České vysoké učení technické v Praze
Fakulta elektrotechnická

DIPLOMOVÁ PRÁCE

Řízení rychlých servopohonů

Autor: Iľja Grudinin

Katedra řídicí techniky

Poďakovanie

Táto práca by nevznikla bez pomoci, podnetov a pripomienok vedúceho diplomovej práce Ing. Pavla Burgeta. Taktiež chcem poďakovať Ing. Romanovi Vančurovi za cenné rady poskytnuté v najťažších chvíľach spracovania diplomovej práce.
Ďalej by som rád poďakoval svojej rodine a priateľom za morálnu a materiálnu podporu počas celého štúdia na FEL ČVUT v Prahe.

Prehlásenie

Prehlasujem, že som svoju diplomovú prácu vypracoval samostatne a použil som iba podklady (literatúru, projekty, SW atď.) uvedené v priloženom zozname.

V Prahe dňa 18. 1. 2008

Ilja Grudinin

Abstrakt

Cieľom tejto práce je rozšíriť možnosti súčasného žonglovacieho stroja o schopnosť žonglovania s tromi guľami. Ďalšou časťou diplomovej práce je vytvorený medzistupeň v podobe troch modelov, na ktorých si študenti majú možnosť osvojiť základy programovania systému zostaveného z PLC, Acoposa a synchrónneho servomotora. Pre tieto modely je vytvorených päť cvičení, ktoré prevedu študentov od založenia projektu až po vytvorenie vačkového automatu. V súvislosti s žonglérom bola zrealizovaná analýza pohybu a navrhnuté technické úpravy v podobe nutného doplnenia prevodovky a programove úpravy na doplnenie tretej osi realizovanej lineárnym motorom. V praxi elektronické vačky úspešne zamieňajú mechanické vačky pre ich flexibilitu programovania a skoro žiadnu opotrebovateľnosť.

Abstract

The goal of this thesis is to extend possibilities of existing juggling machine to machine juggling with three billiard balls. The next part of the diploma thesis is the creation of an intermediate stage of three exemplars, which will be useful for students to gain basics programming skills at system made up of PLC, Acopos and synchronous servomotor. Five lessons were created for this purposes, that should guide students from setting a new project till creation of a cam profile automat. In the context of juggling machine the moving analysis and the proposal for technical changes to supplementation of gearbox there have been carried out. There was also programme changes in completion of third axis of the linear motor. In practise electronical cam profiles commonly replace mechanical cam profiles because of more programing flexibility and superior durability.

České vysoké učení technické v Praze - Fakulta elektrotechnická

Katedra řídicí techniky *Školní rok:* 2006/2007

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Il'ja Grudinin

Obor: Technická kybernetika

Název tématu: Řízení rychlých servopohonů

Zásady pro vypracování:

1. Seznamte se s možnostmi realizace elektronických vaček na systému B&R a Ethernet Powerlink.
2. Realizujte studentské pracoviště pro demonstraci vačkových mechanismů.
3. Rozšiřte stávající model, který přehazuje mezi dvěma držáky kulečníkové koule, o třetí vertikální osu. Pamatujte na možnost automatického sbírání koulí z podlahy modelu.
4. Analyzujte dynamiku pohybu dvou os, pokud si navzájem přehazují tři koule a navrhněte programové řešení. Pokud to mechanická konstrukce modelu dovoluje, program realizujte, případně navrhněte mechanické úpravy modelu.
5. Doplňte model o další PLC, které bude dohlížet na správný a bezpečný pohyb jednotlivých os. Zvažte použití bezpečného PLC.
6. Připravte rozhraní pro vzdálený přístup k modelu.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí diplomové práce: Ing. Pavel Burget

Termín zadání diplomové práce: zimní semestr 2006/2007

Termín odevzdání diplomové práce: leden 2008

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

L.S.

prof. Ing. Zbyněk Škvor, CSc.
děkan

V Praze dne 21.02.2007

Obsah

Zoznam obrázkov

Zoznam tabuliek

Kapitola 1

Úvod

Súčasné trendy sa historicky nezmenili. Tak ako v priebehu celého minulého storočia, tak aj teraz moderným trendom zostáva zvyšovať produkciu zrýchľovaním vyrobného procesu, znižovať náklady a vytlačovať (obmedzovať) ľudský faktor z procesu výroby. Synchrónny servomotor je ukážkovým príkladom presnosti a rýchlosti. Jeho kombináciou s inteligentným servopohonom a výkonným PLC je možné vytvoriť systém schopný vytlačiť z výroby kolosy mechanických neflexibilných vačkových automatov. Precízne polohovanie a vysoké rýchlosti využivajúce cyklickú vysokorýchlostnú sieť Ethernet Powerlink spĺňajú aj tie najnáročnejšie požiadavky. V súvislosti s týmito prednosťami vznikol v našich laboratóriach žonglovací mechanizmus určený pre výuku študentov. Cieľom mojej práce bude rozšírenie jeho možností na žonglovanie s tromi biliardovými guľami. Aby tetno žonglér bol pre študentov prístupnejši tak mojou ďalšou úlohou bude zostrojenie niekoľkých študentských pracovišť pre názornu demonštráciu vačkových mechanizmov.

Kapitola 2

Teoretický rozbor

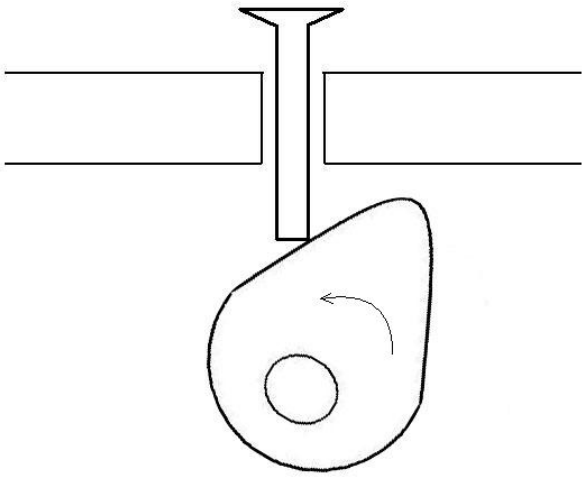
2.1 Vačky

2.1.1 Charakteristika vačiek

Vačka je obvykle vajcovitého tvaru. Tvarom vačky sa dá "mechanicky"naprogramovať doba a výška zdvihu v závislosti na jej natočení. V minulosti boli mechanické vačky využívane vždy v harmonickom komplexnom pohybe sekvencií cyklických strojov. Táto metóda centrálne riadenej osy zabezpečovala synchrónne operácie všetkých mechanických vačiek. Rýchlosť tohoto centrálneho hriadeľa určovala rýchlosť produkcie celého stroja. Najrozšírenejším mechanickým vačkovým automatom je skupina vačiek v aute, ktorá otvára sacie ventily v štvordobom spalovacom motore obr. 2.1. Aplikáciu vačiek dnes môžeme nájsť v baliacom a drevospracujúcom priemysle, v tlačiarenských technológiach a v mnoho iných odvetviach. Napríklad vačkový stroj na výrobu pružin má v oblasti zhybu drôtu niekoľko vačiek so zdvihátkom, na ktoré je napojený nástroj pre ohýbanie drôtu alebo nôž pre odseknutie a ukončenie pružiny. Avšak tento rozšírený princíp mechanických vačiek ma niekoľko obmedzení :

- nedostatok flexibility
- opotrebovanie mechanických komponentov
- plytvanie času pri modifikácii vačkových profilov
- nutnosť hlavného pohonu

S narastajúcimi schopnosťami elektronických riadiacích systémov vznikla možnosť kopírovať funkciu mechanických vačiek, takzvanými elektronickými vačkami. Týmto vačkový pro-



Obrázok 2.1: Ukážka vačky v motore

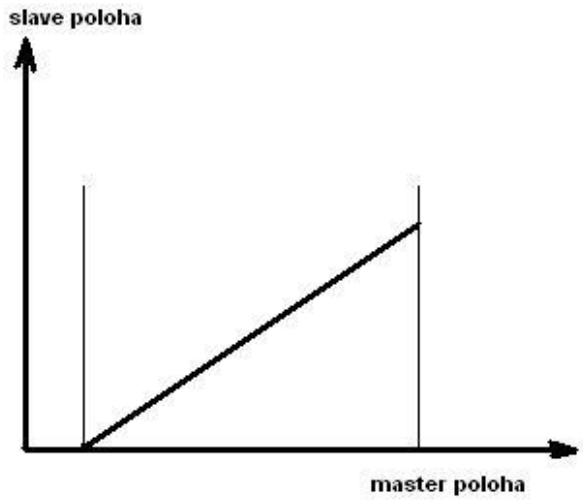
fil už nie je viac konštruovaný špecifickými mechanickými vačkami, ale elektronickými vačkami vytvorenými v Automation Studiu a nahranými priamo v riadiacom systéme. Keďže riadiaci systém môže uchovávať viacero vačiek, ich zámena sa stáva jednoduchým úkonom nahrania novej vačky do acoposu. Zaujímavou výhodou systému elektronických vačiek je schopnosť skrátiť dobu cyklu. Pri vyšších rýchlostiach motoru mechanických vačiek sa stáva, že zdvihatko sa vychýli od vačkového profilu a tým sa stratí presnosť. U elektronických vačiek sa to stať nemôže.

2.1.2 Elektronické vačky

2.1.2.1 Elektronické prepojenie pohonov

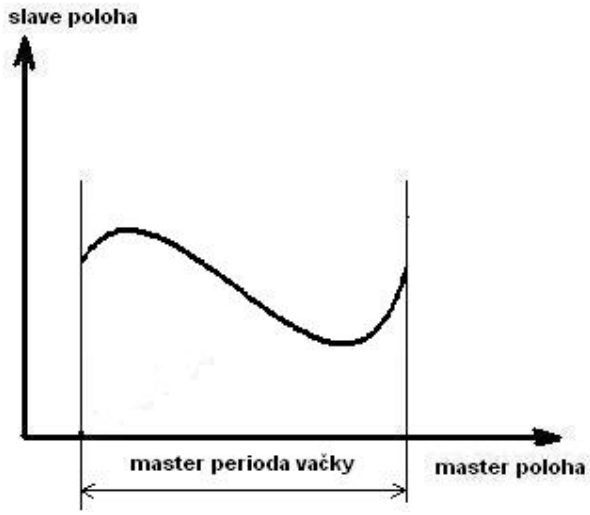
Elektronické prepojenie pohonov vyvoláva predom definovaný synchronizovaný pohyb. Napríklad pohon A je prepojený s pohonom B pomocou pozície. Tým sa myslí, že kým sú pohony aktívne prepojené, tak pohon A musí prispôsobiť svoju polohu špecifickému správaniu polohy pohonu B. V tomto prípade je pohon B master (špecifická referenčná poloha) a pohon A je slave (jeho poloha je závisla na polohe mastra). Na obr. 2.2 je možné vidieť lineárny vzťah polohy medzi slave a master pohonom.

Keď sa master signál mení rovnomerne (pohyb master osy prebieha konštantnou rýchlosťou), rýchlosť slave osy je taktiež konštantná pri tejto špecifikácii. Tento mechanizmus sa využíva pri elektronických prevodovkách. Avšak pri elektronických vačkach



Obrázok 2.2: Linearne prepojenie pohonov

vzťah polôh nemá byť lineárny, ale musí sa vytvoriť polohový profil obr. 2.3.

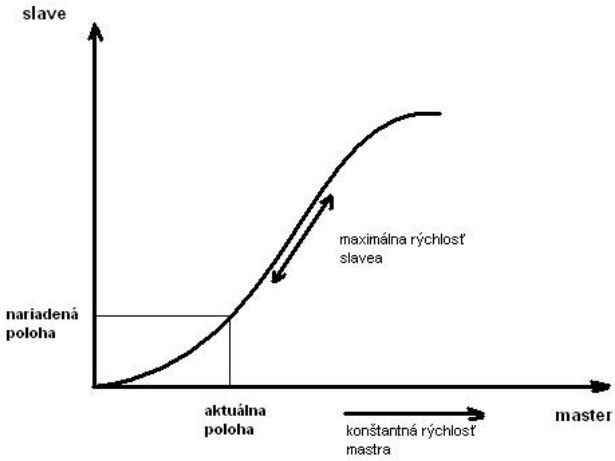


Obrázok 2.3: Profil dynamického polohovania

2.1.2.2 Vačkový profil

Knižnica ACP10_MC poskytuje funkčné bloky pre riadenie nevyhnutných úloh vačkového profilu. Vačkový profil priraďuje príslušnú hodnotu slave polohy každej hodnote master polohy v rámci definovaného rozsahu. Slave pohon musí následovať tento profil, pokiaľ

sú pohony aktívne prepojené. Poloha mastra je konvertovaná na príslušnú polohu slavea prostredníctvom vačkového profilu. To umožňuje mastru pohyb v oboch smeroch. Taktiež to znamená, že hodnoty rýchlosti a zrýchlenia pre slave pohon sú tiež odvodené od rýchlosti a zrýchlenia hodnôt mastra. Preto musí byť overené, že slave pohon môže akceptovať všetky možne rýchlosti a zrýchlenia, ktoré môžu nastať počas riadenia. Kritický rozsah rýchlosti sa vyskytuje v najväčšom spáde vačkového profilu obr. 2.4.



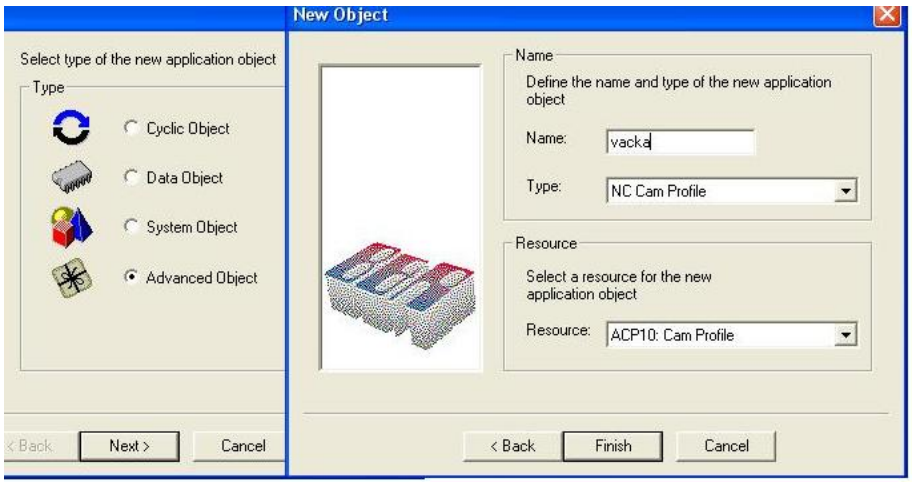
Obrázok 2.4: Úsek profilu s maximálnou rýchlosťou slavea

2.1.3 Vytváranie vačkoveho profilu

Pre vytvorenie vačkového profilu slúži v Atomation Studiu vačkový profilový editor. Vytváranie vačky v editore je možné uskutočniť až po jej vložení do projektu.

2.1.3.1 Vloženie vačky do projektu

Vačkovy profil sa vytvára ako NC software object. V projekte si klikneme na *Insert→New Object*. V dialógovom okne, ktoré sa nám naskytne zaškrtneme *Advanced Object*. Po potvrdení sa musí zvoliť príslušný NC data object. V našom prípade sme si zvolili typ": *NC Cam Profile* a Resource: *ACP10:Cam Profile* obr. 2.5, názov vačkového profilu bude vacka.



Obrázok 2.5: Vloženie vačkového profilu do projektu

Po stlačení tlačítka *Finish*, vačka sa automaticky vloži do projektu a otvorí sa vačkový profilový editor.

2.1.3.2 Editor vačkového profilu

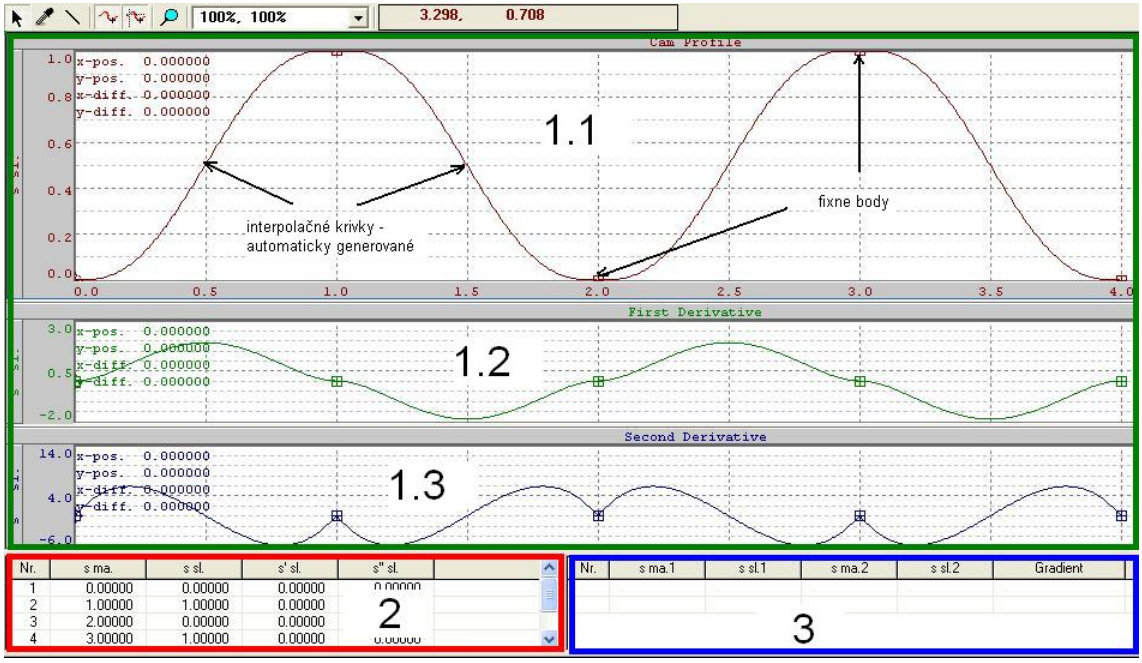
Na obr. 2.6 sa v sekcii 1 nachádza vačkový profil. Pričom v 1.1 je zadefinovaná poloha slavea voči polohe mastra. V sekcii 1.2 je to rýchlosť a v sekcii 1.3 je zrýchlenie. V 2. sekcii sa definujú fixné body profilovej charakteristiky. Tretia sekcia je pole pre zadávanie lineárneho úseku charakteristiky.
Teraz môžeme vkládať fixné body alebo synchronizačné sekcie. Jednotlivé prvky bude editor automaticky spájať interpolačnými krivkami. Fixný bod je bod vo vačkovom profile, kde slave poloha musí mať preddefinovanú závislosť na master polohe. Fixné body môžeme vkládať troma spôsobmi :

- do tabuľky fixných bodov v sekcii 2 na obr. 2.6
- pomocou dialógového okna, ktoré otvoríme pomocou *insert→Fixpoint*
- využitím počitačovej myši priamo na ploche charakteristiky

Popisky v 2. sekcii na obr. 2.6 sú vysvetlené v tab. 2.1

Synchronná sekcia vo vačkovom profile zabezpečuje konštantnú rýchlosť slavea pri konštantnej rýchlosti master osy. Inými slovami vačkový profil obsahuje priamku (ktorá je podobná elektronickej prevodovke) obr. 2.7.

Tuto synchronizačnú sekciu môžme tiež vkládať troma spôsobmi :



Obrázok 2.6: Vačkový editor

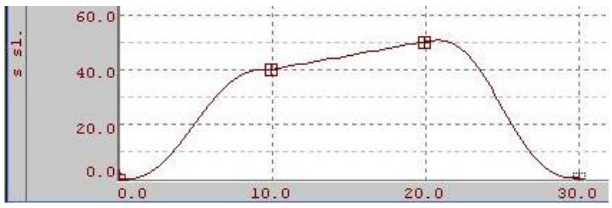
- do tabuľky synchronizačných sekcii v poli číslo 2 na obr. 2.6
- pomocou dialógového okna, ktoré otvoríme pomocou *insert→Synchronous Section*
- využitím počitačovej myši priamo na ploche charakteristiky

Parametre synchronizačnej sekcie sú popísane v tab. 2.2

Interpolačné krivky medzi fixnými bodmi alebo medzi synchronizačnými sekciami a fixnými bodmi sú prepočítavané vačkovým editorom a okamžite medzi ne umiestňované. Tvar tejto krivky sa dá prispôsobovať potrebám užívateľa. Pre zmenu vzhľadu tejto charakteristiky si klikneme na ňu pravým tlačítkom myši a vyberieme *Curve properties*.

Názov stĺpca	Vysvetlenie názvu
Nr.	Vzostupne radené čísla fixných bodov
s ma.	Pozícia fixného bodu na master ose
s sl.	Pozícia fixného bodu na slave ose
s' sl.	Prvá derivácia funkcie vačkového profilu vo fixnom bode
s" sl.	Druhá derivácia funkcie vačkového profilu vo fixnom bode

Tabuľka 2.1: Parametre fixného bodu



Obrázok 2.7: Synchrónna sekcia vačky

Názov stĺpca	Vysvetlenie názvu
Nr.	Vzostupne radené čísla synchronných sekcii
s ma. 1	Štartovacia pozícia synchronnej sekcie na master ose
s sl. 1	Štartovacia pozícia synchronnej sekcie na slave ose
s ma. 2	Koncová pozícia synchronnej sekcie na master ose
s sl. 2	Koncová pozícia synchronnej sekcie na slave ose
Gradient	Sklon synchronnej sekcie alebo pomer prevodu

Tabuľka 2.2: Parametre synchroniza4nej sekcie

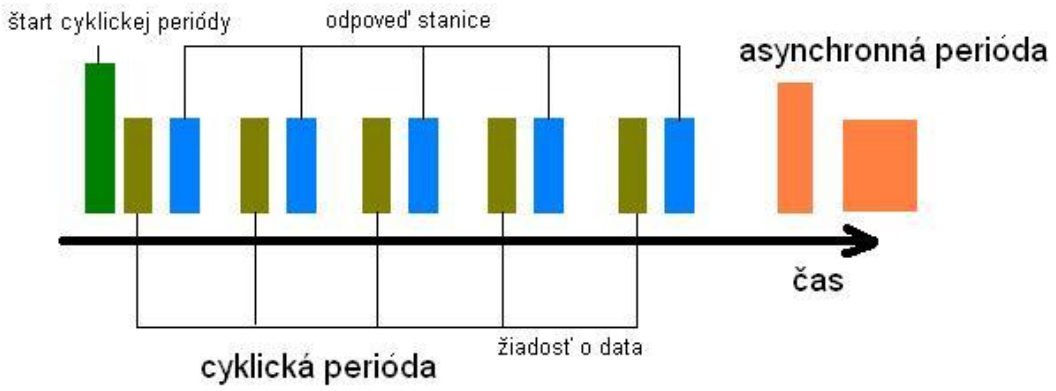
2.2 Ethernet Powerlink

Ethernet Powerlink je deterministický real-time protokol pre štandard Ethernet, ktorý bol vyvinutý spoločnosťou B&R. Je to open protokol, ktorý zastrešuje EPSG (Ethernet POWERLINK Standardization Group). Je to jeden z relevantných štandardov v Európe pre použitie Ethernetu v automatizácií. Ethernet Powerlink je komunikačný profil pre rozšírenie IEEE 802.3 (Fast Ethernet) v automatizácii. Najväčšou zaujímavosťou Ethernetu je schopnosť využiť výhodu média od úrovne senzorov až po úroveň riadiaceho centra. Pwerlink:

- Fast Ethernet podľa IEEE802.3u 100BASE-TX ako prenosové médium.
- použitie štandardných sieťových uzlov (HUB) a štandardných káblov.
- deterministický prenos cyklických dat s minimálnym cyklom 200 μs.
- časová neistota pri synchronizácii komunikujúcich zariadení je menšia ako 1 μs.
- použitie štandardných IP protokolov (TCP, UDP, HTTP).

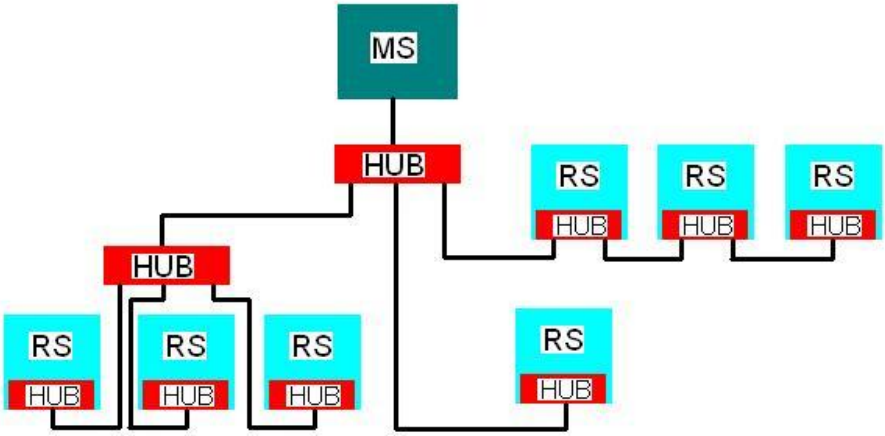
2.2.1 Operačný režim Powerlinku

V Powerlinku sú všetky prenášané dáta kontrolované manažérom siete (MS), aby sa predišlo kolíziam v sieti Ethernet. Sieťové zariadenia, riadené systémy (RS), vysielajú dáta len vtedy, keď im to nariadi manažér. Cyklus Powerlinku je rozdelený do 2 časových úsekov. Rámec "štart cyklu"je vyslaný manažérom ako broadcast správa všetkým staniciam, čím sa striktne deterministicky určí štart cyklickej periódy obr. 2.8.



Obrázok 2.8: Cyklus Powerlinku

V tomto cykle sa uskutočňuje izochronná výmena dát. Manažér siete zasiela žiadosť o dáta (Poll Request) každej stanici jednotlivo za sebou. Adresovaná stanica následne v krátkom čase odpovedá datami (Poll Response). Odpoveď je zaslaná ako multicast správa a nedostane ju len manažer siete ale aj všetky stanice v sieti, ktoré túto správu očakávajú. Následne pokračuje asynchrónna perióda. Manažér opäť vysiela každej stanici samostatne unicast (Invite) rámec. Stanice môžu napríklad vyslať v tejto časti cyklu IP rámec. Trvanie cyklickej izochrónnej periódy a asynchrónnej periódy sa dá nakonfigurovať. Na obr. 2.9 je zobrazená mix štruktúra Ethernet Powerlinku. Na pravej strane sú riadené jednotky (RS) zapojené líniovou štruktúrou a na ľavej strane je použitá hviezdicová štruktúra využivajúca HUB-y. Manažér siete (MS) dokáže zvládnuť maximálne 10 prvkov zapojených do líniovej štruktúry. Celkovo celá sieť nesmie prekročiť 253 uzlov (staníc).



Obrázok 2.9: Ethernet Powerlink - mix štruktúra

2.3 CAN

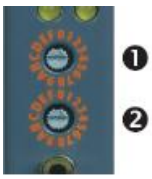
2.3.1 Úvod do systému CAN

Ku komunikácii medzi PLC a Acoposom som využil modul AC110 obr. 2.10, ktorý som nainštaloval do tela Acoposu. Tento modul je vybavený rozhraním *CAN*. Toto rozhranie sa dá využiť aj na pripojenie dodatočného displeja, v prípade žeby sa nejednalo o PP35.

Obrázok 2.10: Modul AC110 rozhrania CAN

Controller Area Network (CAN) je sériový komunikačný protokol, ktorý bol pôvodne vyvinutý firmou Bosch pre poučitie v automobiloch. Väčšina výrobcov integrovaných obvo-

dov implementovali podporu CAN protokolu do svojich výrobkov a preto sa používanie CAN protokolu rozširuje i do iných oblastí, než len do automobilového priemyslu, najme kvôli nízkej cene, jednoduchej implementácii, spoľahlivosti, vysokej prenosovej rýchlosti, jednoduchej rozšíriteľnosti a dostupnosti použitej súčiastkovej základne.
Protokol CAN je definovaný normou ISO 11898. Tá popisuje fyzickú vrstvu protokolu a špecifikáciu CAN 2.0A. Neskôr bola ešte vytvorená špecifikácia CAN 2.0B, ktorá zavádza dva pojmy - štandardný a rozšírený formát správy (rozdiel je v dĺžke identifikátoru správy). ISO 11898 definuje fyzickú a linkovú vrstvu protokolu podľa referenčného modelu ISO/OSI. Aplikačná vrstva protokolu CAN je definovaná vzájomne nekompatibilnými štandardmi (CAL/CANopen, DeviceNet, ...).
CAN je navrhnutý tak, aby prevádzal distribuované riadenie systémov v reálnom čase s prenosovou rýchlosťou do 1Mbit/s a vysokým stupňom zabezpečenia prenosu proti chybám. Protokol sa chová ako *multi-master* čo znamená, že každý uzol zbernice sa môže chovať ako *master* a riadiť tak chovanie iných uzlov. Preto nieje potrebné aby v systéme bol implementovaný nadradený člen, čo zvyšuje spoľahlivosť. Pridelenie adresy na zbernici prebieha nastavením čísla uzlu pomocou prepínača na module obr. 2.11. Tento modul zvládne maximálne 32 členov na zbernici. Po zbernici prebieha komunikácia

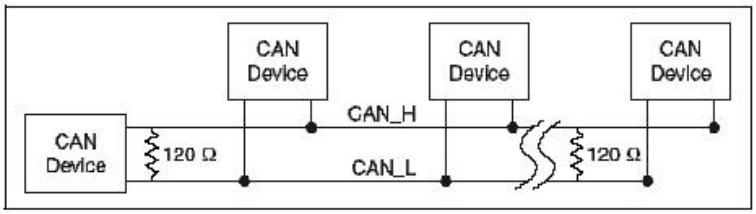


Obrázok 2.11: Prepínač čísla uzlu na module AC110

medzi dvoma uzlami pomocou správ (datová správa a žiadosť o dáta), a management siete (signalizácia chýb, pozastavenie komunikácie) je zaistený pomocou dvoch špeciálnych správ (chybové správy a správy o preťažení). Správy vysielné po zbernici protokolom CAN sú prijímané všetkými uzlami naraz pripojenými k tejto zbernici. Každá správa začína identifikátorom pre určenie adresáta správy, filtrácia správ, ktoré sa daných uzlov netýkajú (Acceptance Filtering) a obsahuje aj prioritu správy, ktorá slúži na rozhodovanie prijatia správy pri kolízii dvoch správ.

2.3.2 Fyzické prenosové médium

Realizácie fyzického média spočíva v implementácii logického súčinu. Protokol CAN definuje dve vzájomne komplementárne hodnoty bitov na sbernici - *dominant* a *recessive*. Pravidlo pre vysielanie v ohľade na logický súčin hovorí, že ak vysielá každé zariadenie *recessive bit*, potom je na zbernici *recessive*. V prípade že aspoň jedna stanica vysiela *dominant bit*, je na sbernici stav *dominant*.
Pre realizáciu fyzického prenosového média sa najčastejšie používa diferenciálna zbernica definovaná podľa normy ISO 11898. Táto norma definuje jednak elektrické vlastnosti vysielacieho budiča a prijímača a zároveň princípy časovania, synchronizácie a kódovánia jednotlivých bitov. Zbernicu na obr. 2.12 tvoria dva vodiče (CAN_H a CAN_L), kde dominant či recessive úroveň na zbernici je definovaná rozdielovým napätím týchto dvoch vodičov. Podľa normy je úroveň recessive veľkosť rozdielového napätia $V_{diff} = 0V$ a pre úroveň dominant $V_{diff} = 2V$.



Obrázok 2.12: Principiálna štruktúra siete CAN podľa ISO 11898

Pre elimináciu odrazov na vedení je zbernica na oboch koncoch prispôsobená zakončovacími odpormi o veľkosti 120 Ω. Presné elektrické charakteristiky zbernice sú detailne v norme ISO 11898. Prehľad základných parametrov je v tab. 2.3.

2.3.3 Linková vrstva protokolu CAN

Linková vrstva protokolu CAN je tvorená dvoma podvrstvami :

- MAC - má na starosti prístup k médiu *MAC (Medium Access Control)* a jej úlohou je kódovať dáta, vkládať doplnkové bity do komunikácie *(Stuffing/Destuffing)*, riadiť prístup všetkých uzlov k médiu s rozlišením priorít správ, detekcia chýb a ich hlásenie a potvrdzovanie správne prijatých správ.

- LCC - *(Logical Link Contol)*, vykonáva filtrovanie prijatých správ *(Acceptance Filtering)* a hlási o preťažení *(Overload Notification)*.

Prenosová rýchlosť	125 kBit/s až 1 Mbit/s
Počet uzlov v sieti	max 30
Maximálna dĺžka zbernice - 1 Mbit/s	
1 Mbit/s	40 m
500 kbit/s	112 m
300 kbit/s	200 m
100 kbit/s	640 m
50 kbit/s	1340 m
20 kbit/s	2600 m
10 kbit/s	5200 m
Typická impedancia vedenia	120 Ω

Tabuľka 2.3: Základné parametre CAN zbernice

2.3.3.1 Riadenie prístupu k médiu a riešenie kolízii

Ak je zbernica voľná, môže ľubovolný uzol zahájiť vysielanie. Ak zaháji niektorý uzol vysielanie skôr než ostatní, získava zbernicu pre seba a ostatné uzly môžu zažať vysielanie až po vyslaní kompletnej správy. Jedinú výnimku tvoria chybové rámce, ktoré môže vyslať ľubovolný uzol, ak detekuje chybu v práve prenášanej správe. Ak zaháji vysielanie súčasne niekoľko uzlov, potom prístup na zbernicu získa ten, ktorý prenáša správu s vyššou prioritou (nižším identifikátorom). Identifikátor je uvedený na začiatku správy. Každý vysielač porovnáva hodnotu práve vysielaného bitu s hodnotou na zbernici a ak zistí, že na zbernici je iná hodnota než vysiela, okamžite preruší ďalšie vysielanie. Tým je zajistené, že správa s vyššou prioritou bude odoslaná prednostne a že nedôjde k jej poškodeniu, čo by malo za následok opakovanie správy a zbytočné predlžovanie doby potrebnej k prenosu správy. Uzol, ktorý nezískal pri kolízii prístup na zbernicu musí vyčkať až bude zbernica opäť voľná, a potom správu vyslať znovu.

2.3.3.2 Zabezpečenie dát, detekcia a signalizácia chýb

Prenášanie dát pomocou protokolu CAN sa považuje za veľmi spoľahlivé aj z dôvodu, že sa na kontrole podieľa viacero mechanizmov naráz :

- **Monitoring** - vysielač porovnáva hodnotu práve vysielaného bitu s úrovňou detekovanou na zbernici. Ak sú obe hodnoty rovnaké, vysielač pokračuje vo vysielaní. V opačnom prípade vysielanie prerušuje.

- **CRC** kód *(*Cyclic Redundancy Check) - na konci každej správy je uvedený 15 bitový CRC kód, ktorý je generovaný zo všetkých predchádzajúcich bitov príslušnej správy podľa polynómu: $x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$. Ak je detekovaná chyba CRC ľubovolným uzlom na zbernici, je vygenerovaná chyba CRC.

- **Vkládanie bitov** (*Bit stuffing*) - vysiela sa na zběrnicu päť po sebe idúcich bitov jednej úrovne, je do správy naviac vložený bit opačnej úrovne. Toto opatrenie slúži jednak k detekcii chýb ale tiež k správnemu časovému zosynchronizovaniu prijímača jednotlivých uzlov. Ak je detekovaná chyba vládania bitov, je vygenerovaná chyba vkládania bitov.

- **Kontrola správy** (*Message Frame Check*) - správa sa kontroluje podľa formátu udaného v špecifikácii a pokiaľ je na nejakej pozícii bitu správy detekovaná nepovolená hodnota, je vygenerovaná chyba rámca (formátu správy).

- **Potvrdenie prijatia správy** (*Acknowlege*) - ak je v poriadku prijatá ľubovolným uzlom, je toto potvrdené zmenou hodnoty jednoho bitu správy (*ACK*). Vysielač vždy na tomto bitu vysiela úroveň *recessive* a detekuje úroveň *dominant*, potom je všetko v poriadku. Potvrdzovanie prijatia správy je prevádzané všetkými uzlami pripojenými ku zbernici bez ohľadu na zapnuté či vypnuté filtrovanie správ (*Acceptance Filtering*).

Každý uzol má zabudované dva interné počítadla chýb udávajúce počet chýb pri príjme a pri vysielaní. Podľa obsahov počítadiel môže uzol prechádzať, medzi troma stavami (*aktívny*, *pasívny*, *odpojený*). Pokiaľ uzol generuje príliš veľké množstvo chýb, je automaticky odpojený. Z hľadiska hlášenia chýb rozdeľujeme uzly do následujôcich troch skupín :

- **Aktívny** (*Error Active*) - uzly sa môžu aktívne podieľať na komunikácii po zbernici a v prípade, že detekujú ľubovolnú chybu v práve prenášanej správe (chyba bitu, chyba CRC, chyba vkládania bitov, chyba rámca), vysielajú na zbernici aktívny príznak chyby (*Active Error Flag*). Aktívny príznak chyby je tvorený šiestimi po sebe idúcimi bitmi *dominant*, kvôli čomu dôjde k poškodeniu prenášanej správy (poruší sa pravidlo vkládania bitov).

- **Pasívny** (*Error Passive*) - tieto uzly sa tiež podieľajú na komunikácii po zbernici, ale z hľadiska hlásenia chýb, vysielajú iba pasívny príznak chyby (*Passive Error*

Flag). Ten je tvorný šiestimi po sebe idúcimi bitmi *recessive*, kvôli čomu nedôjde k deštrukcii práve vysielanej správy.

- **Odpojené** (*Bus-off*) - tieto uzly nemajú žiadny vplyv na zbernicu, ich výstupné budiče sú vypnuté.

2.3.4 Základné typy správ

Špecifikácia protokolu CAN definuje štyri typy sprav :

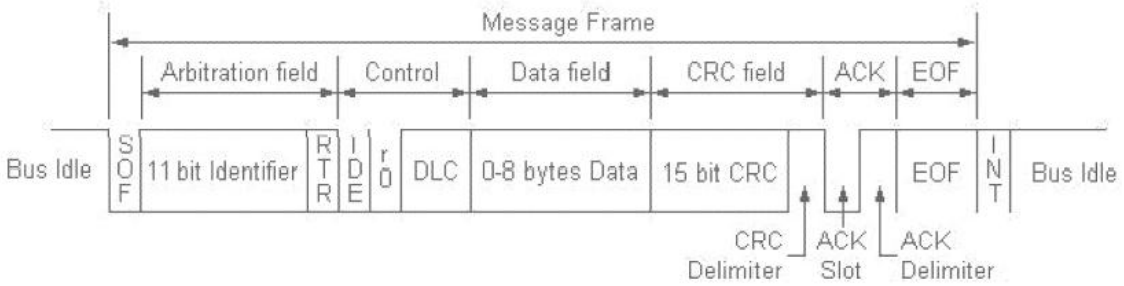
- **Datová správa** (*Data Frame*) - predstavuje základný prvok komunikácie uzlov po zbernici
- **Žiadosť o dáta** (*Remote Frame*) - správa na vyžiadanie dát
- **Chybová správa** (*Error Frame*) - slúži k signalizácii chýb na zbernici
- **Správa o preťažení** (*Overload Frame*) - slúži k oddialeniu vysielania dalšej dátovej správy alebo žiadosti o dáta

2.3.4.1 Datová správa

Datová správa umožňuje vyslať na zbernicu 0 až 8 datových bajtov. Pre jednoduché príkazy uzlom (typu vypni/zapni) nieje nutné prenášať žiadne datové bajty (význam príkazu je daný identifikátorom správy), čo skracuje dobu potrebnú k prenosu správy a zároveň zvetšuje priepustnosť zbernice, obzvlásť pri silnom zaťažení. Protokol CAN používá dva typy datových správ. Prvý typ je definovaný špecifikáciou 2.0A - štandardný formát správy (*Standard Frame*), špecifikácia 2.0B definuje naviac tzv. rozšírený formát správy (*Extended Frame*). Rozdiel medzi oboma formátmi je v dĺžke identifikátoru správy, ktorá je 11 bitov pre štandardný formát a 29 bitov pre rozšírený formát. Oba dva typy správ môžu byť používané na jednej zbernici.
Vyslanie datovej správy je možné iba vtedy, ak je zbernica volná. Akonáhle uzol, ktorý má pripravenú správu k vysielaní, detekuje voľnú zbernicu, začína vysielať. Štruktúra datovej správy podľa špecifikácie 2.0A je na obr. 2.13.

Jednotlivé časti správy znamenajú :

- SOF - začiatok správy (*SOF = Start Of Frame*), 1 bit dominant



Obrázok 2.13: Datová správa podľa špecifikácie CAN 2.0A

- Riadenie prístupu na zbernicu (*Arbitration Field*) - určenie priority správy

  Identifikátor správy - 11 bitov (*Identifier*), udáva význam prenášaej správy

  RTR bit (*Remote Request*) - 1 bit, príznak udává, či sa jedná o datovú správu alebo o žiadosť o vyslanie dát

- Riadiace informácie (*Control Field*)

  R0, R1 - rezervované bity

  Dĺžka dát (*DLC*) - 4 bity, počet prenášaných datových bajtov v správe. Povolené hodnoty sú 0 až 8.

- Datová oblasť (*Data Field*) - datové bajty správy. Maximálne 8 bajtov.

- CRC (*CRC Field*) - 16 bitov, zabezpečovací CRC kód

  CRC kód - 15 bitov

  ERC (*CRC delimiter*) oddelovač - 1 bit recessive.

- Potvrdenie (*ACK Field*) - 2 bity

  ACK (*ACK slot*) potvrdenie - 1 bit

  ACD (*ACK delimiter*) oddelovač - 1 bit recessive

- Koniec správy (*End Of Frame*) - 7 bitov recessive

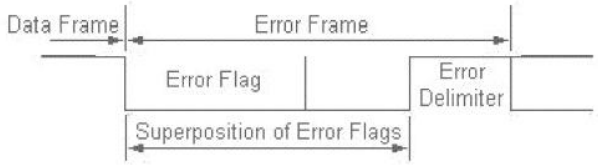
- Medzera medzi správami (*Interframe Space*) - 3 bity recessive, oddeľuje dve správy

2.3.4.2 Žiadosť o dáta

Formát žiadosti o dáta je podobný ako formát datovej správy. RTR bit (pole riadenia prístupu na zbernicu) nastavený do úrovne *recessive* a chýba datová oblasť. Pokiaľ nejaký uzol žiada o zaslanie dát, nastaví taký identifikátor správy, ako má datová správa, ktorej zaslanie požaduje. Tým je zaistené, že pokiaľ v rovnakom okamžiku jeden uzol žiada o zaslanie dát a iný dáta s rovnakým identifikátorom vysiela, prednosť v prístupe na zbernicu získa uzol vysielajúci datovú správu, lebo úroveň RTR bitu datovej správy je dominant a teda má táto správa vyššiu prioritu.

2.3.4.3 Chybová správa

Chybová správa slúži k signalizácii chýb na zbernici CAN. akonáhle ľubovolný uzol na zbernici detekuje v prenášanej správe chybu (chyba bitu, chyba CRC, chyba vkladania bitov, chyba rámca), vygeneruje ihneď na zbernici chybový rámec. Podľa toho, v akom stave pre hlásenie chýb sa uzol, ktorý zistil chybu, práve nachádza, generuje na zbernici buď aktívny (šesť bitov dominant) alebo pasívny (šest bitov recessive) príznak chyby. Pri generováni aktívneho príznaku chyby je prenášana správa poškodená (vzhľadom k porušeniu pravidla na vkládanie bitov), a teda i ostatné uzly začnú vysielať chybové správy. Hlásenie chýb je indikované superpozíciou všetkých chybových príznakov, ktoré vysielajú jednotlivé uzly. Dĺžka tohoto úseku môže byť minimálne šesť a maximálne dvanásť bitov. Chybová správa je znázornená na obr. 2.14



Obrázok 2.14: Chybová zpráva protokolu CAN

2.3.4.4 Správa o preťažení

Správa o preťažení slúži k oddialeniu vysielania ďalšej datovej správy alebo žiadosti o data. Tento zpôsob využívaju zariadenia, ktoré niesu schopné kvôli svojmu vyťaženiu prijímať a spracovávať ďalšie správy. Štruktúra správy je podobná správe o chybe, ale jej vysielanie môže byť zahájené po konci správy, oddelovača chýb alebo predchádzajúceho

oddelovača správ preťaženia. Správa o preťažení je zložená z príznaku preťaženia (šesť bitov dominant) a superpozíciou všetkých príznakov preťaženia, pokiaľ sú generované viacerými uzlami súčasne. Za príznakmi preťaženia následuje ďalších sedem bitov recessive, ktoré tvoria oddelovač správy o preťažení.

Kapitola 3

Praktické riešenie

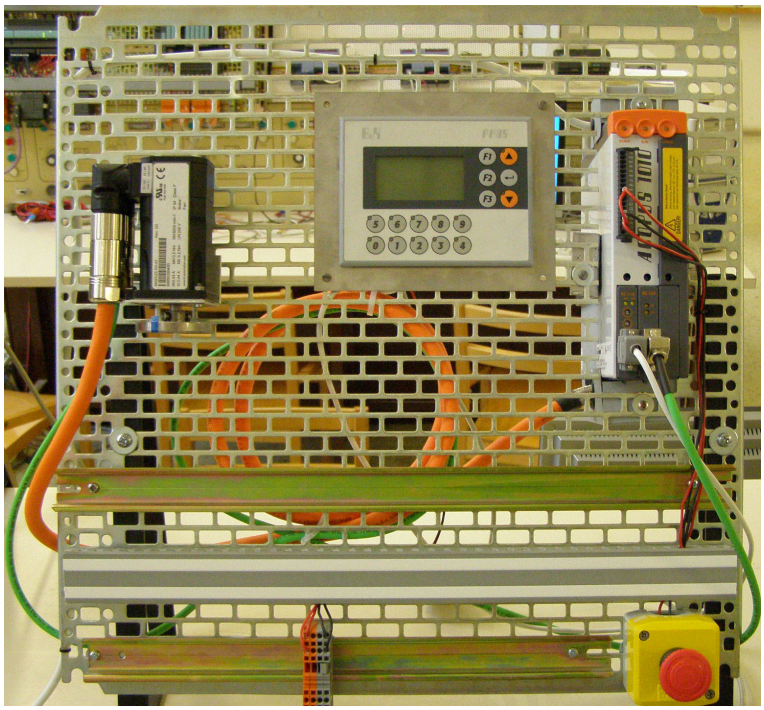
3.1 Laboratórny výučbový model vačiek

3.1.1 Úvod

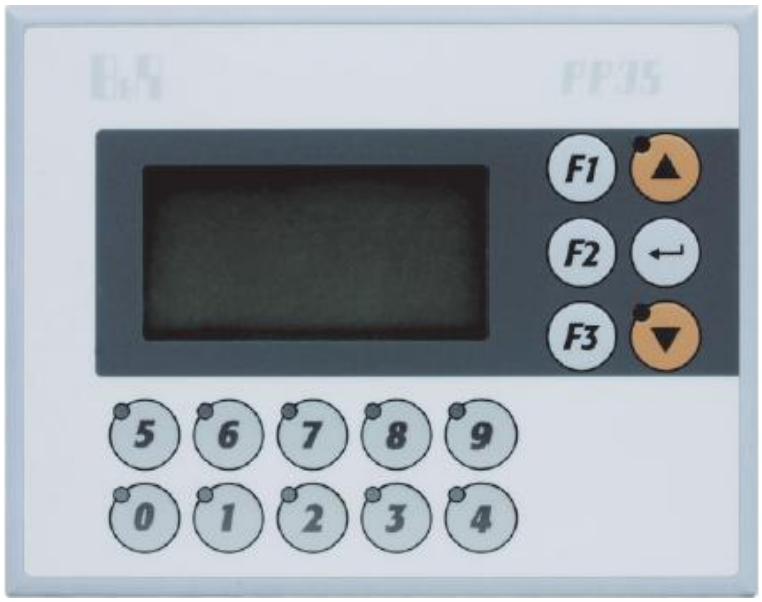
Cieľom tohto modelu je priblížiť študentom systém pozostavajúci z programovateľného logického automatu (PLC), servopohonu a servomotoru. Výstupom mojej práce sú 3 vyučbové modely obr. 3.1 a 5 cvičení. Tieto cvičenia prevedú študentov zostavením projektu, nastavením regulačných konštánt regulátorov v servopohone, základným programom využívajúcim nc_akcie v Ansi C, vytvorením základnej vizualizácie a nakoniec vačkami s pomocou českej knižnice. To by malo poslúžiť ako odrazový mostík pre programovanie zložitejšieho modelu "Žongléra" využívajúceho vačky ako podstatu svojej funkčnosti. Cvičenia sa nachádzajú v prílohe. V následujúcich odstavcoch sú popísané prvky, ktoré boli použité pre tento laboratórny model.

3.1.2 Power Panel PP35

Pre vytvorenie modelu som mal k dispozícii práve PLC s integrovaným displejom *4PP035.0300-01* obr. 3.2. I keď na prvý pohľad sa zdá celkom spôsobily tab. 3.1, v skutočnosti je to najnižšia rada procesorov firmy B&R, ktorá je ešte použiteľná pre vačkový automat. Hlavný problem spočíva v nedostatku flash pamäte. Samotná knižnica ACP10_MC library, ktorá je vytvorená pre programovanie vačkového automatu potrebuje 0,5 MB pamäte flash, čo je polovica z hodnoty ktoru má PP35. Všetky vzniknuté komplikácie odstranila česká knižnica ACP10_CZ od ing. Vančury. Má rovnako široké využitie ako ACP10_MC, ale vyžaduje si o 90% menej pamäte.



Obrázok 3.1: Vyučbový model



Obrázok 3.2: 4PP035.0300-01

Části PLC	Hodnota komponentu
Napájanie	24 VDC
Displej	LCD, podsvietený
Rozlišenie	160x180 pixelov
Tlačítka	16 z toho 10 podsvietených
Pamäť	300 KB SRAM, 1024 KB FlashPROM
Rozhranie	1xRS232, 1xCAN
Vstupy	16 digital
Výstupy	16 digital

Tabuľka 3.1: Parametre PP35

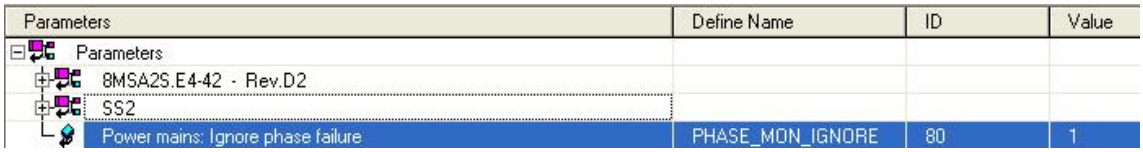
3.1.3 Acopos 1010-50

Ďalšou hlavnou zložkou modelu je servopohon Acopos 1010.50-2 obr. 3.3. Jeho hlavné

Obrázok 3.3: Acopos 1010.50-2

parametre sú uvedené v tab. 3.2. Jeho zvláštnosťou je to, že nemusí mať napájanie 3x230 V, ale stačí mu aj 1x230 V. V jednofázovom napájaní nastáva problém, kedy Acopos začne signalizovať výpadok fázy. To spôsobí pozastavenie jeho činnosti a uvedenie do chybového módu. Bez odstránenia tohto hlásenia nie je možné Acopos používať len s

jednou fázou. Chyba sa dá anulovať špeciálnym parametrom *Ignore phase failure*, ktorý je nutné pridať do parametrizačnej tabuľky. V parametrizačnej tabuľke pravým tlačidlom myši klikneme *Parameters→Insert parameter* a do *ID* vzniknutého parametru napíšeme číslo 80 a do položky "value"napíšeme hodnotu 1 viz. obr. 3.4. Pre komunikáciu s PLC



Obrázok 3.4: Odstranenie hlásenia výpadku fázy v parametrizačnej tabuľke

Části Acoposu 1010.50-2	Hodnota komponentu
Napájanie	3x110-230 VAC ±10% alebo 1x110-230 VAC ±10%
Štartovací prúd	5 A
Frekvencia	50/60 Hz ±4%

Tabuľka 3.2: Parametre Acoposu 1010.50-2

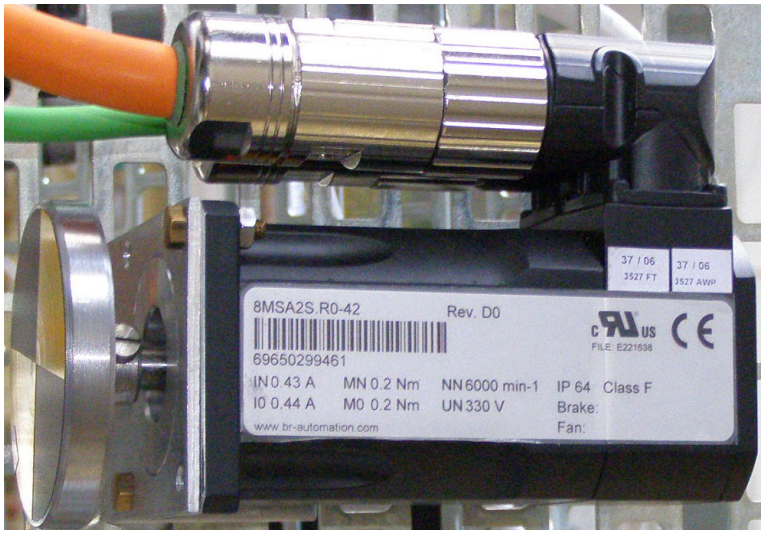
acopos využíva AC120 CAN interface modul, ktorého funkčnosť a princíp celej zbernice CAN je popísana podrobne v teoretickom rozbore. S resolverom zabudovaným v motore komunikuje pomocou modulu AC122 obr. 3.5. Pomocou tohto modulu resolver dostáva

Obrázok 3.5: AC122 - Resolver interface

absolútnu polohu v rámci jednej otáčky.

3.1.4 Synchrónny motor 8MSA2S.R0

Tento motor obr. 3.6 sa vyznačuje vysokou presnosťou oproti asynchronným motorom. Pre určenie polohy ma zabudovaný resolver, ktorý je schopný rozlíšiť 16 000 jednotiek na jednu otáčku. Základne parametre motora 8MSA2S.R0 sú v tab. 3.3. Momentová

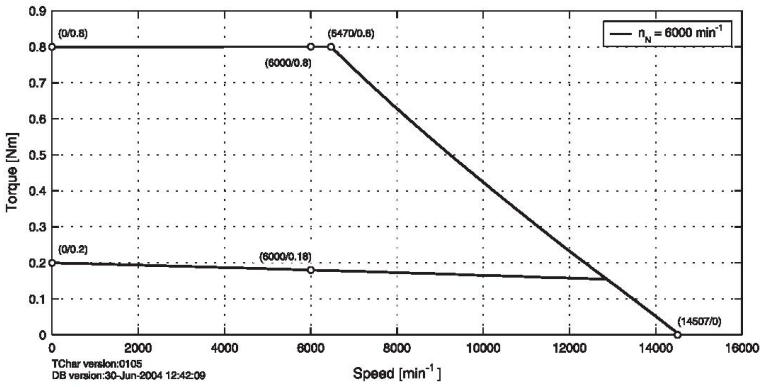


Obrázok 3.6: Servomotor 8MSA2S.R0

Menovitá rýchlosť n_N [min^{-1}]	6000
Menovitý moment M_N [Nm]	0,18
Menovitý prúd I_N [A]	0,43
Moment zvratu M_0 [Nm]	0,2
Špičkový moment M_{max} [Nm]	0,8
Špičkový prúd I_{max} [A]	1,9
Maximálne zrýchlenie bez brzdy a [rad/s^2]	133 333
Maximálna rýchlosť n_{max} [min^{-1}]	12000
Moment zotrvačnosti bez brzdy J_{mot}[$kgcm^2$]	0,06

Tabuľka 3.3: Parametre synchrónneho motora 8MSA2S.R0

charakteristika motora na obr. 3.7 spriehľadňuje tabuľku tab. 3.3.



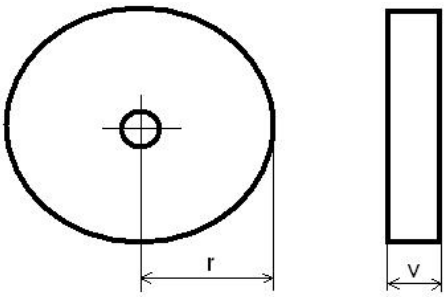
Obrázok 3.7: Momentová charakteristika synchronného motora 8MSA2S.R0

3.1.4.1 Výpočet záťaže motora

Pomer medzi momentom zotrvačnosti záťaže J_{load} a momentom zotrvačnosti motora J_{mot} by nemal prevyšovať hodnotu, ktorá zaručuje dynamické a stabilné riadenie (3.1).

$$\frac{J_{load}}{J_{mot}} < 20 \tag{3.1}$$

Moment zotrvačnosti motora J_{mot} je zadefinovaný v tab. 3.3 a je daný výrobou. Keďže



Obrázok 3.8: Cylindrická zátaž motora

som vychádzal zo záťaže cylindrického tvaru obr. 3.8, na výpočet momentu zotrvačnosti záťaže J_{load} som použil vzorec (3.2).

$$J_{load} = \frac{1}{2} mr^2 \tag{3.2}$$

S pomocou ďalších vzťahov (3.3):

$$V = \pi r^2 v \ , \ m = V\rho \tag{3.3}$$

kde V je objem cylindra a $\rho = 0,0078 kg/cm^3$ hustota ocele.
Pre výrobu záťaže bol k dispozícii válec s polomerom $r = 3cm$. Na základe rovníc (3.1), (3.2) a (3.3) som odvodil rovnicu (3.4)

$$v < \frac{40 J_{mot}}{\pi r^4 \rho} \tag{3.4}$$

Vypočítaná hrúbka cylindrickej záťaže musí byť $v < 1,21cm$. Na základe (3.3) musí byť maximálna hmotnosť $m = 266g$. Pre dostatočný odstup od medznej hodnoty hmotnosti, znížil som jej hmotnosť o 1/4 na $m = 200g$. Hrúbka záťaže sa upravila na $v = 0,9cm$.

3.2 Automatizovaný model žongléra

3.2.1 Úvod

Mojou úlohou je nadviazať na predchádzajucého konštruktéra Jana Pšeničku a prehlbovať možnosti súčasného žonglovacieho automatu obr. 3.9. Tento žonglér pozostáva zo 4 pohonov, realizovaných technikou B&R. Dva synchrónne motory poháňajú dva lineárne pasove moduly MLR 10-110. Tieto pásové moduly transformujú rotačný pohyb motorov na vertikálny lineárny pohyb osi. Ďalšie dva motory sú umiestnené priamo na pásovych moduloch a zabezpečujú horizontálny pohyb žonglovacej gule. Súčasný mechanický stav žongléra je schopný žonglovať nanajvýš dvomi guľami. Pri mojich pokusoch vytvoriť vačku pre žonglovanie s tromi guľami som narazil na mechanické medze modelu. Súčasné zrýchlenie osy vo vertikálnom smere nie je postačujúce. Pri vysokých zrýchleniach dochádza k preťaženiu motorov, ktoré momentovo nestačia. To spôsobuje väčšie neúnosné prúdy vo vinutí, následkom ktorých dochádza k vypnutiu motorov tepelnou poistkou. Vychádzajúc z technických údajov servomotora 8MSA5L tab. 3.4 a z jeho momentovej charakteristiky obr. 3.10 sa ponúka použitie prevodovky. Momentálne využitie motorov je 1038 otáčok za minútu a zároveň maximálne momentové zaťaženie $40,5\,Nm$. Priestor pre zvýšenie výkonu sústavy ponúkaju otáčky, pri ktorých máme zaručený moment $40,5\,Nm$ do $1791\,min^-1$.

3.2.2 Návrh prevodovky pre vertikálnu os

Základné parametre, z ktorých vychádzam sú v tab. 3.5. Pri výpočte prevodu prevo-

Obrázok 3.9: Mechanický žonglér využivajúci vačkový automat

dovky existujú dve obmedzenia. Prvé obmedzenie je zjavné z momentovej charakteristiky motora obr. 3.10, kde je maximálny moment $40,5\,Nm$, ktorý je možné dosiahnúť pri maximálnej rýchlosti 1791 $min^{-}1$. Druhé obmedzenie je maximálny moment 80 Nm, ktorým je možne pôsobiť na lineárny modul MLR10-110.
Pre ochranu lineárneho modulu je v rovnici (3.5) zvýšený maximálny moment motora 8MSA5L o 20%.

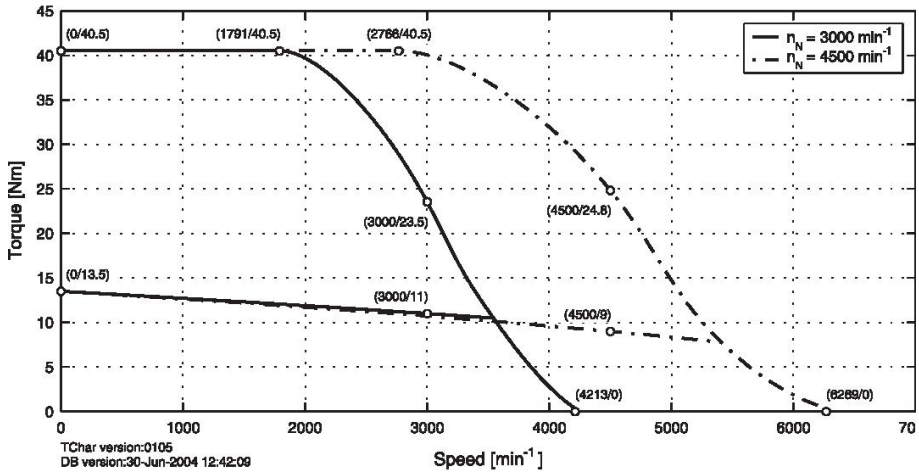
$$M_{motmax} = M_{mot} \cdot 1,2 = 40,5 \cdot 1,2 = 48,6Nm \tag{3.5}$$

Prevod je teda podielom momentov (3.6)

$$Prevod = \frac{M_{maxModul}}{M_{motmax}} = \frac{80}{48,6} = 1,6461 \cong 1,66 \Longrightarrow 5:3 \tag{3.6}$$

Pri využití ďalšieho obmedzenia otáčok vypočítame prevod v (3.7)

$$Prevod = \frac{n_{max} \cdot \pi \cdot d}{v} = \frac{\frac{1791}{60} \cdot \pi \cdot 0,092}{5} = 1,7246 \tag{3.7}$$



Obrázok 3.10: Momentová charakteristika servomotora 8MSA5L

Menovitá rýchlosť n_N [min^{-1}]	3000
Menovitý moment M_N [Nm]	11
Menovitý prúd I_N [A]	7,3
Moment zvratu M_0 [Nm]	13,5
Špičkový moment M_{max} [Nm]	40,5
Špičkový prúd I_{max} [A]	43,2
Maximálne zrýchlenie bez brzdy a [rad/s^2]	55 459
Maximálna rýchlosť n_{max} [min^{-1}]	9000
Moment zotrvačnosti bez brzdy J_{mot}[kgcm^2]	7,3

Tabuľka 3.4: Parametre synchronného motora 8MSA5L

Pri porovnaní oboch prevodov musíme akceptovať nižší prevod 5:3 aby nedošlo k poškodeniu lineárneho modulu . Z tohto prevodu plynú maximálne otáčky (3.8)

$$n_{max} = \frac{Prevod \cdot \ v}{\pi \cdot d} = \frac{1,66 \cdot 5}{\pi \cdot 0,092} = 28,84s^{-1} = 1730min^{-1} \tag{3.8}$$

Maximálny krútiaci moment M_{Kmax} na výstupe prevodovky spôsobí zrýchlenie odvodené v (3.9)

$$M_{Kmax} = 1,2 \cdot m \cdot a \cdot r \Longrightarrow a = \frac{M_{Kmax}}{1,2 \cdot m \cdot r} = \frac{81}{1,2 \cdot 15,6 \cdot 0,046} = 94m/s^2 \tag{3.9}$$

Hmotnosť vertikálnej pohyblivej hmoty	m [kg]	15,6
Maximálna požadovaná rýchlosť	v [m/s]	5
Priemer remenice lin. modulu MLR10-110	d[mm]	92

Tabuľka 3.5: Zadané parametre pre výpočet prevodovky

Čas na dosiahnutie požadovanej maximálnej rýchlosti je:

$$t = \frac{v}{a} = \frac{5}{94} = 53,19ms \tag{3.10}$$

Z predchadzajúcich vzťahov plynie prevod prevodovky 5:3. Na vstupe bude mať maximálny moment po zaokruhlení nahor $50\,Nm$ a na výstupe $100\,Nm$. Vďaka tejto prevodovke sa rýchlost v z 0 na $5\,m/s$ dosiahne za čas $t = 53,19\,ms$ pri zrýchlení $a = 94\,m/s^2$.

3.2.3 Programove rozšírenie o tretiu os

Projekt pre model žongléra bol vytvorený hierarchicky v troch vrstvách. Najvyššiu vrstvu tvorí program obsiahnutý v súbore *hlavní.c*, strednú vrstvu v súbore *zongler.c* a najnižšie vrstvy sú obsiahnuté v súboroch *servo1.c, servo2.c, servo3.c a servo4.c*, ktoré prislúchajú jednotlivým servomotorom. Pre pridanie ďalšej osi sú nutné zmeny vo všetkých vrstvách. V hlavnom strome projektu je nutné pridať ďalšiu *ACOPOS: Parameter Table* a *ACP10: Axis*, ktoré odpovedajú pridanému linearnému motoru. *Parameter Table* servopohonu pre lineárny motor budu dodané z výroby. Následne je nutné všetky parametre lin. motora a servopohonu spojiť v *Deployment Table*, podľa vzoru pôvodných os. Nakoniec je potrebné pridať 5 váčkových profilov určených pre žonglovanie troma guľami napr. *vacka15, vacka25, vacka35, vacka45, vacka55.* Vysporiadanie sa s horeuvedeným postupom dopomôžu cvičenia uvedené v **Dodatku A**.

3.2.3.1 Vytvorenie najnižšej vrstvy tretej osi

Súbor *servo5.c* má obdobnú štruktúru ako súbory *servo2-4.c*. Potrebuje samozrejme vlastné globálne a lokálne premenné:

```
_GLOBAL RealAxis_typ osaServo5;

_GLOBAL BOOL servo5Nahrej;
_GLOBAL BOOL servo5Zavazbi;
```

```
_GLOBAL BOOL servo5Rozvazbi;

_GLOBAL BOOL servo5NahrejOk;
_GLOBAL STRING servo5vacka[8];
_GLOBAL BOOL servo5NahrejRes;

_LOCAL UINT step;
_LOCAL CamAxis_typ camServo5;
_LOCAL DownCamProf_typ downCamProf;
```

V inicializačnej časti príkazom:

```
strcpy(osaServo5.init.name,"servo5");
```

priradíme menom *servo5* vytvoreným v *Deployment Table* reálny pohon, ktorý bol predtým nadefinovaný v *Parameter Table* a *Axis*.
Cyklická časť zostáva rovnaká pri použití horeuvedených nadefinovaných premenných.

3.2.3.2 Doplnenie strednej vrstvy tretej osi

Do súboru *zongler.c* je nutné dodať obdobné globálne premenne ako v najnižšej vrstve. V kroku *ZN_Init* sa doplnia príkazy:

```
strcpy(PPS_Zon_Vacka5s, "00");
strcpy(WSS_Zon_Vacka5s, "00");
```

V kroku *ZN_Wait* je nutné dodať podmienku pre žonglovanie s troma guľami:

```
if ((PPB_Zon_3Ball==1 && V_OVL_FROM==0) || (WSB_Zon_3Ball==1 && V_OVL_FROM==1))
{ step_ZN = ZN_3Ball; }
```

Po pridaní podmienky je nutné vytvorenie kroku *ZN_3Ball*:

```
case ZN_3Ball:
   strcpy (PPS_Zon_Step1, "3ball");
   strcpy (WSS_Zon_Step1, "3ball");
   strcpy (PPS_Zon_Nastaveno, "Prehazovani 3 micku");
   strcpy (WSS_Zon_Nastaveno, "Prehazovani 3 micku");
   strcpy(servo1vacka, "vacka15");
   servo1Nahrej = 1;
```

```
   strcpy(servo2vacka, "vacka25");
   servo2Nahrej = 1;
   strcpy(servo3vacka, "vacka35");
   servo3Nahrej = 1;
   strcpy(servo4vacka, "vacka45");
   servo4Nahrej = 1;
   strcpy(servo5vacka, "vacka55");
   servo5Nahrej = 1;
   WSB_Zon_3Ball = 0;
   step_ZN = ZN_Nastav3Wait;
break;
```

Po kroku *ZN_3Ball* program vykoná krok *ZN_Nastav3Wait*:

```
case ZN_Nastav3Wait:
   if ( servo1NahrejOk == 1 && servo2NahrejOk == 1 && servo3NahrejOk == 1
   && servo4NahrejOk == 1 && servo5NahrejOk == 1)
   {
      step_ZN = ZN_End;
   }
break;
```

V kroku *ZS_Init* sa musi doplniť:

```
osaServo5.cmd.stop = 0;
osaServo5.param.speed = 500;
osaServo5.param.position = 1000;
```

kde hodnoty rýchlosti a pozície sú uvedené ako príklad. V kroku *ZS_Homing* je nutne dodať pokyn pre vykonanie absolutného pohybu a ďalšiu podmienku:

```
osaServo5.cmd.absMove = 1;
if ( osaServo1.info.moveActive == 1 && osaServo2.info.moveActive == 1 &&
osaServo3.info.moveActive == 1 && osaServo4.info.moveActive == 1 &&
osaServo5.info.moveActive == 1 )
   { step_ZS = ZS_HomOk; }
```

V kroku *ZS_Zavazbeni* dochádza k spusteniu vačiek, kde je nutné spustiť aj vačku pre

lineárny motor:

```
servo5Zavazbi = 1;
if (osaServo1.info.moveActive == 1 && osaServo2.info.moveActive == 1 &&
osaServo3.info.moveActive == 1 && osaServo4.info.moveActive == 1
&& osaServo5.info.moveActive == 1)
{ step_ZS = ZS_ZavazbeniOk; }
```

Zastavenie vačkového automatu nastane po splnení podmienky:

```
if ((PPB_Zon_Stop==1 && V_OVL_FROM==0) || (WSB_Zon_Stop==1 && V_OVL_FROM==1))
```

kam je nutné dodať:

```
servo5Rozvazbi = 1;
```

3.2.3.3 Doplnenie najvyššej vrstvy o tretiu os

Súbor *hlavni.c* tvoriaci najvyššiu vrstvu programu potrebuje doplnenie vyhodnotenia koncových členov osi lineárneho motora:

```
KDC_digin_pos_hw_end = pAxis5->dig_in.status.pos_hw_end;
KHC_digin_neg_hw_end = pAxis5->dig_in.status.neg_hw_end;
```

preto je nutné ešte predtým vytvoriť pomocnú štruktúru:

```
pAxis5 = (ACP10AXIS_typ*)osaServo5.intern.pAxis;
```

Všetky novouvedené premenné sa musia zadeklarovať do hlavičkového súboru *promenne.h*.

3.2.3.4 Doplnenie diagnostickej funkcie o tretiu os

Funkcia *diag.c* slúži na zinicializovanie pohonov a uvedenie do READY stavu, popripade odquitovanie hlásených errorov. Pre lineárny motor je nutná taktiež inicializácia. Doplnenie funkcie prebehne nasledujúcim spôsobom:

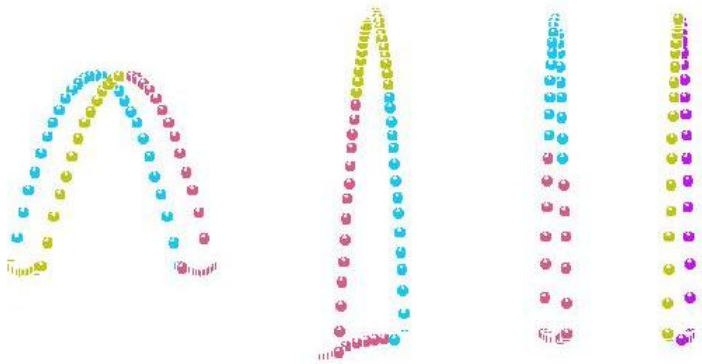
```
if (PPV_Adresa == 7)
```

```
{
   if (PPB_Resume == 1)
      { osaServo5.cmd.resume = 1; }
   if (PPB_SwitchOn == 1)
      { osaServo5.cmd.controllerOn = 1; }
   if (PPB_SwitchOff == 1)
      { osaServo5.cmd.controllerOff = 1; }
   if (PPB_Homing == 1)
      { osaServo5.cmd.homing = 1; }

   if (osaServo5.info.initialized == 1)
      { PPV_InitializedOk = 0; WSV_InitializedOk = 1; }
   else
      { PPV_InitializedOk = 1; WSV_InitializedOk = 0; }

   if (osaServo5.info.controllerOn == 1)
      { PPV_ControllerOn = 0; WSV_ControllerOn = 1; }
   else
      { PPV_ControllerOn = 1; WSV_ControllerOn = 0; }

    if (osaServo5.info.homingOk == 1)
      { PPV_HomingOk = 0; WSV_HomingOk = 1; }
   else
      { PPV_HomingOk = 1; WSV_HomingOk = 0; }

   PPV_ErrorSt = osaServo5.info.status;
   PPV_ErrorNb = osaServo5.info.nbErrors;

   WSV_ErrorSt = osaServo5.info.status;
   WSV_ErrorNb = osaServo5.info.nbErrors;

    strncpy (PPS_Popis_Err1,osaServo5.info.errorText.line1,50);
   strncpy (PPS_Popis_Err2,osaServo5.info.errorText.line2,50);

   strncpy (WSS_Popis_Err1,osaServo5.info.errorText.line1,50);
   strncpy (WSS_Popis_Err2,osaServo5.info.errorText.line2,50);
```

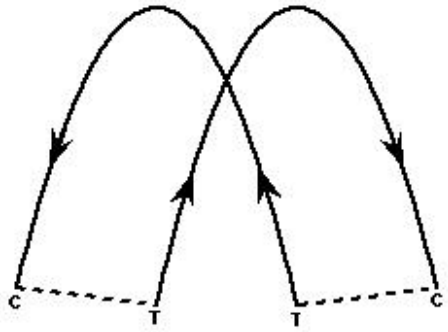
3.2.4 Analýza optimálneho žonglovania

Existuje veľa žonglovacích pomôcok pre žonglovanie počinajúc látkovými loptičkami plnenými ryžou až po horiace fakle. Taktiež je možné žonglovať rôznymi technikami obr. 3.11. Aby bolo žonglovanie prakticky realizovateľné použili sa biliardové gule a kaskadovita technika žonglovania, ktorá je ideálna a najpoužívanejšia pri žonglovaní s nepárnym počtom gúľ. Ruky vyhadzujú loptičky postupne jednu za druhou do dvoch symetrických oblúkov. Každá ruka vyhadzuje loptičku bližšie k stredu, zatial čo druhá ruka ju chytá ďalej od stredu obr. 3.12.

V prípade fyziky žonglovania je nutné v prvom rade zadefinovať parametre:



Obrázok 3.11: Druhy žonglovacích technik (zľava) kaskada, sprcha, fontána



Obrázok 3.12: Dva symetrické obluky kaskadovitej techniky

b - počet žonglovacích loptičiek;

h - počet žonglovacích ruk;

f - čas letu (čas kedy je loptička vo vzduchu medzi vrhom a chytením);

g - gravitačné zrýchlenie;
$\mathbf{V}_h$ - horizontálna rýchlosť hodu;
$\mathbf{V}_v$ - vertikálna rýchlosť hodu;
H - výška hodu;
F - vzdialenosť medzi rukami (horizontálna vzdialenosť letu loptičky);

Claude Shannon, ktorý sa preslávil na poli matematických a informatických vied, bol taktiež prvym, ktorý preskúmal matematiku žonglovania. Z jeho prác vyplývajú ďalšie parametre:

d - čas medzi chytením a vyhodením loptičky;
e - okamih prázdnej ruky medzi vyhodením a chytením ďalšej loptičky;

Shannonov žonglovací teorém elegantne popisuje vzťah medzi časovými premennými a počtom žonglovacích loptičiek a rúk v (3.11).

$$\frac{b}{h} = \frac{d+f}{d+e} \tag{3.11}$$

Z predchadzajúcich premenných je dobre si zadefinovať ďalšie premenné, ktoré vnesu viac svetla do žonglovacej problematiky:

τ - perióda hodu, alebo čas medzi dvoma vrhmi z tej istej ruky. To je to isté ako (d+e).
r - činiteľ obsadenosti ruky, alebo čas kedy je ruka plná. To je d/τ = d/(d+e).
ω - priemerný počet loptičiek vo vzduchu v jednom oblúku. To je f/τ = f/(d+e).

Takto je možné odvodiť rovnice pre ω (3.12):

$$\tau = \frac{f}{\omega} \ , \ f = \tau \cdot \omega \tag{3.12}$$

Hodnota činiteľa obsadenosti ruky r sa pohybuje vždy medzi 0 a 1 a môže byť popísana ako priemerny počet loptičiek nachadzajúcich sa v ruke počas žonglovania. Využitím rovníc (3.11) a (3.12) dostávame dôležite rovnice v (3.13):

$$\frac{b}{h} = \omega + r \ , \ \omega = \frac{b}{h} - r \tag{3.13}$$

Omegu je možné vypozorovať u žongléra pri sledovaní loptičiek nachádzajúcich sa vo vzduchu. Podľa žonglovacej definície žonglér potrebuje o jednu loptičku viac ako má k dispozícii rúk. Takže s dvoma rukami má ω minimálnu hodnotu 0,5 pri $b = 3$ a $r = 1$. Pre hod loptičky ďalej platia tieto vzťahy pre horizontálnu rýchlosť (3.14):

$$V_h = \frac{F}{f} \tag{3.14}$$

vertikálnu rýchlosť (3.15):

$$V_v = \sqrt{2 \cdot g \cdot H} \tag{3.15}$$

a výšku hodu (3.16):

$$H = \frac{V_v^2}{2 \cdot g} \tag{3.16}$$

V prípade, že uvažujeme chytenie a vyhodenie loptičky v rovnakej horizontálnej rovine je možné odvodiť následujúce rovnice pre f (3.17):

$$f = \sqrt{\frac{8 \cdot H}{g}} \tag{3.17}$$

vertikálnu rýchlosť (3.18):

$$V_v = \frac{g \cdot f}{2} \tag{3.18}$$

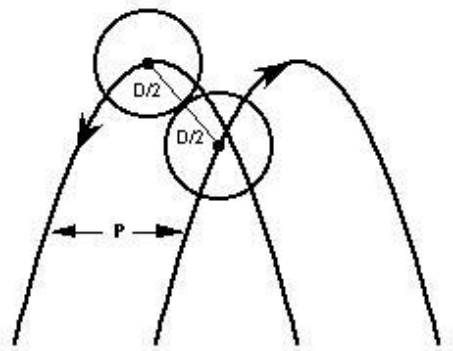
a výšku hodu (3.19):

$$H = \frac{g \cdot f^2}{8} = \frac{g \cdot (\tau \cdot \omega)^2}{8} \tag{3.19}$$

Rovnice (3.17), (3.18) a (3.19) nám hovoria, že pre zdvojnásobenie času letu loptičky sa musí zdvojnásobiť vyhadzovacia rýchlosť a následne zoštvornásobiť výška vrhu!
Pri určení minimálnej bezpečnej vzdialenosti medzi dvoma oblúkmi je nutné zadefinovať ďalšie premenné:

D - priemer žonglovacej gule;
P - vzdialenosť medzi dvoma oblúkmi obr. 3.13;

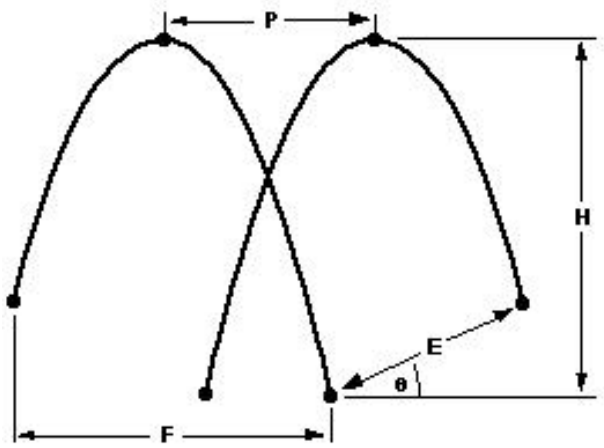
Vzťah pre P/D platí v (3.20):



Obrázok 3.13: Kaskadovitá technika a premenná P

$$\frac{P}{D} = \sqrt{\frac{4 \cdot V_h}{g \cdot \tau}^{2} + 1} \tag{3.20}$$

Optimálna vzdialenosť medzi oblúkmi kaskadovitej techniky sa približuje priemeru žonglovacej gule. Táto vlastnosť kaskadovitej techniky umožňuje i pri väčšom počte žonglovacích gúľ pohyb rúk o relatívne krátke vzdialenosti čo je veľkou výhodou oproti ostatným technikám. Ďalej je nutné zadefinovať vyhadzovaciu a chytaciu pozíciu každého oblúka obr. 3.14. V súvislosti s ďalšími požiadavkami je nutné zadefinovať nové premenné:



Obrázok 3.14: Vyhadzovacie a chytacie pozície oblúkov kaskadovitej techniky

E - vzdialenosť medzi chytacou a vyhadzovacou pozíciou;
θ - chytací uhol obr. 3.14;

Vzťah pre optimálnu hodnotu thety k minimalizovaniu vzdialenosti medzi chytacou a vyhadzovacou pozíciou je (3.21):

$$tan(\theta) = \frac{2 \cdot V_h^2}{g \cdot F} \tag{3.21}$$

ďalej pre E plati (3.22):

$$E = P \cdot cos(\theta) \tag{3.22}$$

pomocou vyšeuvedených vzťahov sa dá napísať (3.23):

$$H = \frac{(F + P \cdot sin^2(\theta))^2}{4 \cdot F \cdot tan(\theta)} \tag{3.23}$$

3.2.4.1 Výpočet parametrov pre praktické využitie v žonglérovi

Pred výpočtom je nutné si stanoviť niekoľko premenných v tab. 3.6. Optimálna hodnota činiteľa obsadenosti ruky r leži niekde medzi 0,5 a 1. Zvyšovaním r dáva žonglérovi viac

Počet žonglovacích gúľ	b	3
Počet rúk	h	2
Činiteľ obsadenosti ruky	r	0,66
Vzdialenosť medzi ramenami	F $[m]$	0,3
Vertikálna rýchlosť	V_v $[m/s]$	5; 4,5; 4; 3,5

Tabuľka 3.6: Zadané parametre pre výpočet kaskadového žonglovania

času na kontrolovanie loptičky a tým aj na presnejšie hody. Zvyšovaním r taktiež sa znižuje ω čim sa znižuje počet kolízii vo vzduchu. Presnu hodnotu r je možné určiť podľa kompromisu žongléra medzi precíznym vyhodenim alebo ľahkým chytením loptičky. Keď sa r znižuje tak zostáva menej času na precízne vyhodenie loptičky. Keď je r príliš veľké, chytenie loptičky začne byť nemožné, lebo nezostane dostatok času na presun ramena do chytacej pozície. Rôzne testy s vysokoskúsenými žonglérmi dokázali, že pre žonglovanie kaskadovitou technikou s 3, 5, 7 alebo 9 loptičkami sa r pohybuje okolo hodnoty $r = 2/3$. Na základe zadaných údajov boli spočítané ostatne parametre v tab. 3.7. Hodnoty boli spočítané pre 4 vertikálne rýchlosti pre vzájomne porovnanie. Porovnaním hodnôt v

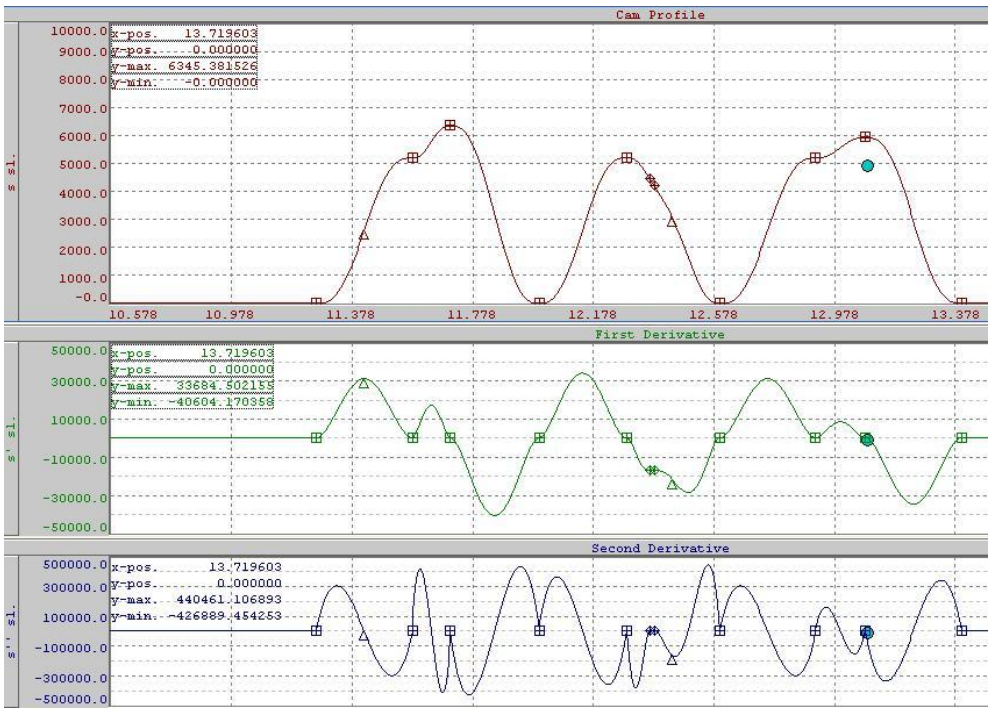
Vertikálna rýchlosť hodu	V_v $[m/s]$	5	4,5	4	3,5
Vyška hodu	H $[m]$	1,274	1,032	0,8155	0,624
Čas letu	f $[s]$	1,019	0,9174	0,8155	0,713
Horizontálna rýchlosť hodu	V_h $[m/s]$	0,2944	0,327	0,3678	0,42
Počet guľ vo vzduchu	ω	0,84	0,84	0,84	0,84
Perióda hodu	τ $[s]$	1,213	1,092	0,97	0,849
Čas medzi chytom a vrhom	d $[s]$	0,8	0,72	0,64	0,56
Čas medzi vrhom a chytom	e $[s]$	0,413	0,372	0,33	0,289
Chytací uhol	θ $[°]$	3,37	4,156	5,25	6,83

Tabuľka 3.7: Vypočítané parametre kaskadového žonglovania

tab. 3.7 je optimálnym riešením vertikálna rýchlosť $V_v = 4,5\,m/s$, pri ktorej výška hodu je $H = 1,032\,m$. So zvyšujúcou sa výškou dochádza k väčším problémom pri chytaní loptičky. Pri nižších vertikálnych rýchlostiach zostáva menej času na rozbeh a spomalenie motorov. Taktiež je nutné uvažovať s časovým oneskorením pri chytaní, ktoré je potrebné pre ustálenie gule v rameni žongléra. Bez tohto oneskorenia guľa nestabilne poskakuje a dochádza k chybným vrhom.

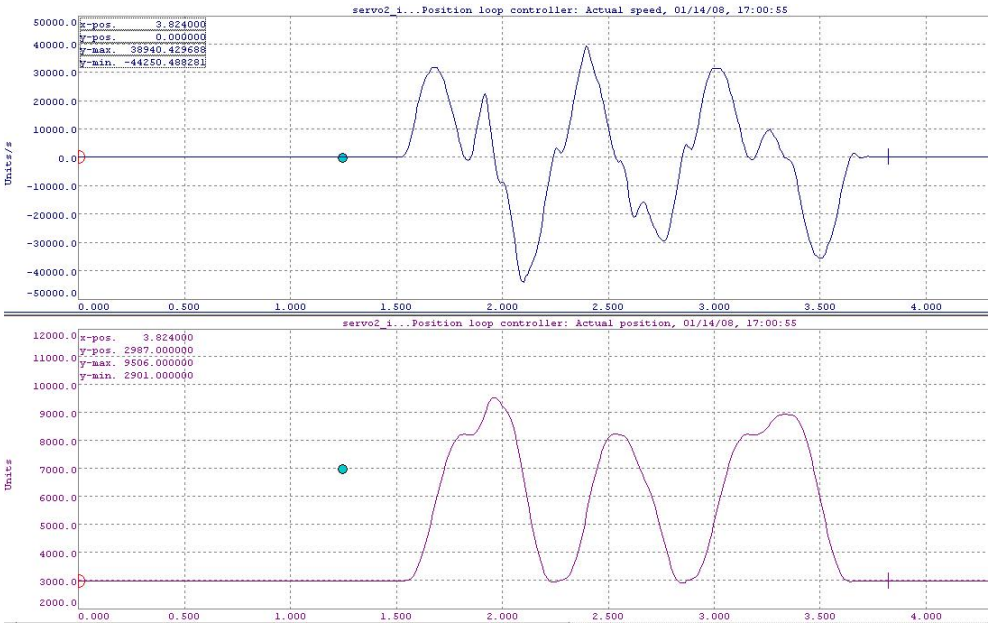
3.3 Sledovanie vačiek pomocou *Trace*

Bez reálnej tretej osi je problematické žonglovať s tromi guľami. Lenže tretia os má slúžiť len ako pomocná pre zapojenie tretej gule do procesu žonglovania. Bez tretej osi som zrealizoval štyri vačky pre žonglovanie s dvoma reálnymi a jednou virtuálnou guľou. Prakticky som overil nedostačujúce momentové schopnosti motorov. U reálnych systémov a tým aj u servomotorov sú všetky priebehy rýchlosti a zrýchlení tvorené oblými krivkami - skok rýchlosti z nuly na nejakú hodnotu prebieha pomalým nábehom, ktorý sa doženie špičkou uprostred nábehovej polohovej krivky. Pri dlhších časových periódach špičky niesu badateľné. Problém nastáva ak chceme od servomotora v čo najkratšom časovom okamžiku žiadať relatívne veľké rýchlosti. Veľkosť zrýchlenia začne dosahovať medzné hodnoty. V tejto situácii je nutné urobiť kompromis medzi maximálnou hodnotou rýchlosti a časom, za ktorý je nutné túto rýchlosť dosiahnuť. V prípade porovnania výstupných charakteristik servomotorov obr. 3.16 a nadefinovaných vačiek obr. 3.15 je u vačky v najnižšej charakteristike uvedene zrýchlenie, ktoré hraniči s medzným zrýchlením $50\,m/s^2$.

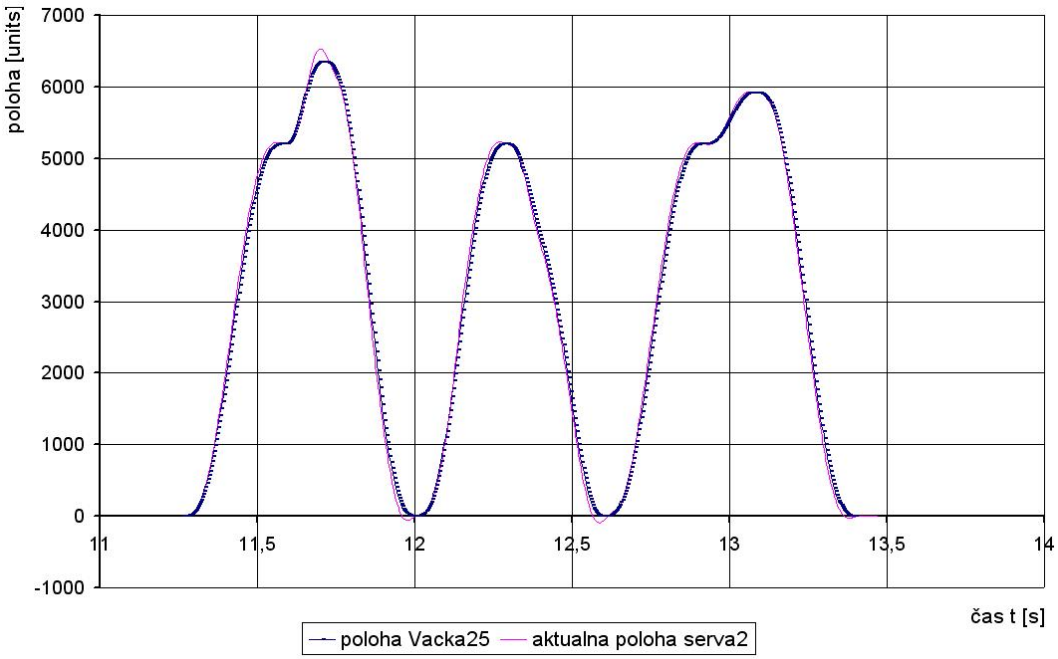


Obrázok 3.15: Vačka 25, servomotor č. 2

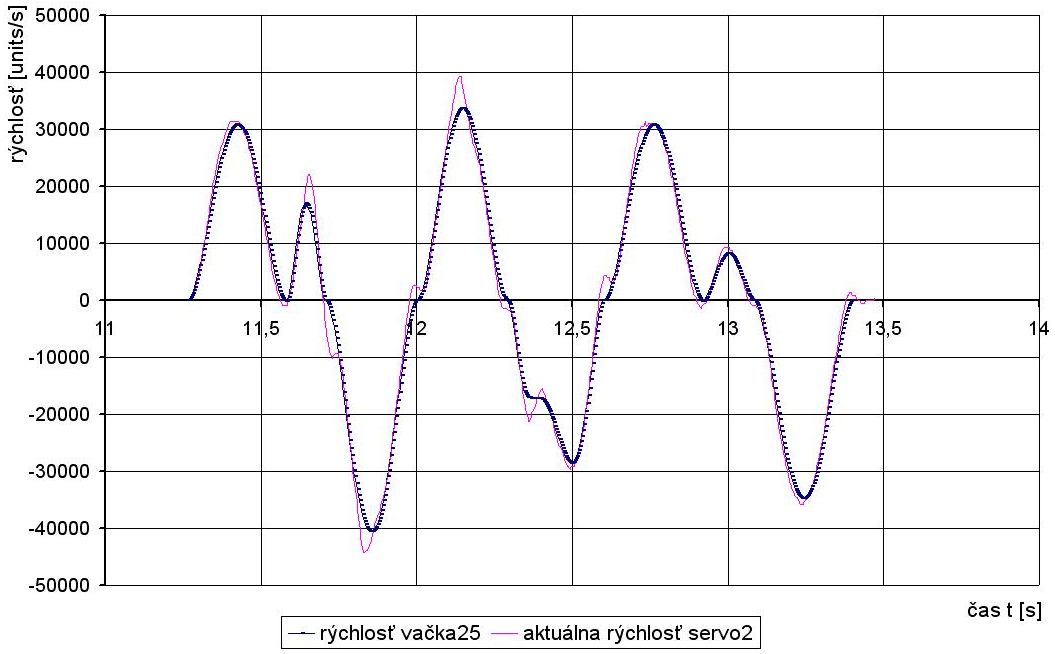
V skutočnosti sú tam patrné veľké rozdiely medzi nadefinovanými (vo vačke) a skutočnými (nameranými v *Trace*) hodnotami. Pre lepšie porovnanie som vyniesol priebehy polohy obr. 3.17 a rýchlosti obr. 3.18 do spoločných grafov. Zatiaľ čo poloha odpovedá namodelovaným vačkám, v rýchlostiach sú hlavne v špičkach veľké rozdiely. U servomotorov číslo 2 a 4 obr. 3.19 sa rozdiely rýchlosti pohybujú od 10% do 22%. Preto aj keď hodnoty zrýchlení vo vačkách dosahuju max $44\,m/s^2$, tak po pripočítani obvyklej chyby vychádza prekoročenie maximálneho zrýchlenia $> 50\,m/s^2$. U servomotorov číslo 1 a 3 obr. 3.20 sa rozdiely rýchlosti pohybujú od 7% do 30%.



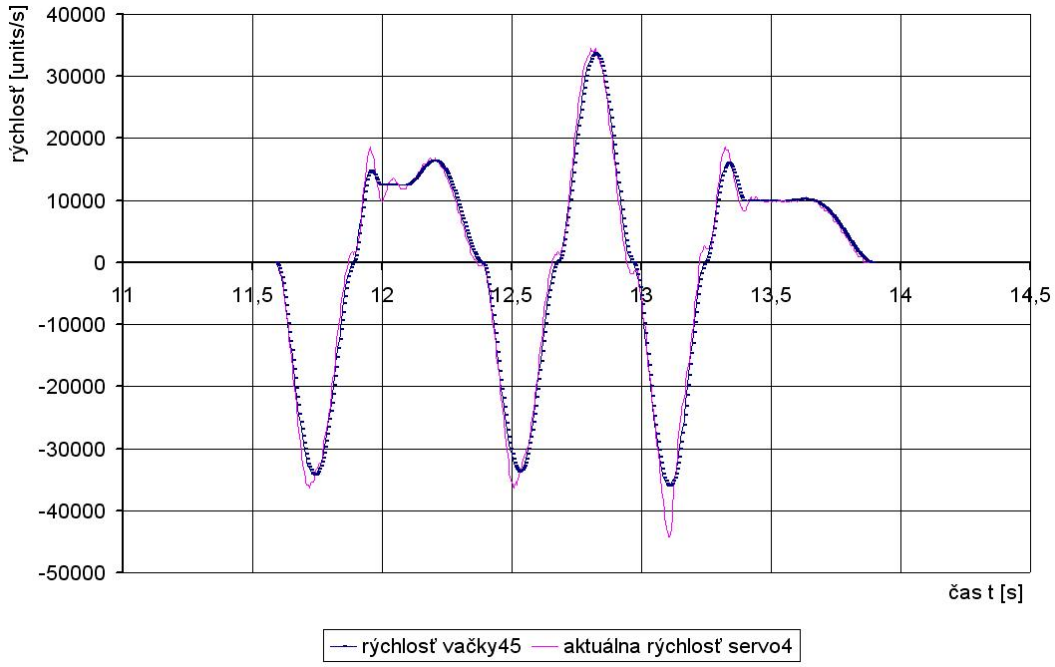
Obrázok 3.16: Priebehy rýchlosti a polohy servomotora č. 2 zmerané v *Trace*



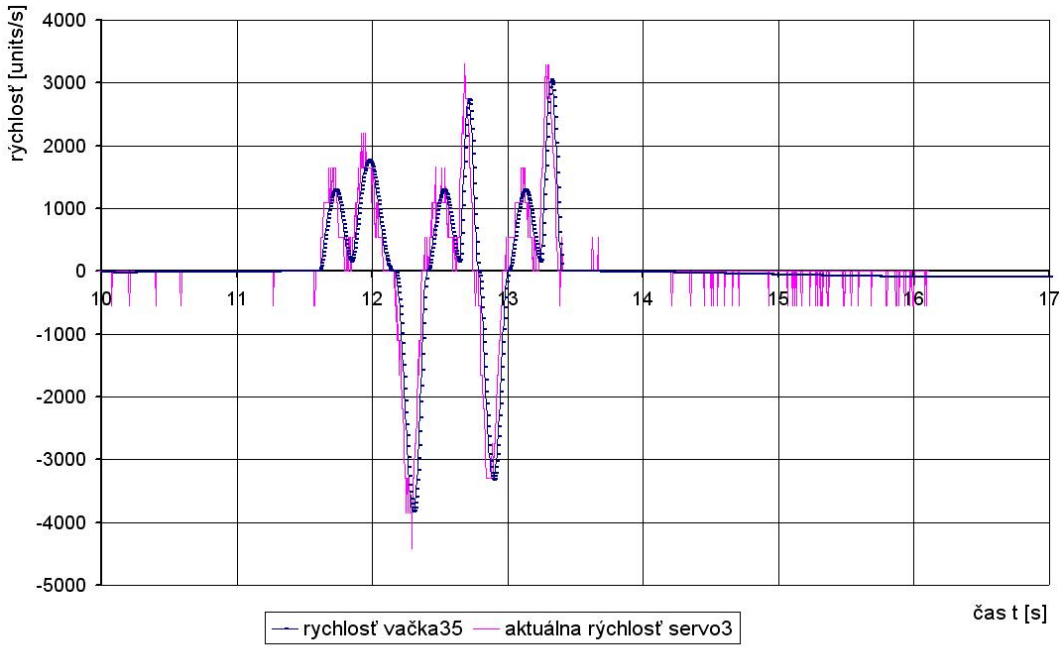
Obrázok 3.17: Porovnanie priebehu namodelovanej a skutočnej polohy servomotora č. 2



Obrázok 3.18: Porovnanie priebehu namodelovanej a skutočnej rýchlosti servomotora č. 2



Obrázok 3.19: Porovnanie priebehu namodelovanej a skutočnej rýchlosti servomotora č. 4



Obrázok 3.20: Porovnanie priebehu namodelovanej a skutočnej rýchlosti servomotora č. 3

Je zrejme, že s takýmto rozptylom hodnôt nie je možné vo vačke nadefinovať optimálne rýchlosti podľa hore spočítaných hodnôt. Preto je nutné prakticky skusmo doladiť vačku pridržiavajúc sa ideálnych spočítaných hodnôt. Takéto precizne ladenie štyroch vačiek pri troch vrhoch a troch chyteniach môže trvať dva dni. V tomto prípade sa hodí vačkový automat, ktorý dokáže vyladene pohyby v cykle opakovať. Pri skusmom dolaďovaní vačiek dochádzalo často k nechcenému prekročeniu maximálnych hodnôt zrýchlenia. Pri nízkych rýchlostiach napr. $3,3\,m/s$ nedochádza k prekračovaniu maximálneho zrýchlenia, ale nastáva časový deficit. Zvýšením rýchlosti na $4,5\,m/s$ musíme špičky v grafe so zrýchlením odstrániť dlhšou dráhou(tým sa predĺži nábehový čas), na ktorej sústava je schopná dosiahnuť vyššiu rýchlosť.

V súčasnom stave sústava dosiahne rýchlosť $4,5\,m/s$ na dráhe $0,85\,m$ pri hraničiacom zrýchlení $43\,m/s^2$ za čas $0,356\,s$. Ak by bolo možné prevodovkou umožniť navýšenie zrýchlenia o 50% (na $60\,m/s^2$), tak sa dráha nábehu skráti o 38% (na $0,53\,m$) a čas nábehu na rýchlosť $4,5\,m/s$ sa skráti tiež o 38% (na $0,222\,s$). Pri takýchto časových hodnotách je možné bezpečne praktizovať žonglovanie kaskadovitou technikou.

Z toho vyplýva, že navýšenie maximálneho zrýchlenia o 50% na $75\,m/s^2$ je postačujúce a taktiež prevodovka s prevodom $5:3$ musí splniť naše požiadavky.

3.4 Bezpečnostné PLC

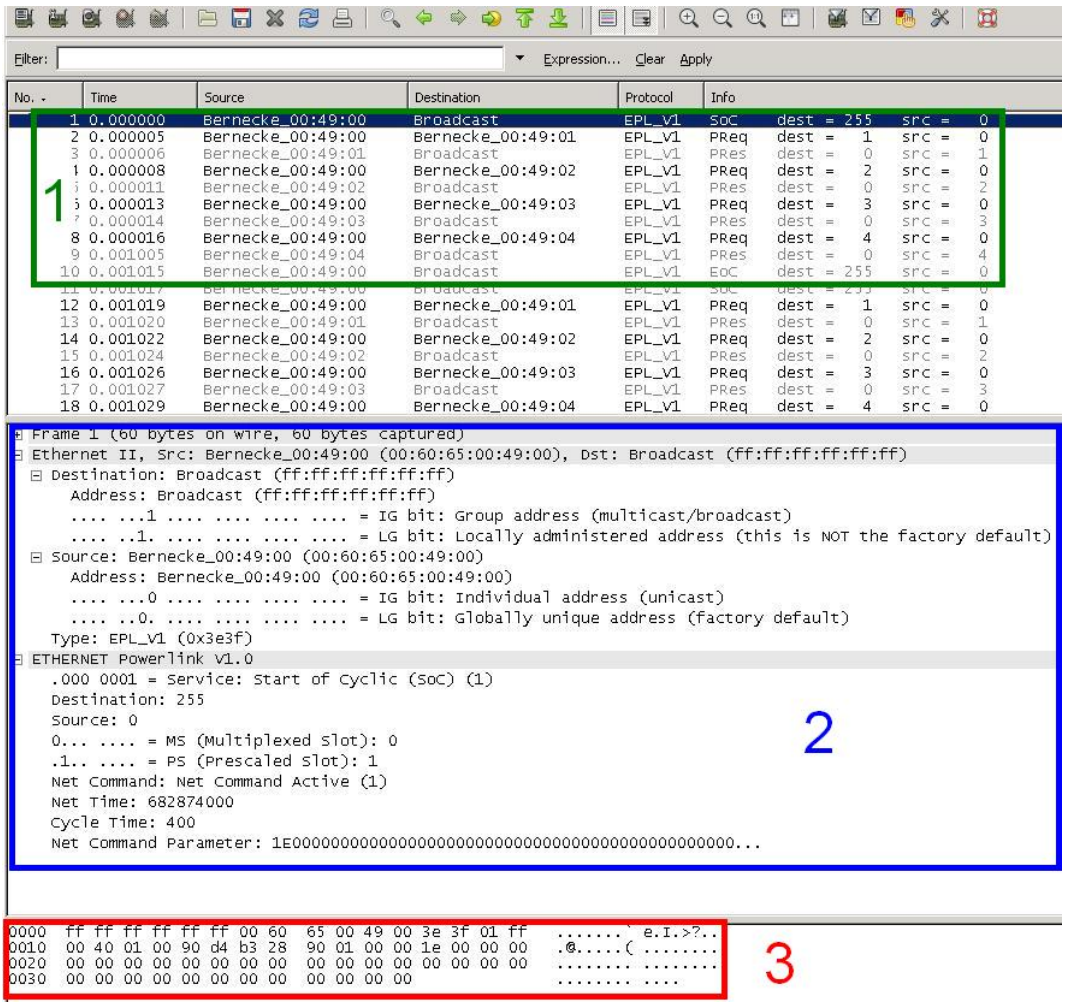
Mojou úlohou je vytvoriť program pre bezpečnostné PLC, ktorý bude dohliadať na bezpečnosť prevádzky žonglovacieho automatu. Bezpečnostné PLC musí byť nezávisle na PLC, ktoré riadi celý žonglovací algoritmus. Jednou z najlacnejších možnosti je pripojiť bezpečnostné PLC na už existujúcu sieť Ethernet Powerlink. Ako som už v teoretickej části uviedol v sieti Powerlink ma hlavnu úlohu máster, ktorý v pravidelných intervaloch (400 μs) žiada od každého uzla v sieti vyslanie dát. Všetko sa vykonáva pomocou broadcast správ. Tieto broadcast správy sa môžu odchytávať s pomocou knižnice *ETHERNET*, ktorá umožňuje funkcie ako *UDPopen, UDPclose, UDPrecv.* Po zadefinovaní príslušných premenných je možné použiť následujúci kód k odchytávaniu broadcast správ:

```
_INIT void init(void)
   { UDPopen_1.enable = 1; /* povoliť FUB */
   UDPopen_1.port = 21001; /* číslo portu*/
   UDPopen(&UDPopen_1); /* otvorenie TCP/IP portu k inému TCP/IP členu */
   receive_ready = 1; /* inicializácia */
   close_ready = 1;
   info_ready = 1;
   node_ready = 1;
   counter = 0;
}

_CYCLIC void cyclic(void)
{ if(counter == 50) /* každých 500ms */
   { if(UDPopen_1.status != 0) /* keď FUB ešte neukončila */
   UDPopen(&UDPopen_1); /* otvorenie TCP/IP portu k inému TCP/IP členu */
else if(receive_ready == 1) /* prijem dát */
   { UDPrecv_1.enable = 1; /* povolenie FUB */
   UDPrecv_1.cident = UDPopen_1.cident; /* kopírovanie ident čísla */
   UDPrecv_1.buffer = (UDINT)&buffer[0]; /* skopírovanie adresy zásobníka */
   UDPrecv_1.buflng = sizeof(buffer); /* veľkosť zásobníka */
   UDPrecv(&UDPrecv_1); /* príjem dát */
   receive_ready = 0;
   }
else if(UDPrecv_1.status != 0) /* keď FUB ešte neskončila */
```

```
   UDPrecv(&UDPrecv_1); /* príjem dát */
else if(close_ready == 1) /* ukonči spojenie */
   { UDPclose_1.enable = 1; /* povoliť FUB */
   UDPclose_1.cident = UDPopen_1.cident; /* skopírovanie ident čísla */
   UDPclose(&UDPclose_1); /* uzatvorenie spojenia */
   close_ready = 0;
   }
else if(UDPclose_1.status != 0)
   UDPclose(&UDPclose_1); /* uzatvorenie spojenia */
else if(info_ready == 1) /* získať info o prenose */
   { ETHinfo_1.enable = 1; /* povoliť FUB */
   ETHinfo(&ETHinfo_1); /* získať info o prenose */
   info_ready = 0;
   }
else if(node_ready == 1) /* získať číslo nodu a IP-adresu */
   { ETHnode_1.enable = 1; /* povoliť FUB */
   ETHnode(&ETHnode_1); /* získať číslo nodu a IP-adresu */
   node_ready = 0;
   }
counter = 0;
}
counter++; /* incrementuj */
}
```

Pred odchytávaním správ bolo nutné zistiť aký druh správ sa v sieti vyskytuje. K tomu som použil voľne dostupný program Wireshark, ktorého prostredie je znázornené na obr. 3.21. Prostredie sa delí na tri časti. V prvej časti sa vyskytujú všetky pakety, ktoré Wireshark bol schopný odchytiť s danou časovou následnosťou. V druhej časti sa nachádzajú podrobnejšie údaje o pakete a to hlavne o protokole, pomocou ktorého bol vyslaný. V tretej časti sa nachádza cely paket v hexa sústave.

V prvej časti obr. 3.22 sa dá vypozorovať celý Powerlinkový cyklus od SoC (Start of Cycle) cez PReq (Poll Request) a PRes (Poll Response) až po koniec cýklu EoC (End of Cycle). Nas zaujímajú hlavne PRes, ktoré sa tam nachádzajú postupne od Acoposu č. 1 až po č. 4 (obr. 3.23 až obr. 3.26).



Obrázok 3.21: Prostredie Wiresharku

Vypozorovať data ako rýchlosť či polohu je veľmi problematické. Pre konkrétnu verziu ACP10 SW je offset dát vždy rovnaký. Offset dát sa dá zistiť iba prečítaním ParID 1008 CYC_MON_DATA_OFFSET. Aby sme mohli prečítať ParID muselo by byť bezpečnostné PLC mastrom Powelinkovej siete. Mastrom je ale primárne PLC, ktoré obsluhuje žongléra a máster v sieti Ethernet Powerlink môže byť len jeden. To znamená, že bezpečnostné PLC môže na sieti len počúvať a nesmie do nej nič vysielať.

Data v paketoch konkrétnych Acoposov sa z nezistiteľných dôvodov štrukturovo líšia. Iný odchytený paket od Acoposu č. 4 je na obr. 3.27. Ich porovnaním by sa s neveľkou pravdepodobnosťou dalo vypozorovať čo odpovedá rýchlosti a čo polohe. Ale ďalší odchytený paket Acoposa č. 1 obr. 3.28 sa nepodobá ani na jeden z dosiaľ uvedených paketov.

1	0.000000	Bernecke_00:49:00	Broadcast	EPL_V1	SoC	dest =	255	src =	0
2	0.000005	Bernecke_00:49:00	Bernecke_00:49:01	EPL_V1	PReq	dest =	1	src =	0
3	0.000006	Bernecke_00:49:01	Broadcast	EPL_V1	PRes	dest =	0	src =	1
4	0.000008	Bernecke_00:49:00	Bernecke_00:49:02	EPL_V1	PReq	dest =	2	src =	0
5	0.000011	Bernecke_00:49:02	Broadcast	EPL_V1	PRes	dest =	0	src =	2
6	0.000013	Bernecke_00:49:00	Bernecke_00:49:03	EPL_V1	PReq	dest =	3	src =	0
7	0.000014	Bernecke_00:49:03	Broadcast	EPL_V1	PRes	dest =	0	src =	3
8	0.000016	Bernecke_00:49:00	Bernecke_00:49:04	EPL_V1	PReq	dest =	4	src =	0
9	0.001005	Bernecke_00:49:04	Broadcast	EPL_V1	PRes	dest =	0	src =	4
10	0.001015	Bernecke_00:49:00	Broadcast	EPL_V1	EoC	dest =	255	src =	0

Obrázok 3.22: Oblasť vo Wiresharku so všetkými odchytenými paketmi

```
0000  ff ff ff ff ff ff 00 60  65 00 49 01 3e 3f 04 00   .......` e.I.>?..
0010  01 01 54 00 90 d4 b3 28  a7 00 00 80 00 00 00 00   ..T....( ........
0020  00 00 01 64 00 00 22 09  18 00 2f 09 00 00 e3 32   ...d..". ../....2
0030  a7 42 00 00 00 00 00 00  00 00 00 00 00 00 00 00   .B...... ........
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0060  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....
```

Obrázok 3.23: Paket patriaci Acoposu č. 1

Východiskom zo situácie by bola podrobne popísana dokumentácia k obsahu dát prenášaných medzi Acoposom a PLC. Lenže vyvojári B&R vyhlásili embargo na obsah Powerlinkového telegramu z dôvodu obchodného tajomstva.

Ďalšou možnosťou je využitie Safety technológie od B&R. Tá poskytuje špeciálny externý EnDat 2.2 encoder, ktorý redundantne vyhodnocuje rýchlosť a polohu motora. Informácie o motore prenáša zabezpečenou cestou a využíva ich ako vstup do safety aplikácie. V tomto prípade je nutné zhodnotiť finančnú stránku riešenia.

```
0000  ff ff ff ff ff ff 00 60  65 00 49 02 3e 3f 04 00   .......` e.I.>?..
0010  02 01 54 00 90 d4 b3 28  5d 00 ea 03 00 00 00 00   ..T....( ].......
0020  00 00 01 52 00 00 20 01  18 00 5d 02 00 00 00 00   ...R.. . ..].....
0030  00 00 5d 02 00 00 00 00  00 00 00 00 00 00 00 00   ..]..... ........
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0060  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....
```

Obrázok 3.24: Paket patriaci Acoposu č. 2

```
0000  ff ff ff ff ff ff 00 60  65 00 49 03 3e 3f 04 00   .......` e.I.>?..
0010  03 01 54 00 90 d4 b3 28  a7 00 00 80 00 00 00 00   ..T....( ........
0020  00 00 01 55 00 00 20 09  18 00 90 fb ff ff 80 65   ...U.. . .......e
0030  26 42 90 fb ff ff 80 65  26 42 00 00 00 00 00 00   &B.....e &B......
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0060  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....
```

Obrázok 3.25: Paket patriaci Acoposu č. 3

```
0000  ff ff ff ff ff ff 00 60  65 00 49 04 3e 3f 04 00   .......` e.I.>?..
0010  04 01 54 00 90 d4 b3 28  5d 00 72 17 00 00 00 00   ..T....( ].r.....
0020  00 00 01 54 00 00 20 01  18 00 6a 01 00 00 00 00   ...T.. . ..j.....
0030  00 00 6a 01 00 00 00 00  00 00 00 00 00 00 00 00   ..j..... ........
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0060  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....
```

Obrázok 3.26: Paket patriaci Acoposu č. 4

```
0000  ff ff ff ff ff ff 00 60  65 00 49 04 3e 3f 04 00   .......` e.I.>?..
0010  04 01 54 00 c0 ff b7 28  a7 00 72 17 00 00 00 00   ..T....( ..r.....
0020  00 00 01 55 00 00 20 09  18 00 90 01 00 00 e0 7f   ...U.. . ........
0030  15 44 90 01 00 00 e0 7f  15 44 00 00 00 00 00 00   .D...... .D......
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0060  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....
```

Obrázok 3.27: Paket patriaci Acoposu č. 4

```
0000  ff ff ff ff ff ff 00 60  65 00 49 01 3e 3f 04 00   .......` e.I.>?..
0010  01 01 54 00 c0 97 46 98  92 01 db f6 ff ff 00 00   ..T...F. ........
0020  00 00 01 69 00 00 20 2b  98 00 fc fa ff ff af 6f   ...i.. + .......o
0030  25 41 0b 00 00 00 fe ff  7f 3f 00 00 00 00 00 00   %A...... .?......
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0050  0b 00 00 00 88 14 52 3f  00 00 00 00 00 00 00 00   ......R? ........
0060  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....
```

Obrázok 3.28: Paket patriaci Acoposu č. 1

Kapitola 4

Záver

Jedným z cieľov tejto práce bolo vytvorenie troch študentských pracovišť. Tieto pracoviština pozostávajú z uceleného systému vytvoreného z Power Panelu, Acoposu (digitálneho servozosilovača) a synchrónneho servomotru. Pre komunikáciu medzi členmi modulu bolo využité rozhranie CAN. K týmto pracovištiam som vypracoval päť cvičení, ktoré umožňujú študentom osvojiť si schopnosti od naladenia polohových a rýchlostných regulátorov, cez základné myšlienky programovania v AnsiC pomocou nc_Akcii, až po vysvetlenie vačiek, ich modelovanie a použitie vo vačkovom automate. Všetky cvičenia sú uvedené v prílohe. Samostatný mnou vytvorený výukový model vytvára základnú stavebnú jednotku žonglovacieho automatu.

Ďalším cieľom práce bolo rozšírenie žonglovacieho automatu o tretiu os. Oneskorenou dodávkou hardwarových části (lineárny motor) sa nepodarilo včas fyzicky nainštalovať tretiu os do samotného žonglovacieho mechanizmu. Ničmenej rozšírenie softwarovej části je podrobne rozpísane v príslušnej kapitole. Využitím poznatkov z cvičení a podrobného popisu programového rozšírenia, umožní nainštalovanej tretej osi jednoduché začlenenie do procesu žonglovania.

Pri analýze dynamiky súčasných dvoch vertikálnych os som zistil, že motory nespĺňajú momentove požiadavky pre dostatočnú akceleráciu potrebnú k žonglovaniu s tromi loptičkami. Pre tento účel som navrhol prevodovku 5:3, ktorá zvýši zrýchlenie skoro na dvojnásobok. Tým sa skráti čas potrebný na presun ramena z vyhadzovacej do chytacej pozície a zároveň skráti dráha potrebná k dosiahnutiu požadovanej vyhadzovacej rýchlosti $4,5\,m/s$.

Pri modelovaní a testovaní vačiek dochádzalo k odchylkám skutočných odmeraných a namodelovaných hodnôt. Odchylky narastali so zvyšujucími sa požiadavkami na vyššiu rýchlosť za kratší čas. Najmarkantnejšie odchylky vznikali v úsekoch hraničiacich s ma-

ximálnym možnym momentom, ktorý bol motor schopný poskytnúť. Pri návrhu prevodovky som bol obmedzený maximálnym momentovým zaťažením lineárneho modulu ($80\,N$) a maximálnymi otáčkami motoru ($1791\,ot/min$) pri ktorých ešte bol schopný poskytnúť maximálny moment $40,5\,N$. Preto po nainštalovaní prevodoviek sa odchylky markantne neznížia, lebo od sústavy motor-prevodovka budeme vyžadovať vyšší výkon.

Pri analýze fungovania bezpečnostného PLC na základe odpočúvania siete, odchytávania broadcast správ od Acoposov a ich vyhodnocovania, som narazil na nedeterministické správanie obsahu dát Powerlinkového telegramu. Dokumentácia k týmto balikom posielaným medzi PLC a Acoposom nieje verejnosti prístupná z dôvodu obchodného tajomstva.

Literatúra

[1] B&R Automation, *ACOPOS User's Manual, Version 1.3.1.* 2004.

[2] B&R Automation, *8MS Three-phase Synchronous Motors User's Manual, Version 1.0.* 2004.

[3] B&R Automation, *Automation Studio Programming, Version 2.0.* 2001.

[4] B&R Automation, *Automation Studio System Introduction, Version 2.0.* 2001.

[5] B&R Automation, *Positioning Introduction, Version 0.47x.0.* 2001.

[6] B&R Automation, *ASiM Multi-Axis Functions TM441.* 2005.

[7] B&R Automation, *Power Panel PP35 User's Manual, Version 2.0.*

[8] Pšenička, J. *Synchronizace servopohonu :* diplomová práce. Praha : ČVUT-České vysoké učení technické v Praze, Fakulta elektrotechnická, 2005.

[9] *CAN in Automation (CiA)* [online], 2008
`http://www.can-cia.org/`

[10] Kalvan, J. *The Analysis and Over-analysis of Juggling Patterns* [online], 2008
`http://www.juggling.org/`

[11] Voss, J. *The Mathematical Theory of Juggling* [online], 2008
`http://seehuhn.de/`

[12] *Ethernet POWERLINK* [online], 2008
`http://www.epl-tools.com/`

[13] *Wikipedia* [online], 2008
`http://en.wikipedia.org/`

Dodatok A

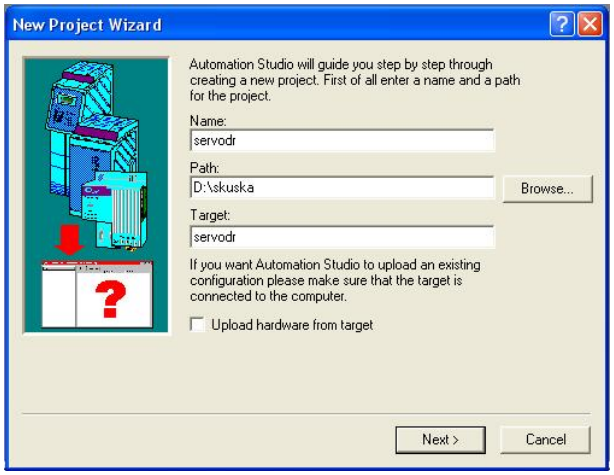
Cvičenia k laboratórnemu vyučbovému modelu

A.1 Cvičení č.1 - Založení projektu

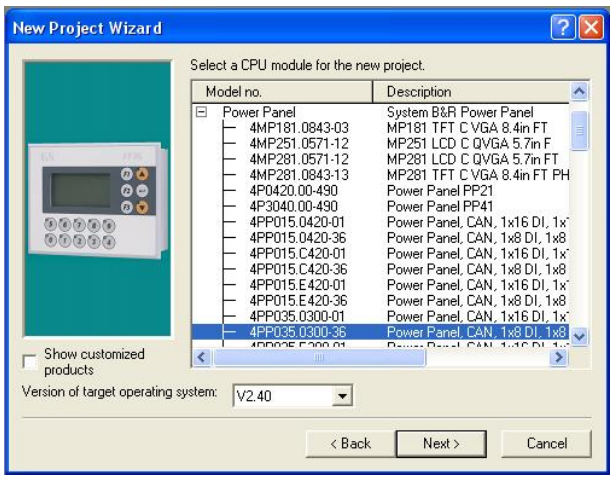
Při rozsáhle výrobě (od pružin až po sváření sáčků v potravinářském průmyslu) se využívá polohování pomocí servomotorů, které zvládají vysoké nároky kladené na dynamické charakteristiky motoru. Motory s takovými dynamickými vlastnostmi jsou preferované před asynchronními motormy kvůli zkvalitnění a hlavně zrychlení výrobních procesů. Tento model, kterého jádro tvoří PLC s displejem, inteligentní servoměnič a servomotor s enkodérem ve zpětné vazbě pomůže studentovi získat ucelenou představu o funkčnosti celého systému. Pro začátek je potřeba všechny prvky tohoto systému provázat aby navzájem komunikovaly. Pro úspěšnou komunikaci je zapotřebí nastavit spoustu tabulek a parametrů přes které vas provede toto cvičení.

A.1.1 Power Panel, Acopos - měnič, Servomotor

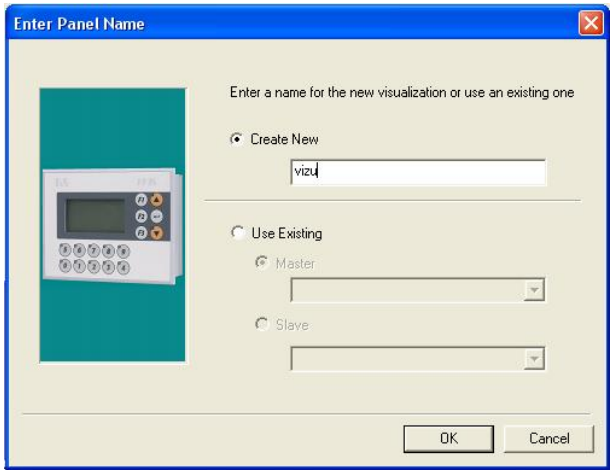
Při zakládání nového projektu obr.A.1 si vybereme z nabídky Power Panel, který máme umístěný v modelu(pozor nejsou všechny stejné i když tak vypadají). Power Panel vybíráme podle výrobního čísla na obr.A.2, které se nachází na štítku v zadní části panelu.



Obrázok A.1: Vytvoření nového projektu



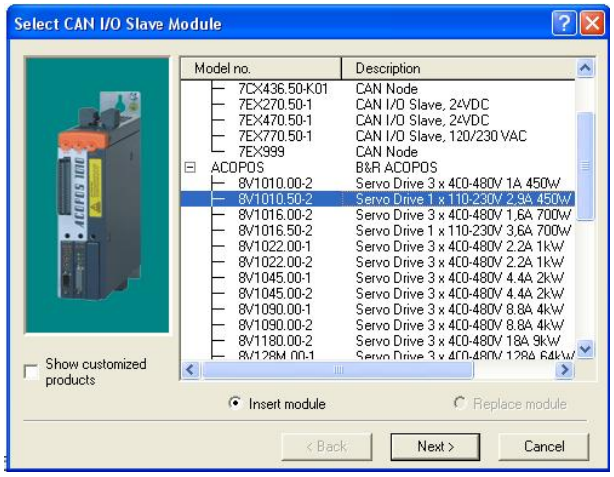
Obrázok A.2: Výběr Power Panelu



Obrázok A.3: Pojmenování vizualizace

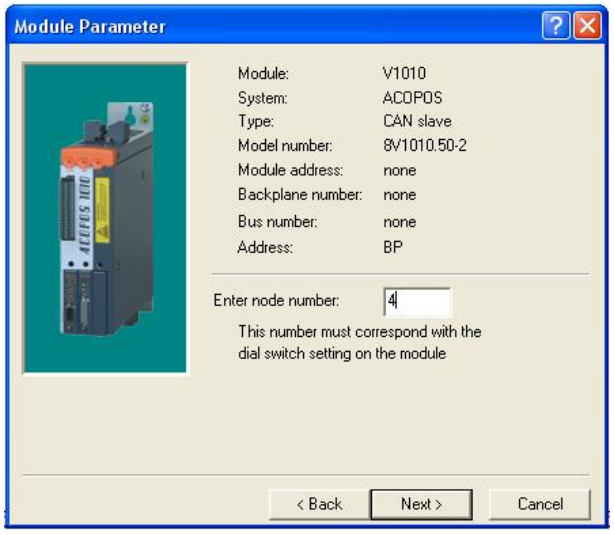
Vytvoření projektu si vyžaduje pojmenování vizualizace obr.A.3, která se připojuje automaticky - pak v dalších cvičeních se k ní vrátíme.

Po vytvoření projektu je třeba do něj vložit acopos - měnič, který je spojen přes



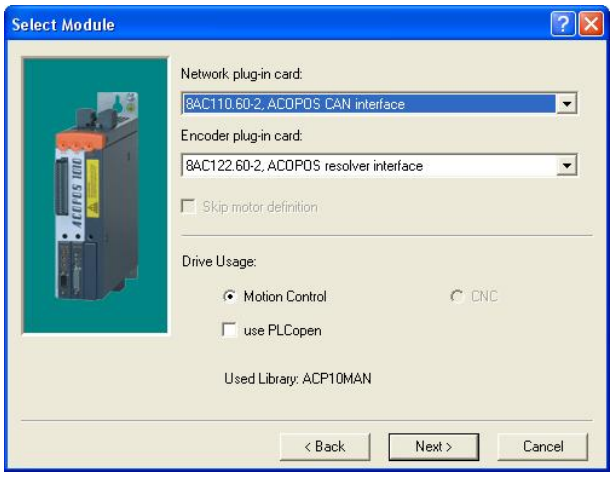
Obrázok A.4: Výběr měniče

CAN s PLC. Výběr meničů na obr.A.4 dosahneme tak, že klikneme na PowerPanel v levém okně a pak v pravém okně klikneme na záložku *CAN* a v ní pravým tlačítkem myši na *IF2(CAN)* → *insert*. Vybraný měnič musí také korespondovat s číslem uvedeném na fyzickém acoposu jako u PLC.



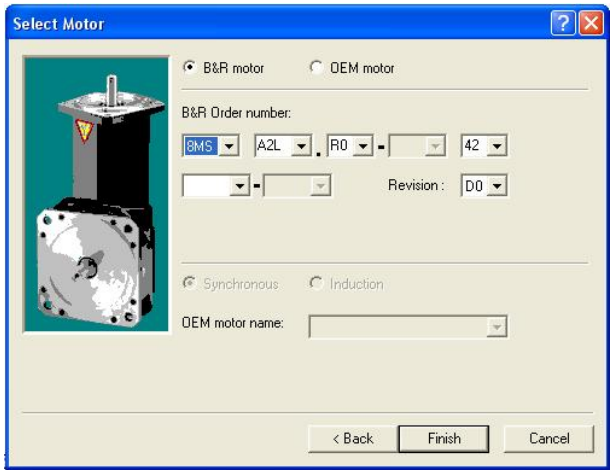
Obrázok A.5: Nastavení identifikačního čísla na sběrnici

Pro komunikaci po sběrnici CAN je nezbytně nutné zadat "node numberöbr.A.5 pro acopos, které slouží pro identifikaci na sběrnici. Toto číslo musí korespondovat s číslem na acoposu na canovské kartě.



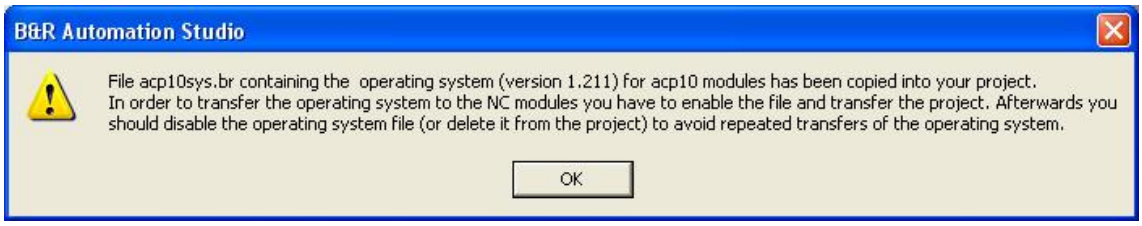
Obrázok A.6: Výběr přidavných modulu acoposu

V dalším okně na obr.A.6 vybereme použité přídavné moduly - CAN pro komunikaci s PLC a Resolver interface pro komunikaci s motorem ve zpětné vazbě. Nezapomeňte odškrtnout políčko *use PLC open*. Tuto knihovnu nemůžeme použít, protože naše PLC nemají dostatek paměti.



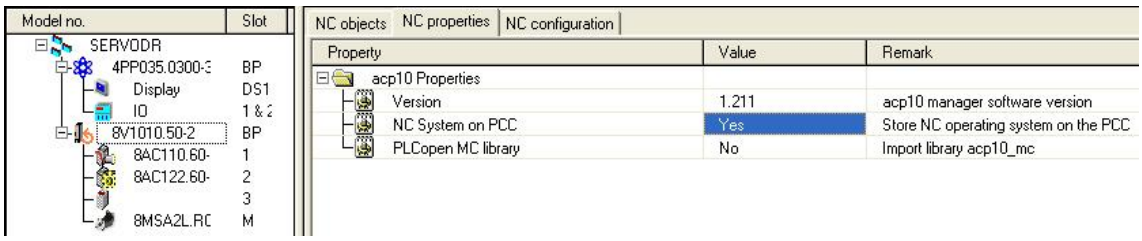
Obrázok A.7: Výběr motoru

V dalším okně na obr.A.7 musíme zadat výrobní číslo motoru kvůli korektnímu načítaní jeho parametrů. Číslo se nachází na štítku motoru.



Obrázok A.8: Hlášení systému

Hlášení na obr.A.8 říká, že po zkompilovaní projektu a nahrání souboru acp10sys do acoposu je potřeba tento soubor deaktivovat aby se nenahrával vždy, když potřebujeme nahrát náš projekt do PLC. Abychom vůbec mohli tento soubor nahrát musíme ho přidat do projektu způsobem - že v levém okně klikneme ve stromu na acopos a v pravém okně pod záložkou *NC properties* → *NC systém on PCC* → *YES* obr.A.9

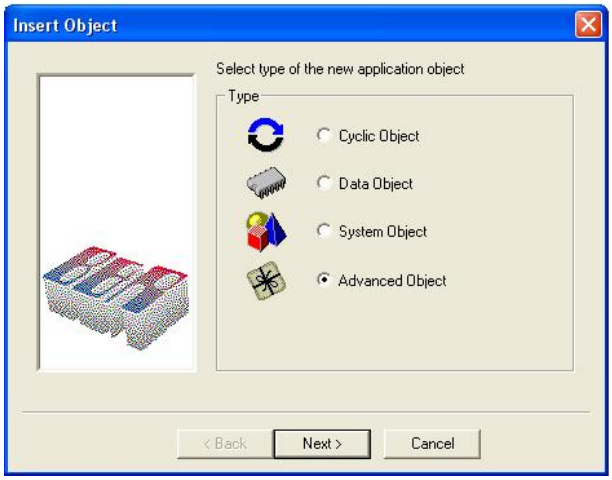


Obrázok A.9: Povolení nahrávání systému do acoposu

A.1.2 Parametrizační tabulka měniče, inicializační tabulka osy, depoyment tabulka

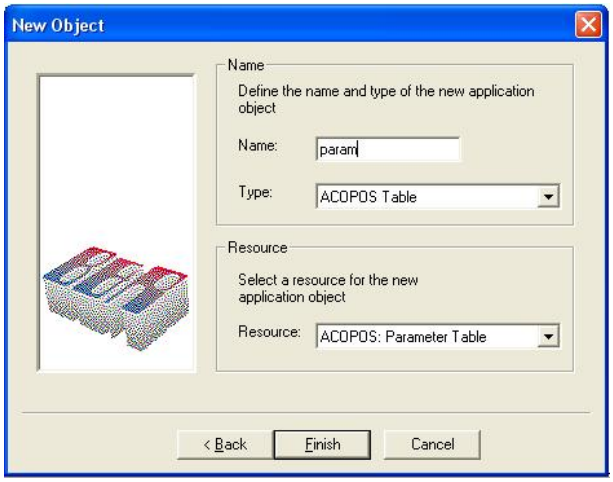
A.1.2.1 Parametrizační tabulka měniče

Parametrizační tabulku vložíme stejně jako cyklický objekt - v levém okně klikneme na PLC a pak v pravém okně pod záložkou *Software* klikneme pravým tlačítkem myši na *CPU* → *Insert Object* - tím se nám zobrazí okno viz obr.A.10



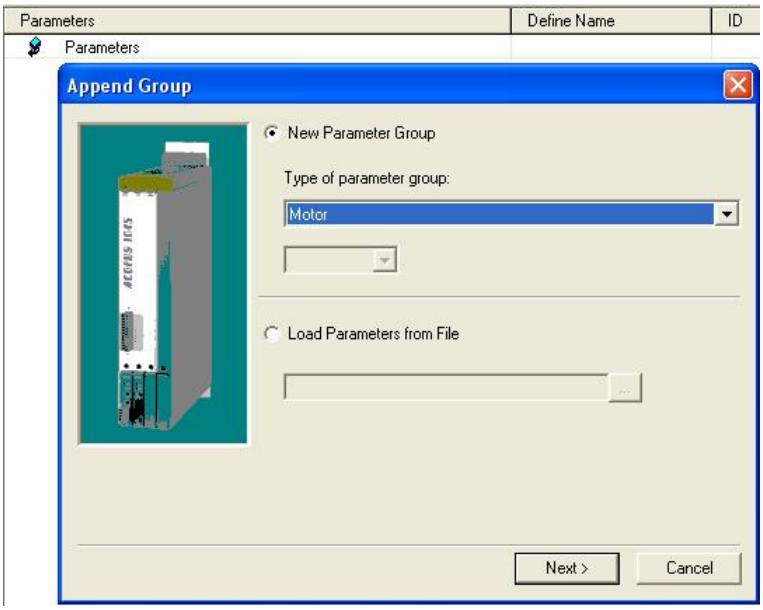
Obrázok A.10: Vkládání objektů do projektu

Pro přidáni parametrizační tabulky si zvolíme *Advanced Object* → *Acopos Table* a pojmenujeme ji maximálně 8 znaky, viz obr.A.11



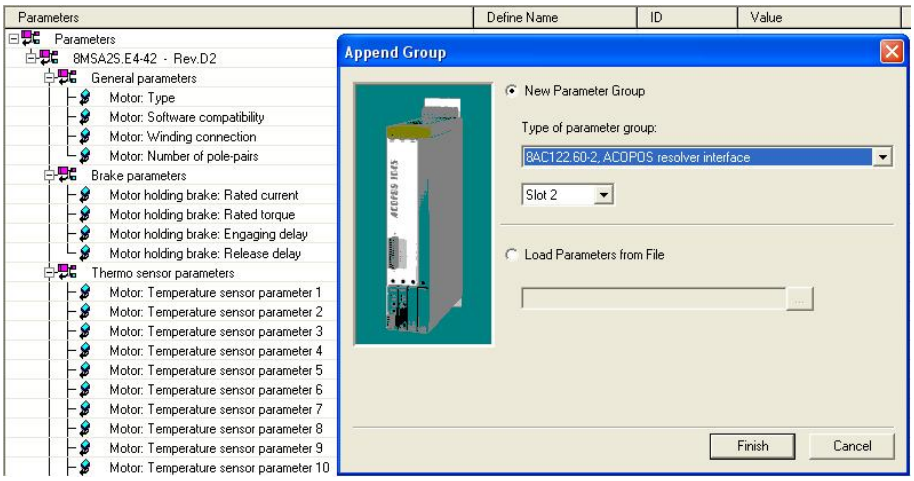
Obrázok A.11: Vytvoření parametrizační tabulky

Po vytvoření parametrizační tabulky se objeví prázdná tabulka s položkou *Parameters*. Tuto tabulku je možné vyvolat kdykoliv, dvojklikem ve stromě hlavního projektu na položku pod *ACOPOS:Parameter table*. Pravým tlačítkem myši klikneme na *Parameters* → *Append Group* a objeví se okno jako na obr.A.12. V nabídce *New Parameter Group* si zvolíte *Motor* a navolíte výrobní číslo podle štítku motoru viz obr.A.7.



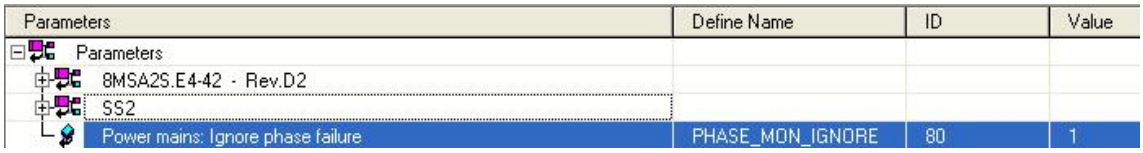
Obrázok A.12: Parametre motoru

Obdobným postupem si navolíte i modul *ACOPOS resolver interface* obr.A.13.



Obrázok A.13: Přidání modulu resolver interface

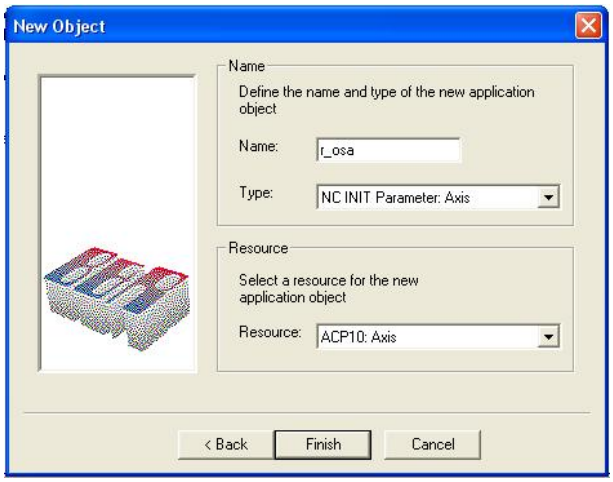
Na konec přidáme důležitý parameter Ignore phase failure - bez kterého se měnič ani nerozběhne, protože jeho systém uvnitř bude hlásit, že vypadla fáze. Jenže my ho napájíme jenom jedinou fází 230V a ne třemi. Pravym tlačítkem klikneme na *Parameters* → *Insert parameter* a do ÏD"vzniklého parametru napíšeme číslo 80 a do hodnoty "value"napíšeme hodnotu 1 viz obr.A.14.



Obrázok A.14: Ignorování výpadku fáze

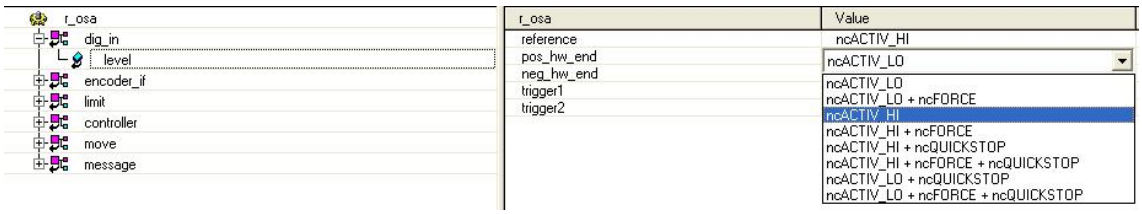
A.1.2.2 Inicializační tabulka osy

Obdobným způsobem jako v podkapitole A.1.2.1 a na obrázcích obr.A.10 a obr.A.11 si vytvoříme inicializační tabulku reálné osy *NC INIT Parameter: Axis* → *ACP10: Axis* obr.A.15



Obrázok A.15: Vytvoření init tabulky reálné osy

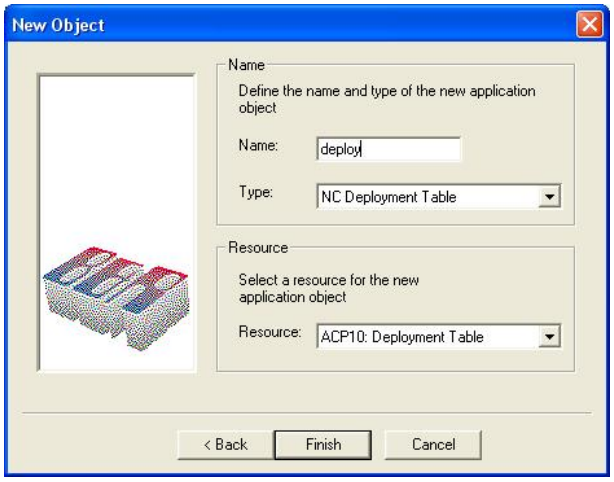
Po vytvoření inicializační tabulky ji otevřeme v hlavním stromě dvojklikem na položku nacházející se pod *ACP10: Axis*. V ní pro začátek pozměníme položku *dig_in* → *level* a všechny parametry změníme na ncActive_HI jako na obr.A.16.



Obrázok A.16: Uprava inicializační tabulky

A.1.2.3 Depoyment tabulka

Pro vytvoření deployment tabulky je postup stejný jako v podkapitole A.1.2.1 a A.1.2.2. Zvolíme si *NC Deployment Table* → *ACP10: Depoloyment Table* jako na obr.A.17. V deployment tabulce si určíme *NC Object Name* - je to jméno naši osy,



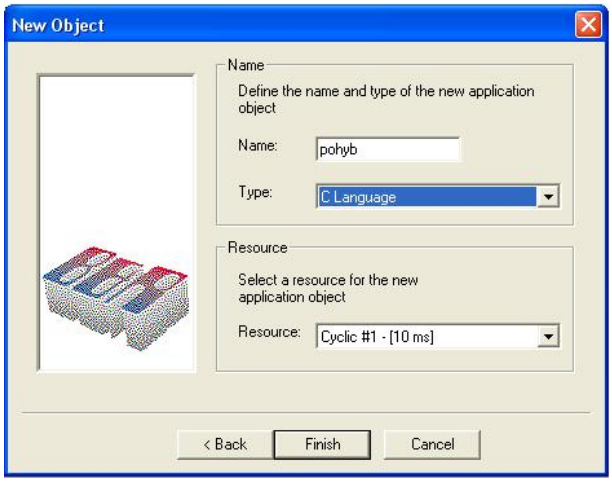
Obrázok A.17: Vytvoření Deployment tabulky

které v budoucnu použijeme v našem programu. Jméno slouží jako ukazatel na konkrétní měnič s motorem (které představují v našem případe reálnou osu) s nějakou inicializační tabulkou. Výhodou deployment tabulky je to, že pokud máme více inicializačních tabulek, tak je můžeme různě kombinovat s danou reálnou osou. V případě, že máme víc stejných reálných os, můžeme použít jednu inicializační tabulku pro všechny osy. Dalším parametrem je *Network Interface* - kde máme na výběr jenom CAN. Další parametr je *Node Number* - to je identifikační číslo na sběrnici CAN. *NC Object Type* je druh naší osy, která je reálná, čili ncAxis. Pak následuje *NC INIT Parameter*, zde si vybereme naši inicializační tabulku a v *ACOPOS Parameter* si vybereme parametrizační tabulku měniče jako na obrA.18.

NC Object Name	Network Interface	Node Number	Advanced	NC Object Type	NC INIT Parameter	ACOPOS Parameter
real_osa	IF2 (CAN)	4		ncAXIS	r_osa	param

Obrázok A.18: Nastavení Deployment tabulky

Nakonec nesmíme opomenout cyklický objekt v jazyce C obr.A.19.

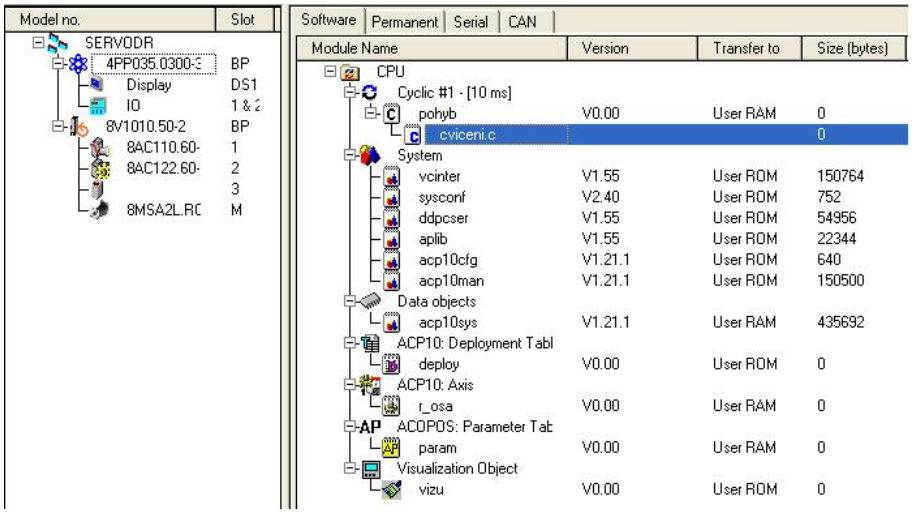


Obrázok A.19: Vytvoření cyklického objektu

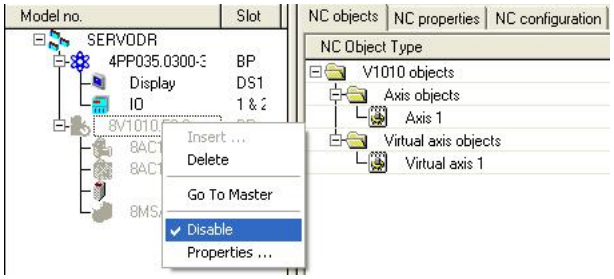
A.1.3 Celkový přehled a sériová komunikace

Celkový přehled projektového stromu v záložce *Software* viz obr.A.20.

Poté můžeme projekt skompilovat a acopos v levém okně deaktivovat viz obr.A.21.

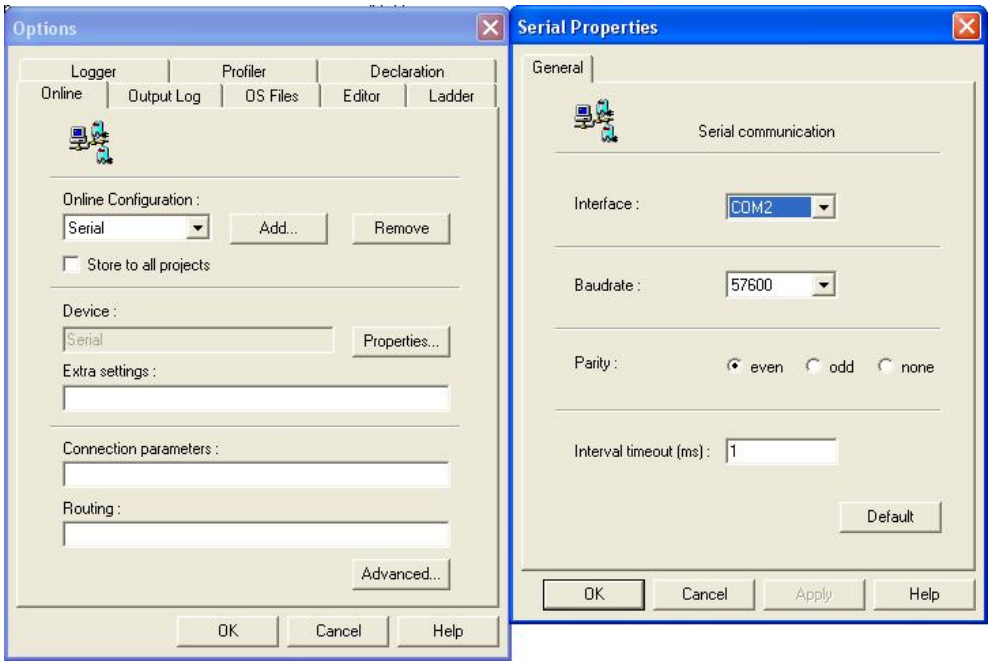


Obrázok A.20: Vytvorený projekt



Obrázok A.21: Deaktivace acoposu

Nahrání projektu do PLC si vyžaduje funkční sériovou komunikaci RS232 počítač ⇒ PLC. Ta se nastavuje v *Tools* → *Options* → *Online* → *Serial* → *Properties* → *COM1* nebo *COM2* podle toho, ve kterém sériovém portu je zapojen konektor viz. obr.A.22



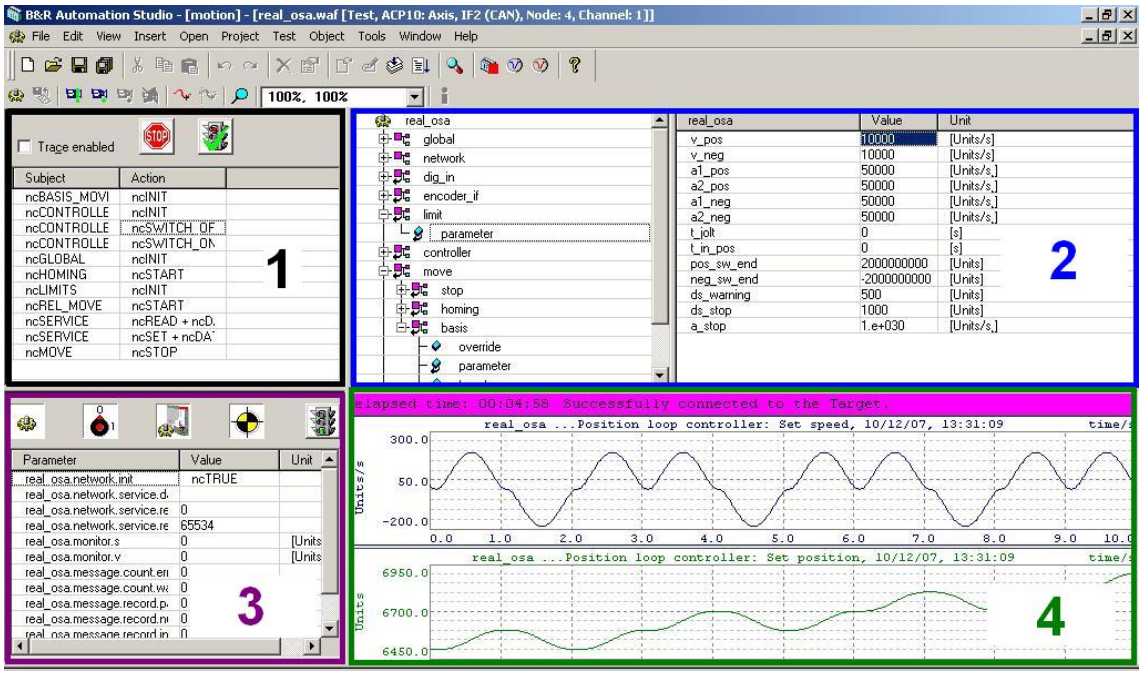
Obrázok A.22: Seriová komunikace

A.2 Cvičení č.2 - Ladění motoru v prostředí Test

Před začátkem programování systému je potřebné nastavit proporcionální a integrační konstantu regulátoru (k_P, k_I), které se nachází v měniči. Pomocí těchto konstant ovlivňujeme statické a dynamické vlastnosti motoru. Kdysi se tyto vlastnosti nastavovaly öd oka"pomocí hmatu a sluchu. Proto B&R vyvinula prostředí *Test* ve kterém můžeme posílat příkazy motoru bez toho abychom museli něco programovat. Pak pomocí zpětné vazby můžeme sledovat stav měniče a motoru (např. chyby, stav spojení atd.) nebo přímo nastavovat parametry jako je zrychlení, dráha relativního posuvu nebo konstanty regulátoru. Tento způsob je velice přehledný a názorný - dnes ho využijeme pro rozhýbání motoru a poznatky se přiblížíme i k jeho programování které nas čeká v příštim cvičení.

A.2.1 Prostředí Test

A.2.1.1 Celkový přehled prostředí Test



Obrázok A.23: Celkový přehled prostředí *Test*

Na obr.A.23 v poli označeném č.1 jsou znázorněné některé nc_akce. NC akce jsou povely pro měnič. Při vytváření projektu s acoposem se nám automaticky přidá do projektu NC Manager. NC Manager se nachází v PLC a vykonáva komunikaci mezi

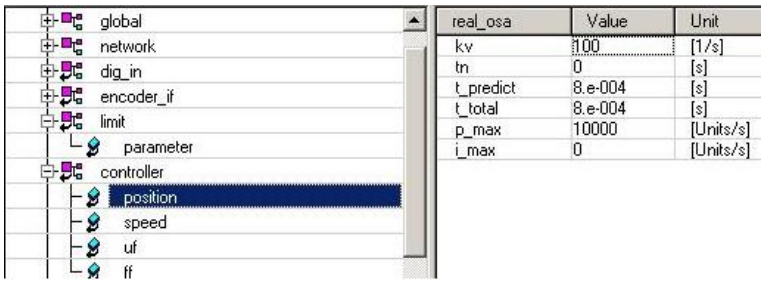
měničem a PLC. V *Testu* můžeme dvojklikem spustit např. ncREL_MOVE → ncStart následně příkaz převezme NC manager a ten ho odešle operačnímu systému měniče. A měnič pomocí regulátorů pohne motorem.

V poli označeném č.2 je znázorněn strom parametrů pro měnič - limitní parametry, parametry regulátoru, globální parametry atd. V poli č.3 jsou zpětnou vazbou přiváděné signály definující stav v jakém se nachází měnič a motor. To že předáme NC manageru nějaký příkaz, např. ncREL_MOVE, ještě neznamená, že se tento příkaz vykoná. Když na vykonání tohoto příkazu nebudou splněné počáteční podmínky, příkaz se do měniče odošle ale nevykoná a ve většině případú vrátí chybu, pomoci které můžeme zjistit, co jsme nesplnili.

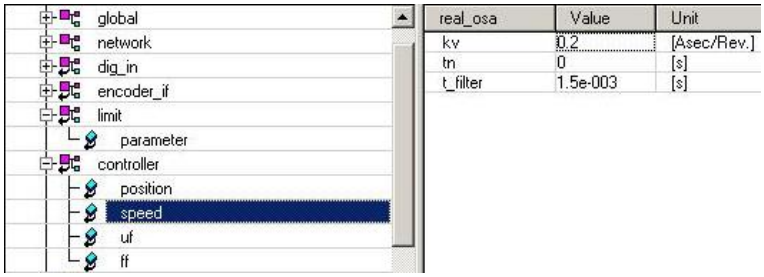
Ve čtvrtém poli jsou umíštené grafy. Například graf aktuální rychlosti nebo graf lag erroru, které nám dopomohou ke správnemu nastavení regulátorů v měniči.

A.2.1.2 Nastavování parametrů regulátoru pro servomotor

Pro zjednodušení ladění budeme používat jenom 3 parametry, které se nachází ve stromu v pravé části Testového okna pod záložkou *controller→position→kv* a *controller→speed→kv, t_filter*, tak jako, to vidíme na obrazcích obr.A.24 a obr.A.25. Před laděním je nutné všechny tři parametry vynulovat



Obrázok A.24: Znázornění parametrů pozičního regulátoru



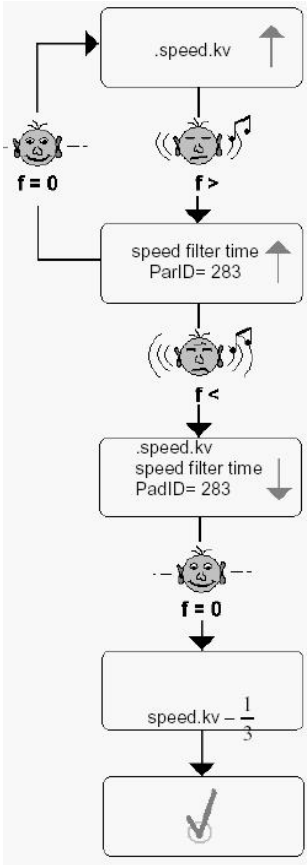
Obrázok A.25: Znázornění parametrů rýchlostního regulátoru

Abychom viděli změny v nastavení, je potřeba motor spustit. Při spuštění motoru je nutné dodržet korektní sekvenci volání nc Akcí, jinak se příkazy nevykonají a semafor bude signalizovat chybu, kterou lze zrušit tlačítkem na obr.A.26.

Obrázok A.26: Semafor signalizující chyby systému a tlačítko pro odstránění chyb

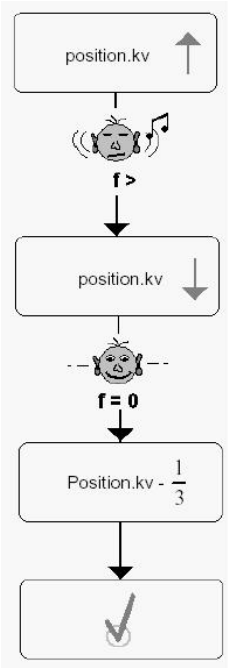
Po každé změne parametrů je potřebné nejdřív vypnout controller pomocí *ncCONTROLLER → ncSWITCH_OFF* pak spustit globální inicializaci pomocí *ncGLOBAL→ncINIT* a pak můžeme zapnout controller *ncCONTROLLER→ncSWITCH_ON*.

Samotné ladění začína parametrem *controller→speed→kv* tak, že jeho počáteční hodnota je 0,1 As/Rev. Na obr.A.27 vidíme, že kv postupně zvyšujeme, dokud hřídel motoru nezačne oscilovat(oscilace zaregistrujeme jenom tehdy, když je controller zapnutý tak, jak je výše uvedeno). Když uslyšíme oscilaci, začneme nastavovat hodnotu t_filter s počáteční hodnotou 0,2msec. Filtrační hodnotu zvyšujeme, dokud oscilace nepřestane. Pak zase zvyšujeme hodnotu kv. Tento cyklus opakujeme dokud neuslyšíme hluboké tóny oscilace. Pak přejdeme do dalšího kroku na obr.A.27 a snížíme obě hodnoty kv i t_filter dokud oscilace nepřestane. V dalším kroku snížíme hodnotu kv o 1/3 na pracovní hodnotu.



Obrázok A.27: Nastavení rýchlostního regulátoru

Po nastavení rýchlostního regulátoru začneme s pozičním regulátorem *controller→position→kv* podle obr.A.28.

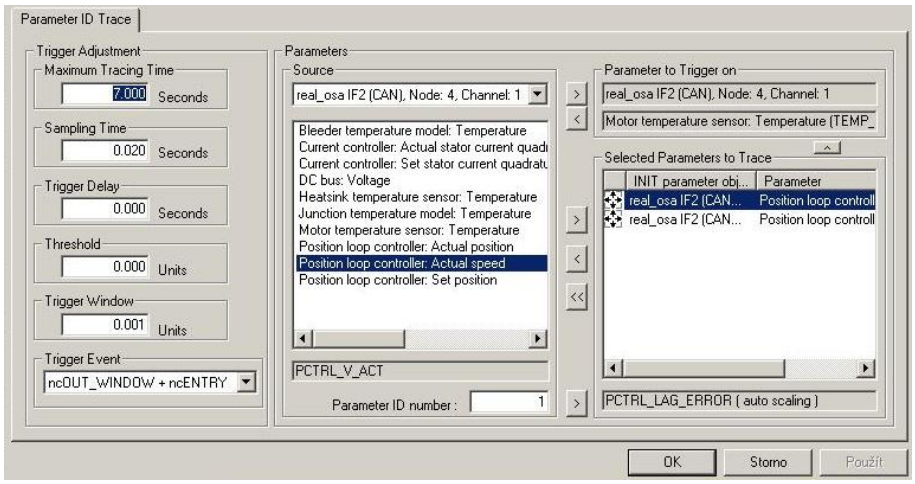


Obrázok A.28: Nastavení pozičního regulátoru

Hodnotu kv zvyšujeme dokud nezaznamenáme oscilaci. Pak kv snížíme na bezoscilační stav a následně odečteme 1/3. Tím nastavíme kv na pracovní hodnotu. Velikost konstanty kv u pozičního regulátoru je přímouměrná síle, která klade odpor rotačním silám vychylujícím rotor z jeho primární polohy.

A.2.1.3 Jemné doladění dynamických vlastností motoru pomocí grafického zobrazení

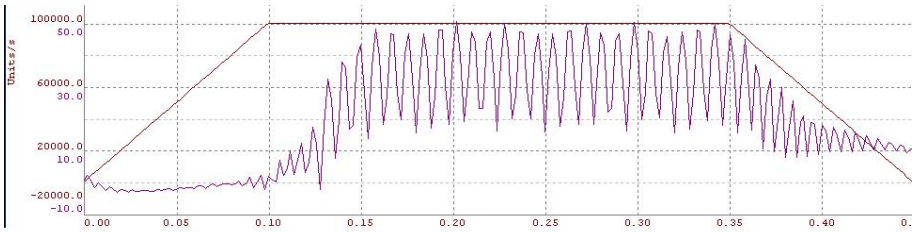
Nyní, po nastavení základních konstant a odstranění oscilací můžeme roztočit motorem. Použijeme relatívní pohyb ncREL_MOVE a pro tuto nc akci musíme mít nastavenou hodnotu *move* →*basis* →*parameter*→*s* například na 1000 units, což je jedna otáčka. Pak pro otočení o jednu otáčku, spustíme *ncGLOBAL* →*ncINIT*, *ncCONTROLLER* →*ncSWITCH_ON*, *ncHOMING* →*ncSTART* a nakonec *ncREL_MOVE*→*ncSTART*. Pomocí grafického zobrazení si můžeme vykreslit například lag error. Lag error je rozdíl mezi nastavenou a aktuální pozici rotoru. Při vačkových profilech, které jsou použité ve složitejších automatizačních procesech, je více os zpřevodovaných za pomocí virtualních převodovek a lag error způsobí veliké zpoždění mezi osami. Tento jev snižuje kvalitu výroby, proto se kompenzuje dodatečným snížením rýchlostí. Lag error si můžeme zobrazit kliknutím na pravé tlačítko myši v oblasti grafů a výběrem položky *Options*. Tím se nám zobrazí panel jako na obr.A.29. V jeho středu jsou křivky, které si můžeme nechat zobrazit a v pravé části panelu jsou křivky, které budou zobrazené při spuštění sledování.



Obrázok A.29: Nastavení grafického zobrazení

Dále můžeme měnit v levé části panelu *Maximum Tracing Time* což je maximální čas sledování. V praxi se to může použít tak, že si vybereme pro vykreslování lag error a set speed. V hlavním okně na obr.A.23 v horní části klikneme na ikonku *Start trace immediately* obr.A.30 a potom na sekvenci pro rozhýbání motoru. Ve výsledku dostaneme následující průběh obr.A.31.

Obrázok A.30: Tlačítko pro odstartování sledování lag erroru



Obrázok A.31: Průběh rýchlosti a lag erroru v prostředí Test

Cílem je dosáhnout pomocí vhodné kombinace parametrů pozičního a rychlostního regulátoru co nejnižší odchylku od nuly - řádově v jednotkách - lag erroru.

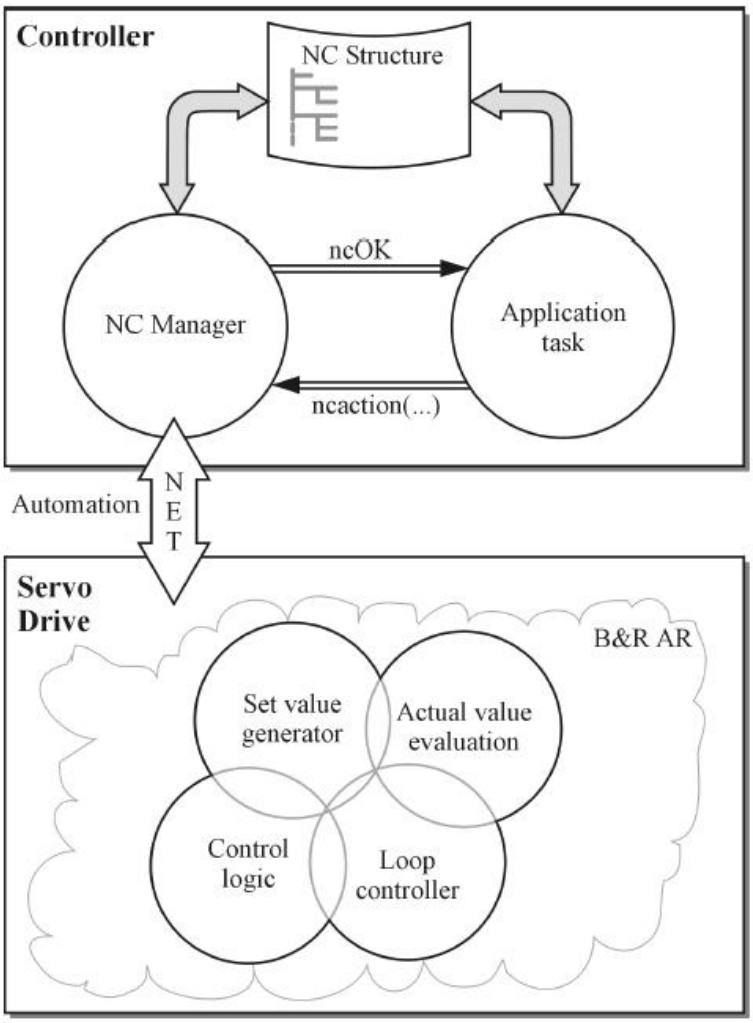
A.3 Cvičení č.3 - Programování motoru prostřednictvím Ansi C

S pojmy jako je NC Manager a nc akce jsme se již potkali v předešlém cvičení. V tomto cvičení navíc ještě přibude NC struktura a na všechno se podíváme podrobněji. Ačkoliv Ansi C je nepochybně silný nástroj a v univerzitních laboratořích nepostrádatelný, v praxi se upřednostňuje spíš B&R Automation Basic - je jednodušší a o něco přehlednější. Důvodem je spíš vyjit zákaznikovi vstříc, který platí svoje lidi pro údržbu a drobné úpravy programu.

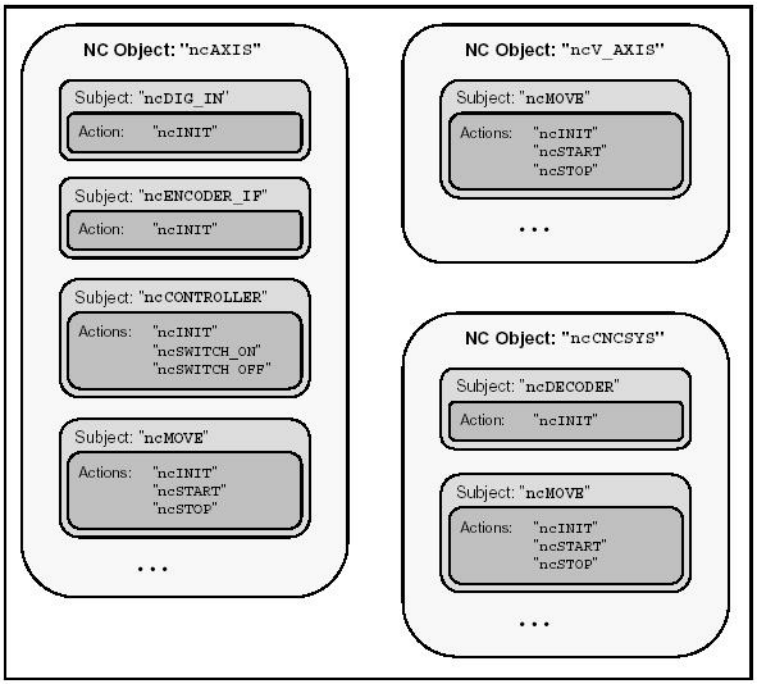
Před začátkem programování si stáhněte z ... vzorový příklad, ke kterému se bude vztahovat níže uvedený popis programování. Vložení souboru provedete importováním přes *File→Import...*.

A.3.1 NC manager, nc akce a nc struktura

Pro pochopení jak naprogramovat reálnou nebo virtuální osu, nám pomůže obrázek obr.A.32. Aplication task je program, který využívá nc strukturu jako prostředníka mezi ním a NC managerem pro zadavání různých parametrů jako je rychlost, zrychlení, nebo pro kontrolu chyb a stavu měniče. Konkrétní příkazy, čili nc akce se zadávají přímo NC manageru pomocí funkce `ncaction()`. Když NC manager obdrží příkaz, odešle zpátky potvrzení o pževzetí příkazu. Vzápětí odešle příkaz operačnímu systému měniče. Všechny statusy, které měnič odešle zpátky NC manageru, jsou NC managerem zapsané do NC struktury. A program využíva NC strukturu k ověření, jestli se náš příkaz v měniči skutečně vykonáva.

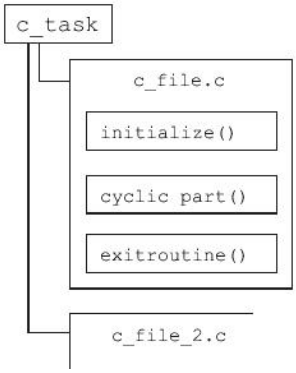


Obrázok A.32: Vztahy mezi NC managerom, aplikací, NC struktúrou a měničem



Obrázok A.33: NC objekt, NC subjekt a NC akce

Na obr.A.33 je popsána hierarchii nc akcí. NC příkaz je popsán NC objektem, NC subjektem a NC akcí. NC objekt nám určuje jestli je to reální nebo virtuální osa. NC sukbjektem říkáme, jestli chceme udělat globálni inicializaci nebo sepnout controller nebo hybat osou. NC akci říkame, jestli chceme zastavit pohyb osy nebo zapnout, vypnout contreller. Příkaz v Ansi C vypadá takhle `ncaction(NC_object,ncSUBJECT,ncACTION);` Ukázkový příklad: `action_status = ncaction(ax_obj,ncCONTROLLER,ncSWITCH_OFF);` Na obr.A.34 vidíme členění programu v Ansi C do 3 částí. Každou čast popíšu v



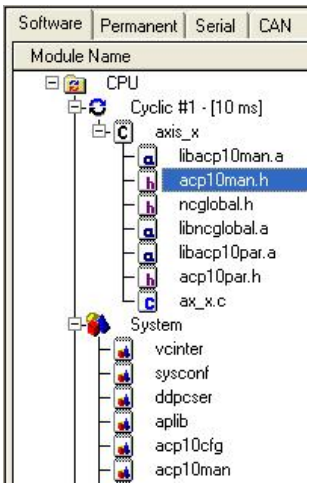
Obrázok A.34: Bloky programu v Ansi C

následujících odstavcích.

A.3.2 Program v Ansi C

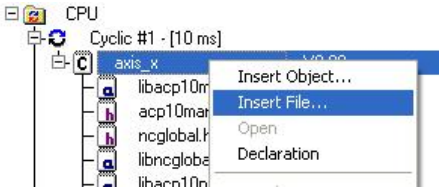
A.3.2.1 Deklarační část

Abychom si nemuseli vytvářet sami NC struktúru, můžeme si ji zpřístupnit vložením příkazu `#include äcp10man.h"` do programu, ale pak nesmíme zapomenout na vložení header souboru `acp10man.h` a knihovny `libacp10man.a` do projektu. Totéž uděláme i se soubory `ncglobal.h, libncglobal.a, acp10par.h, libacp10par.a` tak, jak je to na obr.A.35. Vložení knihovny provedeme tak, že pravým tlačítkem myši klikneme v



Obrázok A.35: Potřebné knihovny pro skompilování kódu

projektu na soubor označující náš program v Ansi C viz. obr.A.36 a vybereme *Insert File*. Otevře se nám náš projekt, ve kterém najdete adresář *Library* a v něm všechny potřebné knihovny.



Obrázok A.36: Vložení knihoven

Pak je nutné si vytvořit lokální proměnnou pro NC object např. `_LOCAL UDINT ax_obj;`. Ukazatel na NC struktúru se vytvoří následovně: `_LOCAL ACP10AXIS_typ *p_ax_dat;`. Pak je nutné mít řadu proměnných pro komunikaci s NC managerem, jako jsou `access_status`, kde se nachází výsledek přístupu k ose nebo `action_status`, kde

se nachází výsledek přijetí nebo nepřijetí nc akce NC managerem. Bitové proměnné jako `_LOCAL plcbit start_pos_mov;` se mohou později využít ve vizualizací jako spouštěcí proměnne.

A.3.2.2 Inicializační část

V inicializační části je důležitá část `access_status = ncaccess(ncACP10MAN,ösa_x",(void *)&ax_obj);`, která zabezpečuje propojení reálne fyzické osy s NC objektem v našem programu. Uprostřed této části je název fyzické osy `ösa_x"`, kterou jsme si vytvořili ještě v prvním cvičení v Deployment tabulce, jak je vidět na obr.A.37. Jména v našem programu a v deployment tabulce musí být stejné.

NC Object Name	Network Interface	Node Number	Advanced	NC Object Type	NC INIT Parameter	ACOPOS Parameter
osa_x	IF2 (CAN)	4		ncAXIS	param	table

Obrázok A.37: Jméno NC ojektu v deployment tabulce

Další důležitou části inicializace je napojení NC objektu na NC strukturu `p_ax_dat = (ACP10AXIS _typ*)ax_obj;`. Dále inicializujeme proměnné, které budeme používat pro nastavování rychlosti nebo zrychleni jako je `v_pos = 10000;` a zároveň vynulujeme bitové proměnné pro startování, například pohybu `start_pos_mov = 0;`. Nesmíme zapomenout na nastavení počátečního kroku `step = W_NET_INIT;`.

A.3.2.3 Cyklická část

Cyklická část vždy probíha v následujících krocích:

```
void _CYCLIC ax_cyclic(void)
{
   switch ( step )
      ...
      case GLOBAL_INIT:
            ...
            break;

      case W_GLOBAL_INIT:
            ...
            break;
```

```
        case INIT_W_CTRL_RDY:
              ...
              break;
        ...
}
```

Každá operace se skládá ze dvou kroků. V prvním kroku se příkaz odešle NC manageru. V podmínce se čeká dokud NC manager potvrdí přijetí příkazu. Když se tak stane, program přeskočí do druhého kroku. Ukážeme si to na operaci Globální Inicializace:

```
case GLOBAL_INIT:
   action_status = ncaction(ax_obj,ncGLOBAL,ncINIT);
   if ( action_status == ncOK )
   {
      step = W_GLOBAL_INIT;
   }
   break;
```

Po přeskočení do druhého kroku globální inicializace `W_GLOBAL_INIT`, program čeká, dokud se změny neprojeví v měniči. Týto změny v měniči NC manager zapíše do NC struktúry a my v podmínce následně indikujeme změny v NC struktúře. Po splnění podmínky program přeskočí do dalšího kroku další operace.

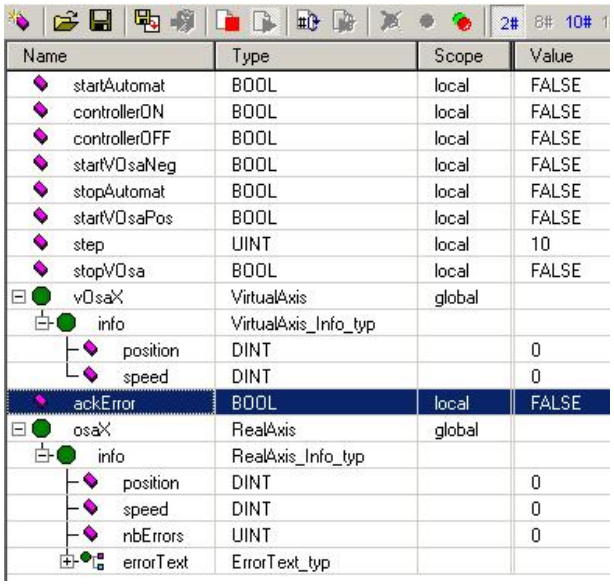
```
case W_GLOBAL_INIT:
   if ( p_ax_dat->global.init == ncTRUE )
   {
      step = SIM_MODE;
   }
   break;
```

Pro přípravení osy k pohybu musíme vykonat operace v definovaném sledu: inicializace sítě, globální inicializace proměnných, zapnutí controlleru a provedení homingu, tak jak to známe již z druhého cvičení.

```
case COMMAND:
```

```
if ( start_pos_mov == 1 )
{
   step = POS_MOVEMENT;
}
else if ( start_neg_mov == 1 )
{
   step = NEG_MOVEMENT;
}
...
break;
```

Pak se dostaneme do kroku `COMMAND`, ve kterém jsou v podmínkách bitové proměnné, které mohou být ovládané z vizualizace nebo ze sledovacího okna *"Watch"* obr.A.38.

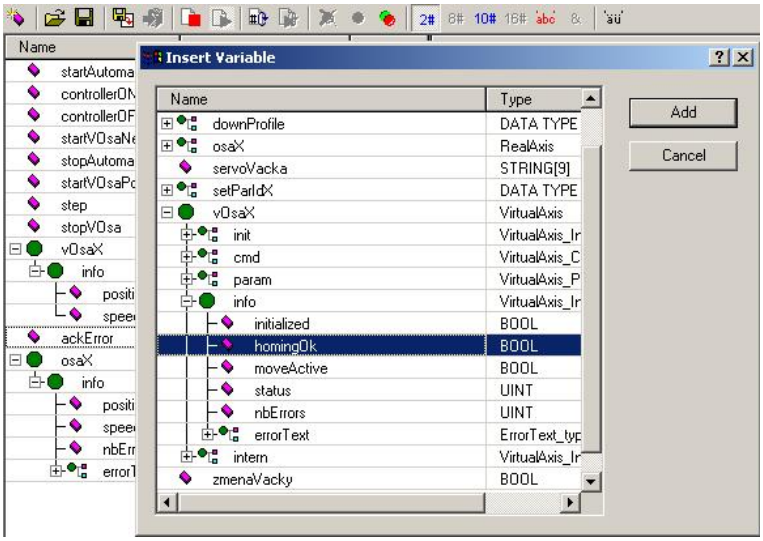


Name	Type	Scope	Value
startAutomat	BOOL	local	FALSE
controllerON	BOOL	local	FALSE
controllerOFF	BOOL	local	FALSE
startVOsaNeg	BOOL	local	FALSE
stopAutomat	BOOL	local	FALSE
startVOsaPos	BOOL	local	FALSE
step	UINT	local	10
stopVOsa	BOOL	local	FALSE
vOsaX	VirtualAxis	global	
info	VirtualAxis_Info_typ		
position	DINT		0
speed	DINT		0
ackError	BOOL	local	FALSE
osaX	RealAxis	global	
info	RealAxis_Info_typ		
position	DINT		0
speed	DINT		0
nbErrors	UINT		0
errorText	ErrorText_typ		

Obrázok A.38: Okno "Watch"

Sledovací okno *Watch* otevřeme v horní liště *Open→Watch*. Proměnné vkládáme v okně Ïnsert"ze stromu struktúry obr.A.39.

Bitové proměnné v podmínce odpovídají například posunu osy v pozitívnim směru nebo absolutní pohyb. Když zadáme z vizualizace příkaz posun v negativním směru, nasetujeme tím proměnnou `start_neg_mov`. Je-li podmínka pro negativní posun osy splněná, program přeskočí z kroku `COMMAND` do kroku `NEG_MOVEMENT`. Po odstartování osy se program vrátí opět do kroku `COMMAND`.



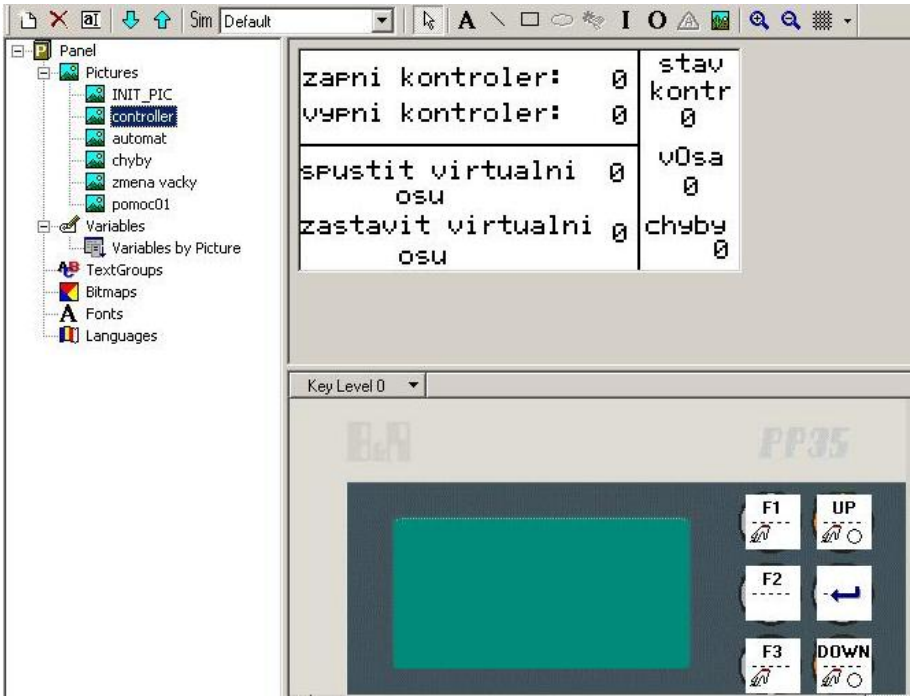
Obrázok A.39: Vkládání proměnných do sledovacího okna

```
case NEG_MOVEMENT:
   p_ax_dat->move.basis.parameter.v_neg = v_neg;
   action_status = ncaction(ax_obj,ncNEG_MOVE,ncSTART);
   if ( action_status == ncOK )
   {
      step = COMMAND;
      start_neg_mov = 0;
   }
   break;
```

A.4 Cvičení č.4 - Vizualizace

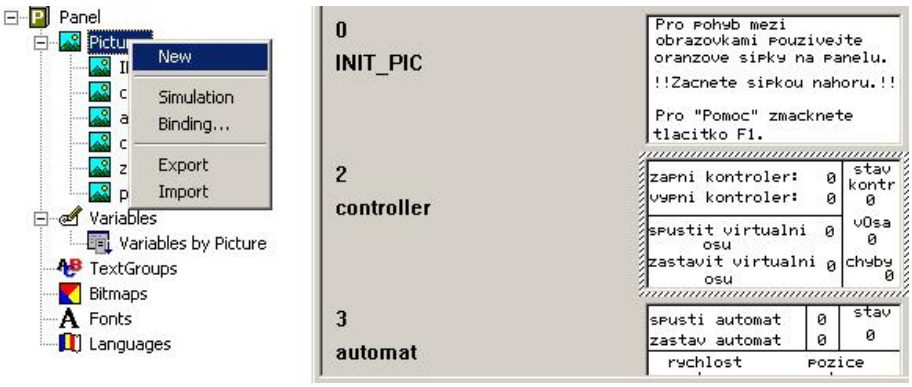
Vizualizace je důležitá pro obsluhu řídící jednotky. V počátcích projektu se často stává prezentačním artiklem a polem k diskuzi a detailnější konkretizaci řídícího systému mezi programátorem a zákazníkem. Vizualizační panely mohou dosahovat i metry čtverečné dotykové funkční plochy. Ne vždy jsou firmy nakloněné tak velkým výdajum, a proto v případech, kde vizualizace není těžítkem, se trh uchyluje i k daleko levnějším řešením. Relativně levné řešení je náš Power Panel se zabudovaným displejem, kde displej obsahuje centimetry čtverečné úžiteční plochy. Tady je pro přehlednost důležité intuitivní uspořádání obrazovek a zadávacích proměnných.

A.4.1 Vizualizace v PP 35



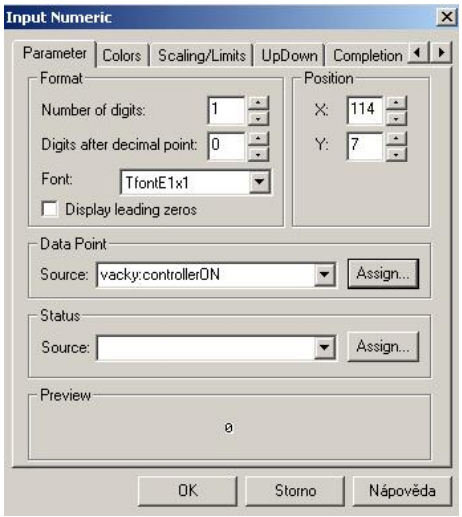
Obrázok A.40: Celkový přehled vizualizační obrazovky

Po dvojkliku na vizualizační list v projektovém stromě se nám zobrazí okno jako na obr.A.49. V levé části vidíme strom celé vizualizace s obrazovkami. Nahoře v pravé části se zobrazuje aktuální obrazovka. Ve spodní částí jsou zobrazené funkční tlačítka panelu. Vložení nové obrazovky provedeme kliknutím pravého tlačítka myši na *Pictures→New* obrA.41.



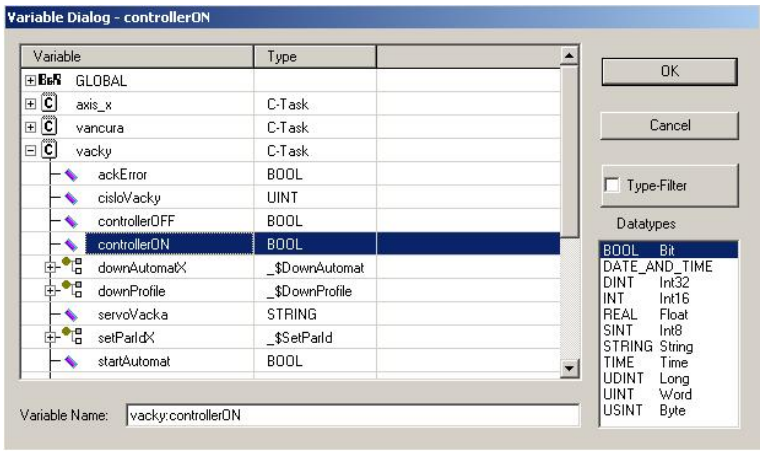
Obrázok A.41: Vytvoření nové obrazovky

V horní pracovní liště se nachází základní užitečné nástroje jako **A** vkládání textu, **I** vkládání vstupních nebo výstupních **O** proměnných. Po vložení vstupní proměnné a po dvojitém kliknutí na ni, se nám otevře panel jejího nastavení obr.A.42.



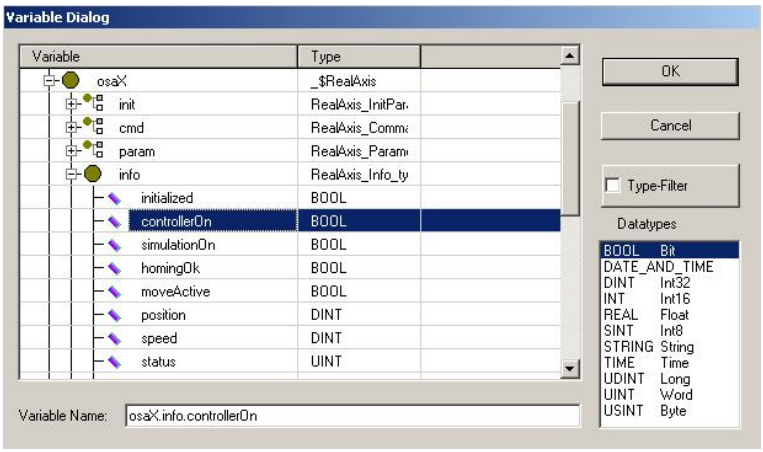
Obrázok A.42: Parametry vstupní proměnné

V tomto panelu musíme nastavit propojení na konkrétní proměnnou v programu. Klikneme do pole "Data Point" na *Assign....* Pak se nám otevře strom všech vstupních proměnných obr.A.43, kde si vybereme tu, co potřebujeme.



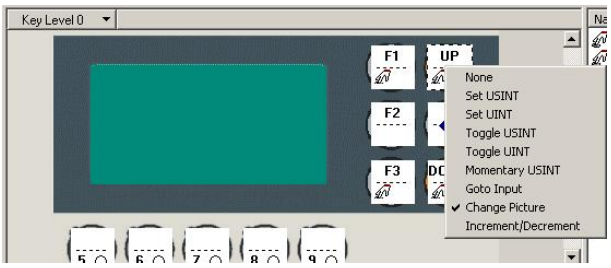
Obrázok A.43: Strom vstupních proměnných

Pro výstupní proměnné jsou užitečné info proměnné v NC struktuře obr.A.44. Například *osaX* → *info* → *ControllerOn* nám bude na obrazovce jedničkou signalizovat zapnutý stav kontroleru a nulou opět jeho vypnutý stav.

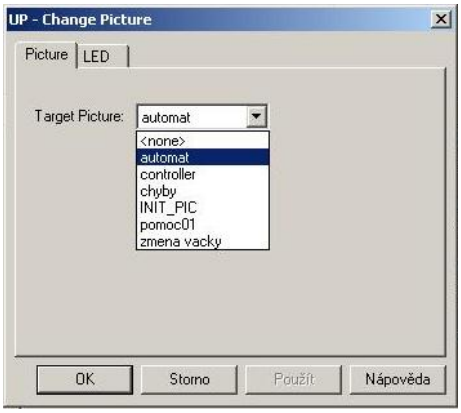


Obrázok A.44: Strom výstupních proměnných

Funkční tlačítka ve spodní částí obrazovky můžeme použít například pro změnu obrazovek. Po kliknutí pravým tlačítkem myši na funkční tlačítko, se nám zobrazí nabídka jeho využití jak to vidíme na obr.A.45. Vybereme si *Change Picture* a následným dvojklikem se nám zobrazí nabídka obrazovek pro změnu obr.A.46.



Obrázok A.45: Funkční tlačítko jako změna obrazovky



Obrázok A.46: Nabídka obrazovek funkčního tlačítka s funkcí "Změna obrazovky"

A.5 Cvičení č.5 - Vačkový automat

Vačka má obvykle vejčitý tvar obrA.47. Tvarem vačky lze mechanicky „naprogramovat“ dobu a výšku zdvihu v závislosti na jejím natočení. Nejrozšírenejším mechanickým váčkovým automatem je skupina vaček v aute, která otevíra sací ventily ve čtyřdobém spalovacím motoru. Ten samý princip se využívá u váčkových strojů. Například vačkový automat na výrobu pružin má v oblasti zhybu drátu několik vaček se zdvihátkem, na které je napojený nástroj pro ohýbání drátu, nebo nůž pro odseknutí a ukončení pružiny. Všechny vačky jsou synchronizované pomocí váčkové hřídele a v rámci otáčky udělají jeden cyklus - například pružinu.

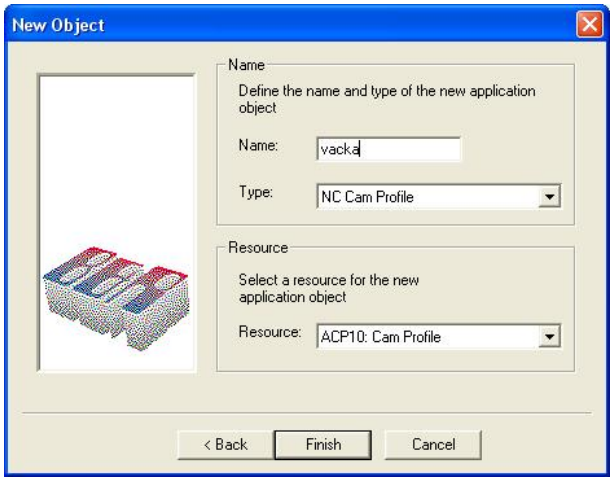
Obrázok A.47: Vačka

Nevýhodou mechanické vačky je její opotřebovatelnost a v případě přeprogramovatelnosti se musí nahradit jinou - čili mnoho mechanických součástek. Všechny nedostatky mechanických vaček odstraňuje elektronická vačka. Takto velký stroj na výrobu pružin skládajíci se z hlavního pohonu, váčkové hřídele a spoustu mechanických převodu se může zjednodušit na velikost našeho modelu, přičemž jeden motor by odpovídal jední vačce. Váčkovu hřídel nahrazuje virtuální osa.

Pro usnadnění práce si stáhněte z ... soubor `camprofile.c` a importujte si ho do projektu.

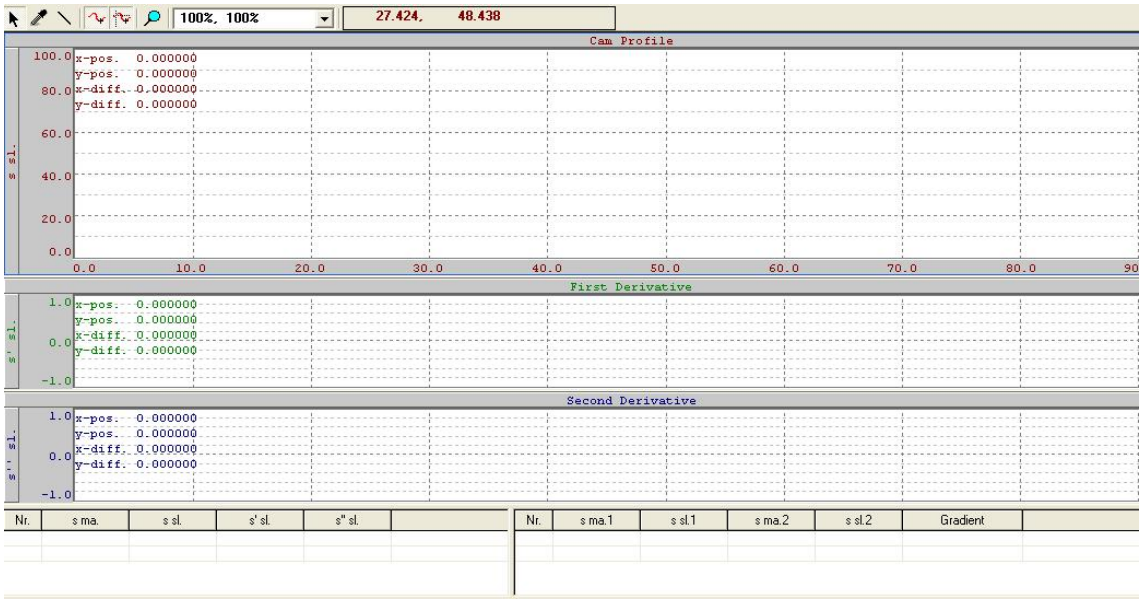
A.5.1 Vačky

Novou vačku vytvoříme tak, že klikneme pravým tlačítkem na strom našeho projektu a vybereme si *Insert Object*. Pak navolíme cam profile tak jako na obr.A.48.



Obrázok A.48: Vytvoření vačky

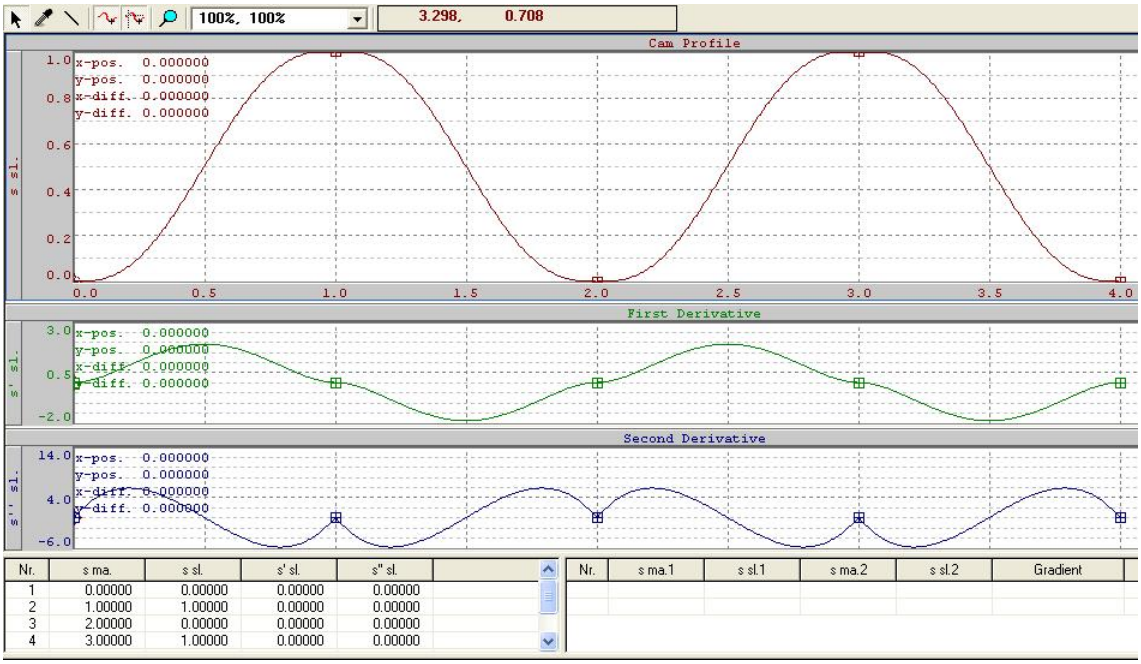
Po vytvoření vačky se nám zobrazí vačkový editor obr.A.49.



Obrázok A.49: Vačkový editor

V jeho horní části se nachází 3 grafy pro dráhu, rychlost a zrychlení vačkového profilu. Ve spodní částí vlevo jsou zobrazené přechodové fixní body vačkové křivky. Jeho spodní

pravá část je pro zadefinování synchronních sekcí ve vačce. Vytvářet vačkové charakteristiky můžeme přímo zápisem hodnot do spodního levého nebo pravého pole. Můžeme také použit myš, klikneme pravým tlačítkem na plochu grafu si vybereme *Insert Fixpoint* nebo *Insert Synchronous Section*. Výsledná vačka může být například jako na obr.A.50.

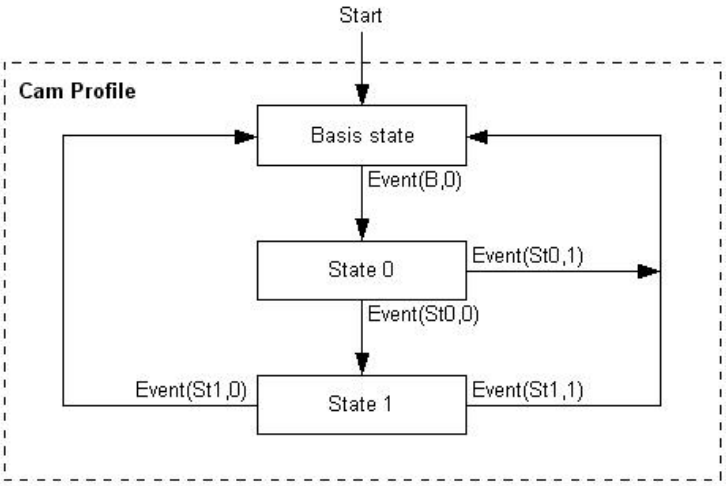


Obrázok A.50: Použitelná vačka

A.5.2 Vačkový automat

A.5.2.1 Stavový diagram

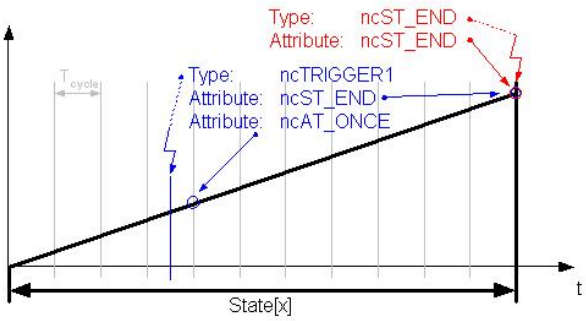
Automat se skláda ze základního stavu a dalších stavů 0, 1, ..., 14. Každý stav má dvě událostí obr.A.51. Událost s nižším indexem má větší prioritu.



Obrázok A.51: Stavový automat

Každá událost má typ (TYPE) a vlastnost (ATTRIBUTE). Vlastnosti události mohou být dvě:

- `ncST_END` - přechod do dalšího stavu se nevykoná dřív než se skončí aktuální stav.
- `ncAT_ONCE` - přechod do dalšího stavu se vykoná okamžitě.



Obrázok A.52: Chování události s vlastnosti ncST_END a ncAT_ONCE

Obrázek A.52 názorně ukazuje, že když nastane událost typu `ncTRIGGER1` s vlastnosti `ncAT_ONCE`, tak automat přejde do dalšího stavu hned v dalším cyklu. Jestli vlastnost

bude `ncST_END`, tak automat přejde do dalšího stavu až se skončí aktuální stav. Typy události jsou:

- `ncOFF` - tahle událost se vynechá.
- `ncST_END` - konec stavu byl dosažen.
- `ncTRIGGER1+ncP_EDGE` - náběžna hrana vstupu prvního spouštěče.
- `ncTRIGGER1+ncN_EDGE` - sestupní hrana vstupu prvního spouštěče.

Další typy události jsou popsané v helpu Automation Studia.

A.5.2.2 Program vačkového automatu v Ansi C

Pro vačkový automat existuje knihovna *PLCopen MC library*, která zaobaluje NC_akce do sebe a odlehčuje programátora od nutnosti komunikace s NC managerom a ověřování, jestli příkaz přijal nebo ne. Tím se kód stáva přehlednějším. V našem případe je knihovna *acp10_mc* příliš náročná na operační pamět PLC, proto brněnský ing. Vančura sestrojil vlastní "českou"knihovnu *acp10cz*, která je kompaktnejší a do PP35 se vejde. Pro využití této knihovny je potřebné vložit do projektu header file *acp10cz.h* a knihovnu *libacp10cz.a*. Taky v deklaraci nesmí chybět `#include <acp10cz.h>`.

A.5.2.3 Deklarace proměnných

Pro fungování vačkového automatu potřebujeme minimálně 2 osy. Jednu reálnou (servomotor) a druhou virtuální. Virtuální osa bude master a realní osa bude slave. Proto si v deklarační části vytvoříme dvě globální proměnné:

```
_GLOBAL RealAxis_typ osaX;
_GLOBAL VirtualAxis_typ vOsaX;
```

A.5.2.4 Inicializace proměnných

V inicializační částí příkazem `memset` vynulujeme všechny zadeklarované struktury.

```
memset(&osaX,0,sizeof(RealAxis_typ));
```

Pro reálnou a virtuální osu musíme zapsat do struktury jejich jména z deployment tabulky.

```
strcpy(osaX.init.name,"r_osa_x");
```

Název reální osy v deployment tabulce je v tomto příkladě uveden v uvozovkách "r_osa_x".

Když si v hlavním projektu přidáme tabulky chyb *ACP10:Error Text Table* přes prave tlačítko myši a *Insert Object*, tak název chybového modulu zapíšeme do struktury např:

```
strcpy(osaX.init.errorTextModule,"chyby");
```

Pak si můžeme ze struktury vybrat nejenom číslo, ale i konkrétní název chyby. Vynulujeme i pomocné bitove proměnné take určené pro vizualizaci.

```
startAutomat = 0;
controllerOFF = 0;
controllerON = 0;
```

Do proměnné `servoVacka`, které jsme deklarovaly jako string, zkopírujeme jméno vačky `"vacka01"`, která se nachází v hlavním stromě.

A.5.2.5 Cyklická část

Cyklická část probíha obdobně jako ve cvičení 3. Příkazy k vykonání (například houmingu) realizujeme zapsaním 1 do command větve (cmd) struktury.

```
vOsaX.cmd.homing = 1;
```

Struktura obsahuje taky info větev, ze které se dovíme, jestli již houming proběhl, pak můžeme přejít do dalšího kroku.

```
if(vOsaX.info.homingOk == 1) step++;
```

Abychom mohli spustit automat nejdřív musíme downloadovat vačkový profil.

```
downProfile.enable = 1;
downProfile.pAxis = (UDINT)&osaX;
strcpy(downProfile.moduleName,servoVacka);
downProfile.index = 1;
DownProfile(&downProfile);
if (downProfile.status == 0)
{
   step++;
}
```

Vačka musí být svázana s reální osou. V našem případě jsme zadeklarovali osu s nazvem `osaX`. Jméno vačky zkopírujeme do `downProfile.moduleName`. Proměnná `downProfile.index` slouží k zaindexovaní více vaček. V automatu pak vybírame jednotlive profily vaček přes indexy.

V dalším kroku zadefinujeme vačkový automat a jeho stavy.

```
downAutomatX.enable = 1;
```

```
downAutomatX.pAxis = (UDINT)&osaX;
memset(&downAutomatX.params, 0, sizeof(downAutomatX.params));
downAutomatX.params.MA_AXIS = ACP10PAR_S_SET_VAX1;
downAutomatX.params.MA_S_START = 0;
downAutomatX.params.MA_IVSTART = 1;
downAutomatX.params.MA_V_MAX = 1000;
#define STATE 0
downAutomatX.params.state[STATE].event[0].EVENT_TYPE = ncST_END;
downAutomatX.params.state[STATE].event[0].EVENT_ATTR = ncST_END;
downAutomatX.params.state[STATE].event[0].EVENT_ST_INDEX = 1;
#undef STATE
#define STATE 1
downAutomatX.params.state[STATE].ST_DATA_INDEX = 1;
downAutomatX.params.state[STATE].COMP_MODE = ncOFF;
downAutomatX.params.state[STATE].MA_FACTOR = 100;
downAutomatX.params.state[STATE].SL_FACTOR = 100;
downAutomatX.params.state[STATE].COMP_MA_S = 0;
downAutomatX.params.state[STATE].COMP_SL_S = 0;
downAutomatX.params.state[STATE].event[0].EVENT_TYPE = ncST_END;
downAutomatX.params.state[STATE].event[0].EVENT_ATTR = ncST_END;
downAutomatX.params.state[STATE].event[0].EVENT_ST_INDEX = 1;
#undef STATE
```

Vačkový automat taky vztáhneme k reálné ose a vynulujeme předcházející parametry `memset (&downAutomatX.params, 0, sizeof(downAutomatX.params));`.
Další parametery:

- `MA_S_START` - startovaci pozice master osy.
- `MA_IVSTART` - startovací interval master osy.
- `MA_V_MAX` - maximální rýchlost master osy.
- `EVENT_TYPE, EVENT_ATTR` - podrobně vysvětleno výše.
- `EVENT_ST_INDEX` - index dalšího stavu.
- `ST_DATA_INDEX` - index vačkového profilu, který jsme zadefinovali funkci `DownProfile`.

- `COMP_MODE` - typ kompenzačního módu.
- `MA_FACTOR` - multiplikační činitel master osy.
- `SL_FACTOR` - multiplikační činitel slave osy.
- `COMP_MA_S` - kompenzační vzdálenost master osy.
- `COMP_SL_S` - kompenzační vzdálenost slave osy.

Po zadefinovaní celého automatu se v dalším kroku zavolá zavolá funkce `DownAutomat(&downAuto`
Když `downAutomatX.status == 0`, tak je automat připraven k nastartování. Pro spuštění automatu musí být připravená reálna osa, tj. `osaX.info.controllerOn == 1`, jinak měnič bude signalizovat chybu. V kroku 10 našeho programu používáme bitové proměnné pro ovládání měniče a vačkového automatu z vizualizace. Z tohoto kroku "Command" skáčeme na místa programu, kde příkazy vykonávame. Pro nastartování automatu se musí hýbat master osa v kladném směru, reálna osa musí být plně připravená a pak nastavíme pro automat *parId→ncStart*.

```
setParIdX.enable = 1;
setParIdX.pAxis = (UDINT) & osaX;
setParIdX.nbItems = 1;
setParIdX.item[0].par_id = ACP10PAR_CMD_AUT_START;/*jaky parametr se ma zapsat*/
*((UINT *) setParIdX.item[0].data_byte) = ncSTART; /* nakopirovani dat */
SetParId (&setParIdX); /* vlastni volani funkce */
if (setParIdX.status == 0)
step=10;
```

Funkce *SetParIdX* slouží k zaslání 32 parametrů měniči. Důležitý je typ parametru `ACP10PAR_CMD_AUT_START` a co se ma poslat `ncSTART` - v případě, že chceme automat nastartovat, a `ncSTOP`, když chceme automat zastavit.