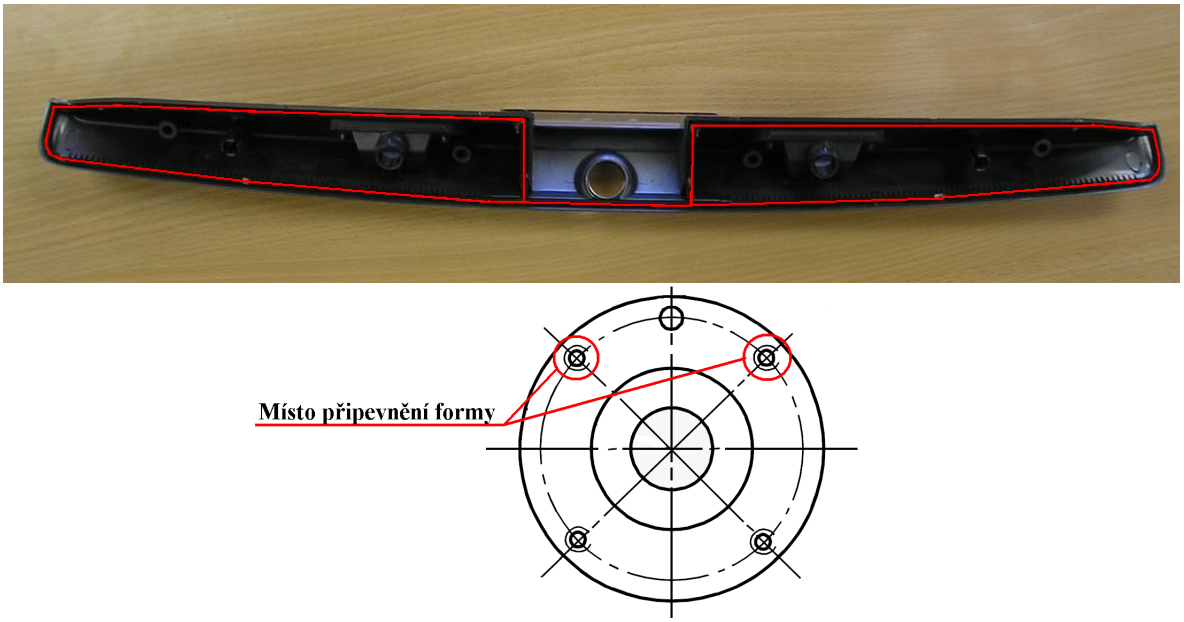
Tabulka 2: Tabulka technických parametrů IRB1400



Obrázek 4: Označení rotačních os a rozsah pracovního rozsahu IRB1400

1.2.1 Pracovní nástroj (plastový výlisek)

Jedná se o plastovou úchytku na otevírání kufru osobního automobilu. V našem případě nám poslouží pouze její vnitřní část, vypadající jako forma s hranami, které budou opracovány. V práci bude dále tato plastová úchytka pojmenována „plastový výlisek“. Nástroj je upevněn dvěma šrouby na robota IRB1400. Na obrázku jsou červeně označeny opracovávané hrany a způsob připevnění k rameni robota (viz obrázek 5).



Obrázek 5: Naznačení opracovaných hran plastového výlisku, včetně označení připevnění k zápěstí robota

1.3 Komunikace mezi roboty

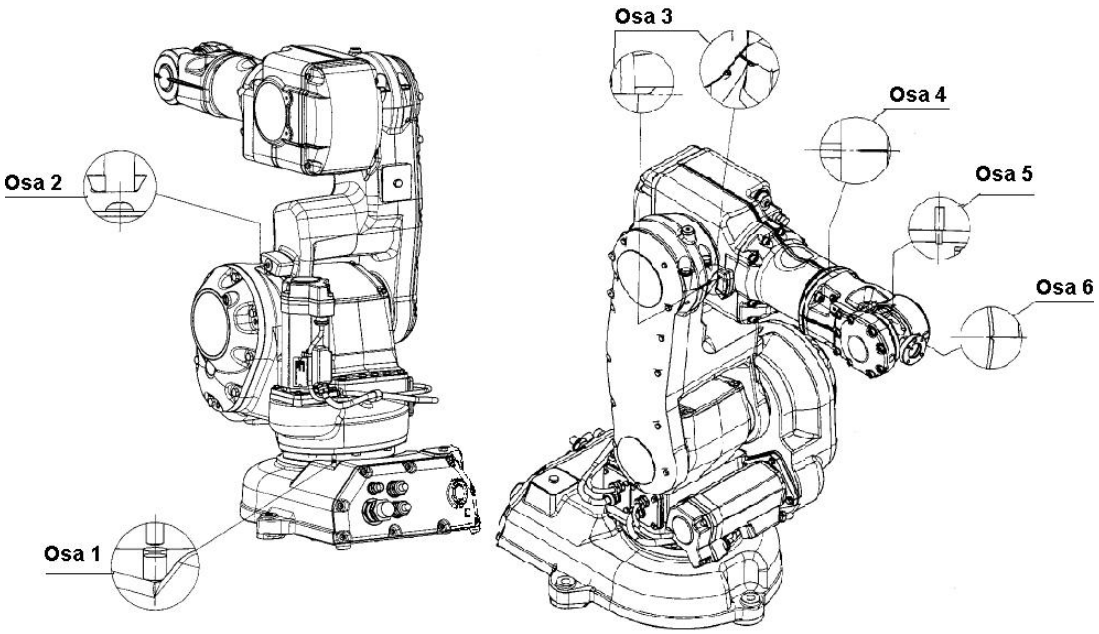
Důležitou součástí při spolupráci robotů je komunikace. Roboti spolu komunikují přez vlastní řídící systém. Mohou využít buď digitální I/O signály, analogové I/O signály, ethernet linkou a dále ještě mohou komunikovat pomocí sériového rozhraní (RS232) pro posílání dat. V úlohách nám postačí využití digitálních I/O signálů a sériový kanál.

- **Digitální I/O signály** - v úlohách jsou využity pouze digitální vstupy značeno DI10 (1 až 4) a digitální výstupy značeno DO10 (1 až 4). Více o propojení mezi oběma roboty je popsáno v (Dokumentace elektrického zapojení robotů, a jejich nástrojů.pdf, *Petr FLODRMAN*, *Karel BLAVKA*, TUL).

- **Komunikace přes sériový kanál** - řídící systémy obou robotů musí být propojeny křiženým sériovým kabelem. V řídících systémech jsou zapojeny do portu „com2“. Pro posílání dat je použita stejná přenosová rychlost 9600 bit/s u řídících systémů obou robotů, která je nezbytná pro správnou komunikaci. Nevýhoda sériové linky je omezení při odesílání paketů. Maximální velikost paketu je pouze 1 byt, tedy 8bitové slovo, které odpovídá číslu o maximální velikosti 255.

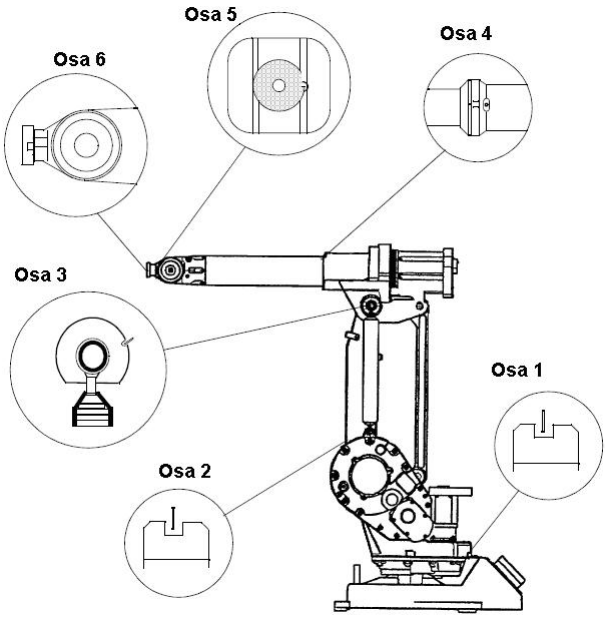
2 Kalibrační proces robota

Pojmem kalibrační proces robota budeme chápat, kdy všechny osy robota nastavíme do kalibračních pozic. To znamená, že všechny inkrementální rotační snímače dostaneme do nulové polohy. Tyto nulové polohy jednotlivých os jsou použity jako základní nebo výchozí pozice. Od základních pozic jsou počítány souřadnice souřadného systému (x, y, z) robota. Kalibrační zóny jsou na dílčích částech kinematického řetězce robota u jednotlivých os vyznačeny výřezy a vrypy do jeho kovového těla, které se pokud možno musejí co nejpřesněji překrývat. Vyznačení kalibračních zón pro jednotlivé typy robotů (viz obrázek 6 a obrázek 7). Je-li robot dobře kalibrovaný, dokáže vypočítat správnou aktuální pozici při spuštění. Nejsou-li osy robota správně kalibrované, má to za následek nepřesný pohyb jeho os včetně TCP, a to by mělo negativní účinek pro veškeré následující činnosti robota.



Obrázek 6: Kalibrační zóny na rameni robota IRB140

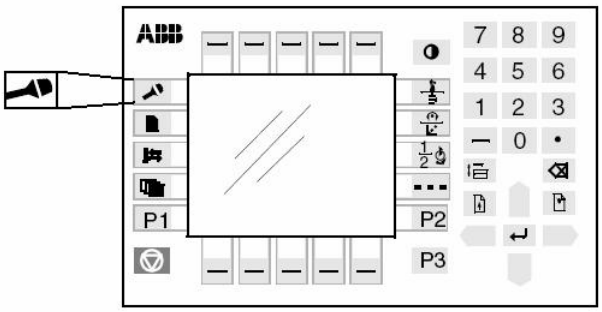
Kalibrační proces se provádí například v případech, kdy nebyl robot kalibrován po nenadálé ztrátě svých kalibračních dat, nebo také při vypnutém stavu a následné výměně záložních baterií, které udržují tyto informace v paměti a dále při vybitém stavu záložních baterií a vypnutém řídícím systému robota. Pokud robot není kalibrován, vyzve k tomu uživatele při prvním spuštění.



Obrázek 7: Kalibrační zóny na rameni robota IRB1400

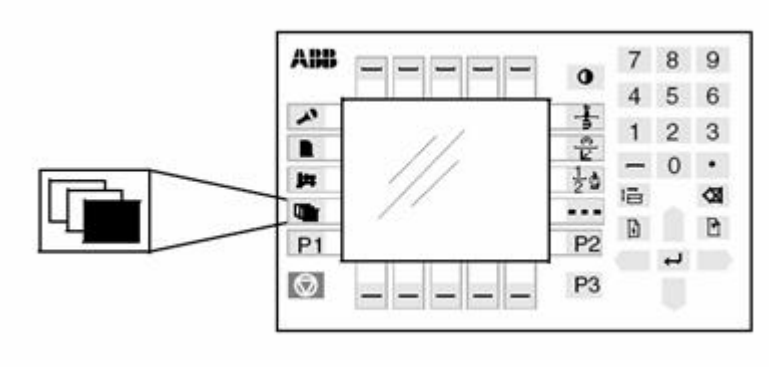
2.1 Jednotlivé kroky kalibrace

1. Nejdříve přesuneme jednotlivě všechny osy robota do nulových poloh pomocí jednotky ručního řízení v angličtině „Teach Pendant“ dále bude použita zkratka (TP) (viz obrázek 8). Každý z použitých robotů obsahuje šest nezávislých os. Pro správnou funkci je nutné nastavit všech šest os do výchozích pozic. Protilehlé výřezy a vrypy na robotově těle musí být samozřejmě proti sobě. Tolerance odchylky je přibližně 5 %.



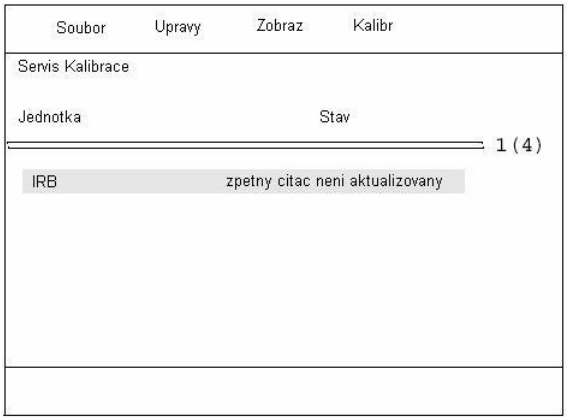
Obrázek 8: Přepnutí do okna pro ruční řízení robota na TP

2. Nyní máme nastavenou výchozí polohu robota a na TP přepneme do „Dalsi okna“ (viz obrázek 9). Dále vybereme „Servis“ a potvrdíme klávesou ENTER.



Obrázek 9: Zvolení položky „Dalsi okna“ na TP

V horním okraji zvolíme „Zobraz“ a vybereme třetí položku „Kalibrace“, kterou potvrdíme klávesou ENTER. Nyní se nám zobrazí okno s hlášením „IRB - zpetny citac neni aktualizovany“, pokud není robot kalibrovaný (viz obrázek 10), nebo „IRB - Synchronizovana“. Pak není potřeba kalibrovat.



Obrázek 10: Okénko se stavem kalibrace

3. Není-li robot kalibrován, zvolíme vpravo nahoře volbu „Kalibr“ a dále označíme „Aktualizovat zpetny citac“ a stiskneme ENTER.

4. Nyní se objeví upozornění o přesné kalibraci a dotaz, zda-li jsou všechny osy v nulových pozicích. Stiskneme „Ano“.

5. Objeví se okno s osami a jejími stavy. Pokud robot ztratil kalibraci všech os, zvolíme „Vse“. Označí se všechny osy znakem „X“, které chceme kalibrovat. Jinak lze vybrat i jednotlivé osy pro samostatnou kalibraci.

6. Po vybrání stiskneme „Ano“.

7. V posledním kroku se Vás systém zeptá, jestli opravdu souhlasíte se změnou hodnot při kalibraci. Stiskneme „Ano“.

Nyní je robot kalibrován a je již použitelný pro vykonávání úloh.

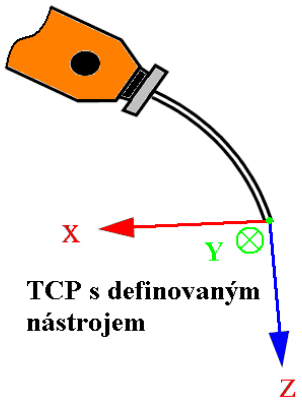
Poznámka:
Po správné kalibraci bude robot při pokusu o lineární pohyb v bodu blízkém kalibrační pozici pravděpodobně hlásit chybu: „Chyba singularity“. Ta je způsobená jeho pozicí v „nule“ - robot má totiž kalibrační „nulu“ jako vztažný bod a v jeho blízkosti není schopen dopočítat údaje k pohybu. Řešením je přepnout se na manuální režim robota a pomocí TP poodjet s ním mimo nulu. Pak již bude bez problému fungovat při vykonání pohybu a nebude hlásit chybu.

3 Získání vzájemných pozic robotů

V této kapitole se dozvíme, jak získat vzájemné pozice robotů v laboratoři. Získání pozic robotů je pro tuto práci velmi důležité, abychom věděli, jak jsou od sebe vzdáleni a natočeni. Pro reálný systém to jsou dva roboti, kteří se nacházejí v kartézském souřadném systému x, y, z. Díky němu se dají zjistit přesné pozice bodů dosažení v prostoru.

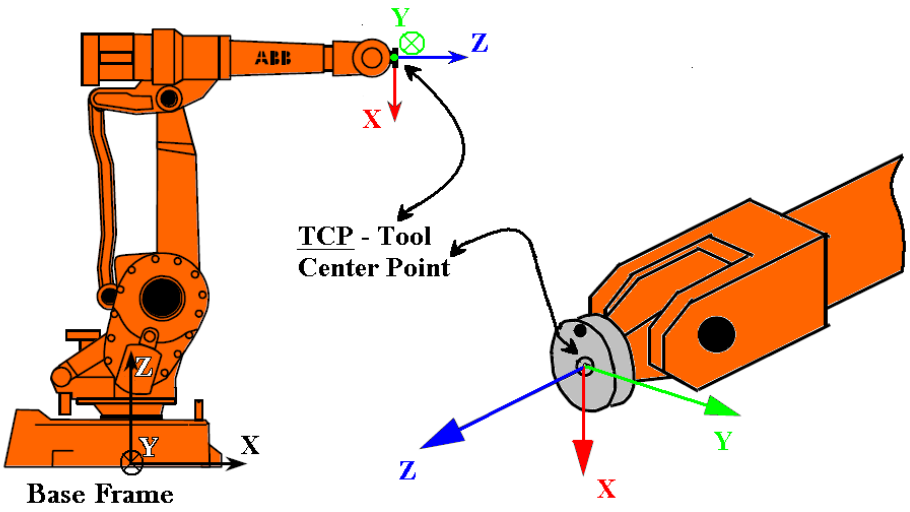
3.1 Metoda získání pozic

Je nutné, aby byli oba roboti pro tuto operaci správně kalibrováni (viz kapitola 2). Po správné kalibraci obou robotů je možné získat „téměř přesné pozice robotů“ tím, že je vůči sobě odměříme pomocí Tool Center Point (TCP). TCP je orientovaný bod na nástroji robota, kterým dosahuje cílových bodů své trajektorie (viz obrázek 11). Polohu tohoto bodu je možné sledovat pomocí zobrazení na TP, nebo programově



Obrázek 11: TCP definováno na nástroji

pomocí jazyku Rapid. Při nedefinovaném nástroji je TCP umístěno na konci ramene robota v tzv. referenčním bodě, kde se upevňují nástroje (viz obrázek 12). Půjde o měření vzdálenosti IRB140 od IRB1400 v osách x, y a z. Pro tuto metodu získávání pozice je důležité, aby na sebe oba roboti dosáhli svými TCP, protože od nich se počítá vzdálenost vlastního středu těla Base Frame (BF) robota. Při odměřování pozic musí být nataven aktuální WObj na TP obou robotů wobj_0, aby robot počítal pozice od jejich nulového BF v souřadném systému.



Obrázek 12: Umístění TCP při nedefinovaném nástroji

3.2 Postup měření pozic

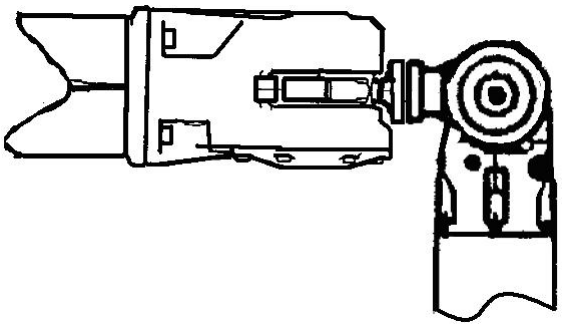
1. Na začátku je nutné mít roboty kalibrované a natočené stejně, jako po jejich kalibraci (viz obrázek 6 a obrázek 7).

2. Natočení IRB140 o 90° pomocí TP a maximální možné přesné přiblížení IRB1400 TCP k IRB140 TCP tak, aby se kryly. Tím získáme z TP IRB1400, vzdálenost IRB140 v ose y a výšku z od IRB1400 (viz obrázek 13).

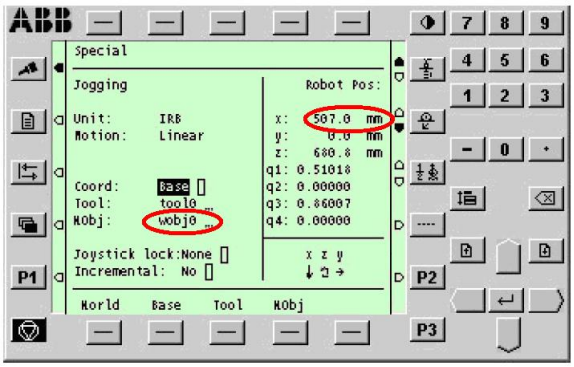
Obrázek 13: Měření pozice y a z

3. Nyní pootočíme opět s IRB140 o 90°, tedy celkově již o 180°, a opakujeme nejbližší a nejpřesnější přiblížení IRB 1400 TCP (viz obrázek 14).

Nyní vidíme na TP IRB1400 vzdálenost v ose x IRB140 od IRB140 (viz obrázek 15). Tímto jsme získali vzdálenost IBR140 od IRB1400 v souřadnicích (x = 524 mm, y = 1165 mm, z = 610 mm). Natočení robotů vůči sobě nebylo měřeno, protože je známo, že jsou v laboratoři natočeni od sebe přibližně o 90°. Toto měření je možné znovu použít pro získání nových pozic robotů, jestliže se fyzicky změnila jejich pozice



Obrázek 14: Měření pozice x



Obrázek 15: Pozice x na TP

v učebně, ale pouze za předpokladu, že na sebe dosáhnou svými TCP. Naměřené pozice robotů budou využity pro přesné programování a optimalizování aplikace pro reálný systém v RobotStudiu a ProgramMakeru.

4 WorkObject

V této kapitole se dozvíme, jak nastavit WorkObject (WObj) pro správnou funkci získání a přepočítávání souřadnic robotů v systému. WObj je součástí jazyku Rapid, ale je natolik důležitý, že je pro jeho vysvětlení věnována tato kapitola.

4.1 WorkObject a jeho nastavení pro oba roboty

Nastavení pozice a orientace WObj je pro oba roboty velice důležité. Jakýkoliv bod prostoru musí být vztažený k nějakému počátku souřadného systému, od kterého jsou počítány jeho souřadnice. Tento počátek souřadného systému zajišťuje v programu právě WObj. Dále je možné měnit jeho polohu a orientaci za běhu programu a tím tak simulovat pohyb nějakého objektu, který je vztažený k tomuto WObj. V základním nastavení má každý robot svůj základní WObj s názvem wobj_0 umístěný v pozici (x = 0, y = 0, z = 0) svého souřadného systému v angličtině Base Frame, zkratka (BF).

Popis složek WObj:

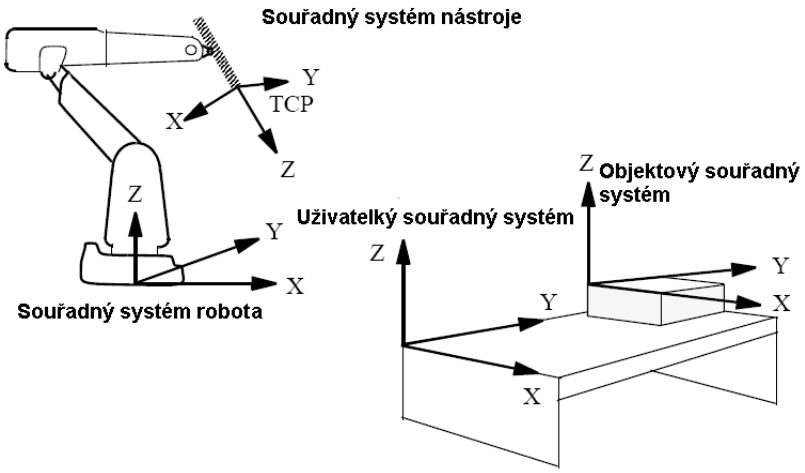
- *robhold* - definuje, zda robot drží WObj.
  - TRUE - robot drží WObj (WObj definovaný na TCP robota), nastaví se při použití statického nástroje (nástroj není připevněn k zápěstí).
  - FALSE - robot nedrží WObj, nastaví se při definovaném nástroji robota (připevněného k zápěstí).

- *ufprog* - definuje, zda je nebo není použitý pevný uživatelský systém.
  - TRUE - je použit pevný uživatelský systém.
  - FALSE - je použit pohyblivý uživatelský souřadný systém.

- *ufmec* - definuje pohybovou jednotku, se kterou robot provádí koordinované pohyby, například posuvný pás. Je možné ho nastavit, má-li *ufprog* hodnotu FALSE.

- *uframe* - určuje pozici a natočení právě aktuálního WObj v uživatelském souřadném systému. Pozice je dána v osách (x, y, z) a natočení v kvaternionech (q_1, q_2, q_3, q_4).

- *oframe* - určuje pozici a natočení právě aktuálního WObj v objektovém souřadném systému. Pozice je dána v osách (x, y, z) a natočení v kvaternionech (q_1, q_2, q_3, q_4).



Obrázek 16: Uživatelský a objektový souřadný systém

Příklad:
Bod A má pozici: x = 500 mm, y = 300 mm, z = 200 mm. Rotace ve všech třech osách je nulová, a je vztažen k vytvořenému WObj s názvem MyWObj, umístěného x = 50 mm, y = 100 mm, z = 150 mm od BF robota s nulovou rotací ve všech osách. Pak je výsledná cílová pozice bodu A pro robota x = 550 mm, y = 400 mm a z = 350 mm.
Ukázka nastavení WObj pro tento příklad v programu:

```
MyWobj.robhold:=FALSE;
MyWobj.ufprog := TRUE;
MyWobj.ufmec := "";
MyWobj.uframe.trans.x := 50;
MyWobj.uframe.trans.y := 100;
MyWobj.uframe.trans.z := 150;
MyWobj.uframe.rot.q1 := 0.707;
MyWobj.uframe.rot.q2 := 0;
MyWobj.uframe.rot.q3 := 0;
```

```
MyWobj.uframe.rot.q4 := -0.707;
MyWobj.oframe.trans.x := 0;
MyWobj.oframe.trans.y := 0;
MyWobj.oframe.trans.z := 0;
MyWobj.oframe.rot.q1 := 1;
MyWobj.oframe.rot.q2 := 0;
MyWobj.oframe.rot.q3 := 0;
MyWobj.oframe.rot.q4 := 0;
```

Ukázka změny pozice v průběhu činnosti programu:

```
MyWObj.uframe.trans.z := MyWObj.uframe.trans.z + 100;
```

Tímto jsme změnili pozici MyWObj v ose z na 200mm od BF robota. Tato změna pozice MyWObj vyvolá i změnu pozice bodu A, který je k němu vztažený.
Pro naše použití je nutností, aby oba WObj byly umístěny ve stejném místě. Tím zajistíme, že každý z robotů bude počítat své souřadnice od shodného počátku.
Jednotlivé nastavení pro roboty bude vypadat následovně:

- Pro IRB1400 je vytvořen WObj s názvem WOBJ1400 s umístěním (0, 0, 0) a s nulovou rotací ve všech osách.
- Pro IRB140 je vytvořen WObj s názvem WOBJ140 s umístěním (-524, 1165, -610) a opět nulová rotace ve všech stupních volnosti. Jeho pozici jsme získali postupem popsaným (viz kapitola 3).

Každý z WObj umístíme do programů daného robota. Všechny cílové body použité v programech budeme vztahovat vždy k jednomu z vytvořených WObj robota dle programu.

5 Kinematika pohybu bodu v prostoru

Kinematika se zabývá pohybem tělesa v rovině nebo v prostoru. Prostorovým pohybem budeme rozumět pohyb pevného tělesa v prostoru. Pro naše úlohy půjde o prostor R^3. Při zvolené souřadné soustavě a známém pohybu tělesa v něm, je pak možné zjistit polohu bodu s tímto tělesem spojeným. V našem případě to budou naměřené body určující trajektorii na plastovém výlisku, který je připevněn k zápěstí robota. Pozici a natočení TCP budeme vždy v potřebnou chvíli znát. Velikost změny pozice bodů na výlisku je dána velikostí úhlu natočení TCP a jeho pozice ve všech třech osách.

5.1 Popis bodu v prostoru

Jakýkoliv bod v prostoru můžeme popsat jeho polohu a orientací od středu souřadného systému.

- **Poloha bodu** - je údaj, vyjadřující umístění bodu v prostoru vzhledem k nějakému souřadnému systému, od kterého je vzdálený v osách x, y a z.
- **Orientace nástroje** - je to velikost natočení nástroje při dosažení cílového bodu v úhlech (α, β, γ) v osách x, y a z od základního (nulového) natočení souřadného systému.

Pro naše úlohy je střed souřadného systému vždy tam, kde se nacházejí oba WObj (WOBJ140 a WOBJ1400).

V robotice se používají pro popis orientace v souřadnicovém systému kvaterniony, značeno (q_1, q_2, q_3, q_4). Kvaternion je tzv. čtveřice obsahující tři imaginární jednotky. Vztah pro výpočet kvaternionu vypadá následovně:

$$q = q_1 + q_2 i + q_3 j + q_4 k \tag{1}$$

$$i,\ j,\ k \in C$$
$$q_1,\ q_2,\ q_3,\ q_4 \in R$$

Pro q_1 až q_4 musí platit výpočet jednotkového kvaternionu:

$$q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1 \tag{2}$$

Pro kvaterniony ovšem neplatí spousta matematických zákonitostí, například komutativnost ab = ba. Pro zjednodušení jsou převáděny kvaterniony na úhly (α, β, γ) v osách x, y a z pomocí jazyku Rapid za použití funkce EulerZYX (viz 6.2.2). Díky těmto hodnotám je můžeme jednodušeji implementovat do matematických vzorců popsaných v této kapitole a opět přesně určit orientaci bodu v prostoru stejně jako za pomocí kvaternionů.

V RS existuje jednoduchý převodník **Qaternion converter**. Pomocí něj lze převádět kvaterniony (q_1, q_2, q_3, q_4) na úhly (α, β, γ) v osách x, y, z a naopak. Zde slouží jen pro kontrolu přepoču, pro Rapid je bohužel nepoužitelný. Tento převaděč naleznete v menu **Tools** → **Add Ins** → Zaškrtnout (Quaternion Converter). Poté se převaděč objeví v menu **Tools**.

5.2 Výpočet bodu v prostoru

Při výpočtu budeme zjišťovat nové pozice a orientace bodu v prostoru, který je pevně spojený s robotovým TCP. Pro přehlednost bude obecný bod spojený pevně s TCP označen jako bod P. Nové pozice a orientace bodu P jsou počítány vzhledem k pevnému souřadnému systému. Z polohy a orientace TCP v cílovém bodu budeme počítat polohu bodu P v prostoru. Pro prostorové zjištění pozice bodu P jsou zmíněny tyto dva matematické aparáty:

- Transformační rotační matice,
- Eulerovy úhly.

Existují i jiné metody výpočtu, ale v této práci se zaměříme na již zmíněné, protože jsou snadno implementovatelné do programu.

5.2.1 Transformační rotační matice pro (3D)

První možností, jak vypočítat pozici bodu P v prostoru při známém natočení TCP o obecný úhel α kolem osy rotace, je Transformační rotační matice. Pozici TCP získáme pomocí instukce Trans (viz jazyk Rapid 6.2.8) a orientaci v úhlech natočení okolo základních os x, y a z získáme pomocí funkce EulerZYX (viz jazyk Rapid 6.2.2). Pro 3D prostor se používají speciální transformační rotační matice pro rotaci kolem všech tří os (viz tabulka 3).

Osa otáčení je rovnoběžná s osou	X			Y			Z		
Pravotočivé otáčení bodu	1	0	0	$cos\alpha$	0	$sin\alpha$	$cos\alpha$	$-sin\alpha$	0
	0	$cos\alpha$	$-sin\alpha$	0	1	0	$sin\alpha$	$cos\alpha$	0
	0	$sin\alpha$	$cos\alpha$	$-sin\alpha$	0	$cos\alpha$	0	0	1
Levotočivé otáčení bodu	1	0	0	$cos\alpha$	0	$-sin\alpha$	$cos\alpha$	$sin\alpha$	0
	0	$cos\alpha$	$sin\alpha$	0	1	0	$-sin\alpha$	$cos\alpha$	0
	0	$-sin\alpha$	$cos\alpha$	$sin\alpha$	0	$cos\alpha$	0	0	1

Tabulka 3: Tabulka transformačních matic pro obecný úhel v jednotlivých osách x, y, z (podle Krause, 1993)

Příklad:
Pro výpočet nové pozice bodu P = (x_1, y_1, z_1) při rotaci kolem osy z vypadá aplikace rotační matice následovně:
Rotační matice pro rotaci kolem osy z při otáčení o úhel α ve směru pohybu otáčení hodinových ručiček.

$$\mathbf{R}(\alpha, \mathbf{z}) = \begin{pmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3}$$

Daná matice pro rotaci v ose z je násobena velikostí polohového vektoru bodu P od robotova TCP.

$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \begin{pmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \tag{4}$$

Pak nová pozice bodu P po pootočení o úhel α ve směru pravotočivého otáčení má pozici $P = (x_2, y_2, z_2)$.

5.2.2 Rotace pomocí eulerových úhlů

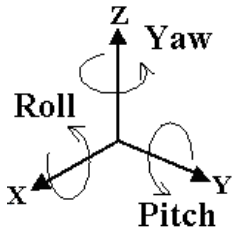
Další možností, jak vypočítat polohu bodu P v prostoru, je řešení pomocí eulerových úhlů. Je to soustava tří úhlů (ψ, ϑ, φ) postupných pootočení souřadnicových

os v určitých souřadnicových rovinách. Jednotlivé úhly (ψ, ϑ, φ) náleží těmto osám rotace $\longrightarrow$ (z, y, x). Velikosti jednotlivých úhlů získáme z funkce EulerZYX (viz jazyk Rapid 6.2.2). Je nutné dodržovat pořadí natáčení os, okolo kterých otáčíme bod. Jiné výsledky například dostaneme pro otáčení XZY. V našem případě budeme otáčet ZYX.

Takto vyjádříme matematicky postupné otáčení:

$$R_{RPY} = R_Z \cdot R_Y \cdot R_X \tag{5}$$

Z angličtiny zkratka RPY:
R - roll, P - pitch, Y - yaw. V češtině se používá „otáčení“, „klopení“ a „bočení“.



Obrázek 17: RPY

Matice R_{RPY} pro postupné natáčení ZYX vypadá takto:

$$\mathbf{R_{RPY}} = \begin{pmatrix} \cos\vartheta\cdot\cos\psi & -\sin\psi\cdot\cos\varphi+\cos\psi\cdot\sin\vartheta\cdot\sin\varphi & \sin\psi\cdot\sin\varphi+\cos\psi\cdot\sin\vartheta\cdot\cos\varphi \\ \sin\psi\cdot\cos\vartheta & \cos\psi\cdot\cos\varphi+\sin\psi\cdot\sin\vartheta\cdot\sin\varphi & -\cos\psi\cdot\sin\varphi+\sin\psi\cdot\sin\vartheta\cdot\cos\varphi \\ -\sin\vartheta & \cos\vartheta\cdot\sin\varphi & \cos\varphi\cdot\cos \end{pmatrix} \tag{6}$$

Aplikace eulerových úhlů (ψ, ϑ, φ) pro obecné pootočení v systému:

$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \begin{pmatrix} \cos\vartheta\cdot\cos\psi & -\sin\psi\cdot\cos\varphi+\cos\psi\cdot\sin\vartheta\cdot\sin\varphi & \sin\psi\cdot\sin\varphi+\cos\psi\cdot\sin\vartheta\cdot\cos\varphi \\ \sin\psi\cdot\cos\vartheta & \cos\psi\cdot\cos\varphi+\sin\psi\cdot\sin\vartheta\cdot\sin\varphi & -\cos\psi\cdot\sin\varphi+\sin\psi\cdot\sin\vartheta\cdot\cos\varphi \\ -\sin\vartheta & \cos\vartheta\cdot\sin\varphi & \cos\varphi\cdot\cos \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \tag{7}$$

Po vynásobení matice polohovým vektorem bodu P = (x_1, y_1, z_1) od robotova TCP, získáme novou pozici bodu P po pootočení systému o obecné úhly (ψ, ϑ, φ).

6 Jazyk Rapid

Jazyk Rapid obsahuje všechny důležité programové instrukce pro roboty ABB. Syntaxe je obdobná programovacímu jazyku Pascal a proto jeho pochopení není nijak složité. Pro editaci programu v tomto jazyku postačí i pouhý poznámkový blok, jinak pro větší uživatelský komfort je vhodnější využít programovací prostředí ProgramMaker, které umožňuje i jednoduchou kompilaci napsaného kódu. V této kapitole bodou zmíněny jen ty příkazy jazyka Rapid, které bude potřeba znát pro řešení daných úloh dle zadání práce. Pro více informací o tomto jazyku nalezneme v (Rapid Reference Manual).

6.1 Příkazy jazyka Rapid pro sériový kanál (RS232)

Při vzájemné koordinaci pohybů obou robotů je velice důležité, aby oba roboti znali aktuální pozici spolupracujícího robota. Díky sériovému kanálu RS232 je možné posílat aktuální pozice, kde se zrovna nachází TCP robota. Dále se dá také využívat k posílání instrukčních číselných, znakových nebo textových kódů, které si druhá strana dokáže dekódovat a využívat dané informace.

6.1.1 Open

Instrukce Open se používá pro otevírání sériového kanálu, nebo souboru pro čtení a zápis. Příkazem Close se sériový kanál uzavírá (deaktivuje).
Syntaxe: Open Objekt \File, IODevice, \Read, \Write, \Append, \Bin;

- *Object* - I/O objekt (I/O zařízení), které má být otevřené například, „HOME:“, „flp1:“, „com2:“ nebo „pc:“.
- *File* - jméno souboru, který má být otevřen.
- *IODevice* - odkaz k otevření souboru nebo sériového kanálu. Tento odkaz je pak užívaný pro čtení a zápis ze nebo do souboru či sériového kanálu.
- *Read* - otevře soubor nebo sériový kanál pro čtení. Při čtení ze souboru vždy začíná na začátku souboru.
- *Write* - otevře soubor nebo sériový kanál pro zápis. Při výběru existujícího souboru, bude obsah smazán a přepsán novým záznamem. Vždy se zapisuje na začátek souboru.

- *Append* - otevírá soubor nebo sériový kanál pro zápis. Když vybraný soubor existuje, tak cokoliv připsané do souboru bude zapsáno na konec.

Argumenty \Read, \Write, \Append se vzájemně vylučují. Když žádný z nich není specifikovaný, instrukce jedná stejným způsobem jako \Write argument pro znakové soubory nebo sériový kanál (bez instrukce \Bin argumentu) a stejným způsobem jako \Append argument pro binární soubory nebo sériový kanál (s instrukcí \Bin argumentu). \Bin - soubor nebo sériový kanál je otevřený pro binární mód. Jestliže nejsou zadané argumenty \Read, \Write nebo \Append, pak instrukce Open otevře binární soubor nebo sériový kanál pro čtení a zápis s ukazatelem na konci souboru.
Přklad:

```
Var iodev chan;
...
Open "com2", chan\Bin;
```

Otevře sériový kanál na portu com2, který se jmenuje „chan“.

6.1.2 ReadBin

Instrukce ReadBin čte byt (8bit) slova ze souboru nebo sériového kanálu.
Syntaxe: ReadBin (IODevice [\Time]);

- *IODevice* - název souboru nebo sériového kanálu, ze kterého se čte.
- *Time* - maximální čas pro čtení ze souboru nebo sériového kanálu v sekundách.

Jestliže argument *Time* není zadán, pak je automaticky nastaveno na 60 sekund. Když je přesaženo 60 sekund, je spuštěna chybová operace čtení, chybový kód ERR_DEV_MAXTIME. Jestliže není definován chybový kód, program bude zastaven.
Příklad:

```
VAR iodev chan;
VAR num cislo;
...
Open "com2", chan\Bin;
cislo := ReadBin(chan);
```

Otevře sériový kanál „chan“ na portu com2 a data o maximální velikosti 8 bitů v

jednom paketu jsou přečtena a uložena do číselné proměnné „cislo“. Když číslo ze sériového kanálu není do 60ti sekund přečteno, program bude ukončen bez chybového kódu, protože nebyl implementován.

6.1.3 WriteBin

Instrukce WriteBin se používá pro zápis 8bitových slov v jednom paketu do sériového kanálu. Jedná-li se o číslo, pak jeho hodnota je v rozmezí 0 - 255. Záporná čísla se musejí označit a znaménko poslat zvlášť.
Syntaxe: WriteBin IODevice, Buffer, NChar;

- *IODevice* - jméno používaného sériového kanálu.
- *Buffer* - seznam (pole), obsahující čísla nebo znak, které mají být napsané.
- *NChar* - počet znaků z Buffer, které mají být zapsány.

Příklad:

```
VAR iodev chan;
VAR num cislo;
VAR num delka;
...
Open "com2", chan\Bin;
cislo := 255;
delka := 1;
WriteBin chan, cislo, delka;
```

Otevře sériový kanál „chan“ na portu com2 a data o maximální velikosti 8bitů tedy číslo max. velikosti 255 a délka 1, jsou zapsána do sériového kanálu.

6.1.4 ClearIOBuff

Používá se pro mazání vstupních dat ve vyrovnávací paměti (buffer) sériového kanálu.
Syntaxe: ClearIOBuff IODevice;

- *IODevice* - jméno používaného sériového kanálu.

Příklad:

```
VAR iodev chan;
```

...
Open "com2", chan \Bin;
ClearIOBuff chan;
V tomto příkladě jsme provedli vymazání vstupní vyrovnávací paměti pro sériový kanál „chan“.

6.2 Další intrukce jazyka Rapid

6.2.1 TPWrite

Tato instrukce vypisuje data jako text na Teach Pendant.
Syntaxe: TPWrite String \Num \Bool \Pos \Orient;

- *String* - text, který bude vypsán na Teach Pendant o maximální velikosti 80 znaků.
- *Num* - vypíše číslo z proměnné za text.
- *Bool* - vypíše logickou proměnnou za text.
- *Pos* - data, která představují pozici se vypíšou za text.
- *Orient* - data, která představují orientaci, se vypíšou za text.

Příklad:
VAR num cislo;
...
Cislo := 255;
TPWrite "Hodnota = " \Num := cislo;
Na TeachPendant se vypíše: „Hodnota = 255“.

6.2.2 EulerZYX - (Euler ZYX rotace)

Užívá se pro získání eulerových úhlů natočení (z, y, x) z orientace objektu vzhledem k počátku soustavy. Hodnoty orientace objektu jsou v quaternionech. Jde tedy o funkci, která převádí qaterniony na velikosti úhlů v osách rotace (x, y, z). Odpovídající Eulerův úhel, vyjádřený ve stupních, je v rozsahu [- 180, + 180]. Vrací hodnotu (num).
Syntaxe: EulerZYX ([\X] [\Y] [\Z] ObjektRotace);

Argumenty \X, \Y a \Z se zadávají vždy samostatně do funkce, jinak bude vyvoláno chybové hlášení. Tato funkce vrací pouze jen hodnotu jednoho daného úhlu.

- *[\X]* - vrací úhel v ose rotace X vzhledem k počátku soustavy.
- *[\Y]* - vrací úhel v ose rotace Y vzhledem k počátku soustavy.
- *[\Z]* - vrací úhel v ose rotace Z vzhledem k počátku soustavy.

Příklad:

```
VAR num xuhel;
VAR num yuhel;
VAR num zuhel;
VAR pose object;
...
xuhel := EulerZYX(\X, object.rot);
yuhel := EulerZYX(\Y, object.rot);
zuhel := EulerZYX(\Z, object.rot);
```

Výsledkem tohoto příkladu jsou uložené úhly natočení objektu v osách (x, y, z) do proměnných „xuhel“, „yuhel“, „zuhel“.

6.2.3 CRobT

Načte aktuální pozici robota a externích os. Tato funkce vrací hodnotu robtarget s pozicí (x, y, z), orientací (q1, q2, q3, q4), uspořádáním os robota a umístěním externích os.

Syntaxe: CRobT ([\Tool] [\WObj]);

- *Tool:* nástroj užívaný pro výpočet aktuální pozice robota. Pokud není uvedeno, je použit aktuální nástroj.
- *WObj:* WorkObject, ke kterému je pozice vypočítána.

6.2.4 Sin

Vypočítá hodnotu sinus úhlu. Vrací hodnotu (num).

Syntaxe: Sin(uhel);

- *uhel* - velikost úhlu.

Příklad:

```
VAR num uhel;
VAR num cislo;
...
cislo := Sin(uhel);
```

6.2.5 Cos

Vypočítá hodnotu cosinus úhlu. Vrací hodnotu (num).
Syntaxe: Cos(uhel);

- *uhel* - velikost úhlu.

Příklad:

```
VAR num uhel;
VAR num cislo;
...
cislo := Cos(uhel);
```

6.2.6 Trunc

Používá se ke zkrácení hodnoty desetinného čísla na požadovanou délku, nebo přímo na integer (celé číslo). Vrací hodnotu (num).
Syntaxe: Trunc (Val [\Dec]);

- *Val* - hodnota desetinného čísla, které má být zkráceno.
- *Dec* - počet desetinných míst, které mají zůstat za desetinou čarou. Při nezadání parametru Dec se automaticky nastaví na 0 a výsledná číselná hodnota bude bez desetinné části.

Příklad:

```
VAR num cislo;
cislo := Trunc(0.38521\Dec := 3);
```

Do proměnné „cislo“ bude vložena hodnota 0.385.

6.2.7 Round

Používá se pro zaokrouhlení desetinné části čísla na požadovaný počet míst. Používá se klasické matematické zaokrouhlování **(číslo 5-9 zaokrouhluje nahoru a číslo menší jak 5 zaokrouhluje směrem dolů).** Vrací hodnotu (num).
Syntaxe: Round (Val [\Dec]);

- *Val* - hodnota desetinného čísla, které má být zaokrouhleno.
- *Dec* - číslo udává, na jaké pozici desetinného čísla má být provedeno zaokrouhlení.

Příklad:

```
VAR num cislo;
cislo := Round(0.38521 \Dec := 1);
```

Do proměnné „cislo“ bude vložena hodnota 0.4.

6.2.8 Trans, rot

Trans se používá pro získání informací, týkajících se pozice a orientace bodu vzhledem k aktuálnímu WObj. Orientace je udávána v kvaternionech.
Syntaxe: robtarget.trans. [\x], [\y], [\z].
robtarget.rot. [\q1], [\q2], [\q3], [\q4].
Příklad:

```
VAR num xpos;
VAR num ypos;
VAR num zpos;
VAR robtarget point;
xpos := point.trans.x;
ypos := point.trans.y;
zpos := point.trans.z;
```

Do proměnných „xpos“, „ypos“, „zpos“ jsou vloženy vzdálenosti bodu „point“ od aktuálního WObj.
Příklad:

```
VAR num q1;
VAR num q2;
VAR num q3;
```

```
VAR num q4;
VAR robtarget point;
q1 := point.rot.q1;
q2 := point.rot.q2;
q3 := point.rot.q3;
q4 := point.rot.q4;
```

Do proměnných „q1“, „q2“, „q3“ a „q4“ budou postupně vloženy velikosti natočení v kvaternionech bodu „point“ od aktuálního WObj. Trans a rot se dají použít obráceně a upravovat jednotlivé vzdálenosti a orientace bodů vzhledem k WObj.

6.2.9 Clock

Clock se používá pro měření času. Funkci lze rovněž použít jako stopky. Jedná se o data typu (clock). Naměřený čas se ukládá v sekundách s přesností na 0.01 sekundy. Clock používá tyto instrukce:

- ClkStart - start času,
- ClkStop - zastavení času,
- ClkReset - resetování času na 0,
- ClkRead - pro přečtení délky časového úseku v sekundách od ClkStart.

Příklad:

```
VAR clock clock2;
VAR num time;
...
ClkReset clock2;
ClkStart clock2;
WaitUntil DInput(DI10) = 1;
ClkStop clock2;
time := ClkRead(clock2);
```

Do proměnné „time“ je vložena hodnota délky čekání systému, dokud se nezmění DI10 na hodnotu 0. Čtení času je možné, také i bez nutnosti zastavení času instrukcí ClkStop.

6.2.10 Speeddata

Tento datový typ obsahuje spoustu částí, zabývající se rychlostí pohybu. Nám stačí jen ta část, ve které můžeme kombinovat rychlost a čas prováděného pohybu. Jako další parametr v datovém typu speeddata je čas, značeno [\T]. Tento čas určuje dobu, po kterou se robot bude přesouvat z jednoho bodu do druhého.
Syntaxe: MoveL *, Speed [\V] | [\T], Zone, Tool\ Wobj;
Příklad:
MoveL pointA, v1000\T := 5, fine, tool0;
Robot se bude pohybovat po lineární trajektorii z aktuální pozice do bodu „pointA“. Doba, než se robot dostane do „pointA“, bude trvat 5 sekund bez ohledu na rychlost v1000.

6.2.11 ITimer

Tato instrukce se používá při časovém přerušení systému a následném spuštění „trap rutiny“. Tato instrukce může být použita například pro aktualizování výpočtu vždy při jeho spuštění na základě času, který uběhl od spuštění či předchozího přerušení.
Příklad:
VAR intnum timeint;
...
IEnable;
CONNECT timeint WITH pocitej;
ITimer 0.3, timeint;
...
IDisable;
V tomto příkladu nejprve aktivujeme spuštění přerušení. Poté systém informujeme o tom, ke které rutině se má připojit když nastane přerušení. V našem případě to je „trap rutina“ - „pocitej“. Do ITimeru nastavujeme 0.3, tj. přerušení se spustí po každých 0.3 sekundách. Vykoná se rutina „pocitej“. Na konec deaktivujeme spouštění přerušení.
Deklarace Trap rutiny:
TRAP pocitej
...
ENDTRAP

7 Řešení ukázkových úloh

Tato kapitola obsahuje úlohy na vzájemnou koordinaci pohybu robotů při opracování obecného tvaru trajektorie výlisku. Jsou zde podrobně popsány čtyři způsoby řešení, které byly úspěšně odzkoušeny na reálném systému v laboratoři inteligentních robotů. Při řešení jednotlivých úloh jsou využity poznatky ze všech předchozích kapitol.

7.1 Získání trajektorie na plastovém výlisku

Ještě než bude možné provádět úlohy, je potřeba určit několik bodů pro definování trajektorie pohybu nástroje na výlisku. Ta bude určovat, jaké hrany se mají opracovat. Opracování výlisku bude provádět robot IRB140, který má připevněný nástroj „ocelová tyčka“. Trajektorii získanou z bodů bude postupně robot projíždět a tím simulovat opracování. Následuje postup, jak získat body určující trajektorii na plastovém výlisku:

1. Podmínkou správného naměření bodů na plastovém výlisku je, aby byli oba roboti správně kalibrováni (viz kapitola 2).

2. Dále je potřeba, aby oba roboti měli nastavený jako aktuální svůj WObj na TP. Pro IRB140 to je WOBJ140 a pro IRB1400 to je WOBJ1400.

3. Nyní přesuneme robota IRB1400 do pozice, ze které druhým robotem IRB140 bude získána trajektorie bodů na plastové fromě. Pro tuto pozici robota IRB1400 zavedeme označení pod názvem „pozice vytváření bodů“. Tato pozice musí být v takové vzdálenosti od robota IRB140, aby byl schopný naměřit všechny body na výlisku potřebné k sestavení trajektorie s orientací kolmo proti bodu na výlisku v ose z. Body budeme postupně číst z TP IRB140 a pak je zapíšeme do programu. Při měření je použita „pozice vytváření bodů“ ve vzdálenosti od počátku soustavy (x = 800, y = 400, z = 1085) a natočením (x = 0, y = 90, z = 0).

4. Po naměření bodů určujících trajektorii na plastovém výlisku stačí tyto body jen zadat do programu robota IRB140, který s nimi pak dále pracuje. Při zadávání bodů do programu musí být zapisovány postupně podle indexů od

prvního bodu do posledního tak, jak je má IRB140 projíždět. Všechny body na výlisku jsou naměřeny vzhledem ke shodnému počátku systému obou robotů.

5. Jedinou podmínkou, aby následující algoritmy fungovaly i po nově naměřených bodech trajektorie je ta, že „pozice vytváření bodů“ musí mít vždy fixovanou orientaci vzhledem k počátku souřadného systému, která je: (x = 0, y = 90, z = 0). Změna „pozice vytváření bodů“ je bez problému možná, ale pak se musí tato pozice v programu IRB140 aktualizovat. Při změně orientace „pozice vytváření bodů“ je nutné pro správnou funkci programu vhodně poupravit algoritmus v proceduře VypBody() dále (viz procedura 7.2.3), která obsahuje i „pozice vytváření bodů“.

7.2 První úloha (statická)

V tomto příkladě na vzájemnou koordinaci pohybů se bude jednat o tzv. statickou úlohu z hlediska robota IRB1400. Robot IRB1400 bude postupně projíždět svoji definovanou trajektorii. V každém bodě své trajektorie se zastaví a pošle pozici TCP sériovým kanálem do řídícího systému IRB140. Robot IRB140 pak vykoná konstantní rychlostí objetí nové trajektorie plastového výlisku připevněného na zápěstí IRB1400. Poté se IRB1400 přesune do dalšího bodu své trajektorie a znovu se opakuje příjem nových pozičních dat do řídícího systému IRB140 a projetí trajektorie nově vypočítaných bodů na výlisku.

7.2.1 Podrobný popis řešení

Při startu programu se oba roboti synchronizují v „Home“ pozici pomocí čekání na I/O signály. „Home“ pozice je definovaná kvůli bezpečnému startu programu obou robotů. Robot IRB1400 se bude postupně přesouvat s připevněným výliskem na jeho zápěstí do předem definovaných bodů. Z každého předdefinovaného bodu vyšle svoji pozici a natočení TCP (vzhledem k souřadnému systému) přes sériový kanál do řídícího systému robota IRB140. Tento robot IRB140 si pomocí kinematických vztahů vypočítá nové pozice bodů na výlisku a projede jejich novou trajektorii a tím provede i opracování. Po projetí trajektorie se vrátí IRB140 do definovaného bodu, který má bezpečnou vzdálenost od IRB1400.

Po dosažení bezpečného bodu IRB140 se může IRB1400 přesunout do dalšího předdefinovaného bodu své trajektorie a opět se opakuje poslání pozice TCP přes

sériový kanál do řídícího systému IRB140. Podmínkou bezproblémového provedení je, aby všechny body určující trajektorii na plastovém výlisku byly v každé předem definované pozici pro IRB1400 v dosažitelnosti externích os IRB140, jinak zahlásí IRB140 chybu cílové pozice a program se zastaví. Dále je důležité, aby byli oba roboti správně kalibrováni, protože při provádění programu se bude hrot, připevněný na IRB140, pohybovat ve vzdálenosti cca 5 mm od plastového výlisku. Při špatné kalibraci by hrozilo poškození plastového výlisku, nebo by mohla vzniknout kolize obou robotů (viz kapitola 2). Tato úloha je již použitelná například při nanášení lepidla.

7.2.2 Popis programu pro IRB1400

Nejprve je nutné popsat program robota IRB1400, protože IRB140 čeká na dosažení jeho prvního bodu, jehož pozici a natočení mu poté pošle k přepočítání nové pozice trajektorie, určující hrany k opracování výlisku.

- Při startu programu je nutné smazat I/O buffer ze sériové linky (viz jazyk Rapid 6.1.4). Při jeho nesmazání se může stát, že v něm zůstala nějaká data z jiné předchozí úlohy, která by v lepším případě skončila po přečtení chybou. Jinak by hrozilo vykonání neznámých dat vedoucích například k poškození zařízení.

- Dále je nutná synchronizace pohybů obou robotů, kterou zajišťují signály pro digitální vstupy a výstupy. Tato synchronizace je použita průběžně v programu, kde by bez těchto signálů hrozila kolize. Dále jsou použity pro informaci, že se posílají data po sériové lince $\longrightarrow$ otevřít sériový kanál pro čtení a zápis. Použité DI a DO signály (SET, RESET, WAITDI).

- Po dosažení každého z předdefinovaných bodů robotovým TCP pošle IRB1400 přes sériový kanál svoji pozici (realizováno procedurou Pozice()).

Popis použité procedury:

PROC Pozice()

Tato procedura slouží pro přečtení a odeslání pozice TCP v daném bodu přes sériový kanál druhému robotovi. V této proceduře je nutné vytvořit pomocný robtarget nazvaný „position“, do kterého budeme ukládat pozici a natočení TCP pomocí funkce CRobT() vzhledem k WOBJ1400. Pro čtení natočení v jednotlivých osách v

používáme EulerZYX a dále pro čtení pozice v prostoru používáme position.trans.x. Po přečtení všech natočení a pozic zpracujeme data do potřebné podoby, aby byly korektní pro odeslání přes sériový kanál pomocí instrukce WriteBin.

7.2.3 Popis programu pro IRB140

- Po startu programu opět nutné vymazání I/O bufferu stejné jako u IRB1400.
- Synchronizace pohybů pomocí I/O digitálních vstupů a výstupů.
- Po příchodu I/O signálu, upozorňujícího na otevření sériového kanálu pro čtení a zápis, se spustí procedura ReadChan().

Popis použitých procedur:

PROC ReadChan()

Po otevření sériového kanálu se pomocí instrukce ReadBin začnou načítat příchozí byty v cyklu. Cykl čtení má velikost rovnou počtu příchozích bytů sériovou linkou. Pro přečtení každého bytu se musí celý cykl provést právě jednou. Proto je dobré mezi první odeslané byty na straně IRB1400 zapsat i délku relace (počet bytů k přečtení), aby nedošlo k zacyklení. Po přečtení všech příchozích bytů se automaticky spustí procedura ReadPosition().

PROC ReadPosition()

Tato procedura začne z došlých bytů z předchozí procedury skládat a vypočítávat data s údaji o pozici a natočení TCP IRB1400 a ukládat je do proměnných, se kterými bude dále pracováno. Poté se spustí procedura VypBody().

PROC VypBody()

Procedura obsahuje naměřené pozice bodů na výlisku pomocí TP, které tvoří trajektorii na výlisku. Tyto pozice byly měřeny při pozici „pozice vytváření bodů“ (viz podkapitola 7.1). Pomocí cyklu se začnou vypočítávat nejdříve rozdíly naměřených bodů na výlisku od bodu „pozice vytváření bodů“, abychom dostali pouze vzdálenosti bodů od TCP. Poté se nové pozice těchto bodů vypočítají na základě dat o nové pozici a natočení TCP IRB1400, které byly přijaty sériový kanálem. Pro výpočty nových pozic jsou použity transformační rotační matice (viz kapitola 5). Pak se provede opracování nově vypočítané trajektorie na plastovém výlisku.

Program je vytvořen tak, že při změně tvaru výlisku stačí jen aktualizovat pozice nových bodů, určující trajektorii k opracování výlisku. Program bude fungovat dále bez jakéhokoliv zásahu za podmínky:

- Nově naměřené body byly měřeny při pozici IRB1400 v bodu „pozice vytváření bodů“. Jestliže bylo provedeno měření nových bodů v jiné pozici než „pozice vytváření bodů“, je nutné zadat do procedury VypBody() i tuto novou pozici, od které se budou přepočítávat vzdálenosti bodů výlisku od TCP (viz podkapitola 7.1).
- Počet nově naměřených bodů je stejný jako předtím, jinak je nutné aktualizovat i proměnnou „pocetbodu“ na aktuální počet. Pak bude program opět bez problémů fungovat.

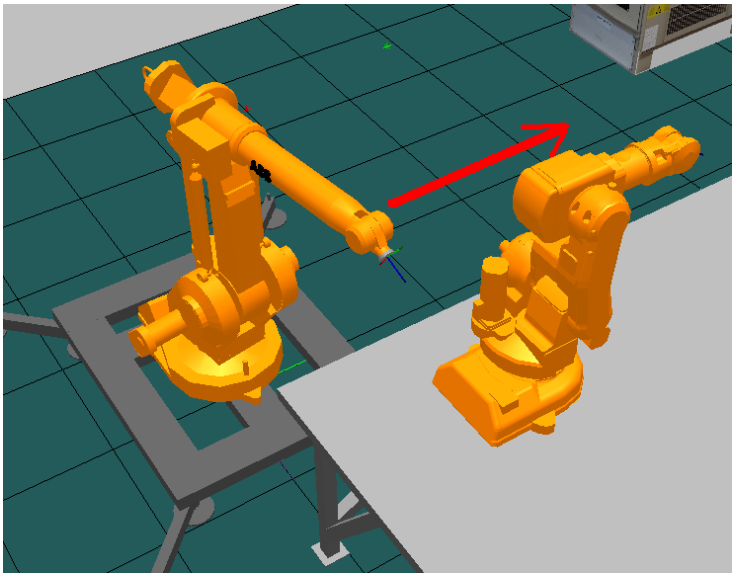
7.3 Druhá úloha (pohyb řízený časem)

V druhé úloze na vzájemnou koordinaci pohybů půjde už o trochu složitější příklad než v předchozí úloze. Zde se jedná o současný pohyb obou robotů při opracovávání plastového výlisku, který má IRB1400 připevněn na svém zápěstí. Opracování bude provádět IRB140 během lineárního pohybu IRB1400 v jedné ose po předem definované trajektorii.

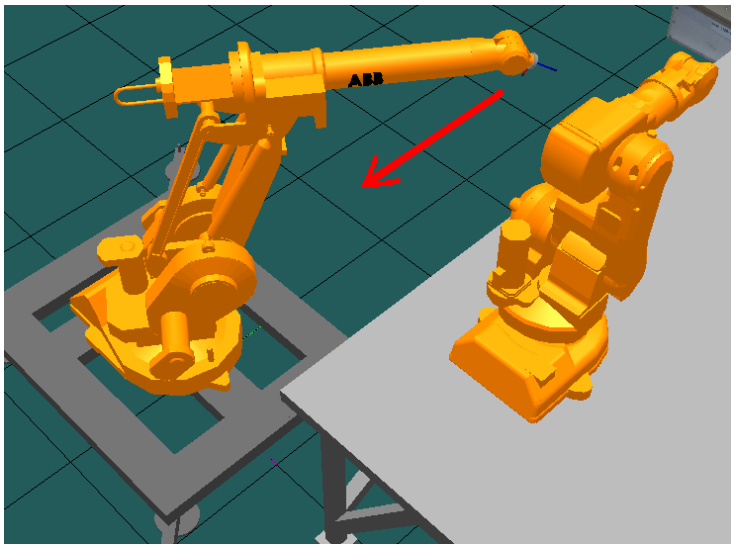
Robot IRB1400 má v programu předdefinované dvě trajektorie, které jsou pro ukázku postačující:

1. Lineární pohyb se bude provádět nejdříve na delší trajektorii směrem od těžiště robota IRB1400 (viz obrázek 18). Poznnámka: na obrázku nejsou znázorněny pracovní nástroje.
2. A poté na kratší trajektorii opět lineární, ale opačným směrem, tedy zpět k těžišti robota IRB1400 (viz obrázek 19). Poznámka: na obrázku nejsou znázorněny pracovní nástroje.

Důležitou částí celého řešení je pohyb robota IRB140 při opracování plastového výlisku. Pomocí programovacího jazyka Rapid lze realizovat řízení rychlosti pohybu robota časem, kde se poupraví argument typu speeddata Rapid 6.2.10. To znamená, že je možné přesně definovat čas, jak dlouho bude trvat robotovi přemístění z bodu A do bodu B.



Obrázek 18: Lineární pohyb od těžiště IRB1400



Obrázek 19: Lineární pohyb k těžišti IRB1400

Z tohoto poznatku budeme také vědět, za jaký čas dokáže projet robot IRB140 celou trajektorii na plastovém výlisku. Jedinou nevýhodou je to, že se robot při objíždění trajektorie nepohybuje konstantní rychlostí, protože čas na projetí je ve všech usecích stejný bez ohledu na vzdálenost.

7.3.1 Podrobný popis řešení

Po startu programu je opět nutné, aby došlo k synchronizaci obou robotů najetím do „Home“ pozice, kde čekají na příslušné I/O signály. „Home“ pozice je definovaná kvůli bezpečnému startu programu obou robotů.

Programy se začnou vykonávat následovně:

Robot IRB1400 se přesune do prvního bodu, který bude zároveň „startovní“ pro lineární pohyb směrem od IRB1400, při kterém bude provedeno opracování plastového výlisku. Současně se IRB140 přesune do čekací přiblížené pozice, kde bude čekat na data ze sériového kanálu od IRB1400.

Po dosažení „startovního“ bodu robotem IRB1400 předá svoje poziční data přes sériový kanál do řídícího systému robota IRB140. Mezi daty, která jsou poslána sériovou linkou, jsou i informace o tom, jakou délku dráhy bude provádět IRB1400 lineárním pohybem, a také jakou rychlostí včetně údaje o směru pohybu plastového výlisku (nejprve od těžiště IRB1400 v programu značeno číslem „1“, nebo zpět k těžišti IRB1400 - v programu značeno „0“ (viz obrázek 18 a 19). Poznámka: na obrázku nejsou znázorněny pracovní nástroje. Z těchto dat si IRB140 - vypočítá čas provádění lineárního pohybu robota IRB1400. Dále se počítá průměrný čas na bod, který je pro všechny stejný, a dále přírůstek změny pozice jednotlivých bodů na plastového výlisku v čase. Díky konstantnímu času pro přesouvání mezi body, určující trajektorii na plastovém výlisku, je velikost přírůstku změny pozice konstantní. Tyto konstantní přírůstky jsou pro jednotlivé body kumulovanými součty. Blíže popsáno v proceduře VypRychVzdal() (viz 7.3.3). Pak se vypočítají nové pozice bodů na startu, ke kterým se buď přičte kumulovaný kladný přírůstek, nebo kumulovaný záporný přírůstek, záleží na směru pohybu plastového výlisku. Po vypočítání všech nových bodů na plastovém výlisku provede IRB140 přiblížení k prvnímu bodu trajektorie. Toto přiblížení je dobré udělat, aby se při spuštění lineárního pohybu plastového výlisku IRB140 nemusel k prvnímu bodu dostávat z velké vzdálenosti a aby byly tak pohyby plynulejší.

Po dosažení přiblíženého bodu se pomocí I/O signálů roboti informují o startu lineárního pohybu IRB1400 a pohybu projíždění trajektorie na výlisku IRB140, který poté začne. Po projetí trajektorií obou robotů se IRB1400 zastaví a čeká, než se IRB140 vzdálí do bezpečné vzdálenosti čekací pozice. Poté se IRB1400 přesune k dalšímu „startovnímu“ bodu a totéž se opakuje pro lineární pohyb směrem zpět

při kratší vzdálenosti. Jediný rozdíl je v tom, že se zde kumulované přírůstky dráhy budou odčítat od startovní pozice, jak již bylo zmíněno, protože pohyb bude směrem k počátku souřadného systému v ose y.

7.3.2 Popis programu pro IRB1400

Nejprve je nutné popsat program robota IRB1400, protože IRB140 čeká na dosažení jeho „startovního“ bodu, jehož pozici mu poté pošle k přepočítání nové pozice trajektorie, určující hrany k opracování na výlisku.

- Při startu programu je smazání I/O bufferu stejné jako (viz 7.2.2).
- Dále je nutná průběžná synchronizace pohybů obou robotů, kterou zajišťují signály pro digitální vstupy a výstupy I/O, vše jako u první úlohy (viz 7.2.2).
- Rychlost lineárního pohybu pro odeslání sériovou linkou do řídícího systému IRB140. Je nutné je zadat v pouze numerickém tvaru a ne ve tvaru speeddata, protože jazyk Rapid neobsahuje funkci pro konvertování speeddata na num. Toto je nutné udělat kvůli matematickým výpočtům IRB140, jinak by vznikla chyba výpočtu a program by se zastavil.

Popis použité procedury:

PROC Pozice()

Tato procedura slouží pro přečtení a odeslání pozice TCP v daném bodu přes sériový kanál druhému robotovi, stejné jako u první úlohy (viz 7.2.2).

7.3.3 Popis programu pro IRB140

- Po startu programu je opět nutné vymazání I/O bufferu, stejné jako u IRB1400.
- Synchronizace pohybů pomocí I/O digitálních vstupů a výstupů.
- Po příchodu I/O signálu, upozorňujícího na otevření sériového kanálu pro čtení a zápis, se spustí procedura ReadChan().

Popis použitých procedur:

PROC ReadChan()

Po otevření sériového kanálu se pomocí instrukce ReadBin začnou načítat příchozí byty v cyklu, stejné jako (viz První úloha (statická) 6.1.2). Po přečtení všech příchozích

bytů se automaticky spustí procedura ReadPosition().

PROC ReadPosition()
Tato procedura začne z došlých bytů z předchozí procedury skládat a vypočítávat data s údaji o pozici a natočení TCP IRB1400 a ukládat je do proměnných, se kterými bude dále pracováno. Poté se spustí procedura VypRychVzdal().

PROC VypRychVzdal()
Z přijatých dat sériovou linkou se jednoduše vypočítá čas, po který bude trvat lineární pohyb plastového výlisku. Dále se z tohoto času podílem počtu bodů na výlisku vypočítá nový čas, který bude použit ve speeddata, tj. průměrný čas na bod. To znamená, že ve stejný čas oba začnou s pohybem a shodně ve stejný čas skončí. A na konec se spočítá kumulovaný přírůstek, tj. přírůstek pozice v ose y jednotlivých bodů. Je to opět násobení rychlostí lineárního pohybu a času na bod.

PROC VypBody()
Procedura obsahuje naměřené pozice bodů na výlisku pomocí TP, které tvoří trajektorii. Tyto pozice byly měřeny při pozici, označené jako „pozice vytváření bodů“. Pomocí cyklu se začnou vypočítávat nejdříve rozdíly naměřených bodů na výlisku od bodu „pozice vytváření bodů“, abychom dostali pouze vzdálenosti bodů od TCP. Poté se nové pozice těchto bodů vypočítají na základě dat o nové pozici a natočení TCP IRB1400, které byly přijaty sériovým kanálem. Pro souřadnici y každého bodu se ještě musí buď přičítat, nebo odečítat kumulované přírůstky, záleží na směru pohybu výlisku. Pro výpočty nových pozic jsou použity transformační rotační matice z kapitoly 5.

Program je opět tvořen tak, že při změně tvaru výlisku stačí jen aktualizovat pozici nových bodů, určující trajektorii na výlisku. Program funguje dále bez jakéhokoliv zásahu za podmínky:

- Nově naměřené body byly měřeny při pozici IRB1400 v bodu „pozice vytváření bodů“. Jestliže bylo provedeno měření nových bodů v jiné pozici než „pozice vytváření bodů“, je nutné zadat do procedury VypBody() i tuto novou pozici, od které se budou přepočítávat vzdálenosti bodů výlisku od TCP (viz podkapitola 7.1).

- Počet nově naměřených bodů je stejný jako předtím, jinak je nutné aktualizovat i proměnnou „pocetbodu“ na aktuální počet. Pak bude program opět bez problémů fungovat.

7.4 Třetí a čtvrtá úloha (pohybující se WObj)

Tato úloha je obdobná té předchozí, ale je vylepšená o vyřešení problému s nerovnoměrnou rychlostí v každém úseku během opracování výlisku robotem IRB140. Opět se zde bude jednat o současný pohyb obou robotů při opracování plastového výlisku, který má IRB1400 připevněý na svém zápěstí. Opracování bude provádět IRB140 během lineárního pohybu IRB1400 v jedné ose po předem definované trajektorii.
Robot IRB1400 má v programu předdefinované dvě trajektorie, které pro ukázku stačí:

1. Lineární pohyb se bude provádět nejdříve na delší trajektorii směrem od těžiště robota IRB1400 (viz obrázek 18). Poznámka: na obrázku nejsou znázorněny pracovní nástroje.

2. A poté na kratší trajektorii opět lineární, ale opačným směrem, tedy zpět k těžišti robota IRB1400 (viz obrázek 19). Poznámka: na obrázku nejsou znázorněny pracovní nástroje.

Důležitou částí celého řešení je vylepšení o konstantní rychlost pohybu robota IRB140 při opracování plastového výlisku. V této úloze se změní postup vypočítávání pozic jednotlivých bodů na výlisku. Nyní nám bude stačit si vypočítat pozice všech bodů na výlisku jen na „startovní“ pozici lineárního pohybu IRB1400. O nové pozice jednotlivých bodů na plastovém výlisku se bude starat programově pohybující se WObj, ke kterému budou body během pohybu vztažené. Pak je potřeba vytvořit pomocný WObj, kterým budeme pohybovat. Tento WObj nám bude reprezentovat pohyb robota IRB1400. V programu je nazvaný MoveWobj.

7.4.1 Podrobný popis řešení

Po startu programu je opět nutné, aby došlo k synchronizaci obou robotů najetím do „Home“ pozice, kde čekají na příslušné I/O signály. „Home“ pozice je definovaná kvůli bezpečnému startu programu obou robotů.

Pak začnou vykonávat programy následující činnosti:
Robot IRB1400 se přesune do prvního bodu, který bude zároveň „startovní“ pro lineární pohyb směrem od IRB1400, při kterém bude provedeno opracování plastového výlisku. Současně se IRB140 přesune do čekací přiblížené pozice, kde bude čekat na data ze sériového kanálu od IRB1400.

Po dosažení „startovního“ bodu robotem IRB1400 předá svoje poziční data přes sériový kanál do řídícího systému robota IRB140. Mezi daty, které jsou poslány sériovou linkou, jsou i informace o tom jaká bude rychlost a směr pohybu plastového výlisku (nejprve od těžiště robota IRB1400 - v programu značeno číslem „1“, nebo zpět k těžišti robota IRB1400 - v programu značeno „0“ (viz obrázek 18 a 19). Poznámka: na obrázku nejsou znázorněny pracovní nástroje. Z těchto dat si IRB140 spočítá nové startovní pozice bodů na plastovém výlisku. Nyní před startem celého pohybu je potřeba nastavit pomocný MoveWobj, ke kterému budou vztažené body po dobu lineárního pohybu. Tento WObj zde bude zastupovat pohyb ramene IRB1400. WObj umístíme do počáteční pozice středu soustavy celé aplikace, kde se nacházejí WOBJ1400 a WOBJ140. Je důležité pro začátek lineárního pohybu MoveWobj umístit na střed soustavy, protože všechny body na výlisku jsou zatím měřeny od stejného středu. Po spuštění lineárního pohybu nám tedy bude stačit jen přepočítávat nové pozice pohybujícího se WObj, ke kterému jsou při pohybu body vztažené. Výpočet MoveWobj je nutné provést vždy před pohybovou instrukcí MOVE během opracování výlisku. Tím se zajistí, že daný bod na výlisku bude vztažený k právě aktuální pozici WObj. Pomocí programovacího jazyka Rapid lze měnit poziční data WObj, od kterého se vypočítávají vzdálenosti jednotlivých bodů. Podrobnější informace o výpočtu WObj je v proceduře PocWobj() (viz 7.4.3).

Tento způsob řešení je velice elegantní, ale je potřeba ještě ošetřit jeden malý problém, které toto řešení obsahuje. Naměřené body, tvořící trajektorii na plastovém výlisku, jsou od sebe různě vzdálené, vždy tak, aby jejich počet byl co nejmenší, ale současně aby bylo zajištěno objetí všech hran, které je potřeba opracovat. Pro body od sebe vzdálenější na plastovém výlisku by tato metoda nebyla příliš přesná, protože robot získá pozici cílového bodu ještě dříve než se začne přesouvat vzhledem k MoveWobj. Pak by hrozilo, že po dojetí na cílový bod by reálný bod byl už v jiné pozici. Abychom předešli těmto nepřesnostem, je potřeba vzdálenosti mezi všemi body trajektorie na výlisku rozdělit na několik kratších úseků stejné délky, což je popsáno v proceduře Generuj() (viz 7.4.3). Za každým z těchto kratších úseků se

aktualizuje pozice pohybujícího se MoveWobj, což nám velice zpřesní pohyb.

Program je ošetřen i pro ten případ, že robot IRB1400, nesoucí plastový výlisek, se dostane na konec své lineární trajektorie dříve, než projede svoji trajektorii na plastovém výlisku robot IRB140, provádějící opracování. Toto ošetření u předešlé úlohy nebylo nutné, protože se časy pohybu při opracování obou robotů shodovaly.

7.4.2 Popis programu pro IRB1400

Nejprve je nutné popsat program robota IRB1400, protože IRB140 čeká na dosažení jeho prvního bodu, jehož pozici mu poté pošle k přepočítání nové pozice trajektorie, určující hrany k opracování na výlisku.

- Při startu programu je nutné smazat I/O buffer.
- Dále je nutná průběžná synchronizace pohybů obou robotů, kterou zajišťují signály pro digitální vstupy a výstupy, vše jako u první úlohy (viz 7.2.2).
- Rychlost lineárního pohybu výlisku pro odeslání sériovou linkou. Je nutné zadání v numerickém tvaru.

Popis použité procedury:

PROC Pozice()

Tato procedura slouží pro přečtení a odeslání pozice TCP v daném bodu přes sériový kanál druhému robotovi (stejné jako u první úlohy (viz 7.2.2)).

7.4.3 Program pro IRB140

- Po startu programu je opět nutné vymazání I/O bufferu, stejné jako u IRB1400.
- Synchronizace pohybů pomocí I/O digitálních vstupů a výstupů.
- Po příchodu I/O signálu, upozorňující na otevření sériového kanálu pro čtení a zápis, se spustí procedura ReadChan().

Popis použitých procedur:

PROC VypBody()

Procedura obsahuje naměřené pozice bodů na výlisku pomocí TP, které tvoří trajektorii na výlisku. Tyto pozice byly měřeny v bodu „pozice vytváření bodů“. Pomocí cyklu se začnou vypočítávat nejdříve rozdíly naměřených bodů na výlisku od bodu

„pozice vytváření bodů“, abychom dostali pouze vzdálenosti bodů od TCP. Poté se nové pozice těchto bodů vypočítají na základě dat o nové pozici a natočení TCP IRB1400, které byly přijaty sériový kanálem. Pro výpočty nových pozic jsou použity transformační rotační matice (viz kapitola 5).

PROC Generuj()
Tato procedura nejdříve vypočítá Eukleidovy vzdálenosti, ze které získáme vzdálenosti mezi všemi sebou sousedícími body na výlisku.
Eukleidovská vzdálenost mezi body A = $[x_1, y_1, z_1]$, B = $[x_2, y_2, z_2] \in \mathrm{R}^3$ **je dána vzorcem:**

$$d = \sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1))^2} \tag{8}$$

d - je vzdálenost mezi body A a B

Dále se s jednotlivými vzdálenostmi počítá následovně:
Je potřeba jednotlivé délky mezi body rozdělit na několik menších částí, aby byla zajištěna větší přesnost dosažení bodu po výpočtu MoveWobj. V programu je zvoleno generování bodu každých 5 mm, které postačují pro dostatečnou přesnost.

Obecný vzorec pro výpočet pozice bodu C mezi body A, B:

- A = $[x_1, y_1, z_1]$, B = $[x_2, y_2, z_2]$
- L_{AB} - Délka přímky mezi body A, B
- L_{AC} - vzdálenost bodu C od A
- platí podmínka $0 \leq L_{AC} \leq L_{AC}$

$$C = \left(x_1 + (x_2 - x_1) \cdot \frac{L_{AC}}{L_{AB}}, y_1 + (y_2 - y_1) \cdot \frac{L_{AC}}{L_{AB}}, z_1 + (z_2 - z_1) \cdot \frac{L_{AC}}{L_{AB}}\right) \tag{9}$$

(Výpočet souřadnic bodu C, psáno vektorově.)

PROC PocWobj()

Tato procedura je velice důležitá pro pohyb IRB140. Po spuštění lineárního pohybu se stará o výpočet nové pozice MoveWobj. V proceduře se vždy získá čas z pomocného časovače a díky známé rychlosti IRB1400 můžeme také spočítat přesnou pozici pohybující se ho WObj, který zde vytváří programový pohyb ramene IRB1400. Tato procedura se volá vždy před instrukcí MOVE, provádějící pohyb na plastovém výlisku během opracování.

Opět je program vytvořen tak, že při změně tvaru výlisku stačí pouze aktualizovat pozici nových bodů, určující trajektorii na výlisku. Program funguje dále bez jakéhokoliv zásahu za podmínky:

1. Nově naměřené body byly měřeny při pozici IRB1400 v bodu „pozice vytváření bodů“. Jestliže bylo provedeno měření nových bodů v jiné pozici než „pozice vytváření bodů“, je nutné zadat do procedury VypBody() i tuto novou pozici, od které se budou přepočítávat vzdálenosti bodů výlisku od TCP (viz podkapitola 7.1).

2. Počet nově naměřených bodů je stejný jako předtím, jinak je nutné aktualizovat i proměnou „pocetbodu“ na aktuální počet. Pak bude program opět bez problémů fungovat.

7.4.4 Alternativní řešení pomocí systémového přerušení

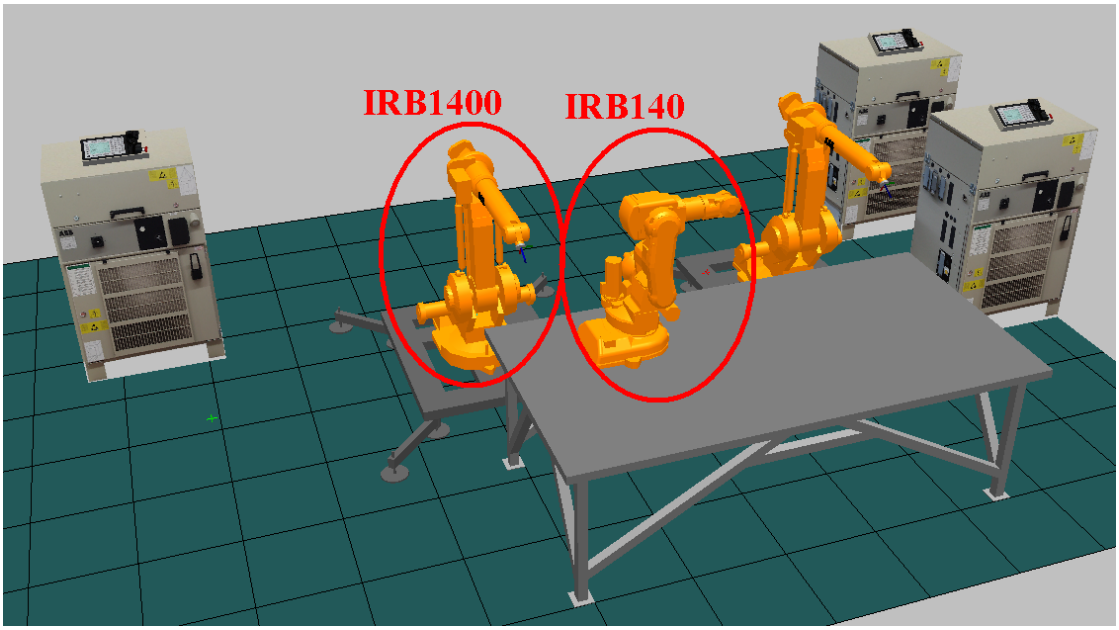
Předchozí úlohu je možné také řešit pomocí systémového přerušení „interrupt“. Přerušení, o které se zde jedná, bude vyvoláno na základě uplynutí určitého časového úseku. Při přerušení se automaticky spustí „trap rutina“, což jsou námi zadané příkazy, které se mají v tomto přerušení vykonat. Díky tomuto řešení můžeme vyvolat zadané instrukce kdykoliv za běhu programu po uplynutí námi zadaného časového intervalu. Toto je výhoda před procedurami, které se spustí vždy, když na ně přijde řada při zpracování instrukcí.

Pro předešlé řešení se pouze upraví program pro robota IRB140 tak, že se nahradí procedura PocWobj() (viz 7.4.3) za „trap rutinu“ s názvem TRAP pocitej. Dále je také nutné implementovat do hlavního programu instrukce o přerušení (viz jazyk Rapid 6.2.11). Instrukce v proceduře PocWobj() a Trap pocitej jsou téměř

shodné jen s tím rozdílem, že v „trap rutině“ není potřeba používat časovač, když víme, po jakém časovém intervalu se spustí přerušení.

Velikost časového úseku, po kterém se spustí přerušení, můžeme v programu zadat dle potřeby v intervalu od minimální hodnoty 0.05 sekundy. Ovšem pro cyklické přerušení je doporučeno používat čas minimálně 0.25 sekundy. Kvůli tomuto omezení minimální hodnoty cyklického časového přerušení je toto řešení v některých úsecích trajektorie méně přesné než předešlé řešení při stejné rychlosti opracovávaného výlisku. Jedinou možností, jak tuto nepřesnost minimalizovat, je zmenšit rychlost posuvu výlisku. Tím dojde ke zmenšení chyby, protože reálné body na výlisku se díky menší rychlosti budou jen málo vzdalovat od bodů, vypočítávaných pomocí systémového přerušení. Nové pozice těchto bodů jsou počítány opět vzhledem MoveWobj, který vypočítává svoji novou pozici každých 0.25 sekundy během přerušení.

Všechny zde popsané programy pro oba roboty jsou k nahlédnutí v přiloženém CD včetně videa všech úloh.



Obrázek 20: Model laboratoře inteligentních robotů vytvořený v RobotStudiu 4.0

Závěr

Výsledkem této práce je čtveřice úloh, řešících opracování plastového výlisku při vzájemné koordinaci pohybů obou robotů. Úlohy byly řešeny pomocí programovacího jazyka Rapid a prostředí ProgramMaker. Veškeré programy pro robota IRB140, který prováděl opracování plastového výlisku, byly vytvořeny pro obecný tvar dle zadání práce. Všechny způsoby řešení byly úspěšně odzkoušeny na reálném systému v laboratoři inteligentních robotů v učebně S15, budova A. Pro pohyb plastového výlisku bylo použito lineárního pohybu v jedné ose s možností natočení v ose z. Všechny zde popsané úlohy je možné použít ve výrobních procesech, typicky například pro nanášení lepidla nebo sváření.

Při řešení bylo dále prostudováno a popsáno:

- popis a postup při kalibraci použitých robotů,
- metoda získání vzájemných pozic robotů, která je použitelná pro obecné získávání vzájemných pozic robotů za uvedených podmínek (viz kapitola 3),
- vysvětlení a popis vybraných instrukcí a funkcí jazyka Rapid, potřebných pro realizaci této práce. Tyto funkce jsou typické pro komunikaci sériovým kanálem, matematické výpočty a dále získání souřadnic pozice ramene robota v prostoru atd.,
- některé metody pro kinematické výpočty pozic bodů vázaných na pohyb ramene robota,
- nastavení sériového kanálu pro správnou komunikaci mezi oběma použitými řídicími systémy robotů.

Při řešení úloh byl kladen důraz na přesnost pohybu hrotu, kterým bylo prováděno opracování plastového výlisku. Přesto se od sebe jednotlivá řešení liší přesností a způsobem pohybu během opracování:
V prvním řešení je přesnost pohybu hrotu po trajektorii na výlisku nejlepší, protože výlisek je statický a nepohybuje se. Toto je jediné statické řešení z uvedených řešených úloh.
Druhé řešení opět přináší velikou přesnost, ale pohyb hrotu se nepohybuje konstantní rychlostí po trajektorii na výlisku. Pro praktické využití by to znamenalo,

že by musela být řízena rychlost nanášení lepidla v závislosti na rychlosti pohybu nanášecího hrotu.
Třetí řešení přináší velkou přesnost pohybu současně i s konstantní rychlostí pohybu po trajektorii na plastovém výlisku.
Čtvrté řešení je alternativou, vycházející z třetího řešení. Přesností pohybu ovšem nepřekoná předešlé řešení. Dále je nutné používat menší rychlosti pohybu výlisku z důvodu vyšší přesnosti.

Třetí způsob řešení společně s prvním je pro nanášení lepidla nejlepším řešením s ohlédnutím na složitost implementace a přesnost pohybu při opracování. V příloze na CD je možné shlédnout video soubor zachycující různé způsoby opracovávání plastového výlisku, které byly odzkoušeny na reálném systému.

Při řešení některých částí jednotlivých úloh bylo využito i simulačního softwaru RobotStudio 4.0, kterým disponuje univerzita. Tato verze ovšem nepodporuje více než jeden spuštěný řídicí systém robotů v jednom okamžiku a nebylo možné simulovat spolupráci v jednotlivých úlohách během řešení. Vždy muselo být využito reálného systému pro odzkoušení funkčnosti a správnosti řešení. Během této práce se narazilo také na problém stálé kalibrace robotů. Údaje o kalibraci robotů jsou uloženy v řídicích systémech, které po vypnutí udržují záložní baterie. Z důvodu vybitých záložních baterií byly vždy tyto informace ztraceny. Po vypnutí se muselo znovu kalibrovat. Bohužel na kalibraci, která určuje přesnost pohybu robota, je vázaná i další činnost. Naměřené body, určující trajektorii na plastovém výlisku, musely být znovu naměřeny a aktualizovány v programu.

Možností jak navázat na tuto práci je vylepšit a rozšířit popsané metody. Úlohy lze doplnit o možnost pohybu plastového výlisku ve více osách během opracování a dále natočení nástroje při opracování pro obecnou orientaci vůči pevnému souřadnému systému. Dále lze rozšířit a popsat možnosti jazyka Rapid, který obsahuje ještě mnoho jiných funkcí a instrukcí.

Literatura

[1] RobotStudio 3.1 User's Guide. *Firemní dokumentace k programu RobotStudio, Švédsko 2004.*

[2] Rapid Reference Manual. *Firemní dokumentace k programu Rapid, Švédsko 2004.*

[3] Product Manual IRB 140. *Firemní dokumentace k robotu IRB 140, Švédsko.*

[4] Product Manual IRB 1400. *Firemní dokumentace k robotu IRB 1400, Švédsko.*

[5] GREPL, Robert. *Modelování mechatronických systémů v Matlab SimMechanics*. 1. vyd. Praha: Ben, 2007. 143 s. ISBN 978-80-7300-226-8.

[6] Stavitel Pyramid. *Výukový materiál*. TUL FM, 13 s. ČR.

[7] FLODRMAN, Petr a BLAVKA, Karel. *Dokumentace elektrického zapojení robotů, a jejich nástrojů*. TUL FM, 27 s. ČR.

[8] PŠENIČKA, Radek a TVRZNÍK, Michal. *Demonstrační aplikace v RobotStudiu*. Ročníkový projekt, TUL FM 2007, 40 s. ČR.

[9] VÁVRA, Václav a LOSOS, Zdeněk. *Odvození transformačních matic pro různé typy rotace* [on-line]. Aktualizováno: 11.1.2007. [cit. 19. 2. 2008]. Dostupné na: <http://mineralogie.sci.muni.cz/kap_1_3_symetrie/rotace_priklad.htm>.

[10] KAVAN, Ladislav. *Kvaterniony a interpolace* [on-line]. ČVUT [cit. 18.2. 2008]. Dostupné na: <http://www.cgg.cvut.cz/~kavanl1/Y36PHA>.

Přílohy

1. **CD**

   Obsah CD:

   - Literatura
     - User's Guide 4.0.80.pdf
     - Rapid Reference 4.0.80.pdf
     - Product Manual IRB 140 3HAC 7564-1 rev.1 M2000.pdf
     - Product Manual IRB 1400 3HAC 7617-1 rev. 1 M2000.pdf
     - Stavitel pyramid.pdf
     - Demonstrační aplikace v RobotStudiu.pdf
     - Dokumentace elektrického zapojení robotů, a jejich nástrojů.pdf
   - Software RobotStudio 4.0
     - RobotStudio 4.0 CD Image.exe
     - dptech.lic
   - Programy řešených úloh
     - 1uloha IRB140.prg
     - 1uloha IRB1400.prg
     - 2uloha IRB140.prg
     - 2uloha IRB1400.prg
     - 3uloha IRB140.prg
     - 3uloha IRB1400.prg
     - 4uloha IRB140.prg
     - 4uloha IRB1400.prg
   - Video řešených úloh
     - video.wmv