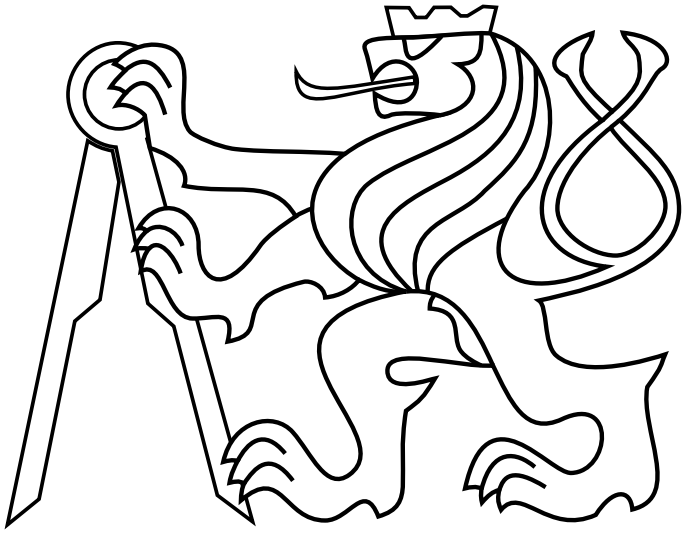
České vysoké učení technické v Praze

Fakulta elektrotechnická

Katedra řídicí techniky



BAKALÁŘSKÁ PRÁCE

Návrh a řízení stroje na badminton
Design and control of a badminton launcher

Autor : Ondřej Maslikiewicz

Vedoucí práce : Ing. Pavel Burget, Ph.D.

2014

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, dne 22.5.2014 ..... Mosliková

Podpis

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Ondřej Maslikiewicz**

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

Název tématu: **Návrh a řízení stroje na badminton**

Pokyny pro vypracování:

1. Navrhněte princip stroje na vystřelování badmintonových míčků.
2. Seznamte se s řídicím systémem B+R a servozesilovači ACOPOS. Navrhněte a realizujte synchronní řízení kotoučů, které vystřelují badmintonové míčky. Prozkoumejte možnosti změny letu míčku podle způsobu jeho uchopení rotujícími kotouči.
3. Doplňte stroj o automatický podavač míčků a zapojte jej do řídicího systému. Navrhněte mechanismus pro změnu směru letu míčku v horizontální rovině.
4. Rozšiřte řídicí systém o vizualizaci primárně na mobilním telefonu s využitím běžně dostupného prostředí pro vizualizaci řídicích systémů. Vytvořte scénáře vystřelování míčků a připravte pro ně vhodné ovládací prvky, aby bylo možné snadno nastavovat jejich parametry a vybírat jeden jako aktivní.

Seznam odborné literatury:

1. Jaroš, P. Rozšíření modelu žonglér a řízení CNC stroje. Diplomová práce, K13135 ČVUT v Praze FEL, 2011.
2. Kohout, T. Model žonglér pro vzdálenou výuku a řízení CNC stroje. Diplomová práce, K13135 ČVUT v Praze FEL, 2011.
3. Firemní literatura společnosti B&R Automation.

Vedoucí: Ing. Pavel Burget, Ph.D.

Platnost zadání: do konce zimního semestru 2014/2015

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
L.S.

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 27. 9. 2013

Poděkování

Zde bych rád poděkoval vedoucímu své práce Ing. Pavlu Burgetovi, Ph.D. za cenné rady, vedení a pomoc při realizaci badmintonového stroje. Dále bych rád poděkoval Ing. Milanovi Bartošovi, CSc. za konzultace k mechanice stroje, za jeho konstrukci a výrobu jednotlivých dílů prototypu stroje s podavačem míčků. Poděkování také patří Ing. Josefu Kultovi za výkresy kotoučů a výpočet momentu setrvačnosti. V neposlední řadě bych rád poděkoval Ing. Jiřímu Maslikiewiczovi za pomoc s produkty B&R a také za zapůjčení powerlinkové karty V1 a PLC PP65.

Anotace

Tato bakalářská práce se zabývá návrhem stroje pro odpalování badmintonových míčků. Řešení sestává z návrhu mechaniky stroje, pohonů a řídicího systému. Mechanika se skládá z odpalovacích kotoučů a podávacího mechanismu míčků. Pro řešení pohonu jsou použity servoměniče ACOPOS a motory firmy B&R, pro řízení stroje je použito PLC téže firmy. K programování servoměničů a PLC je využito Automation Studio.

Klíčová slova

Badminton; odpalování; mechanika; pohon; řídicí systém; servozesilovač; PLC

Abstract

This thesis deals with the design of a machine for striking badminton shuttlecocks. The solution involves a design of the machinery mechanics, the drives and the control system. The drive consists of launch discs and a shuttlecock feed mechanism. As a solution for the drive, ACOPOS servo drives and the motors of the B&R company are used. A PLC of the same company is used for machine control. Automation Studio is used for programming servo drives and the PLC.

Key words

Badminton; badminton launch; mechanism; drive; control system; servo drive; PLC

Obsah

Seznam obrázků

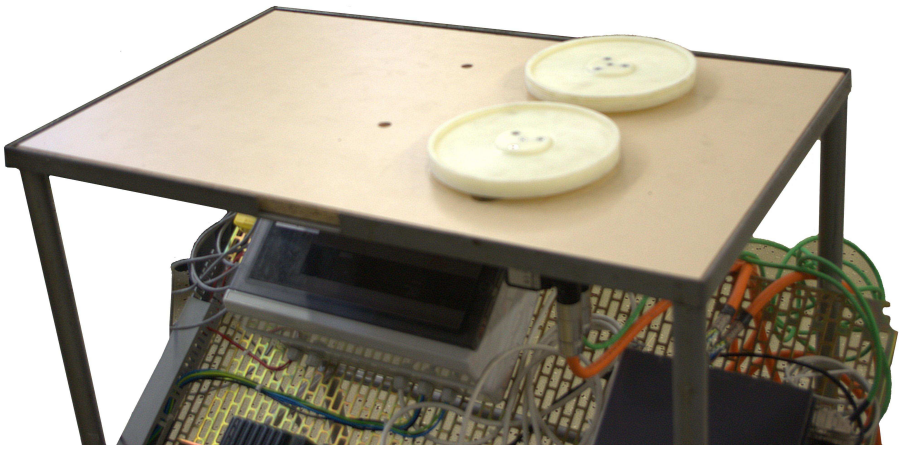
Seznam tabulek

Seznam zkratek

- PLC - Programmable Logic Controller (Programovatelný logický automat)
- B&R - Bernecker + Rainer Industrie Elektronik GmbH
- HMI - Human-machine interface
- RAM - random-access memory
- EPSG - Ethernet POWERLINK Standardization Group
- CiA - CAN in Automation
- MN - Managing Node
- SoC - Start of Cycle

1 Úvod

Badminton je v dnešní době velice populární sport. Přibývá jak amatérských hráčů, tak profesionálů nejvyšší světové třídy. Pro úspěchy v jakémkoliv sportu je zapotřebí tréninku. Takový trénink není náročný pouze pro samotného hráče, ale i pro trenéra. Ten musí neustále dokola opakovat jednu a tu samou činnost např. nahazování míčů na základní čáru. Usnadnění a zpřesnění takového tréninku má za cíl badmintonový stroj. Ten je určen k nahazování badmintonových míčů, a tím umožní sportovci hrát (trénovat), aniž by měl reálného soupeře (trenéra).



Obrázek 1: Badmintonový stroj

Na takovýto stroj jsou pak kladeny požadavky především na přesnost, rychlost a rozmanitost nadhozů. Stroj musí být dostatečně rychlý. Rychlost závisí na tom, pro koho se nadhazuje (pro děti, dospělé, reprezentanty). Přesnost je důležitá obzvláště při pokročilejších programech. Samozřejmě i při pouhém nadhazování je třeba docílit toho, aby míček dopadal na stejné místo. To s ohledem na nepravidelnost míčků (dva přesně identické nenajdeme) nebude vždy stejné. Rozmanitost nadhozů je důležitá pro praktické využití badmintonového stroje. Házet míček na jedno místo s danou frekvencí je vcelku nezajímavé a nepoužitelné.

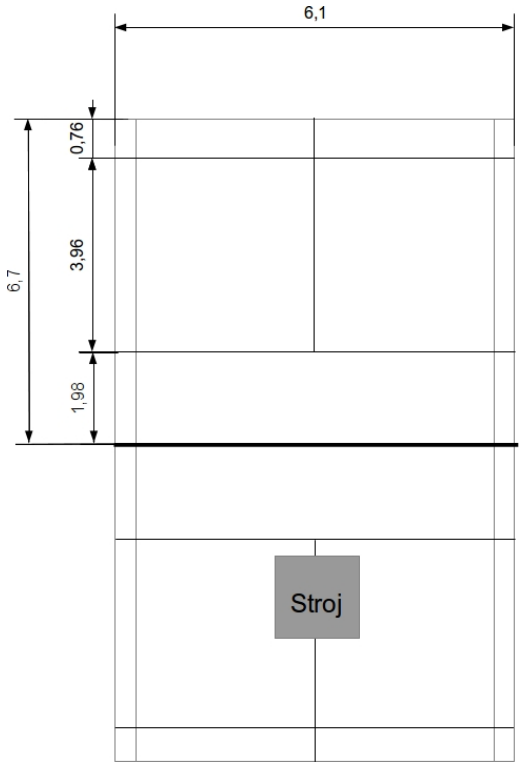
2 Badmintonový stroj

2.1 Požadavky trenéra na stroj

V ideálním případě by výsledný stroj měl splňovat požadavky, které pocházejí od badmintonového trenéra. S Mgr. Michalem Turoňem jsem konzultoval, jaké jsou tréninkové metody pro trénink na nejvyšší úrovni. Z této konzultace vzešly požadavky na stroj.

2.1.1 Hra do kurtu

Pro badmintonový trénink jsou, při hře do kurtu, možné tyto situace, které by bylo dobré strojem pokrýt.



Obrázek 2: Badmintonový kurt (výška sítě *1,55* m)

Zadní část

Stroj v tomto módu střílí míče do zadní části kurtu. Míče by měly létat s periodou 1,5 - 2 s a k tomu střídat strany kurtu. Měření periody bylo provedeno při konzultaci programů s trenérem. Pro lepší představu situace tréninku byly představeny a při té příležitosti změřeny některé parametry.

Střední část

Do střední části kurtu by bylo dobré míče pálit jak na střídačku, co se týče stran, tak také způsobu dopadu. Buď míč na místo dopadu letí přímo, nebo velkým obloukem a na místo určení padá. Perioda míčů je v tomto případě 1 - 1,5 s

Střední část 2

Druhá možnost, jak střílet do střední části kurtu, je nastřelování z větší výšky a zápornou elevací stroje. Tímto se dá simulovat smeč a tím i trénovat její vybrání.

2.1.2 Hra na síti

V tomto případě se míče střílí těsně nad síť, a to po celé délce sítě. Perioda takového nahazování je zhruba 0,25 - 0,5 s. Takováto rychlost je, vzhledem k požadavků změny směru letu, takřka nedosažitelná. Nejen že se musí pokrýt celý efektivní prostor nad sítí tzn. zhruba 5 m, ale současný způsob podávání míče nejspíše neumožní takovouto frekvenci.

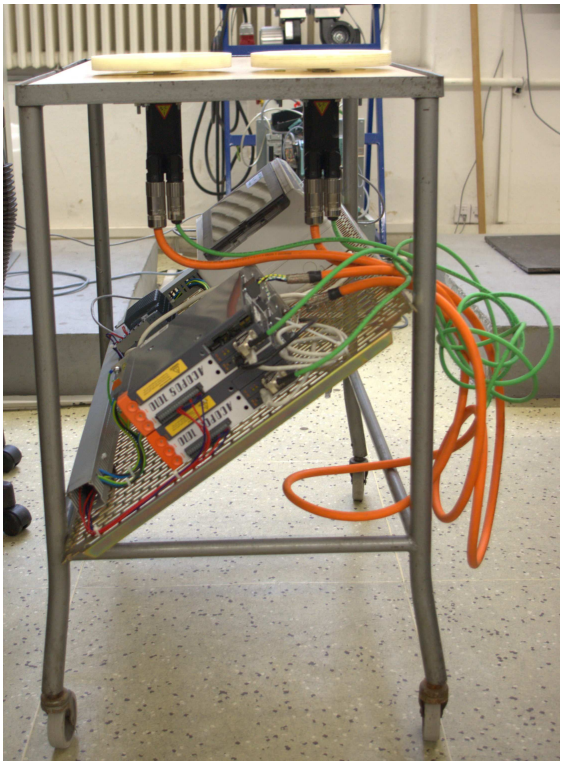
2.1.3 Nahazování na raketu

Stroj nepálí míče nikam daleko, ale těsně před sebe. Hráč odehrává míče, které ze stroje vypadávají s velkou frekvencí, zhruba 1 míč za 0,2 - 0,3 s. Tento mód není

náročný na samotný mechanizmus odpalování, ale na podávání míčů. Takovéto frekvence není v současné době podavač schopen poskytnout.

2.2 Základní princip

Stroj pracuje na vcelku jednoduchém principu. Badmintonovému míčku je udělena rychlost za pomocí proti sobě se točících kotoučů. Základem stroje jsou tedy dva kotouče, které rotují proti sobě. Mezi kotouči je mezera o cca 2 - 4 mm menší než je průměr odpalovací části míčku (hlavička). Mezera mezi kotouči musí být menší z



Obrázek 3: Badmintonový stroj

důvodu dobrého přenosu energie z kotoučů na míček. V případě stejně velké mezery jako je průměr hlavičky míčku, se jedná o bodový styk. V tomto případě není míčku udělena téměř žádná energie a tím pádem míček nikam nedoletí. Při zmenšení me-

zery se míček zmáčkne a úsek, po kterém se míčku předává energie, se prodlouží. Ideálně míček dosáhne stejné rychlosti jako je obvodová rychlost kotoučů. Podle koeficientu tření mezi kotouči a hlavičkou míče je jeho rychlost vždy menší než obvodová rychlost kotoučů. Dochází k prokluzu mezi kotoučem a povrchem hlavičky míče.

3 Laboratorní stroj

První verze stroje byla sestavena ze stolku s dřevěnou deskou a z motorů připevněných ve vyfrézovaných drážkách (obrázek 1/strana 1).

Na tomto stroji bylo možné odzkoušet princip výstřelu. Tento jednoduchý stroj ověřil teoretické předpoklady, a to především, že dva kotouče rotující proti sobě vystřelí míček. Zároveň se na tomto stroji dala vyzkoušet velikost mezery mezi kotouči a následné chování míčku při odpalu. Jedna z nejdůležitějších věcí, která byla na tomto stroji odzkoušena, byla možnost změny směru letu míčku. Teoretický předpoklad, že při změně rychlosti pouze jednoho kotouče, změně převodového stupně ve virtuální převodovce, dojde ke změně směru letu, se ukázal mylný. Po řadě experimentů se mi nepodařilo tímto způsobem míček donutit letět jiným směrem než rovně. K maximální změně směru došlo v případě úplného zastavení jednoho z kotoučů. Míček pak sice zatáčí, ale doletí metr před stroj a metr do strany. Toto je zcela nevyhovující a tento systéme řízení směru letu míčku, jsem tedy opustil.

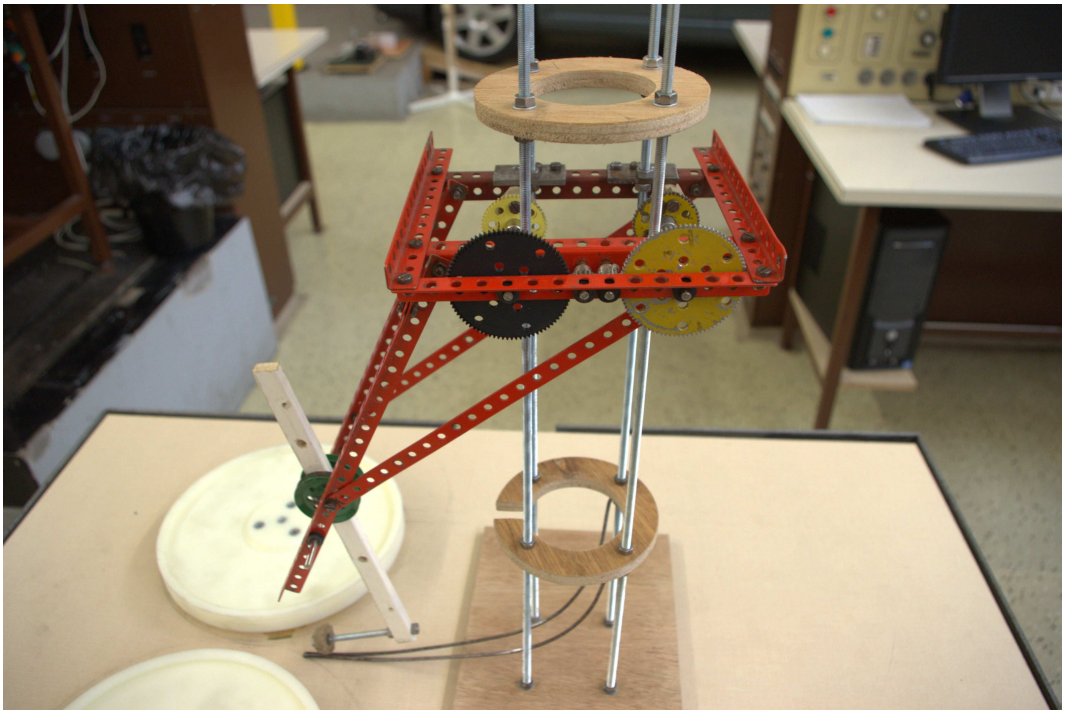
Další možností, jak změnit směr výstřelu, je posun kotoučů v ose odpalu dopředu a dozadu. Při posunutí jen jednoho z kotoučů a zachování mezery mezi nimi se směr odpalu změní. Míčky jsou odpalovány na stranu, na které je kotouč posunut víc dozadu. Toto je v podstatě velmi jednoduché, protože kotouče stále pálí míčky přímo před sebe, ale rovnoběžné tečny, které tento směr určují, se vzájemným posunutím natočí. Stejného efektu lze dosáhnout i při mírném natočení desky s motory. V tomto

případě se natočí pouze motory a celý stroj zůstane na místě. Není zapotřebí tak velkého a silného motoru ke změně směru letu míčku. Výhodou oproti posouvání kotoučů proti sobě je jednodušší konstrukce mechanizmu natáčení. Při posunu kotoučů je zapotřebí dosáhnout velké přesnosti a zároveň s posunem dopředu a dozadu se musí kotouče pohybovat k sobě. Při pouhém posunutí dopředu, dozadu dojde ke změně velikosti mezery, a tím i účinnosti stroje. Pro změnu úhlu odpalu míče je v prvním prototypu použito ruční natáčení stroje. První prototyp má za úkol odzkoušet hlavně základní principy podávání míčků a dynamiky výstřelu, tedy u něj není nutné směr měnit automaticky.

3.1 Podavač míčků

K podávání míčku bylo zapotřebí vymyslet podavač. Ten je sestaven z několika částí. Míčky jsou umístěny v tubusu podobně, jako je tomu v prodejních tubusech. Jediným rozdílem je, že tubusem v podavači míčky volně propadají. Zastaví se až na konci, kde je zúžení dostatečně velké na zadržení sloupce míčků nad ním. Z této části se míček, pomocí dvou excentrických podavačů umístěných proti sobě, vytrhne a přemístí do nižšího patra. Tam je zachycen druhým zúžením. Excentrické podavače uchopí za hlavičku vždy jen jeden míček. Ze spodního patra je míček mezi kotouče dopraven ramenem, které míček nabere a po kruhové dráze ho dopraví mezi kotouče. Tím je míčku udělena základní rychlost a kotouče míček neuvádí v pohyb z klidu. Rameno a excentrické podavače jsou poháněny jedním pohonem.

Laboratorní model podavače byl sestaven z materiálů dostupných doma. Pro konstrukci byly použity závitové tyče, překližková mezikruží a díly Merkuru (obrázek 4/strana 7).



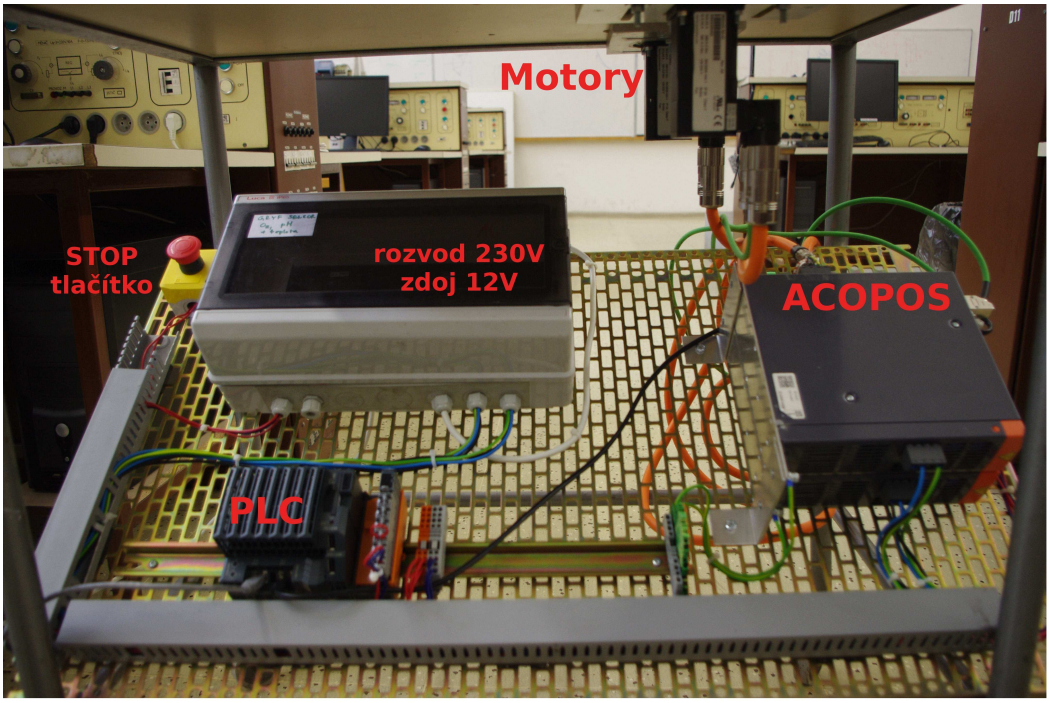
Obrázek 4: Podavač míčku

3.2 Elektroinstalace

Po elektrické stránce je stroj postaven na produktech firmy B&R. Jedná se o řídicí prvek PLC *X20 CP1485* a servo zesilovače ACOPOS *8V1010*. U prvního stroje bez podavače míčků byla elektroinstalace provedena na montážní matraci. (obrázek 5/strana 8)

4 První prototyp

Po konzultacích s Ing. Milanem Bartošem, CSc. a po ověření funkčnosti některých předpokladů byl navrhnut (obrázek 6/strana 9) a postaven první prototyp badmintonového stroje. Tento stroj je zhotoven z profilů item a komponent přímo vyrobených pro stroj. Stroj v laboratorní verzi byl pouze stolek na kolečkách.

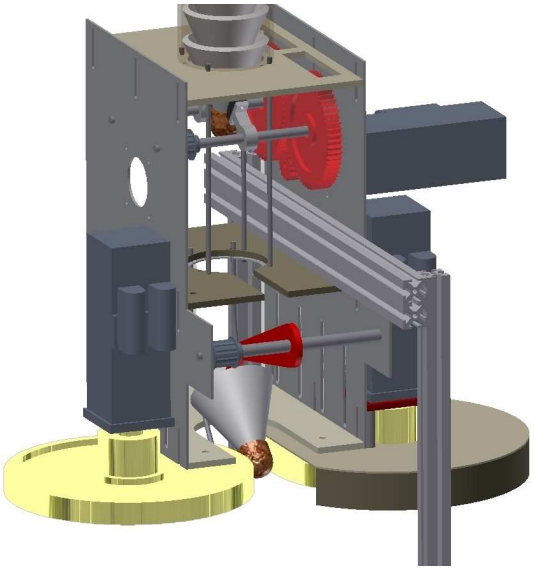


Obrázek 5: Badmintonový stroj

K prvnímu prototypu se vyrobil i skutečný podavač míčků. Model z Merkuru, i když v životní velikosti, se hodil pouze k ověření myšlenky podávání míčků.

Jak je vidět na obrázku 7/strana 10, prototyp a jeho podavač kopíruje návrh z Merkuru. Motor s mechanizmem podávání míčků je spojen ozubenými koly v poměru *1:4*. Podavače (excentrická kola) pro připravení jednoho míče jsou s kotvou pro podání míčů do kotoučů propojena zubatým řemenem s poměrem *1:1*.

První prototyp se podařilo sestavit a oživit. K většímu testování nedošlo, pouze se ověřila základní funkčnost a je třeba nadále testovat a vyvíjet. K účelům dalšího vývoje je prototyp sestaven tak, že je velmi jednoduché změnit některé proporční vzdálenosti stroje. Například se dá změnit vzdálenost kotoučů, výška kotoučů vzhledem k podavači, výška zádržných mezikruží vůči samotnému mechanizmu, v neposlední řadě to je velikost zádržných mezikruží.

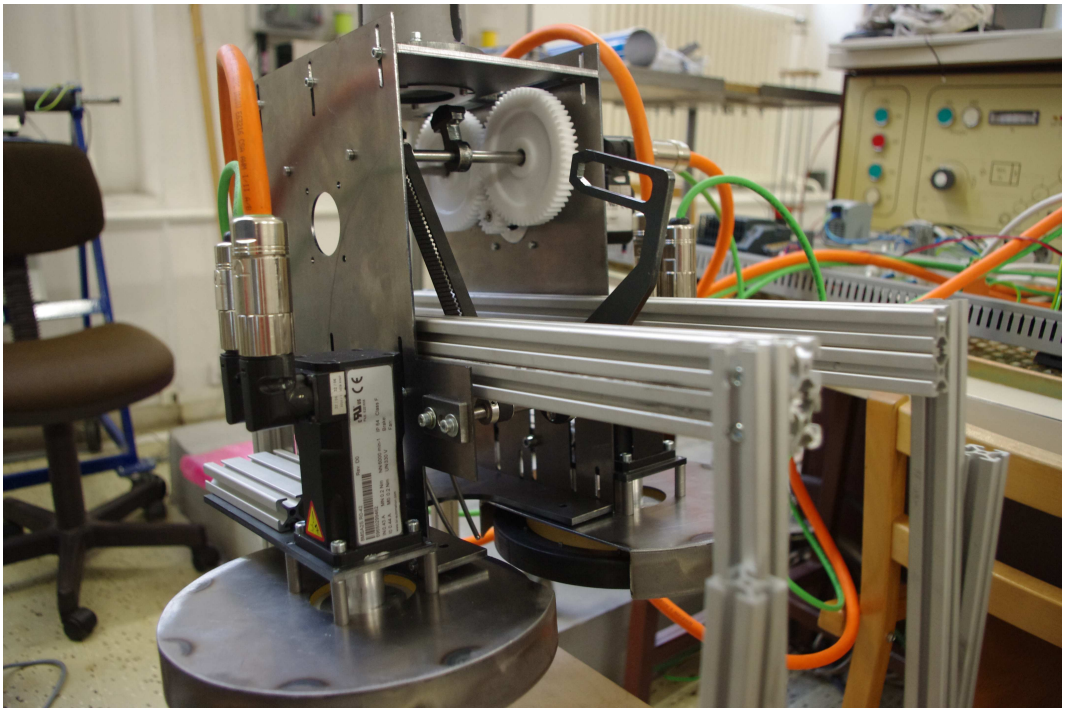


Obrázek 6: 3D model prototypu badmintonového stroje

4.1 Hardware

4.1.1 Řídicí jednotka

Hlavním členem použitým k řízení stroje je PLC. Jedná se o starší typ *X20 CP1485* (obrázek 8/strana 11). Vzhledem k nenáročnosti na výpočetní výkon je, co se týče procesorové stránky, PLC zatím dostačující. Pro lepší možnosti vizualizace je ovšem tento model nevyhovující. Velikost paměti RAM značně omezuje možnosti vizualizace. K vizualizaci a tím i ovládání celého stroje je použita vizualizace přes VNC server, který běží přímo na PLC. Při paměti *64* MB jsem byl nucen použít obrazovku s rozlišením 320×240, což v dnešní době je dávno překonané a bylo by vhodné uživatelské prostředí vylepšit použitím jiného PLC s větší pamětí. Tím by nastala možnost zvýšit rozlišení. U novějších PLC od firmy B&R s větší pamětí lze použít vizualizaci až do FullHD rozlišení. To je sice pro první prototypy nepotřebné, ale ve finální verzi pro případného uživatele vhodné. Dále při nahrazování PLC je vhodné použít takové, které umožňuje použít openSAFETY. To umožní větší



Obrázek 7: První prototyp

bezpečnost, ta je vzhledem k velkým rychlostem rotujících kotoučů důležitá.

Při realizaci prvního prototypu s podavačem míčků jsem narazil na problém nedostatku RAM paměti. A to nejen pro větší vizualizaci, ale po přidání třetí osy (motoru) není v PLC dostatek místa na jakoukoliv vizualizaci. Tento problém jsem dočasně vyřešil provozováním stroje na Power Panelu *PP65* (PLC spolu s HMI). Jedná se o PLC s vlastním displejem a vizualizace tím pádem není pouze jako VNC server, ale i na dotykové obrazovce panelu *PP65*.

4.1.2 Řízení motorů

Na prvním prototypu jsou použity servozesilovače ACOPOS *8V1010* . Společně s PLC se jedná o řídicí systém, který byl na fakultě nevyužit a dal se tedy použít na toto zařízení. ACOPOSy mají ve svých slotech zasunuty dvě karty. Jedna je

Obrázek 8: X20 CP1485

AC122 k této kartě je připojen rezolvér v motoru a slouží pro zpětnou vazbu polohy a rychlosti motoru. Druhá kata *AC114* je powerlinková karta pro komunikaci s ostatními částmi řídicího systému. Komunikace po powerlinku je popsána v samostatné kapitole 4.6 na straně 19.

Rozměry ACOPOSu jsou v porovnání s rozměry a výkony požadovanými na stroj naddimenzované a servozesilovače již v prvním stroji zabíraly poměrně hodně místa. Na obrázku 5/strana 8 je první elektrické zapojení pouze pro dvě osy, tzn. ovládání kotoučů bez podavače.

I tak je celá konstrukce poměrně těžká. Řešením je například použití systému ACOPOSmulti. Ten z jednoho zařízení může ovládat více os, nebo ACOPOSmikro, to je navíc ještě velikostně a výkonově přijatelnější. Při použití jakéhokoliv jiného řešení na bázi ACOPOS je velkou výhodou zachování celého řídicího programu. Změní se pouze fyzická konfigurace pohonů.

4.1.3 Motory

K pohánění celého stroje jsou použity motory firmy B&R. Jedná se o nejmenší třífázové synchronní motory z řady 8MS. Přesněji motory s označením *8MSA2S.R0-42 Rev. D0* . Tyto motory jsou sice rozměrově přijatelné, ale jejich výkon není příliš vysoký. Kroutící moment těchto malých motorů je 0,2 Nm. Tímto parametrem je limitováno maximální zrychlení, kterým se dají kotouče roztáčet, popřípadě brzdit.

Obrázek 9: Motory řady 8MS

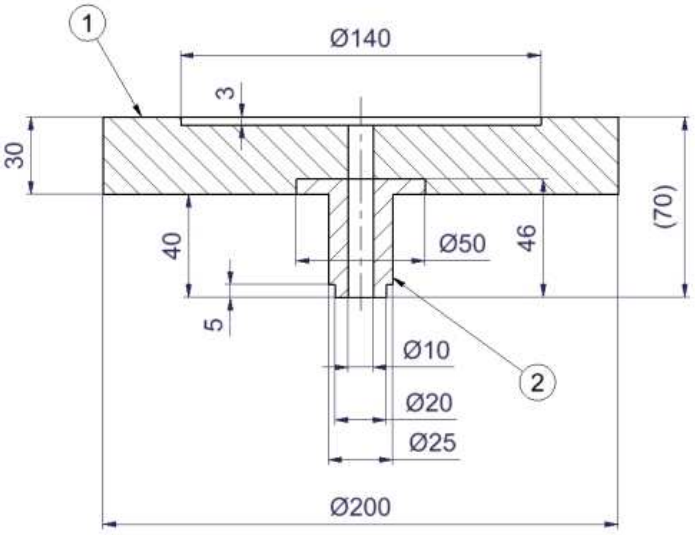
Výpočet týkající se toho, jaký motor je vhodný, popřípadě co s danými motory jsme schopni dokázat, je v kapitole 4.3 na straně 15 .

4.2 Kotouče

K odpalování, jak už je popsáno výše, jsou použity dva kotouče, které se točí proti sobě. Tyto kotouče jsou vyrobeny ze silonu. Na obrázku 10/strana 13 je náčrtek původních kotoučů. Výkres kotoučů byl proveden v programu Autodesk Invertor, za což děkuji Ing. Josefu Kultovi. Tento kotouč měl několik nedostatků. Pro použitý typ motoru je moc těžký a má velký moment setrvačnosti, a proto měl celý stroj špatnou dynamiku. Výpočet je uveden v kapitole 4.3 na straně 15. Dále pak při odpalování dochází k velkému prokluzu mezi povrchem kola a hlavičkou badmintonového míčku, tím je předávání energie značně neefektivní a dolet je poté limitován na nižší hodnoty, než by bylo teoreticky možné dosáhnout. Kotouče již při poloviční rychlosti, než které jsou maximálně schopny dosáhnout, střílejí míčky na maximální vzdálenost. Měření je dokumentováno v kapitole 6 na straně 29.

4.2.1 Odlehčení kotouče

Pro dynamiku roztočení kotoučů není důležitá jejich hmotnost, ale moment setrvačnosti (kapitola 4.3/strana 15). Proto bylo nutné udělat odlehčení ne za účelem

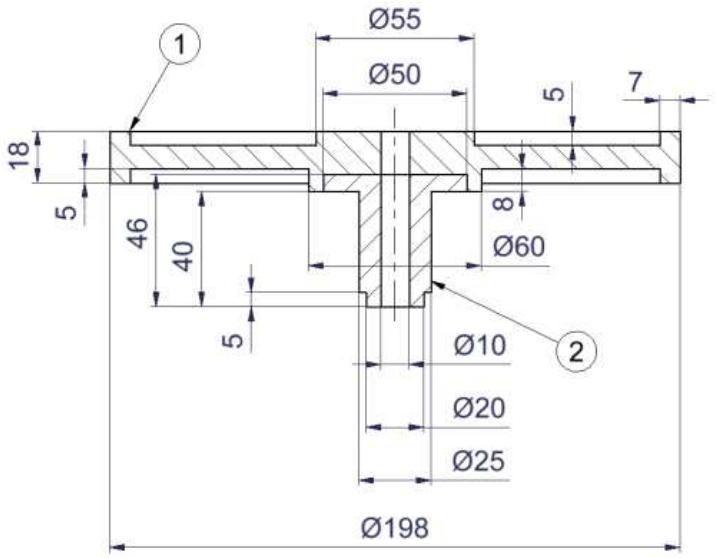


Obrázek 10: Výkres původního kotouče

snížení váhy, ale minimalizace momentu setrvačnosti. Prvním návrhem bylo odvrtání materiálu. Tato možnost se na výsledném momentu neprojeví tolik, kolik byl předpoklad, a proto jsem od ní odstoupil. Další možností bylo zmenšení rozměrů kotoučů odsoustružením přebytečného materiálu. Vzhledem k tomu, že kotouče jsou vyrobeny z vcelku pevného materiálu (silonu) a namáhání není moc velké, dochází pouze ke zmáčknutí míčku. Kotouče je tedy je možné zmenšit, a to na rozměry na obrázku 11/strana 14. Podle tohoto náčrtu byl skutečně kotouč upraven a v příloze je výsledný výkres.

4.2.2 Pogumování kotoučů

Pro zlepšení přenosu energie mezi kotoučem a míčkem by bylo dobré povrch kotoučů po obvodu pogumovat. Teoreticky se tím zvýší přilnavost a míček nebude tolik prokluzovat. Nastal ovšem problém s tím, jak gumu na kotouči udržet. Vzhledem k tomu, že kotouče jsou vyrobeny ze silonu, nedrží na něm téměř žádné lepidlo. Silon je velmi mastný materiál, nehledě na to, že na gumě také ne každé lepidlo drží. Dalším



Obrázek 11: Výkres upraveného kotouče

faktorem je, že při nabrání a zmáčknutí míčku se guma na kotouči začne hrnout. Vytvoří se jakási vlna a ta zapříčiní větší zátěž lepeného spoje, až do okamžiku, kdy lepidlo povolí. Tento fakt lze pozorovat i na mnohem pevnějších materiálech, jako je například kov.

První varianta, kterou jsem odzkoušel, bylo vyříznutí pruhu gumy z duše na jízdní kolo. Tento proužek byl skoro stejně dlouhý jako obvod kotouče a konce k sobě byly přilepeny. Hrana řezu byla pro lepší spojení šikmá a spoj byl proveden podlepením pevnou modelářskou páskou se skelnými vlákny. Takto zhotovené pásky jsem natáhl na kotouče a po roztočení se objevil nový problém, a to velká odstředivá síla. Na gumu již při zhruba 200 ot s^{-1} působí velká odstředivá síla, dochází k oddálení gumy od kotouče. Guma se postupně tak natáhne, až z kotouče odpadne a odstřelí do prostoru.

Druhý způsob byl pokus s podlepením pásku z předešlého pokusu. Pásky jsem ke kotouči zkusil přilepit chemoprénem. Na některých místech lepidlo drželo, ale

nechytlo se všude a navíc došlo k efektu s vlnou a guma se odlepovala a po nějaké době by došlo k odlepení a odletu gumy.

Třetí možností bylo proužek gumy vystřihnout z duše k automobilu. Takto vystřižený pásek je celistvý a odpadá tedy problém s lepením atd. Další výhodou je, že pásek se na gumu musí opravdu natáhnout. Průměr duše je menší než průměr kotouče. I přes fakt, že na kotouči byla guma již dost natažená, při roztočení se natáhla ještě více a z kotouče, stejně jako v prvním případě, odletěla.

Poslední vyzkoušenou možností je přišroubování gumy mezikružím. Pan Bartoš navrhl a zhotovil mezikruží, které zapadá za vysoustruženou část na kotoučích (obrázek 11/strana 14). Guma na kotoučích je pak ustřižena tak, aby se konce daly přetáhnout až za vysoustružený okraj a přišroubovat mezikružím. Tímto způsobem sice vzroste moment setrvačnosti, ale guma pak na kotoučích drží a to i při roztočení na maximální otáčky. Toto řešení sice není ideální, ale pro prototyp a otestování vlastností odpalu s pogumováním postačující. Do budoucna by bylo vhodné kotouče vyrobit z jiného materiálu např. celogumové s železným středem, nebo je po obvodě nechat pogumovat. Guma by se na obvod nanesla nějakým lepším způsobem, než je lepení chemoprénem a přišroubování mezikružím.

4.3 Návrh parametrů motorů

Při výběru motorů je třeba vybrat takový, který odpovídá požadavkům na dynamiku stroje. V případě badmintonového stroje je situace otočená a to tak, že k motorům, které mám k dispozici, je třeba zjistit, jakým maximálním zrychlením lze roztáčet kotouče. V obou případech se vychází ze stejného vztahu 1 pro výpočet momentu síly.

$$M = J\alpha \tag{1}$$

Kde M je moment síly, J je moment setrvačnosti a α je úhlové zrychlení. Pro naše potřeby se vztah upraví na 2

$$\alpha = \frac{M}{J} \tag{2}$$

Moment síly je dán použitým typem motoru a je 0,2 Nm. Moment setrvačnosti je dán samotným kotoučem J_1, kovovým středem J_2 a momentem setrvačnosti samotného motoru J_3.

$$J_c = J_1 + J_2 + J_3 \tag{3}$$

Program Invertor nebyl použit pouze k výkresům kotoučů, ale především k výpočtu momentů setrvačností a to jak kotoučů, tak kovového středu. V podkapitole 4.2.1 na straně 12 je kotouč s jeho parametry popsán lépe. Pro výpočet maximálního zrychlení je důležitý pouze moment setrvačnosti. Moment setrvačnosti kovové neměnné části je dle výkresu $J_2 = 13,7\,\mathrm{kgmm^2}$. Z manuálu k motoru nebo z Helpu v Automation Studiu lze zjistit, že moment setrvačnosti motoru *8MSA2S* je $J_3 = 60\,\mathrm{kgmm^2}$. Z výkresů ke kotoučům v příloze je vidět, že původní kotouč měl moment setrvačnosti $J_{11} = 5460\,\mathrm{kgmm^2}$ a po osoustružení (zmenšení rozměru) je jeho moment setrvačnosti $J_{12} = 1916\,\mathrm{kgmm^2}$.

Původní moment setrvačnosti soustavy, kotouč a motor, je po dosazení do vztahu 3 tedy $J_{c1} = 5533,7\,\mathrm{kgmm^2}$, soustava s odlehčeným kotoučem má moment setrvačnosti $J_{c2} = 1989,7\,\mathrm{kgmm^2}$.

Pro výpočet maximálního zrychlení a tím i určení samotné dynamiky roztočení kotoučů použijeme vztah 2. Po dosazení dostáváme dvě hodnoty pro maximální zrychlení a to pro původní kotouč $\alpha_1 = 36,14\,\mathrm{rad\,s^{-2}}$ a pro odlehčený kotouč $\alpha_2 = 100,51\,\mathrm{rad\,s^{-2}}$, neboli $\alpha_1 = 5,75\,\mathrm{ot\,s^{-2}}$ a $\alpha_2 = 16,00\,\mathrm{ot\,s^{-2}}$.

Pro lepši představu o dynamice je vhodné znát čas, za který se kotouče roztočí.

$$t_{min} = \frac{\omega_{max}}{\alpha_{max}} \tag{4}$$

Kde t_{min} je minimální čas, za který se kotouče roztočí na maximální úhlovou rychlost ω_{max}, při maximálním možném zrychlení α_{max}. Maximální otáčky motoru jsou $\omega_{max} = 100\,\mathrm{ot\,s^{1}}$ při daných zrychleních lze těchto otáček dosáhnout za $t_1 = 17{,}39\,\mathrm{s}$ respektive $t_2 = 6{,}25\,\mathrm{s}$. Z těchto údajů je vidět, že odlehčení kotoučů bylo nezbytné pro lepší dynamiku, která se tím téměř ztrojnásobila.

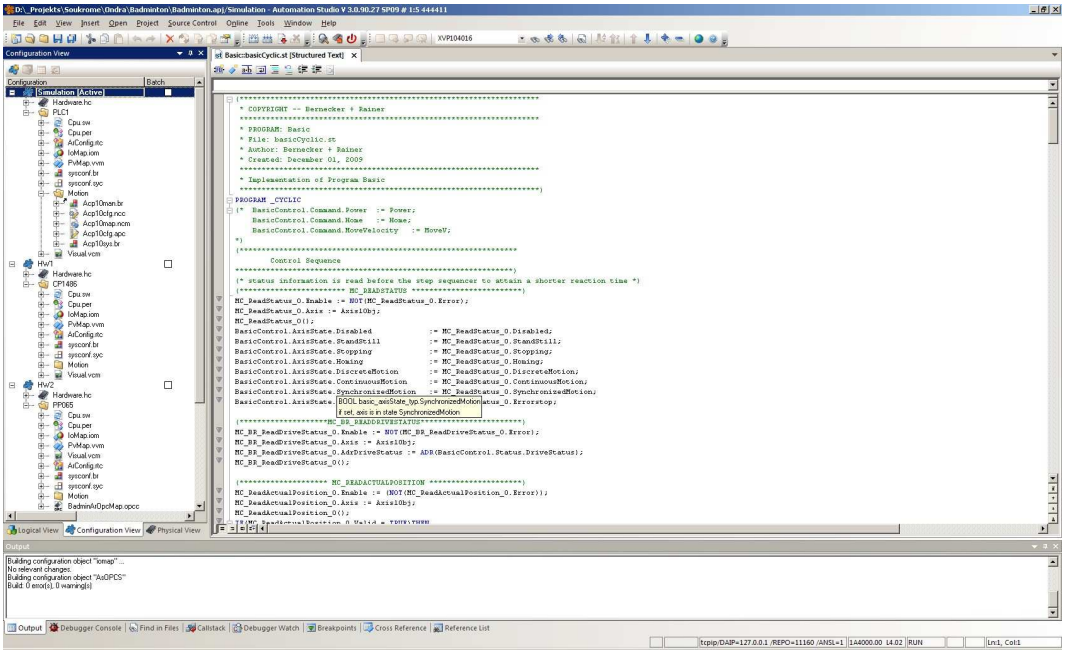
4.4 Software

4.4.1 Automation Studio

Firma B&R vyvíjí vlastní vývojové studio. Současná verze má číslo *4*. Já jsme ovšem používal studio ve verzi *3*. Licenci k tomuto studiu má škola k dispozici. Celé studio je velmi kvalitně zpracováno a je dost přehledné již při základním nastavení (obrázek 12/strana 18). Na rozdíl od většiny jiných programovacích nástrojů je ve studiu kvalitně zpracována nápověda (Help). Ta se opravdu dá použít, a to nejen k řešení problémů, ale i jako studijní materiál. V nápovědě jsou nejen údaje spojené se samotným PLC a jeho programováním, ale dají se zde najít i údaje k ACOPOSu atd. Jsou zde i manuály k ostatnímu hardwaru, a to například k motorům.

4.4.2 Programování PLC

PLC od firmy B&R se dá programovat ve všech jazycích dle normy *IEC EN 61131-3* a v B&R Automation Basic, což je velkou výhodou. Já jsem si zvolil programování ve strukturovaném textu a B&R Basicu. Ke všem způsobům programování se dají dohledat potřebné informace v Helpu. Ke každému jazyku je uvedena syntaxe s příkladem použití a není tedy třeba vyhledávat podporu z externích zdrojů. Při programování lze využít spousty příkladů, které napomáhají k



Obrázek 12: Automation studio 3.

rychlému a efektivnímu řešení daného problému. Při programovaní badmintonového stroje byly použity příklady pro ovládání servopohonů a jejich synchronizace (virtuální převodovka).

4.4.3 Vizualizace

Pro ovládaní můžou být použita hardwarová tlačítka nebo vizualizace s virtuálními tlačítky. Na některých PLC automatech je k dispozici LCD panel s dotykovými tlačítky nebo s celou dotykovou plochou. Mezi další možnosti patří vzdálená vizualizace přes VNC server. Vzhledem k použitému HW a požadavkům na ovládání jsem si vybral vzdálenou vizualizaci pomocí VNC serveru.

Pro tvorbu vizualizace na panelu i pro VNC je přímo ve studiu implementován grafický editor. Tento editor funguje na jednoduchém principu rozmístění požadovaných prvků a nastavení jejich vlastností. Pro potřeby vizualizace můžeme použít například : tlačítka, popisky, vstupní pole atd. U těchto prvků je možné na-

stavit jejich inicializační hodnoty, funkce po kliknutí, možnosti vkládání dat atd.

4.5 Ovládání motorů

Na badmintonovém stroji jsou v současnosti tři motory, které je třeba řídit. Výhodou použití ACOPOSu s knihovnou PLCopen Motion Control je snadné ovládání. Nemusíme se starat o programování samotného ovládání jako je rozběh, zastavení po rampách, PID regulátor proud, otáček, polohy. Část programu, která se o toto stará, lze nalézt v Helpu ke studiu.

Motory, které na stroji jsou, je možné rozdělit na dvě skupiny. Dva se starají o vyhazování míčků. Tyto motory pohánějí kotouče a jsou navzájem synchronizované virtuální převodovkou. Třetí motor se stará o podávání míčků. Konstrukce tak, jak byla navržena a vyrobena, umožňuje na podávání míčku použít pouze jeden motor.

4.5.1 Virtuální převodovka

Jak už je napsáno výše, kotouče jsou poháněny dvěma motory, které jsou mezi sebou spojeny virtuální převodovkou. Jeden z motorů je označen jako MASTER, a tento motor přímo řídíme. Druhý motor - SLAVE, pouze kopíruje mastra v daném převodu. V případě badmintonového stroje je převod virtuální převodovky *-1*, a to z toho důvodu, že potřebujeme, aby se kotouče točily proti sobě. Zdrojový kód programů je uveden v příloze a společně s celým projektem na CD.

4.6 Powerlink

V této podkapitole je stručný popis POWERLINKU. Při tomto popisu jsem čerpal z [4].

Jedna z možností, jak mezi sebou může komunikovat PLC a servozesilovač (ACOPOS), je použití technologie POWERLINK. Tato komunikace neslouží pouze ke

komunikaci mezi PLC a ACOPOSem, ale dá se použít i ke komunikaci mezi více PLC, vizualizačním panelem, senzory atd. POWERLINK je komunikace v reálném čase založená na technologii Ethernetu. Ethernet jako takový, jak jej známe z počítačových sítí, je nedeterministický. Při přenášení velkého množství dat v počítačové síti nám nevadí skutečnost, že nevíme, kdy přesně daný paket dojde, nebo pokud se jeden ztratí. V případě průmyslových sítí by takovýto výpadek mohl mít vážné následky, obzvlášť jednalo-li by se o příkazy motorům. Na druhou stranu použití Ethernetu v průmyslových sítích je vzhledem k jednoduchosti propojení velmi žádané. Navíc se do takové sítě dají použít běžné prvky z PC sítě, jako je například HUB. Proto firma B&R v roce 2001 představila protokol POWERLINK zajišťující deterministickou real-time komunikaci na bázi Ethernetu.

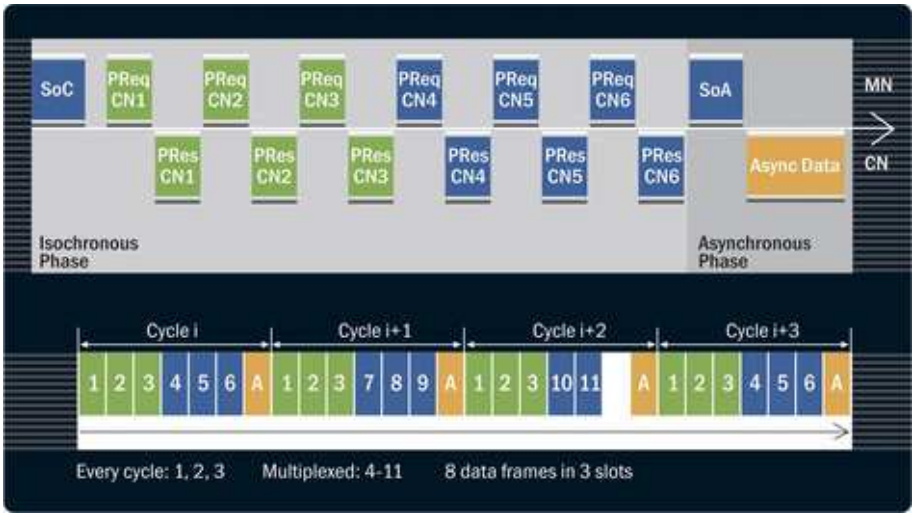
Obrázek 13: Logo EPSG

Ethernet POWERLINK je otevřený protokol vyvíjen EPSG (Ethernet POWERLINK Standardization Group).

Technologie nemá nic společného s napájením přes ethernet PoE (Power over Ethernet). Vývoj protokolu probíhá ve spolupráci s CiA (CAN in Automation) a díky tomu je POWERLIK kompatibilní s profilem CANopen.

Po nastartování systému komunikaci na síti řídí hlavní uzel MN (Managing Node). Na síti je vytvořena komunikace v základním cyklu, který zajišťuje determinističnost komunikace a zároveň možnost komunikace v reálném čase (obrázek 14/strana 21). Na začátku každého cyklu je vyslán od MN startovací rámec dat SoC (Start of Cycle). Tento rámec přijímají všichni a slouží k synchronizaci.

Poté následuje izochronní část komunikace. Ta je určena k přenosu všech časově kritických dat. Jako jsou například příkazy k řízení a údaje od senzorů. Hlavní uzel



Obrázek 14: Komunikace po POWERLINKu

MN (obvykle PLC) vysílá tzv. Preq - Poll Request rámce, tím odešla data určená každému uzlu. Daný uzel odpovídá tzv. Pres - Poll Response rámcem a tím naopak uzel pošle časově kritické údaje do MN. Tímto způsobem se obslouží všechny uzly na síti. Komunikaci na síti po celou dobu poslouchají všechny body, jedná se tedy o model Producent Konzument. Časový úsek pro Pres a Preq - Roll Request rámec pro jeden uzel se nazývá "časový úsek pro adresovaný uzel (bod)".

Po ukončení izochronní části komunikace následuje asynchronní část. Tento úsek je určen k odeslání všech ostatních dat, která chceme přenést po síti. Hlavní uzel MN vyšle právo na jeden konkrétní bod vysílat Ad-hoc a to zasláním rámce SoA (Start of Asynchronous). Při této komunikaci funguje standardní IP protokol. Tento stav přetrvává až do vyslání dalšího startovacího rámce SoC.

V případě velkého množství uzlů a pro zkrácení doby nutné k obsloužení všech je možné použít multiplexování. To znamená, že ne každý uzel musí být obsloužen v daném cyklu. Například uzly 1 - 3 je nutné obsluhovat v každém cyklu, ale uzly 4 - 11 je možné obsloužit jednou za 3 cykly (obrázek 14/strana 21). Tím se zmenší délka izochronní části a dojde ke zlepšení real-time komunikace. Při větší délce trvání se

synchronizace rozchází.

4.7 Programy pro odpal míče

Výše popsané principy řízení motorů jsou použity k realizaci programů pro odpalování badmintonových míčů.

V současné době se na prototypu nedají realizovat požadavky pocházející z metod tréninku. Stroj je na začátku vývoje a zatím je veškerá práce soustředěna na návrh a realizaci principu výstřelu a podávání míče. Dalšími kroky pak bude automatická změna elevace a úhlu výstřelu míče.

V současné podobě stroje se dají měnit pouze dva parametry, které ovlivňují chování. Jedním z parametrů je rychlost kotoučů. Na této rychlosti závisí dostřel stroje a ten se pohybuje od nuly až do cca osmi metrů (kapitola 6 strana 29). Dostřel je možné ovlivnit elevací výstřelu a výškou, z které se střílí.

Druhý parametr, který se v současné době dá měnit, je interval výstřelu. Tento parametr do stroje zanáší změnu dynamiky hry. I v této základní verzi je dobré mít možnost stroje zkoušet a ověřovat v reálném prostředí. Proto je důležité měnit tempo výstřelu pro zpestření a zajímavost tréninku.

4.7.1 Automatické programy

Na základě možností, které na současném stroji mám, jsem připravil tři automatické programy odpalování míčků.

Programy

Náhoda (Program 1) V tomto programu je jak rychlost kotoučů, tak interval odpalu míče určen pseudonáhodným číslem.

Kotouče jsou roztáčeny a brzděny po náhodně generované rampě. Rampa je generována jako minimální rychlost kotoučů (ω_{min}), plus generovaná hodnota ($\Delta\omega$). Generovaná hodnota je v rozsahu $0 \ldots (\omega_{max} - \omega_{min})$, kde ω_{max} je maximální úhlová rychlost.

$$\omega = \omega_{min} + \Delta\omega \tag{5}$$

Jako generátor náhodných čísel byl použit Lineární kongruentní generátor (vztah 6). Ten pseudonáhodně generuje čísla od nuly až po maximální zadanou hodnotu (m).

$$x_{i+1} = (ax_i + c) \mod m \tag{6}$$

x_{i+1} je následující hodnota, x_i je současná hodnota.
c a m jsou nesoudělná čísla
Pro a platí : $a-1$ je dělitelné všemi provočíselnými faktory m a je násobek 4, jestliže m je násobek 4.

V programu pak lze nastavit minimální otáčky ω_{min}, maximální otáčky ω_{max} a periodu, se kterou se mění hodnota generátoru.

Stejným způsobem je generovaný interval odpalování míčků. U něj se dá nastavit minimální časový interval, maximální interval a perioda generátoru. Oba generátory jsou na sobě nezávislé, a tím i perioda přepočítání nové hodnoty.

Zavedením dvou náhodných generátorů je tento program opravdu náhodný a to v obou parametrech, které se u prototypu dají měnit.

Náhodná rychlost (Program 2) Tento program má náhodně generované rampy na motorech s kotouči stejně jako v Programu 1, tzn. pseudonáhodně vygenerovaná hodnota, která se přičte k minimální hodnotě. Interval odpalu je v tomto programu pevně nastavený. Dá se změnit v nastavení k programu.

Konstantní (Program 3) V tomto programu je zatím jak rychlost kotoučů, tak interval odpalu konstantní a nastavitelný v nastavení k programu.

Takovýto program je vhodný například pro měření doletu míčů nebo testování chování různých míčů atd.

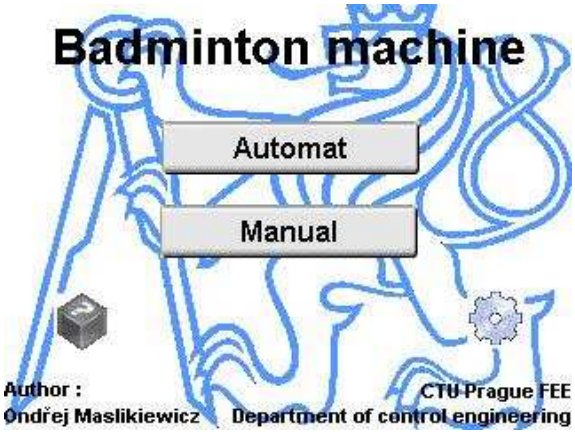
Všechny programy by bylo vhodné v budoucnu doplnit o definici počtu míčů, které se mají odpálit. Prozatím všechny programy běží, dokud nejsou vypnuty, a to bez ohledu na to, zda jsou ve stroji míče. S tímto souvisí i možnost doplnit stroj o senzor na detekci přítomnosti míčů.

5 Vizualizace badmintonového stroje

Pro vizualizaci badmintonového stroje a tím i jeho ovládání, jsem zvolil možnost vizualizace pomocí vzdáleného přístupu zprostředkovávaného VNC server implementovaným přímo v PLC. Podoba vizualizace byla navržena a vytvořena v prostředí Automation studia 3.

Samotná vizualizace se dá rozdělit do dvou částí, a to pro běžné použití a poté pro servisní obsluhu. Obě části jsou součástí jedné vizualizace v rozlišení 320×240. Toto rozlišení limituje počet komponent, které se vejdou na jednu stránku. Celá vizualizace je dvojjazyčná a to v anglické a české jazykové mutaci. Jako nativní jazyk je zvolena angličtina. Jazyk se ovšem dá ve vizualizaci přepnout.

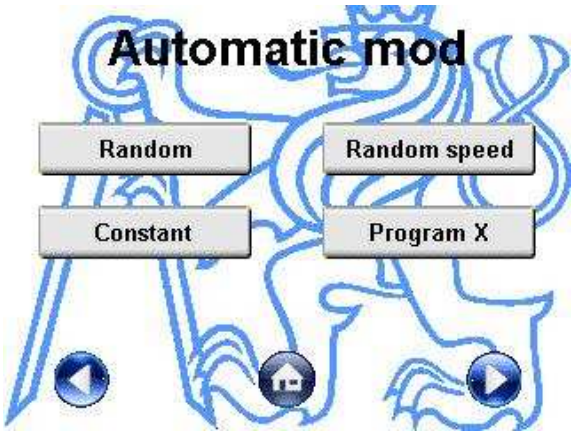
Při startu PLC a tím i VNC serveru se zobrazí úvodní obrazovka (obrázek 15/strana 25), na které je rozvětvení na automatický a manuální režim odpalování míčků. Dále je zde nápověda, která je navržena jako vrstva, nikoliv jako samostatná stránka. Poslední důležitá věc je možnost přepnutí na stránku s nastavením.



Obrázek 15: Vizualizace : hlavní obrazovka

5.1 Automatický režim

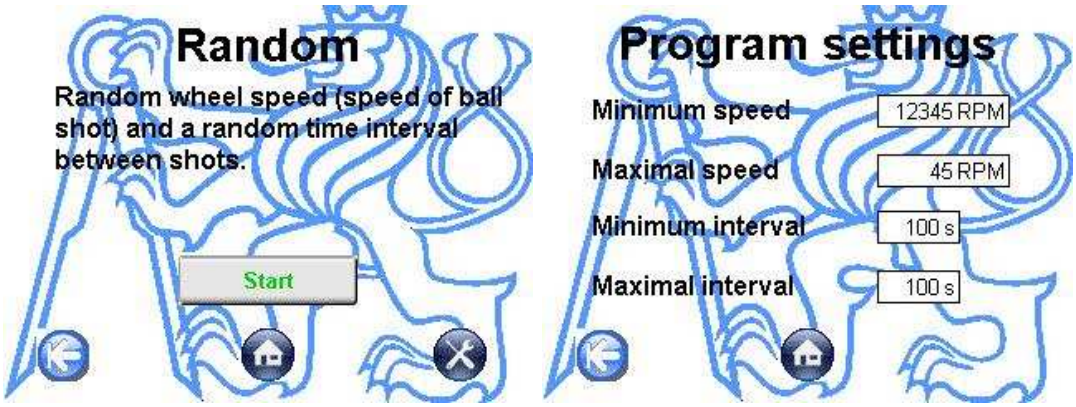
Při volbě automatického režimu se vizualizace přepne na obrazovku



Obrázek 16: Vizualizace : Automatické režimy

(obrázek 16/strana 25), na které je možnost volby různých automatických programů. V současném provedení vizualizace, která odpovídá současnému stavu a schopnostem stroje, je automatických režimů pouze pár, a jsou pouze pro ukázku a testování stroje. Ve finální verzi stroje zde bude volba režimů, které pro badmintonový trénink budou užitečné.

Ve spodní části obrazovky jsou šipky pro listování mezi volbou programů a tlačítko Domů (Home). Tímto tlačítkem se dostaneme na úvodní stránku.



Obrázek 17: Vizualizace : Program odpalování

Každý z automatických programů má pro vizualizaci vlastní stránku (obrázek 17/strana 26), na které je krátký popis toho, co má stroj dělat. Umožňuje-li program nějaké nastavení, je toto nastavení na této stránce. Například nastavení intervalů výhozu míče, maximální rychlosti atd. Z této obrazovky je možnost se vrátit buď na stránku s volbou programů, nebo na úvodní stranu.

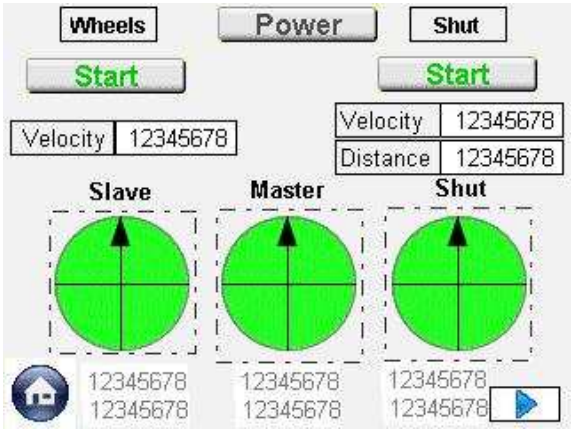
5.2 Manuální režim

Pod tlačítkem manuálního režimu na hlavní obrazovce se skrývá obrazovka s možností kompletního ručního nastavení stroje. Veškeré parametry pro odpálení míče se zde dají nastavit a tím si vytvořit vlastní téměř poloautomatický režim. Z těchto důvodů je možnost nastavení počtu míčů, které se mají odpálit. Správným nastavením stroje lze dostřelit kamkoliv v limitech samotného stroje.

Z této stránky se přes tlačítko Home dostaneme na hlavní obrazovku.

5.3 Servisní vizualizace

Vizualizace je kromě uživatelského módu pro běžné použití vybavena i servisní



Obrázek 18: Servisní vizualizace : Hlavní obrazovka

částí. Tato část je určena k testování samotného stroje a motorů jako takových, nikoliv k běžnému použití pro hru.

Na první obrazovce po přepnutí do servisního modu je možnost ovládání všech tří motorů najednou. Po zapnutí tlačítka *Power* se zapnou všechny motory, proběhne metoda home na všech osách a motory pro výstřel míčku se spojí převodovkou. Tento proces lze sledovat jak v reálu přímo na kotoučích, tak ve vizualizaci v grafickém znázornění kotoučů. Ty jsou ve vypnutém stavu šedé, při pohotovostním režimu se zbarví do žluta, a ve stavu ready jsou zelené se šipkou označující natočení (po zapnutí nahoru).

Na stránce se dá nastavit rychlost, kterou se mají kotouče roztočit, a tlačítkem *Start* se roztočí se zrychlením, které se dá změnit v nastavení pro Master motor. To, že se motory točí, je ve vizualizaci znázorněno otáčením grafických koleček a pod nimi je hodnota aktuální rychlosti a pozice. Motory se zastaví vymáčknutím tlačítka *Stop*.

U třetího motoru se nastavuje nejen rychlost, ale i vzdálenost, kterou má motor

urazit. Pro odpálení míče je potřeba mechanizmem udělat celý kruh. Při zmáčknutí druhého tlačítka *Start* se motor roztočí danou rychlostí s daným zrychlením a urazí zadanou vzdálenost.

Obrázek 19: Servisní vizualizace : Ovládání motoru

Při kliknutí na obrázek znázorňující kotouče se obrazovka přepne na příslušnou stranu vizualizace s ovládáním daného motoru. Ve spodní části je vlevo tlačítko pro návrat na hlavní obrazovku uživatelské vizualizace. A vpravo dole je tlačítko pro přepnutí stránky.

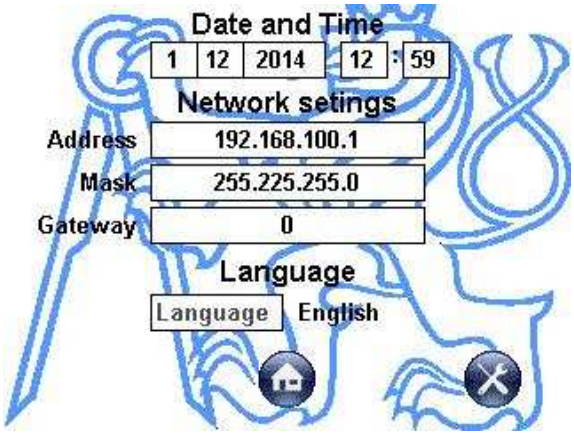
Na další straně je vizualizace obsluhy motoru *Master* . Na této stránce je možnost kompletního ovládání motoru. Je zde možnost vyzkoušet všechny úkony, které se s motorem dají dělat. Dále je zde zapínání a vypínání virtuální převodovky a zobrazení aktuálního stavu, a to rychlost, poloha a error ID. Ve spodní části je přepínání obrazovky.

Další strana je nastavení parametrů motoru *Master* . Je umožněno nastavit veškeré parametry, které jsou zpřístupněny pro ovládání motoru.

Další strany jsou ovládání a nastavení zbývajících dvou motorů a jejich parametrů. Jedná se o motory *Slave* a *Shut* .

5.4 Nastavení

Poslední volbou na hlavní obrazovce je možnost přepnutí na nastavení stroje. Na této obrazovce (obrázek 20/strana 29) lze nastavit čas a síťové parametry stroje. PLC je totiž pro komunikaci s vnějším světem vybaveno ethernetovým portem. Ko-



Obrázek 20: Vizualizace : Nastavení

munikaci po síti je tedy nutno někde nastavit. Standardně se toto dá nastavit při programování ve studiu, ale pro reálné používaní stroje je dobré mít možnost toto nastavení přímo na stroji změnit.

Další důležitou věcí na této stránce je přepínání jazyků. Jak již bylo popsáno výše, celá vizualizace je provedena ve dvou jazykových mutacích. V případě nutnosti není problém do programu zavést další jazyky.

Samozřejmě i na této stránce je možnost návratu na hlavní obrazovku. Kromě této volby je zde přepnutí i na servisní část vizualizace.

6 Měření dostřelu badmintonového stroje

Úkolem měření je zjistit závislost mezi otáčkami kotoučů a vzdáleností, do které doletí vystřelený míček. Závislost je nutno znát pro možnost přesného střílení míčku

po kurtu. Let míčku je v podstatě balistická křivka, vzhledem k neznámým parametrům popisujících chování míčku ve vzduchu nelze vzdálenost dostřelu spočítat přesně. Můžeme ji určit pouze přibližně.

Pro výpočet je také nutné znát skutečnou rychlost míčku při opuštění stroje. Ta je teoreticky stejná jako je obvodová rychlost kotoučů, ale prakticky bude vždy menší. Během krátké doby styku kotoučů s míčem se nepřenese dostatečné množství energie na míček, a ten tím nemá stejnou rychlost jako je rychlost kotoučů. Dále dochází k prokluzu míčku mezi kotouči a zároveň rychlost míčku může být ovlivněna třením košíku (brk) o kotouče a desku.

6.1 Zadání

Za stejných výchozích podmínek (pozice stroje, elevace, rychlost kotoučů ...) bude postupně vypáleno několik (20) míčků. Vzdálenost měřená od stroje (čáry odpalu) k místu dopadu míčku bude měřena a zaznamenána do tabulky. Opakování měření má za účel zmenšit až odstranit chyby při odpalu. Při podávání míčku do stroje bez podavače, tzn. rukou nejde zajistit naprosto přesně shodné podání. Vliv na odpal může mít úhel naklonění i natočení míčku, hloubka podání míčku (vzdálenost mezi středem hlavy míčku a horním okrajem kotouče) atd.

Po změně rychlosti otáčení se postup měření zopakuje pro danou rychlost. Měření se musí provést na dostatečném počtu rychlostí v rozsahu minimum až maximum otáček. Maximální otáčky jsou dány parametry motoru 100 ot/s (6000 ot/min), v současném programu 100000 unit/s. Minimum otáček je zapotřebí určit, a to postupným zvyšování otáček z nuly na nejnižší hodnotu, při které má smysl měřit. Míček musí doletět za hranu stroje (stolečku), při nižších rychlostech se pouze po stolku posune nebo z něj spadne.

Na vzdálenost doletu míčku bude mít vliv kromě rychlosti kotoučů i elevace stroje. Ta při jednom měření zůstává stejná. Stejně tak i pozice stroje se nemění,

aby bylo možno měřit vzdálenost dopadu. Kromě těchto faktorů má vliv i výška stroje, ta je momentálně neměnná, ale je nutno ji zaznamenat (změřit).

6.2 Postup měření

1. Umístit stroj na bezpečné a dostatečně velké (dlouhé) místo.
2. Označit čáru odpalu, hranice stroje.
3. Připravit stroj na měření tzn. vše nastavit a pokud možno se strojem už nehýbat. Změřit výšku stroje.
4. Postupným zvyšováním rychlosti otáček, po malých krocích, zjistit minimální otáčky pro samotné měření.
5. Z rozsahu otáček min/max určit krok měření.
6. Při jedné rychlosti provést a zaznamenat několik měření vzdálenosti dopadu míčku.
7. Zvýšit otáčky o předem určený krok a opakovat bod 6.
8. Vynést závislost dopadu míčku na otáčkách do tabulky a grafu.

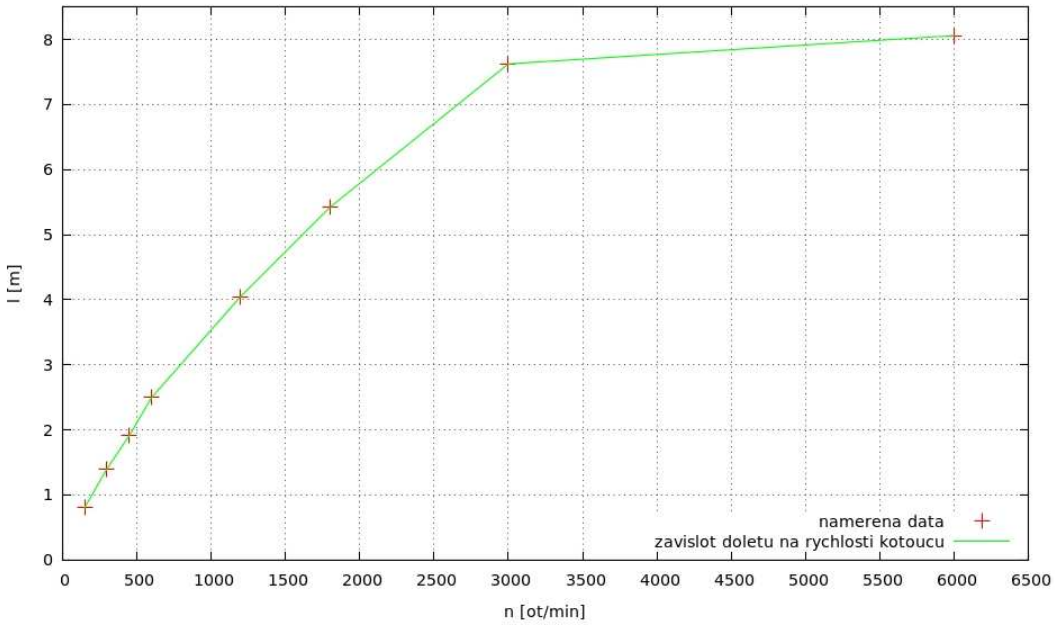
6.3 Měření bez pogumovaných kotoučů

Jako první jsem změřil závislost doletu míčků na rychlosti otáčení kotoučů bez pogumovaných kotoučů. Měření jsem provedl podle postupu, který jsem vytvořil a popsal výše. Naměřená data jsou zaznamenána v tabulce 1 na straně 38. U měření jsem neměl pevný krok zvyšování rychlosti kotoučů, protože závislost není lineární a měřit po malém kroku mezi 3000 -6000 ot/min by nemělo smysl.

Měření jsme provedl s dvaceti míčky, z toho 8 bylo plastových a zbylé byly péřové. Kotouče byly ve výšce 0,95 m a jejich vzdálenost byla 23 mm. Úhel odpalu byl cca 0,02 rad.

6.4 Závěr měření

Z naměřených dat, vynesených do grafu (obrázek 21/strana 32), lze vidět, že



Obrázek 21: Graf měření doletu bez pogumovaných kotoučů

závislost doletu míčků na rychlosti otáčení kotoučů je nelineární. Maximální vzdálenost při této konfiguraci stroje je zhruba 8 m. Nelinearita je zapříčiněna především faktem, že při otáčkách nad 3000 ot/min není možné míčku kvůli prokluzu předat dostatečné množství energie a tím rychlost, kterou míček opustí stroj, je téměř stejná jako při 3000 ot/min.

7 Problémy při realizaci

Při stavbě badmintonového stroje, respektive jeho prototypu, jsem narazil hned na několik problémů, které bylo nutné vyřešit. I přes fakt, že řešení těchto problémů bylo velice časově náročné, jsou pro mě cenným poznatkem. Jejich řešení je dobrou zkušeností pro případné zaměstnání v oboru řízení a automatizace.

7.1 ACOPOS a POWERLINK

Jeden z prvních problémům, který se vyskytl hned na začátku práce, při pokusu motory jenom roztočit, byla špatná komunikace servozesilovače po POWERLINKu. Tento problém nastal pouze u jednoho ze dvou motorů, což ztížilo jeho řešení, vzhledem k tomu, že na první dojem byly oba servozesilovače stejné.

Problém se projevoval, tím že servozesilovač vůbec nekomunikoval s PLC. Signalizace diodami na přední straně ACOPOSu naznačovala chyby komunikace po POWERLINKu. Tato chyba přetrvávala i po výměně „chybného“ servozesilovače za nový. Po přetrvávání problému jsme přišel na to, že v jednom ACOPOSU je již aktualizovaný Firmware. Podle Helpu a všech dostupných informací [3] totiž daná revize servozesilovče nepodporuje POWERLIKovou kartu verze 2 (*AC144*).

Přehrání Firmwaru u ostatní servozelivovačů není nic složitého, po zapůjčení POWERLINKOVé karty V1, která je nezbytná, se pouze do programu přidá knihovna, která FW v ACOPOSu aktualizuje a poté se dá ke komunikaci použít karta *AC144*.

7.2 První testování

Při pokusu o první testování, ještě s prvním strojem zhotoveným na stole jsem narazil na problém s napájením stroje. Na kurtech v tělocvičně stroj vyhazoval jističe respektive proudový chránič v něm integrovaný. Po několika testech jsem přišel na

to, že tento jev je způsoben připojením obou servozesilovačů. Vzhledem k jejich jmenovitému odběru to nemohlo být způsobeno nadproudem. V zapojení nebyl ani žádný zkrat, to by stroj nefungoval ani ve škole ve strojovně.

Po nastudování informací o vnitřním elektrickém zapojení ACOPOSu [3] jsem přišel na to, že na vstupu napájení je filtr zapojený proti zemi. Svod tohoto filtru respektive dvou, je dostatečně velký na to, aby proudový chránič obvod rozpojil.

S podavačem míčků jsou celkově použity tři ACOPOSy a bylo tedy nutné přistoupit ke galvanickému oddělení stroje od elektrické sítě. K oddělení jsem použil oddělovací transformátor, dostatečně dimenzovaný pro tři pohony. Pro lepší spínaní a ochranu v rámci stroje byl každý servozesilovač vybaven vlastním jističem.

7.3 Nedostatek paměti PLC

Tento problém je již výše zmíněn, jedná se o to, že původní PLC *X20CP1485* nemá dostatečně velkou paměť pro provozování větší vizualizace než v rozlišení 320×240 bodů. Toto by ve výsledku ani moc nevadilo, ale po přidání třetí osy (třetí pohonné jednotky) byla paměť v PLC nedostatečná pro jakoukoliv vizualizaci.

Pro dostavění a testování prototypu mi bylo zapůjčeno, Ing. Jiřím Maslikiewiczem z firmy Farmet a. s., PLC *PP65*, které má dostatečně velkou paměť RAM. Jedná se o panel s displejem vizualizace, tím pádem nebyla pouze na VNC serveru. Tím se celé testování dost zjednodušilo, stroj se dá ovládat přímo na displeji PLC.

Do budoucna je potřeba pořídit PLC, které paměťové nároky na tři osy pohonů a VNC server s vizualizací bude schopné splňovat.

8 Závěr

Cílem této práce bylo primárně postavit stroj na vystřelování badmintonových míčů. V rámci práce, kterou jsem na stroji dělal, jsem navrhl a zprovoznil první opravdový prototyp, který je doplněn o podávání míčků. K řízení a pohonu jsem použil produkty firmy B&R. Seznámil jsem se se servozesilovači ACOPOS a zajistil synchronizaci kotoučů pro odpalování míčů.

V rámci testování na stroji, a to již v původní podobě (bez podavače míčů), jsem ověřil některé teoretické možnosti změny směru letu míčů. Vzhledem k nepotvrzení teoretických předpokladů je současná verze stroje bez možnosti změny směru, ale v práci je uveden návrh, jak stroj modifikovat.

Vzhledem k tomu, že vývoj podavače a testování na původním stroji (motory na desce), nebylo triviální a ve skutečnosti bylo časově velmi náročné a výroba prototypu se i přes snahu udělat vše nejjednodušeji zpozdila, nezbylo mnoho času pro testování prototypu.

Provedl jsem měření závislosti doletu míče na rychlosti otáčení kotoučů, bohužel z časových důvodů a složitosti úprav na kotoučích nebylo možné v čas provést porovnání mezi pogumovanými a nepogumovanými kotouči. Toto měření doplním a budu pokračovat v úpravě kotoučů tak, aby v době obhajoby byla k dispozici další měření.

Pro lepší ovládání stroje jsem vytvořil vizualizaci, která běží na VNC serveru. Tímto způsobem je zajištěna možnost stroj ovládat z mobilních zařízení např. z chytrého telefonu. K tvorbě vizualizace jsem použil možnosti Automation studia od firmy B&R. V rámci vizualizace bylo navrženo i několik automatických programů pro ovládání stroje. A to jak teoretické, které vycházejí z požadavků trenéra, tak v současné podobě stroje realizovatelné.

Stroj na vystřelování badmintonových míčků je zatím pouze ve fázi vývoje, čemuž odpovídají i jeho současné schopnosti. Pro reálné použití je zapotřebí na stroji dále

pracovat. Například nahradit stávající programy pro ovládání motorů (ne všechny funkce jsou zapotřebí), nebo řádně otestovat prototyp a odladit jeho chyby. V případě svého budoucího studia bych se rád nadále věnoval této problematice a stroj zdokonalil a přiblížil finální podobě.

Literatura

[1] *Jaroš, P. .:* **Rozšíření modelu žonglér a řízení CNC stroje.**
Diplomová práce, K13135 ČVUT v Praze FEL, 2011

[2] *Kohout, T. .:* **Model žonglér pro vzdálenou výuku a řízení CNC stroje.**
Diplomová práce, K13135 ČVUT v Praze FEL, 2011

[3] *Firemní literatura společnosti B&R Automation.*

[4] *Ethernet POWERLINK Standardization Group [online] [cit. 17-5-2014]*
`http://www.ethernet-powerlink.org/`

[5] *Maslikiewicz, O.:* **Badmintonový stroj**
Individuální projekt, K13135 ČVUT v Praze FEL, 2014

[6] *Kirsch, P.:* **Individuální projekt**
K13135 ČVUT v Praze FEL, 2013

A Tabulky

Tabulka 1: Měření doletu míčku bez pogumovaných kotoučů

Měření doletu bez pogumování								
n [ot/min]	150	300	450	600	1200	1800	3000	6000
CM	l [m]	l [m]	l [m]	l [m]	l [m]	l [m]	l [m]	l [m]
1	0,53	1,49	1,89	2,45	3,90	5,24	6,92	7,34
2	0,88	1,45	1,92	2,55	3,98	5,30	6,94	8,04
3	0,88	1,53	1,90	2,38	4,08	5,42	7,42	7,94
4	0,94	1,28	1,79	2,45	3,99	5,58	7,52	7,84
5	1,00	1,30	1,85	2,59	4,14	5,60	7,42	7,79
6	0,79	1,44	2,02	2,58	4,04	5,21	7,74	8,64
7	0,89	1,52	1,94	2,53	3,99	5,42	7,59	7,94
8	0,79	1,44	2,02	2,63	3,86	5,65	7,87	8,14
9	0,79	1,49	1,99	2,58	3,86	5,43	7,74	7,84
10	0,94	1,47	1,92	2,64	4,24	5,59	7,88	8,31
11	0,69	1,28	1,94	2,47	4,02	5,33	7,36	7,89
12	0,77	1,54	1,74	2,52	4,24	5,42	7,55	7,99
13	0,78	1,24	2,02	2,65	4,04	5,49	8,04	8,06
14	0,86	1,41	1,82	2,64	4,24	5,66	7,94	8,32
15	0,92	1,35	2,04	2,53	4,14	5,64	8,04	8,34
16	0,64	1,41	1,74	2,16	3,87	5,18	7,59	8,39
17	0,64	1,51	1,87	2,28	3,93	4,74	7,62	7,89
18	0,75	1,24	1,89	2,34	4,24	5,39	7,77	8,04
19	0,82	1,32	1,90	2,40	4,18	5,47	7,73	8,34
20	0,83	1,32	2,03	2,56	4,04	5,67	7,83	8,14
průměr	0,81	1,40	1,91	2,50	4,05	5,42	7,63	8,06

B Obsah přiloženého CD

- Text páce v PDF
- Celý projekt ve vývojovém prostředí AutomationStudio
  - Zdrojové kódy
  - Vizualizace

C Zdrojové kódy

- Basic - *Ovládání hlavního motoru*
- Gear - *Ovládání převodovky a druhého motoru*
- Cam - *Ovládání motoru pro podavač míčků, vačková hřídel*
- KontrolAut - *Ovládání celého stroje v uživatelském módu, současné automatické programy*
- Kontrol - *Ovládání stroje v servisním módu*

```
(*********************************************************************
 * COPYRIGHT -- Bernecker + Rainer
 *********************************************************************
 * PROGRAM: Basic
 * File: basicCyclic.st
 * Author: Bernecker + Rainer and OM
 * Created: December 01, 2009; 11.11.2013
 *********************************************************************
 * Implementation of Program Basic
 *********************************************************************)
PROGRAM _CYCLIC

(*************************************************************
      Control Sequence
*************************************************************)
(* status information is read before the step sequencer to attain a shorter reaction time *)
(************************ MC_READSTATUS ************************)
MC_ReadStatus_0.Enable := NOT(MC_ReadStatus_0.Error);
MC_ReadStatus_0.Axis := Axis1Obj;
MC_ReadStatus_0();
BasicControl.AxisState.Disabled            := MC_ReadStatus_0.Disabled;
BasicControl.AxisState.StandStill          := MC_ReadStatus_0.StandStill;
BasicControl.AxisState.Stopping            := MC_ReadStatus_0.Stopping;
BasicControl.AxisState.Homing              := MC_ReadStatus_0.Homing;
BasicControl.AxisState.DiscreteMotion      := MC_ReadStatus_0.DiscreteMotion;
BasicControl.AxisState.ContinuousMotion    := MC_ReadStatus_0.ContinuousMotion;
BasicControl.AxisState.SynchronizedMotion  := MC_ReadStatus_0.SynchronizedMotion;
BasicControl.AxisState.ErrorStop           := MC_ReadStatus_0.Errorstop;

(********************MC_BR_READDRIVESTATUS************************)
MC_BR_ReadDriveStatus_0.Enable := NOT(MC_BR_ReadDriveStatus_0.Error);
MC_BR_ReadDriveStatus_0.Axis := Axis1Obj;
MC_BR_ReadDriveStatus_0.AdrDriveStatus := ADR(BasicControl.Status.DriveStatus);
MC_BR_ReadDriveStatus_0();

(******************** MC_READACTUALPOSITION **********************)
MC_ReadActualPosition_0.Enable := (NOT(MC_ReadActualPosition_0.Error));
MC_ReadActualPosition_0.Axis := Axis1Obj;
MC_ReadActualPosition_0();
IF(MC_ReadActualPosition_0.Valid = TRUE)THEN
   BasicControl.Status.ActPosition := MC_ReadActualPosition_0.Position;
END_IF

(******************** MC_READACTUALVELOCITY **********************)
MC_ReadActualVelocity_0.Enable := (NOT(MC_ReadActualVelocity_0.Error));
MC_ReadActualVelocity_0.Axis := Axis1Obj;
MC_ReadActualVelocity_0();
IF(MC_ReadActualVelocity_0.Valid = TRUE)THEN
   BasicControl.Status.ActVelocity := MC_ReadActualVelocity_0.Velocity;
END_IF

(************************ MC_READAXISERROR **********************)
MC_ReadAxisError_0.Enable := NOT(MC_ReadAxisError_0.Error);
MC_ReadAxisError_0.Axis := Axis1Obj;
MC_ReadAxisError_0.DataAddress := ADR(BasicControl.Status.ErrorText);
MC_ReadAxisError_0.DataLength := SIZEOF(BasicControl.Status.ErrorText);
MC_ReadAxisError_0.DataObjectName := 'acp10etxen';
MC_ReadAxisError_0();
```

```
(* central monitoring OF stop command attains a shorter reaction TIME in CASE OF emergency
stop *)
(******************CHECK FOR STOP COMMAND************************)
   IF (BasicControl.Command.Stop = TRUE) THEN
      IF ((AxisStep >= STATE_READY) AND (AxisStep < STATE_ERROR)) THEN
         (* reset all FB execute inputs we use *)
         MC_Home_0.Execute := 0;
         MC_Stop_0.Execute := 0;
         MC_MoveAbsolute_0.Execute := 0;
         MC_MoveAdditive_0.Execute := 0;
         MC_MoveVelocity_0.Execute := 0;
         MC_ReadAxisError_0.Acknowledge := 0;
         MC_Reset_0.Execute := 0;

         (* reset user commands *)
         BasicControl.Command.Stop := 0;
         BasicControl.Command.Home := 0;
         BasicControl.Command.MoveJogPos := 0;
         BasicControl.Command.MoveJogNeg := 0;
         BasicControl.Command.MoveVelocity := 0;
         BasicControl.Command.MoveAbsolute := 0;
         BasicControl.Command.MoveAdditive := 0;
         AxisStep := STATE_STOP;
      END_IF
   END_IF

(***************** CHECK FOR GENERAL AXIS ERROR ******************)
          IF ((MC_ReadAxisError_0.AxisErrorID <> 0) AND (MC_ReadAxisError_0.Valid =
TRUE)) THEN
                    Mas_err := TRUE;
      AxisStep := STATE_ERROR_AXIS;
(***************** CHECK IF POWER SHOULD BE OFF ******************)
   ELSIF ((BasicControl.Command.Power = FALSE) AND (MC_ReadAxisError_0.Valid =
TRUE)) THEN
      IF ((MC_ReadStatus_0.Errorstop = TRUE) AND (MC_ReadStatus_0.Valid = TRUE)) THEN
         AxisStep := STATE_ERROR_RESET;
      ELSE
         AxisStep := STATE_WAIT;
      END_IF
   END_IF

CASE AxisStep OF

(******************** WAIT *************************)
   STATE_WAIT:  (* STATE: Wait *)
          (*        BasicControl.Command.Power := Power;          *)
      IF (BasicControl.Command.Power = TRUE) THEN
         AxisStep := STATE_POWER_ON;
      ELSE
         MC_Power_0.Enable := FALSE;
      END_IF

      (* reset all FB execute inputs we use *)
      MC_Home_0.Execute := FALSE;
      MC_Stop_0.Execute := FALSE;
      MC_MoveAbsolute_0.Execute := FALSE;
      MC_MoveAdditive_0.Execute := FALSE;
```

```
      MC_MoveVelocity_0.Execute := FALSE;
      MC_ReadAxisError_0.Acknowledge := FALSE;
      MC_Reset_0.Execute := FALSE;

      (* reset user commands *)
      BasicControl.Command.Stop := FALSE;
      BasicControl.Command.Home := FALSE;
      BasicControl.Command.MoveJogPos := FALSE;
      BasicControl.Command.MoveJogNeg := FALSE;
      BasicControl.Command.MoveVelocity := FALSE;
      BasicControl.Command.MoveAbsolute := FALSE;
      BasicControl.Command.MoveAdditive := FALSE;

      BasicControl.Status.ErrorID := 0;

(******************** POWER ON **********************)
   STATE_POWER_ON:  (* STATE: Power on *)
      MC_Power_0.Enable := TRUE;
      IF (MC_Power_0.Status = TRUE) THEN
                              AxisStep := STATE_READY;

      END_IF
      (* if a power error occured go to error state *)
      IF (MC_Power_0.Error = TRUE) THEN
         BasicControl.Status.ErrorID := MC_Power_0.ErrorID;
         AxisStep := STATE_ERROR;
      END_IF

(******************** READY **********************)
   STATE_READY:  (* STATE: Waiting for commands *)
      Mas_ON := TRUE;
          //vizualizace motor je redy
                    IF (BasicControl.Command.Home = TRUE)THEN
         BasicControl.Command.Home := FALSE;
         AxisStep := STATE_HOME;

      ELSIF (BasicControl.Command.Stop = TRUE) THEN
         AxisStep := STATE_STOP;

      ELSIF (BasicControl.Command.MoveJogPos = TRUE) THEN
         AxisStep := STATE_JOG_POSITIVE;

      ELSIF (BasicControl.Command.MoveJogNeg = TRUE) THEN
         AxisStep := STATE_JOG_NEGATIVE;

      ELSIF (BasicControl.Command.MoveAbsolute = TRUE) THEN
         BasicControl.Command.MoveAbsolute := FALSE;
         AxisStep := STATE_MOVE_ABSOLUTE;

      ELSIF (BasicControl.Command.MoveAdditive = TRUE) THEN
         BasicControl.Command.MoveAdditive := FALSE;
         AxisStep := STATE_MOVE_ADDITIVE;

      ELSIF (BasicControl.Command.MoveVelocity = TRUE) THEN
         BasicControl.Command.MoveVelocity := FALSE;
         AxisStep := STATE_MOVE_VELOCITY;

      ELSIF (BasicControl.Command.Halt = TRUE) THEN
```

```
        BasicControl.Command.Halt := FALSE;
        AxisStep := STATE_HALT;
    END_IF

(******************** HOME **********************)
  STATE_HOME:  (* STATE: start homing process *)
    MC_Home_0.Position := BasicControl.Parameter.HomePosition;
    MC_Home_0.HomingMode := BasicControl.Parameter.HomeMode;
    MC_Home_0.Execute := TRUE;
    IF (MC_Home_0.Done = TRUE) THEN
      MC_Home_0.Execute := FALSE;
      AxisStep := STATE_READY;
    END_IF
    (* if a homing error occured go to error state *)
    IF (MC_Home_0.Error = TRUE) THEN
      MC_Home_0.Execute := FALSE;
      BasicControl.Status.ErrorID := MC_Home_0.ErrorID;
      AxisStep := STATE_ERROR;
    END_IF

(************************HALT_MOVEMENT**************************)
  STATE_HALT:   (* STATE: Halt movement *)
    MC_Halt_0.Deceleration := BasicControl.Parameter.Deceleration;
    MC_Halt_0.Execute := TRUE;
    IF (MC_Halt_0.Done = TRUE) THEN
      MC_Halt_0.Execute := FALSE;
      AxisStep := STATE_READY;
    END_IF
    (* check if error occured *)
    IF (MC_Halt_0.Error = TRUE) THEN
      BasicControl.Status.ErrorID := MC_Halt_0.ErrorID;
      MC_Halt_0.Execute := FALSE;
      AxisStep := STATE_ERROR;
    END_IF

(************************ STOP MOVEMENT *************************)
  STATE_STOP:  (* STATE: Stop movement *)
    MC_Stop_0.Deceleration := BasicControl.Parameter.Deceleration;
    MC_Stop_0.Execute := TRUE;
    (* if axis is stopped go to ready state *)
    IF ((MC_Stop_0.Done = TRUE) AND (BasicControl.Command.Stop = FALSE)) THEN
      MC_Stop_0.Execute := FALSE;
                    AxisStep := STATE_READY;
    END_IF
    (* check if error occured *)
    IF (MC_Stop_0.Error = TRUE) THEN
      BasicControl.Status.ErrorID := MC_Stop_0.ErrorID;
      MC_Stop_0.Execute := FALSE;
      AxisStep := STATE_ERROR;
    END_IF

(******************** START JOG MOVEMENT POSITVE **********************)
   STATE_JOG_POSITIVE:  (* STATE: Start jog movement in positive direction *)
    MC_MoveVelocity_0.Velocity      := BasicControl.Parameter.JogVelocity;
    MC_MoveVelocity_0.Acceleration  := BasicControl.Parameter.Acceleration;
    MC_MoveVelocity_0.Deceleration  := BasicControl.Parameter.Deceleration;
    MC_MoveVelocity_0.Direction     := mcPOSITIVE_DIR;
    MC_MoveVelocity_0.Execute := TRUE;
```

```
      IF (BasicControl.Command.MoveJogPos = FALSE) THEN
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_HALT;
      END_IF
      (* check if error occured *)
      IF (MC_MoveVelocity_0.Error = TRUE) THEN
         BasicControl.Status.ErrorID := MC_MoveVelocity_0.ErrorID;
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_ERROR;
      END_IF

(******************** START JOG MOVEMENT NEGATIVE **********************)
   STATE_JOG_NEGATIVE:  (* STATE: Start jog movement in negative direction *)
      MC_MoveVelocity_0.Velocity      := BasicControl.Parameter.JogVelocity;
      MC_MoveVelocity_0.Acceleration  := BasicControl.Parameter.Acceleration;
      MC_MoveVelocity_0.Deceleration  := BasicControl.Parameter.Deceleration;
      MC_MoveVelocity_0.Direction           := mcNEGATIVE_DIR;
      MC_MoveVelocity_0.Execute := TRUE;
      IF (BasicControl.Command.MoveJogNeg = FALSE) THEN
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_HALT;
      END_IF
      (* check if error occured *)
      IF (MC_MoveVelocity_0.Error = TRUE) THEN
         BasicControl.Status.ErrorID := MC_MoveVelocity_0.ErrorID;
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_ERROR;
      END_IF

(******************** START ABSOLUTE MOVEMENT **********************)
    STATE_MOVE_ABSOLUTE:  (* STATE: Start absolute movement *)
      MC_MoveAbsolute_0.Position      := BasicControl.Parameter.Position;
      MC_MoveAbsolute_0.Velocity      := BasicControl.Parameter.Velocity;
      MC_MoveAbsolute_0.Acceleration  := BasicControl.Parameter.Acceleration;
      MC_MoveAbsolute_0.Deceleration  := BasicControl.Parameter.Deceleration;
      MC_MoveAbsolute_0.Direction     := BasicControl.Parameter.Direction;
      MC_MoveAbsolute_0.Execute := TRUE;
      (* check if commanded position is reached *)
      IF (BasicControl.Command.Halt) THEN
         BasicControl.Command.Halt := FALSE;
         MC_MoveAbsolute_0.Execute := FALSE;
         AxisStep := STATE_HALT;
      ELSIF (MC_MoveAbsolute_0.Done = TRUE) THEN
         MC_MoveAbsolute_0.Execute := FALSE;
         AxisStep := STATE_READY;
      END_IF
      (* check if error occured *)
      IF (MC_MoveAbsolute_0.Error = TRUE) THEN
         BasicControl.Status.ErrorID := MC_MoveAbsolute_0.ErrorID;
         MC_MoveAbsolute_0.Execute := TRUE;
         AxisStep := STATE_ERROR;
      END_IF

(******************** START ADDITIVE MOVEMENT **********************)
   STATE_MOVE_ADDITIVE:  (* STATE: Start additive movement *)
      MC_MoveAdditive_0.Distance      := BasicControl.Parameter.Distance;
      MC_MoveAdditive_0.Velocity      := BasicControl.Parameter.Velocity;
      MC_MoveAdditive_0.Acceleration  := BasicControl.Parameter.Acceleration;
```

```
      MC_MoveAdditive_0.Deceleration  := BasicControl.Parameter.Deceleration;
      MC_MoveAdditive_0.Execute := TRUE;
      (* check if commanded distance is reached *)
      IF (BasicControl.Command.Halt) THEN
         BasicControl.Command.Halt := FALSE;
         MC_MoveAdditive_0.Execute := FALSE;
         AxisStep := STATE_HALT;
      ELSIF (MC_MoveAdditive_0.Done = TRUE) THEN
         MC_MoveAdditive_0.Execute := FALSE;
         AxisStep := STATE_READY;
      END_IF
      (* check if error occured *)
      IF (MC_MoveAdditive_0.Error = TRUE) THEN
         BasicControl.Status.ErrorID := MC_MoveAdditive_0.ErrorID;
         MC_MoveAdditive_0.Execute := FALSE;
         AxisStep := STATE_ERROR;
      END_IF

(******************** START VELOCITY MOVEMENT **********************)
   STATE_MOVE_VELOCITY:  (* STATE: Start velocity movement *)
      MC_MoveVelocity_0.Velocity      := BasicControl.Parameter.Velocity;
      MC_MoveVelocity_0.Acceleration  := BasicControl.Parameter.Acceleration;
      MC_MoveVelocity_0.Deceleration  := BasicControl.Parameter.Deceleration;
      MC_MoveVelocity_0.Direction     := BasicControl.Parameter.Direction;
      MC_MoveVelocity_0.Execute := TRUE;
      (* check if commanded velocity is reached *)
      IF (BasicControl.Command.Halt) THEN
         BasicControl.Command.Halt := FALSE;
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_HALT;
      ELSIF (MC_MoveVelocity_0.InVelocity = TRUE) THEN
                              MC_MoveVelocity_0.Execute := FALSE;
                              Mas_run := TRUE;
          //zapnuti vizualizace
         AxisStep := STATE_READY;
      END_IF
      (* check if error occured *)
      IF (MC_MoveVelocity_0.Error = TRUE) THEN
         BasicControl.Status.ErrorID := MC_MoveVelocity_0.ErrorID;
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_ERROR;
      END_IF

(******************** FB-ERROR OCCURED *************************)
   STATE_ERROR:  (* STATE: Error *)
      (* check if FB indicates an axis error *)
                  IF (MC_ReadAxisError_0.AxisErrorCount<>0) THEN
                              Mas_err := TRUE;
         AxisStep := STATE_ERROR_AXIS;
      ELSE
         IF (BasicControl.Command.ErrorAcknowledge = TRUE) THEN
            BasicControl.Command.ErrorAcknowledge := FALSE;
            BasicControl.Status.ErrorID := 0;
            (* reset axis if it is in axis state ErrorStop *)
            IF ((MC_ReadStatus_0.Errorstop = TRUE) AND (MC_ReadStatus_0.Valid = TRUE))
THEN
               AxisStep := STATE_ERROR_RESET;
            ELSE
```

```
            AxisStep := STATE_WAIT;
         END_IF
      END_IF
   END_IF

(******************** AXIS-ERROR OCCURED *************************)
   STATE_ERROR_AXIS:  (* STATE: Axis Error *)
      IF (MC_ReadAxisError_0.Valid = TRUE) THEN
                              IF (MC_ReadAxisError_0.AxisErrorID <> 0) THEN
                                    Mas_err := TRUE;
         BasicControl.Status.ErrorID := MC_ReadAxisError_0.AxisErrorID;
      END_IF
      MC_ReadAxisError_0.Acknowledge := FALSE;
      IF (BasicControl.Command.ErrorAcknowledge = TRUE) THEN
         BasicControl.Command.ErrorAcknowledge := FALSE;
         (* acknowledge axis error *)
         IF (MC_ReadAxisError_0.AxisErrorID <> 0) THEN
            MC_ReadAxisError_0.Acknowledge := TRUE;
         END_IF
      END_IF
      IF (MC_ReadAxisError_0.AxisErrorCount = 0) THEN
         (* reset axis if it is in axis state ErrorStop *)
         BasicControl.Status.ErrorID := 0;
         IF ((MC_ReadStatus_0.Errorstop = TRUE) AND (MC_ReadStatus_0.Valid = TRUE))
THEN
            AxisStep := STATE_ERROR_RESET;
         ELSE
            AxisStep := STATE_WAIT;
         END_IF
      END_IF
   END_IF

(******************** RESET DONE *************************)
   STATE_ERROR_RESET:  (* STATE: Wait for reset done *)
      MC_Reset_0.Execute := TRUE;
      (* reset MC_Power.Enable if this FB is in Error*)
      IF (MC_Power_0.Error = TRUE) THEN
         MC_Power_0.Enable := FALSE;
      END_IF
      IF(MC_Reset_0.Done = TRUE)THEN
         MC_Reset_0.Execute := FALSE;
         AxisStep := STATE_WAIT;
      ELSIF(MC_Reset_0.Error = TRUE) THEN
         MC_Reset_0.Execute := FALSE;
         AxisStep := STATE_ERROR;
      END_IF
(******************** SEQUENCE END *************************)
END_CASE

(*****************************************************************
      Function Block Calls
*****************************************************************)

(************************** MC_POWER ****************************)
MC_Power_0.Axis := Axis1Obj;  (* pointer to axis *)
MC_Power_0();

(************************** MC_HOME *****************************)
```

```
MC_Home_0.Axis := Axis1Obj;
MC_Home_0();

(********************** MC_MOVEABSOLUTE ************************)
MC_MoveAbsolute_0.Axis := Axis1Obj;
MC_MoveAbsolute_0();

(********************** MC_MOVEADDITIVE ************************)
MC_MoveAdditive_0.Axis := Axis1Obj;
MC_MoveAdditive_0();

(********************** MC_MOVEVELOCITY ************************)
MC_MoveVelocity_0.Axis := Axis1Obj;
MC_MoveVelocity_0();

(************************** MC_STOP ******************************)
MC_Stop_0.Axis := Axis1Obj;
MC_Stop_0();

(**************************MC_HALT*******************************)
MC_Halt_0.Axis := Axis1Obj;
MC_Halt_0();

(************************** MC_RESET *****************************)
MC_Reset_0.Axis := Axis1Obj;
MC_Reset_0();

    END_PROGRAM
```

```
(*********************************************************************
 * COPYRIGHT -- Bernecker + Rainer
 *********************************************************************
 * PROGRAM: Gear
 * File: gearCyclic.st
 * Author: Bernecker + Rainer and OM
 * Created: December 01, 2009; 10.12.2013
 *********************************************************************
 * Implementation of Program Gear
 *********************************************************************)
PROGRAM _CYCLIC

(*************************************************************
      Control Sequence
*************************************************************)

(* status information is read before the step sequencer to attain a shorter reaction time *)
(************************ MC_READSTATUS *************************)
MC_ReadStatus_0.Enable := NOT(MC_ReadStatus_0.Error);
MC_ReadStatus_0.Axis := Axis2Obj;
MC_ReadStatus_0();
GearControl.AxisState.Disabled            := MC_ReadStatus_0.Disabled;
GearControl.AxisState.StandStill          := MC_ReadStatus_0.StandStill;
GearControl.AxisState.Stopping            := MC_ReadStatus_0.Stopping;
GearControl.AxisState.Homing              := MC_ReadStatus_0.Homing;
GearControl.AxisState.DiscreteMotion      := MC_ReadStatus_0.DiscreteMotion;
GearControl.AxisState.ContinuousMotion    := MC_ReadStatus_0.ContinuousMotion;
GearControl.AxisState.SynchronizedMotion  := MC_ReadStatus_0.SynchronizedMotion;
GearControl.AxisState.ErrorStop           := MC_ReadStatus_0.Errorstop;

(********************MC_BR_READDRIVESTATUS***********************)
MC_BR_ReadDriveStatus_0.Enable := NOT(MC_BR_ReadDriveStatus_0.Error);
MC_BR_ReadDriveStatus_0.Axis := Axis2Obj;
MC_BR_ReadDriveStatus_0.AdrDriveStatus := ADR(GearControl.Status.DriveStatus);
MC_BR_ReadDriveStatus_0();

(******************** MC_READACTUALPOSITION **********************)
MC_ReadActualPosition_0.Enable := (NOT(MC_ReadActualPosition_0.Error));
MC_ReadActualPosition_0.Axis := Axis2Obj;
MC_ReadActualPosition_0();
IF(MC_ReadActualPosition_0.Valid = TRUE)THEN
   GearControl.Status.ActPosition := MC_ReadActualPosition_0.Position;
END_IF

(******************** MC_READACTUALVELOCITY **********************)
MC_ReadActualVelocity_0.Enable := (NOT(MC_ReadActualVelocity_0.Error));
MC_ReadActualVelocity_0.Axis := Axis2Obj;
MC_ReadActualVelocity_0();
IF(MC_ReadActualVelocity_0.Valid = TRUE)THEN
   GearControl.Status.ActVelocity := MC_ReadActualVelocity_0.Velocity;
END_IF

(************************ MC_READAXISERROR **********************)
MC_ReadAxisError_0.Enable := NOT(MC_ReadAxisError_0.Error);
MC_ReadAxisError_0.Axis := Axis2Obj;
MC_ReadAxisError_0.DataAddress := ADR(GearControl.Status.ErrorText);
MC_ReadAxisError_0.DataLength := SIZEOF(GearControl.Status.ErrorText);
MC_ReadAxisError_0.DataObjectName := 'acp10etxen';
```

```
MC_ReadAxisError_0();

(* central monitoring OF stop command attains a shorter reaction TIME in CASE OF emergency
stop *)
(******************CHECK FOR STOP COMMAND***********************)
   IF (GearControl.Command.Stop = TRUE) THEN
      IF ((AxisStep >= STATE_READY) AND (AxisStep < STATE_ERROR)) THEN
         (* reset all fb execute inputs we use *)
         MC_Home_0.Execute := FALSE;
         MC_Stop_0.Execute := FALSE;
         MC_MoveAbsolute_0.Execute := FALSE;
         MC_MoveAdditive_0.Execute := FALSE;
         MC_MoveVelocity_0.Execute := FALSE;
         MC_GearIn_0.Execute := FALSE;
         MC_GearOut_0.Execute := FALSE;
         MC_ReadAxisError_0.Acknowledge := FALSE;
         MC_Reset_0.Execute := FALSE;

         (* reset user commands *)
         GearControl.Command.Stop := FALSE;
         GearControl.Command.Home := FALSE;
         GearControl.Command.MoveJogPos := FALSE;
         GearControl.Command.MoveJogNeg := FALSE;
         GearControl.Command.MoveVelocity := FALSE;
         GearControl.Command.MoveAbsolute := FALSE;
         GearControl.Command.MoveAdditive := FALSE;
         GearControl.Command.DisengageSlave := FALSE;
         GearControl.Command.StartSlave := FALSE;

         AxisStep := STATE_STOP;
      END_IF
   END_IF


(**************** CHECK FOR GENERAL AXIS ERROR ******************)
   IF ((MC_ReadAxisError_0.AxisErrorID <> 0) AND (MC_ReadAxisError_0.Valid = TRUE))
THEN
      AxisStep := STATE_ERROR_AXIS;
(***************** CHECK IF POWER SHOULD BE OFF ******************)
   ELSIF ((GearControl.Command.Power = FALSE) AND (MC_ReadAxisError_0.Valid = TRUE))
THEN
      IF ((MC_ReadStatus_0.Errorstop = TRUE) AND (MC_ReadStatus_0.Valid = TRUE)) THEN
         AxisStep := STATE_ERROR_RESET;
      ELSE
         AxisStep := STATE_WAIT;
      END_IF
   END_IF

CASE AxisStep OF

(******************** WAIT ************************)
          STATE_WAIT:  (* STATE: Wait *)
          (*         GearControl.Command.Power := Power;           *)
          IF (GearControl.Command.Power = TRUE) THEN
         AxisStep := STATE_POWER_ON;
      ELSE
         MC_Power_0.Enable := FALSE;
      END_IF
```

```
      (* reset all fb execute inputs we use *)
      MC_Home_0.Execute := FALSE;
      MC_Stop_0.Execute := FALSE;
      MC_MoveAbsolute_0.Execute := FALSE;
      MC_MoveAdditive_0.Execute := FALSE;
      MC_MoveVelocity_0.Execute := FALSE;
      MC_GearIn_0.Execute := FALSE;
      MC_GearOut_0.Execute := FALSE;
      MC_ReadAxisError_0.Acknowledge := FALSE;
      MC_Reset_0.Execute := FALSE;

      (* reset user commands *)
      GearControl.Command.Stop := FALSE;
      GearControl.Command.Home := FALSE;
      GearControl.Command.MoveJogPos := FALSE;
      GearControl.Command.MoveJogNeg := FALSE;
      GearControl.Command.MoveVelocity := FALSE;
      GearControl.Command.MoveAbsolute := FALSE;
      GearControl.Command.MoveAdditive := FALSE;
      GearControl.Command.DisengageSlave := FALSE;
      GearControl.Command.StartSlave := FALSE;

      GearControl.Status.ErrorID := 0;

(******************** POWER ON **********************)
   STATE_POWER_ON:  (* STATE: Power on *)
      MC_Power_0.Enable := TRUE;
      IF (MC_Power_0.Status = TRUE) THEN
         AxisStep := STATE_READY;
      END_IF
      (* if a power error occured go to error state *)
      IF (MC_Power_0.Error = TRUE) THEN
         GearControl.Status.ErrorID := MC_Power_0.ErrorID;
         AxisStep := STATE_ERROR;
      END_IF

(******************** READY **********************)
   STATE_READY:  (* STATE: Waiting for commands *)
      IF (GearControl.Command.Home = TRUE)THEN
         GearControl.Command.Home := FALSE;
         AxisStep := STATE_HOME;

      ELSIF (GearControl.Command.Stop = TRUE) THEN
         AxisStep := STATE_STOP;

      ELSIF (GearControl.Command.MoveJogPos = TRUE) THEN
         AxisStep := STATE_JOG_POSITIVE;

      ELSIF (GearControl.Command.MoveJogNeg = TRUE) THEN
         AxisStep := STATE_JOG_NEGATIVE;

      ELSIF (GearControl.Command.MoveAbsolute = TRUE) THEN
         GearControl.Command.MoveAbsolute := FALSE;
         AxisStep := STATE_MOVE_ABSOLUTE;

      ELSIF (GearControl.Command.MoveAdditive = TRUE) THEN
         GearControl.Command.MoveAdditive := FALSE;
```

```
        AxisStep := STATE_MOVE_ADDITIVE;

      ELSIF (GearControl.Command.MoveVelocity = TRUE) THEN
        GearControl.Command.MoveVelocity := FALSE;
        AxisStep := STATE_MOVE_VELOCITY;

      ELSIF (GearControl.Command.StartSlave = TRUE) THEN
        GearControl.Command.StartSlave := FALSE;
        AxisStep := STATE_GEAR_START;

      ELSIF (GearControl.Command.Halt = TRUE) THEN
        GearControl.Command.Halt := FALSE;
        AxisStep := STATE_HALT;
      END_IF

(******************** HOME **********************)
   STATE_HOME: (* STATE: start homing process *)
      MC_Home_0.Position := GearControl.Parameter.HomePosition;
      MC_Home_0.HomingMode := GearControl.Parameter.HomeMode;
      MC_Home_0.Execute := TRUE;
      IF (MC_Home_0.Done = TRUE) THEN
        MC_Home_0.Execute := FALSE;
        AxisStep := STATE_READY;
      END_IF
      (* if a homing error occured go to error state *)
      IF (MC_Home_0.Error = TRUE) THEN
        MC_Home_0.Execute := FALSE;
        GearControl.Status.ErrorID := MC_Home_0.ErrorID;
        AxisStep := STATE_ERROR;
      END_IF

(***********************HALT_MOVEMENT**************************)
   STATE_HALT:   (* STATE: Halt movement *)
      MC_Halt_0.Deceleration := GearControl.Parameter.Deceleration;
      MC_Halt_0.Execute := TRUE;
      IF (MC_Halt_0.Done = TRUE) THEN
        MC_Halt_0.Execute := FALSE;
        AxisStep := STATE_READY;
      END_IF
      (* check if error occured *)
      IF (MC_Halt_0.Error = TRUE) THEN
        GearControl.Status.ErrorID := MC_Halt_0.ErrorID;
        MC_Halt_0.Execute := FALSE;
        AxisStep := STATE_ERROR;
      END_IF

(************************ STOP MOVEMENT *************************)
   STATE_STOP: (* STATE: Stop movement *)
      MC_Stop_0.Deceleration := GearControl.Parameter.Deceleration;
      MC_Stop_0.Execute := TRUE;
      (* if axis is stopped go to ready state *)
      IF ((MC_Stop_0.Done = TRUE) AND (GearControl.Command.Stop = FALSE)) THEN
         MC_Stop_0.Execute := FALSE;
         AxisStep := STATE_READY;
      END_IF
      (* check if error occured *)
      IF (MC_Stop_0.Error = TRUE) THEN
        GearControl.Status.ErrorID := MC_Stop_0.ErrorID;
```

```
        MC_Stop_0.Execute := FALSE;
        AxisStep := STATE_ERROR;
      END_IF


(******************** START JOG MOVEMENT POSITVE ***********************)
   STATE_JOG_POSITIVE: (* STATE: Start jog movement in positive direction *)
      MC_MoveVelocity_0.Velocity      := GearControl.Parameter.JogVelocity;
      MC_MoveVelocity_0.Acceleration  := GearControl.Parameter.Acceleration;
      MC_MoveVelocity_0.Deceleration  := GearControl.Parameter.Deceleration;
      MC_MoveVelocity_0.Direction     := mcPOSITIVE_DIR;
      MC_MoveVelocity_0.Execute := TRUE;
      IF (GearControl.Command.MoveJogPos = FALSE) THEN
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_HALT;
      END_IF
      (* check if error occured *)
      IF (MC_MoveVelocity_0.Error = TRUE) THEN
         GearControl.Status.ErrorID := MC_MoveVelocity_0.ErrorID;
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_ERROR;
      END_IF

(******************** START JOG MOVEMENT NEGATIVE ***********************)
   STATE_JOG_NEGATIVE: (* STATE: Start jog movement in negative direction *)
      MC_MoveVelocity_0.Velocity      := GearControl.Parameter.JogVelocity;
      MC_MoveVelocity_0.Acceleration  := GearControl.Parameter.Acceleration;
      MC_MoveVelocity_0.Deceleration  := GearControl.Parameter.Deceleration;
      MC_MoveVelocity_0.Direction     := mcNEGATIVE_DIR;
      MC_MoveVelocity_0.Execute := TRUE;
      IF (GearControl.Command.MoveJogNeg = FALSE) THEN
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_HALT;
      END_IF
      (* check if error occured *)
      IF (MC_MoveVelocity_0.Error = TRUE) THEN
         GearControl.Status.ErrorID := MC_MoveVelocity_0.ErrorID;
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_ERROR;
      END_IF

(******************** START ABSOLUTE MOVEMENT ***********************)
   STATE_MOVE_ABSOLUTE: (* STATE: Start absolute movement *)
      MC_MoveAbsolute_0.Position      := GearControl.Parameter.Position;
      MC_MoveAbsolute_0.Velocity      := GearControl.Parameter.Velocity;
      MC_MoveAbsolute_0.Acceleration  := GearControl.Parameter.Acceleration;
      MC_MoveAbsolute_0.Deceleration  := GearControl.Parameter.Deceleration;
      MC_MoveAbsolute_0.Direction     := GearControl.Parameter.Direction;
      MC_MoveAbsolute_0.Execute := TRUE;
      (* check if commanded position is reached *)
      IF (GearControl.Command.Halt) THEN
         GearControl.Command.Halt := FALSE;
         MC_MoveAbsolute_0.Execute := FALSE;
         AxisStep := STATE_HALT;
      ELSIF (MC_MoveAbsolute_0.Done = TRUE) THEN
         MC_MoveAbsolute_0.Execute := FALSE;
         AxisStep := STATE_READY;
      END_IF
```

```
      (* check if error occured *)
      IF (MC_MoveAbsolute_0.Error = TRUE) THEN
         GearControl.Status.ErrorID := MC_MoveAbsolute_0.ErrorID;
         MC_MoveAbsolute_0.Execute := FALSE;
         AxisStep := STATE_ERROR;
      END_IF

(********** START ADDITIVE MOVEMENT **********)
   STATE_MOVE_ADDITIVE: (* STATE: Start additive movement *)
      MC_MoveAdditive_0.Distance      := GearControl.Parameter.Distance;
      MC_MoveAdditive_0.Velocity      := GearControl.Parameter.Velocity;
      MC_MoveAdditive_0.Acceleration  := GearControl.Parameter.Acceleration;
      MC_MoveAdditive_0.Deceleration  := GearControl.Parameter.Deceleration;
      MC_MoveAdditive_0.Execute := TRUE;
      (* check if commanded distance is reached *)
      IF (GearControl.Command.Halt) THEN
         GearControl.Command.Halt := FALSE;
         MC_MoveAdditive_0.Execute := FALSE;
         AxisStep := STATE_HALT;
      ELSIF (MC_MoveAdditive_0.Done = TRUE) THEN
         MC_MoveAdditive_0.Execute := FALSE;
         AxisStep := STATE_READY;
      END_IF
      (* check if error occured *)
      IF (MC_MoveAdditive_0.Error = TRUE) THEN
         GearControl.Status.ErrorID := MC_MoveAdditive_0.ErrorID;
         MC_MoveAdditive_0.Execute := FALSE;
         AxisStep := STATE_ERROR;
      END_IF

(********** START VELOCITY MOVEMENT **********)
   STATE_MOVE_VELOCITY: (* STATE: Start velocity movement *)
      MC_MoveVelocity_0.Velocity      := GearControl.Parameter.Velocity;
      MC_MoveVelocity_0.Acceleration  := GearControl.Parameter.Acceleration;
      MC_MoveVelocity_0.Deceleration  := GearControl.Parameter.Deceleration;
      MC_MoveVelocity_0.Direction           := GearControl.Parameter.Direction;
      MC_MoveVelocity_0.Execute := TRUE;
      (* check if commanded velocity is reached *)
      IF (GearControl.Command.Halt) THEN
         GearControl.Command.Halt := FALSE;
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_HALT;
      ELSIF (MC_MoveVelocity_0.InVelocity = TRUE) THEN
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_READY;
      END_IF
      (* check if error occured *)
      IF (MC_MoveVelocity_0.Error = TRUE) THEN
         GearControl.Status.ErrorID := MC_MoveVelocity_0.ErrorID;
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_ERROR;
      END_IF

(********** START GEAR MOVEMENT **********)
   STATE_GEAR_START: (* STATE: Start electronic gear coupling *)
      MC_GearIn_0.RatioNumerator  := GearControl.Parameter.RatioNumerator;
      MC_GearIn_0.RatioDenominator:= GearControl.Parameter.RatioDenominator;
      MC_GearIn_0.Acceleration    := GearControl.Parameter.Acceleration;
```

```
      MC_GearIn_0.Deceleration    := GearControl.Parameter.Deceleration;
      MC_GearIn_0.Execute := TRUE;
      (* wait for gear stop *)
      IF (GearControl.Command.Halt) THEN
         GearControl.Command.Halt := FALSE;
         MC_GearIn_0.Execute := FALSE;
         AxisStep := STATE_HALT;
      ELSIF (GearControl.Command.DisengageSlave = TRUE) THEN
         GearControl.Command.DisengageSlave := FALSE;
         MC_GearIn_0.Execute := FALSE;
         AxisStep := STATE_GEAR_STOP;
      END_IF
      (* check if error occured *)
      IF (MC_GearIn_0.Error = TRUE) THEN
         GearControl.Status.ErrorID := MC_GearIn_0.ErrorID;
         MC_GearIn_0.Execute := FALSE;
         AxisStep := STATE_ERROR;
      END_IF

(********************* STOP GEAR MOVEMENT **********************)
   STATE_GEAR_STOP: (* STATE: Stop electronic gear coupling *)
      MC_GearOut_0.Execute := TRUE;
      (* check if coupling is stopped *)
      IF (MC_GearOut_0.Done = TRUE) THEN
         MC_GearOut_0.Execute := FALSE;
         AxisStep := STATE_READY;
      END_IF
      (* check if error occured *)
      IF (MC_GearOut_0.Error = TRUE) THEN
         GearControl.Status.ErrorID := MC_GearOut_0.ErrorID;
         MC_GearOut_0.Execute := FALSE;
         AxisStep := STATE_ERROR;
      END_IF

(********************* FB-ERROR OCCURED *************************)
   STATE_ERROR: (* STATE: Error *)
      (* check if FB indicates an axis error *)
      IF (MC_ReadAxisError_0.AxisErrorCount<>0) THEN
         AxisStep := STATE_ERROR_AXIS;
      ELSE
         IF (GearControl.Command.ErrorAcknowledge = TRUE) THEN
            GearControl.Command.ErrorAcknowledge := FALSE;
            GearControl.Status.ErrorID := 0;
            (* reset axis if it is in axis state ErrorStop *)
            IF ((MC_ReadStatus_0.Errorstop = TRUE) AND (MC_ReadStatus_0.Valid = TRUE))
THEN
               AxisStep := STATE_ERROR_RESET;
            ELSE
               AxisStep := STATE_WAIT;
            END_IF
         END_IF
      END_IF

(********************* AXIS-ERROR OCCURED *************************)
   STATE_ERROR_AXIS: (* STATE: Axis Error *)
      IF (MC_ReadAxisError_0.Valid = TRUE) THEN
         IF (MC_ReadAxisError_0.AxisErrorID <> 0) THEN
            GearControl.Status.ErrorID := MC_ReadAxisError_0.AxisErrorID;
```

```
        END_IF
        MC_ReadAxisError_0.Acknowledge := FALSE;
        IF (GearControl.Command.ErrorAcknowledge = TRUE) THEN
           GearControl.Command.ErrorAcknowledge := FALSE;
           (* acknowledge axis error *)
           IF (MC_ReadAxisError_0.AxisErrorID <> 0) THEN
              MC_ReadAxisError_0.Acknowledge := TRUE;
           END_IF
        END_IF
        IF (MC_ReadAxisError_0.AxisErrorCount = 0) THEN
           (* reset axis if it is in axis state ErrorStop *)
           GearControl.Status.ErrorID := 0;
           IF ((MC_ReadStatus_0.Errorstop = TRUE) AND (MC_ReadStatus_0.Valid = TRUE))
THEN
              AxisStep := STATE_ERROR_RESET;
           ELSE
              AxisStep := STATE_WAIT;
           END_IF
        END_IF
     END_IF

(******************** RESET DONE ************************)
   STATE_ERROR_RESET: (* STATE: Wait for reset done *)
      MC_Reset_0.Execute := TRUE;
      (* reset MC_Power.Enable if this FB is in Error*)
      IF (MC_Power_0.Error = TRUE) THEN
         MC_Power_0.Enable := FALSE;
      END_IF
      IF(MC_Reset_0.Done = TRUE)THEN
         MC_Reset_0.Execute := FALSE;
         AxisStep := STATE_WAIT;
      ELSIF(MC_Reset_0.Error = TRUE) THEN
         MC_Reset_0.Execute := FALSE;
         AxisStep := STATE_ERROR;
      END_IF
(******************** SEQUENCE END ************************)
END_CASE

(*************************************************************
      Function Block Calls
*************************************************************)

(************************** MC_POWER ***************************)
MC_Power_0.Axis := Axis2Obj; (* pointer to axis *)
MC_Power_0();

(************************** MC_HOME ****************************)
MC_Home_0.Axis := Axis2Obj;
MC_Home_0();

(********************** MC_MOVEABSOLUTE ************************)
MC_MoveAbsolute_0.Axis := Axis2Obj;
MC_MoveAbsolute_0();

(********************** MC_MOVEADDITIVE ************************)
MC_MoveAdditive_0.Axis := Axis2Obj;
MC_MoveAdditive_0();
```

```
(********************** MC_MOVEVELOCITY ************************)
MC_MoveVelocity_0.Axis := Axis2Obj;
MC_MoveVelocity_0();

(********************** MC_GEARIN ************************)
MC_GearIn_0.Master := Axis1Obj;
MC_GearIn_0.Slave := Axis2Obj;
MC_GearIn_0();

(********************** MC_GEAROUT ************************)
MC_GearOut_0.Slave := Axis2Obj;
MC_GearOut_0();

(************************** MC_STOP ******************************)
MC_Stop_0.Axis := Axis2Obj;
MC_Stop_0();

(**************************MC_HALT*******************************)
MC_Halt_0.Axis := Axis2Obj;
MC_Halt_0();

(************************** MC_RESET *****************************)
MC_Reset_0.Axis := Axis2Obj;
MC_Reset_0();

END_PROGRAM
```

```
(*********************************************************************
 * COPYRIGHT -- Bernecker + Rainer
 *********************************************************************
 * PROGRAM: Cam
 * File: camCyclic.st
 * Author: Bernecker + Rainer and OM
 * Created: December 01, 2009; 28.3.2014
 *********************************************************************
 * Implementation of Program Cam
 *********************************************************************)
PROGRAM _CYCLIC

(*************************************************************
      Control Sequence
*************************************************************)

(* status information is read before the step sequencer to attain a shorter reaction time *)
(************************ MC_READSTATUS ************************)
MC_ReadStatus_0.Enable := NOT(MC_ReadStatus_0.Error);
MC_ReadStatus_0.Axis := Axis2Obj;
MC_ReadStatus_0();
CamControl.AxisState.Disabled            := MC_ReadStatus_0.Disabled;
CamControl.AxisState.StandStill          := MC_ReadStatus_0.StandStill;
CamControl.AxisState.Stopping            := MC_ReadStatus_0.Stopping;
CamControl.AxisState.Homing              := MC_ReadStatus_0.Homing;
CamControl.AxisState.DiscreteMotion      := MC_ReadStatus_0.DiscreteMotion;
CamControl.AxisState.ContinuousMotion    := MC_ReadStatus_0.ContinuousMotion;
CamControl.AxisState.SynchronizedMotion  := MC_ReadStatus_0.SynchronizedMotion;
CamControl.AxisState.ErrorStop           := MC_ReadStatus_0.Errorstop;

(********************MC_BR_READDRIVESTATUS**********************)
MC_BR_ReadDriveStatus_0.Enable := NOT(MC_BR_ReadDriveStatus_0.Error);
MC_BR_ReadDriveStatus_0.Axis := Axis2Obj;
MC_BR_ReadDriveStatus_0.AdrDriveStatus := ADR(CamControl.Status.DriveStatus);
MC_BR_ReadDriveStatus_0();

(******************** MC_READACTUALPOSITION **********************)
MC_ReadActualPosition_0.Enable := (NOT(MC_ReadActualPosition_0.Error));
MC_ReadActualPosition_0.Axis := Axis2Obj;
MC_ReadActualPosition_0();
IF(MC_ReadActualPosition_0.Valid = TRUE)THEN
   CamControl.Status.ActPosition := MC_ReadActualPosition_0.Position;
END_IF

(******************** MC_READACTUALVELOCITY **********************)
MC_ReadActualVelocity_0.Enable := (NOT(MC_ReadActualVelocity_0.Error));
MC_ReadActualVelocity_0.Axis := Axis2Obj;
MC_ReadActualVelocity_0();
IF(MC_ReadActualVelocity_0.Valid = TRUE)THEN
   CamControl.Status.ActVelocity := MC_ReadActualVelocity_0.Velocity;
END_IF

(************************ MC_READAXISERROR **********************)
MC_ReadAxisError_0.Enable := NOT(MC_ReadAxisError_0.Error);
MC_ReadAxisError_0.Axis := Axis2Obj;
MC_ReadAxisError_0.DataAddress := ADR(CamControl.Status.ErrorText);
MC_ReadAxisError_0.DataLength := SIZEOF(CamControl.Status.ErrorText);
MC_ReadAxisError_0.DataObjectName := 'acp10etxen';
```

```
MC_ReadAxisError_0();

(* central monitoring OF stop command attains a shorter reaction TIME in CASE OF emergency
stop *)
(******************CHECK FOR STOP COMMAND***********************)
   IF (CamControl.Command.Stop = TRUE) THEN
      IF ((AxisStep >= STATE_READY) AND (AxisStep < STATE_ERROR)) THEN
         (* reset all fb execute inputs we use *)
         MC_Home_0.Execute := FALSE;
         MC_Stop_0.Execute := FALSE;
         MC_MoveAbsolute_0.Execute := FALSE;
         MC_MoveAdditive_0.Execute := FALSE;
         MC_MoveVelocity_0.Execute := FALSE;
         MC_CamTableSelect_0.Execute := FALSE;
         MC_CamIn_0.Execute := FALSE;
         MC_CamOut_0.Execute := FALSE;
         MC_ReadAxisError_0.Acknowledge := FALSE;
         MC_Reset_0.Execute := FALSE;

         (* reset user commands *)
         CamControl.Command.Stop := FALSE;
         CamControl.Command.Home := FALSE;
         CamControl.Command.MoveJogPos := FALSE;
         CamControl.Command.MoveJogNeg := FALSE;
         CamControl.Command.MoveVelocity := FALSE;
         CamControl.Command.MoveAbsolute := FALSE;
         CamControl.Command.MoveAdditive := FALSE;
         CamControl.Command.DisengageSlave := FALSE;
         CamControl.Command.StartSlave := FALSE;

         AxisStep := STATE_STOP;
      END_IF
   END_IF

(**************** CHECK FOR GENERAL AXIS ERROR ******************)
   IF ((MC_ReadAxisError_0.AxisErrorID <> 0) AND (MC_ReadAxisError_0.Valid = TRUE))
THEN
      AxisStep := STATE_ERROR_AXIS;
(***************** CHECK IF POWER SHOULD BE OFF ******************)
   ELSIF ((CamControl.Command.Power = FALSE) AND (MC_ReadAxisError_0.Valid = TRUE))
THEN
      IF ((MC_ReadStatus_0.Errorstop = TRUE) AND (MC_ReadStatus_0.Valid = TRUE)) THEN
         AxisStep := STATE_ERROR_RESET;
      ELSE
         AxisStep := STATE_WAIT;
      END_IF
   END_IF

CASE AxisStep OF

(******************** WAIT ************************)
   STATE_WAIT: (* STATE: Wait *)
      IF (CamControl.Command.Power = TRUE) THEN
         AxisStep := STATE_POWER_ON;
      ELSE
         MC_Power_0.Enable := FALSE;
      END_IF
```

```
    (* reset all fb execute inputs we use *)
    MC_Home_0.Execute := FALSE;
    MC_Stop_0.Execute := FALSE;
    MC_MoveAbsolute_0.Execute := FALSE;
    MC_MoveAdditive_0.Execute := FALSE;
    MC_MoveVelocity_0.Execute := FALSE;
    MC_CamTableSelect_0.Execute := FALSE;
    MC_CamIn_0.Execute := FALSE;
    MC_CamOut_0.Execute := FALSE;
    MC_ReadAxisError_0.Acknowledge := FALSE;
    MC_Reset_0.Execute := FALSE;

    (* reset user commands *)
    CamControl.Command.Stop := FALSE;
    CamControl.Command.Home := FALSE;
    CamControl.Command.MoveJogPos := FALSE;
    CamControl.Command.MoveJogNeg := FALSE;
    CamControl.Command.MoveVelocity := FALSE;
    CamControl.Command.MoveAbsolute := FALSE;
    CamControl.Command.MoveAdditive := FALSE;
    CamControl.Command.DisengageSlave := FALSE;
    CamControl.Command.StartSlave := FALSE;

    CamControl.Status.ErrorID := 0;

(********** POWER ON **********)
  STATE_POWER_ON: (* STATE: Power on *)
    MC_Power_0.Enable := TRUE;
    IF (MC_Power_0.Status = TRUE) THEN
      AxisStep := STATE_READY;
    END_IF
    (* if a power error occured go to error state *)
    IF (MC_Power_0.Error = TRUE) THEN
      CamControl.Status.ErrorID := MC_Power_0.ErrorID;
      AxisStep := STATE_ERROR;
    END_IF

(********** READY **********)
  STATE_READY: (* STATE: Waiting for commands *)
    IF (CamControl.Command.Home = TRUE)THEN
      CamControl.Command.Home := FALSE;
      AxisStep := STATE_HOME;

    ELSIF (CamControl.Command.Stop = TRUE) THEN
      AxisStep := STATE_STOP;

    ELSIF (CamControl.Command.MoveJogPos = TRUE) THEN
      AxisStep := STATE_JOG_POSITIVE;

    ELSIF (CamControl.Command.MoveJogNeg = TRUE) THEN
      AxisStep := STATE_JOG_NEGATIVE;

    ELSIF (CamControl.Command.MoveAbsolute = TRUE) THEN
      CamControl.Command.MoveAbsolute := FALSE;
      AxisStep := STATE_MOVE_ABSOLUTE;

    ELSIF (CamControl.Command.MoveAdditive = TRUE) THEN
      CamControl.Command.MoveAdditive := FALSE;
```

```
        AxisStep := STATE_MOVE_ADDITIVE;

      ELSIF (CamControl.Command.MoveVelocity = TRUE) THEN
        CamControl.Command.MoveVelocity := FALSE;
        AxisStep := STATE_MOVE_VELOCITY;

      ELSIF (CamControl.Command.StartSlave = TRUE) THEN
        CamControl.Command.StartSlave := FALSE;
        AxisStep := STATE_CAM_SELECT;

      ELSIF (CamControl.Command.Halt = TRUE) THEN
        CamControl.Command.Halt := FALSE;
        AxisStep := STATE_HALT;

      END_IF

(******************** HOME **********************)
  STATE_HOME: (* STATE: start homing process *)
      MC_Home_0.Position := CamControl.Parameter.HomePosition;
      MC_Home_0.HomingMode := CamControl.Parameter.HomeMode;
      MC_Home_0.Execute := TRUE;
      IF (CamControl.Command.Stop = TRUE) THEN
        MC_Home_0.Execute := FALSE;
        AxisStep := STATE_STOP;
      ELSIF (MC_Home_0.Done = TRUE) THEN
        MC_Home_0.Execute := FALSE;
        AxisStep := STATE_READY;
      END_IF
      (* if a homing error occured go to error state *)
      IF (MC_Home_0.Error = TRUE) THEN
        MC_Home_0.Execute := FALSE;
        CamControl.Status.ErrorID := MC_Home_0.ErrorID;
        AxisStep := STATE_ERROR;
      END_IF

(***********************HALT_MOVEMENT*************************)
  STATE_HALT:   (* STATE: Halt movement *)
      MC_Halt_0.Deceleration := CamControl.Parameter.Deceleration;
      MC_Halt_0.Execute := TRUE;
      IF (MC_Halt_0.Done = TRUE) THEN
        MC_Halt_0.Execute := FALSE;
        AxisStep := STATE_READY;
      END_IF
      (* check if error occured *)
      IF (MC_Halt_0.Error = TRUE) THEN
        CamControl.Status.ErrorID := MC_Halt_0.ErrorID;
        MC_Halt_0.Execute := FALSE;
        AxisStep := STATE_ERROR;
      END_IF

(************************ STOP MOVEMENT *************************)
  STATE_STOP: (* STATE: Stop movement *)
      MC_Stop_0.Deceleration := CamControl.Parameter.Deceleration;
      MC_Stop_0.Execute := TRUE;
      (* if axis is stopped go to ready state *)
      IF ((MC_Stop_0.Done = TRUE) AND (CamControl.Command.Stop = FALSE)) THEN
        MC_Stop_0.Execute := FALSE;
        AxisStep := STATE_READY;
```

```
      END_IF
      (* check if error occured *)
       IF (MC_Stop_0.Error = TRUE) THEN
         CamControl.Status.ErrorID := MC_Stop_0.ErrorID;
         MC_Stop_0.Execute := FALSE;
         AxisStep := STATE_ERROR;
      END_IF

(********************* START JOG MOVEMENT POSITVE ***********************)
   STATE_JOG_POSITIVE: (* STATE: Start jog movement in positive direction *)
      MC_MoveVelocity_0.Velocity      := CamControl.Parameter.JogVelocity;
      MC_MoveVelocity_0.Acceleration  := CamControl.Parameter.Acceleration;
      MC_MoveVelocity_0.Deceleration  := CamControl.Parameter.Deceleration;
      MC_MoveVelocity_0.Direction     := mcPOSITIVE_DIR;
      MC_MoveVelocity_0.Execute := TRUE;
      IF (CamControl.Command.Stop) THEN
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_STOP;
      (* check if jog movement should be stopped *)
      ELSIF (CamControl.Command.MoveJogPos = FALSE) THEN
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_HALT;
      END_IF
      (* check if error occured *)
      IF (MC_MoveVelocity_0.Error = TRUE) THEN
         CamControl.Status.ErrorID := MC_MoveVelocity_0.ErrorID;
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_ERROR;
      END_IF

(********************* START JOG MOVEMENT NEGATIVE ***********************)
   STATE_JOG_NEGATIVE: (* STATE: Start jog movement in negative direction *)
      MC_MoveVelocity_0.Velocity      := CamControl.Parameter.JogVelocity;
      MC_MoveVelocity_0.Acceleration  := CamControl.Parameter.Acceleration;
      MC_MoveVelocity_0.Deceleration  := CamControl.Parameter.Deceleration;
      MC_MoveVelocity_0.Direction     := mcNEGATIVE_DIR;
      MC_MoveVelocity_0.Execute := TRUE;
      IF (CamControl.Command.Stop) THEN
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_STOP;
      (* check if jog movement should be stopped *)
      ELSIF (CamControl.Command.MoveJogNeg = FALSE) THEN
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_HALT;
      END_IF
      (* check if error occured *)
      IF (MC_MoveVelocity_0.Error = TRUE) THEN
         CamControl.Status.ErrorID := MC_MoveVelocity_0.ErrorID;
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_ERROR;
      END_IF

(********************* START ABSOLUTE MOVEMENT ***********************)
   STATE_MOVE_ABSOLUTE: (* STATE: Start absolute movement *)
      MC_MoveAbsolute_0.Position      := CamControl.Parameter.Position;
      MC_MoveAbsolute_0.Velocity      := CamControl.Parameter.Velocity;
      MC_MoveAbsolute_0.Acceleration  := CamControl.Parameter.Acceleration;
      MC_MoveAbsolute_0.Deceleration  := CamControl.Parameter.Deceleration;
```

```
      MC_MoveAbsolute_0.Direction     := CamControl.Parameter.Direction;
      MC_MoveAbsolute_0.Execute := TRUE;
      (* check if commanded position is reached *)
      IF (CamControl.Command.Stop = TRUE) THEN
         MC_MoveAbsolute_0.Execute := FALSE;
         AxisStep := STATE_STOP;
      ELSIF (CamControl.Command.Halt) THEN
         CamControl.Command.Halt := FALSE;
         MC_MoveAbsolute_0.Execute := FALSE;
         AxisStep := STATE_HALT;
      ELSIF (MC_MoveAbsolute_0.Done = TRUE) THEN
          MC_MoveAbsolute_0.Execute := FALSE;
          AxisStep := STATE_READY;
      END_IF
      (* check if error occured *)
      IF (MC_MoveAbsolute_0.Error = TRUE) THEN
         CamControl.Status.ErrorID := MC_MoveAbsolute_0.ErrorID;
         MC_MoveAbsolute_0.Execute := FALSE;
         AxisStep := STATE_ERROR;
      END_IF

(********************* START ADDITIVE MOVEMENT **********************)
   STATE_MOVE_ADDITIVE: (* STATE: Start additive movement *)
      MC_MoveAdditive_0.Distance      := CamControl.Parameter.Distance;
      MC_MoveAdditive_0.Velocity      := CamControl.Parameter.Velocity;
      MC_MoveAdditive_0.Acceleration  := CamControl.Parameter.Acceleration;
      MC_MoveAdditive_0.Deceleration  := CamControl.Parameter.Deceleration;
      MC_MoveAdditive_0.Execute := TRUE;
      (* check if commanded distance is reached *)
      IF (CamControl.Command.Stop = TRUE) THEN
         MC_MoveAdditive_0.Execute := FALSE;
         AxisStep := STATE_STOP;
      ELSIF (CamControl.Command.Halt) THEN
         CamControl.Command.Halt := FALSE;
         MC_MoveAdditive_0.Execute := FALSE;
         AxisStep := STATE_HALT;
      ELSIF (MC_MoveAdditive_0.Done = TRUE) THEN
         MC_MoveAdditive_0.Execute := FALSE;
         AxisStep := STATE_READY;
      END_IF
      (* check if error occured *)
      IF (MC_MoveAdditive_0.Error = TRUE) THEN
         CamControl.Status.ErrorID := MC_MoveAdditive_0.ErrorID;
         MC_MoveAdditive_0.Execute := FALSE;
         AxisStep := STATE_ERROR;
      END_IF

(********************* START VELOCITY MOVEMENT **********************)
   STATE_MOVE_VELOCITY: (* STATE: Start velocity movement *)
      MC_MoveVelocity_0.Velocity      := CamControl.Parameter.Velocity;
      MC_MoveVelocity_0.Acceleration  := CamControl.Parameter.Acceleration;
      MC_MoveVelocity_0.Deceleration  := CamControl.Parameter.Deceleration;
      MC_MoveVelocity_0.Direction     := CamControl.Parameter.Direction;
      MC_MoveVelocity_0.Execute := TRUE;
      (* check if commanded velocity is reached *)
      IF (CamControl.Command.Stop = TRUE) THEN
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_STOP;
```

```
      ELSIF (CamControl.Command.Halt) THEN
         CamControl.Command.Halt := FALSE;
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_HALT;
      ELSIF (MC_MoveVelocity_0.InVelocity = TRUE) THEN
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_READY;
      END_IF
      (* check if error occured *)
      IF (MC_MoveVelocity_0.Error = TRUE) THEN
         CamControl.Status.ErrorID := MC_MoveVelocity_0.ErrorID;
         MC_MoveVelocity_0.Execute := FALSE;
         AxisStep := STATE_ERROR;
      END_IF

(******************** SELECT CAM TABLE **********************)
   STATE_CAM_SELECT: (* STATE: Select the CAM *)
      MC_CamTableSelect_0.CamTable := 'profile';
      MC_CamTableSelect_0.Periodic := mcPERIODIC;
      MC_CamTableSelect_0.Execute := TRUE;
      (* wait for select done *)
      IF (MC_CamTableSelect_0.Done = TRUE) THEN
         CamTableID := MC_CamTableSelect_0.CamTableID;
         MC_CamTableSelect_0.Execute := FALSE;
         AxisStep := STATE_CAM_START;
      END_IF
      (* check if error occured *)
      IF (MC_CamTableSelect_0.Error = TRUE) THEN
         CamControl.Status.ErrorID := MC_CamTableSelect_0.ErrorID;
         MC_CamTableSelect_0.Execute := FALSE;
         AxisStep := STATE_ERROR;
      END_IF

(******************** START CAM MOVEMENT **********************)
   STATE_CAM_START: (* STATE: Start electronic gear coupling *)
      MC_CamIn_0.MasterOffset := CamControl.Parameter.MasterOffset;
      MC_CamIn_0.SlaveOffset := CamControl.Parameter.SlaveOffset;
      MC_CamIn_0.MasterScaling := CamControl.Parameter.MasterScaling;
      MC_CamIn_0.SlaveScaling := CamControl.Parameter.SlaveScaling;
      MC_CamIn_0.StartMode := CamControl.Parameter.StartMode;
      MC_CamIn_0.CamTableID := CamTableID;
      MC_CamIn_0.Execute := TRUE;
      (* wait for CAM stop *)
      IF (CamControl.Command.Stop = TRUE) THEN
         MC_CamIn_0.Execute := FALSE;
         AxisStep := STATE_STOP;
      ELSIF (CamControl.Command.Halt) THEN
         CamControl.Command.Halt := FALSE;
         MC_CamIn_0.Execute := FALSE;
         AxisStep := STATE_HALT;
      ELSIF (CamControl.Command.DisengageSlave = TRUE) THEN
         CamControl.Command.DisengageSlave := FALSE;
         MC_CamIn_0.Execute := FALSE;
         AxisStep := STATE_CAM_STOP;
      END_IF
      (* check if error occured *)
      IF (MC_CamIn_0.Error = TRUE) THEN
         CamControl.Status.ErrorID := MC_CamIn_0.ErrorID;
```

```
            MC_CamIn_0.Execute := FALSE;
            AxisStep := STATE_ERROR;
        END_IF

(******************** STOP CAM MOVEMENT **********************)
    STATE_CAM_STOP: (* STATE: Stop electronic gear coupling *)
        MC_CamOut_0.Execute := TRUE;
        (* check if coupling is stopped *)
        IF (MC_CamOut_0.Done = TRUE) THEN
            MC_CamOut_0.Execute := FALSE;
            AxisStep := STATE_READY;
        END_IF
        (* check if error occured *)
        IF (MC_CamOut_0.Error = TRUE) THEN
            CamControl.Status.ErrorID := MC_CamOut_0.ErrorID;
            MC_CamOut_0.Execute := FALSE;
            AxisStep := STATE_ERROR;
        END_IF

(******************** FB-ERROR OCCURED *************************)
    STATE_ERROR: (* STATE: Error *)
        (* check if FB indicates an axis error *)
        IF (MC_ReadAxisError_0.AxisErrorCount<>0) THEN
            AxisStep := STATE_ERROR_AXIS;
        ELSE
            IF (CamControl.Command.ErrorAcknowledge = TRUE) THEN
                CamControl.Command.ErrorAcknowledge := FALSE;
                CamControl.Status.ErrorID := FALSE;
                (* reset axis if it is in axis state ErrorStop *)
                IF ((MC_ReadStatus_0.Errorstop = TRUE) AND (MC_ReadStatus_0.Valid = TRUE))
THEN
                    AxisStep := STATE_ERROR_RESET;
                ELSE
                    AxisStep := STATE_WAIT;
                END_IF
            END_IF
        END_IF

(******************** AXIS-ERROR OCCURED *************************)
    STATE_ERROR_AXIS: (* STATE: Axis Error *)
        IF (MC_ReadAxisError_0.Valid = TRUE) THEN
            IF (MC_ReadAxisError_0.AxisErrorID <> 0) THEN
                CamControl.Status.ErrorID := MC_ReadAxisError_0.AxisErrorID;
            END_IF
            MC_ReadAxisError_0.Acknowledge := FALSE;
            IF (CamControl.Command.ErrorAcknowledge = TRUE) THEN
                CamControl.Command.ErrorAcknowledge := FALSE;
                (* acknowledge axis error *)
                IF (MC_ReadAxisError_0.AxisErrorID <> 0) THEN
                    MC_ReadAxisError_0.Acknowledge := TRUE;
                END_IF
            END_IF
            IF (MC_ReadAxisError_0.AxisErrorCount = 0) THEN
                (* reset axis if it is in axis state ErrorStop *)
                CamControl.Status.ErrorID := 0;
                IF ((MC_ReadStatus_0.Errorstop = TRUE) AND (MC_ReadStatus_0.Valid = TRUE))
THEN
                    AxisStep := STATE_ERROR_RESET;
```

```
            ELSE
               AxisStep := STATE_WAIT;
            END_IF
         END_IF
      END_IF

(******************** RESET DONE ************************)
   STATE_ERROR_RESET: (* STATE: Wait for reset done *)
      MC_Reset_0.Execute := TRUE;
      (* reset MC_Power.Enable if this FB is in Error*)
      IF (MC_Power_0.Error = TRUE) THEN
         MC_Power_0.Enable := FALSE;
      END_IF
      IF(MC_Reset_0.Done = TRUE)THEN
         MC_Reset_0.Execute := FALSE;
         AxisStep := STATE_WAIT;
      ELSIF(MC_Reset_0.Error = TRUE) THEN
         MC_Reset_0.Execute := FALSE;
         AxisStep := STATE_ERROR;
      END_IF
(******************** SEQUENCE END ************************)
END_CASE

(*************************************************************
      Function Block Calls
*************************************************************)

(************************** MC_POWER ***************************)
MC_Power_0.Axis := Axis2Obj; (* pointer to axis *)
MC_Power_0();

(************************** MC_HOME ****************************)
MC_Home_0.Axis := Axis2Obj;
MC_Home_0();

(********************** MC_MOVEABSOLUTE ************************)
MC_MoveAbsolute_0.Axis := Axis2Obj;
MC_MoveAbsolute_0();

(********************** MC_MOVEADDITIVE ************************)
MC_MoveAdditive_0.Axis := Axis2Obj;
MC_MoveAdditive_0();

(********************** MC_MOVEVELOCITY ************************)
MC_MoveVelocity_0.Axis := Axis2Obj;
MC_MoveVelocity_0();

(********************** MC_CAMTABLESELECT ************************)
MC_CamTableSelect_0.Master:= Axis1Obj;
MC_CamTableSelect_0.Slave:= Axis2Obj;
MC_CamTableSelect_0();

(********************** MC_CAMIN ************************)
MC_CamIn_0.Master := Axis1Obj;
MC_CamIn_0.Slave := Axis2Obj;
MC_CamIn_0();

(********************** MC_CAMOUT ************************)
```

```
MC_CamOut_0.Slave := Axis2Obj;
MC_CamOut_0();

(************************* MC_STOP *****************************)
MC_Stop_0.Axis := Axis2Obj;
MC_Stop_0();

(*************************MC_HALT*****************************)
MC_Halt_0.Axis := Axis2Obj;
MC_Halt_0();

(************************* MC_RESET ****************************)
MC_Reset_0.Axis := Axis2Obj;
MC_Reset_0();

END_PROGRAM
```

```
(*********************************************************************
 * COPYRIGHT --
 *********************************************************************
 * Program: KontrolAut
 * File: KontrolAut.ab
 * Author: OM
 * Created: May 11, 2014
 *********************************************************************
 * Implementation of program KontrolAut
 *********************************************************************)

PROGRAM _INIT

    IF NkPoc = 0 THEN
        NkPoc         = 240                    ;pocatecni otacky kotouèe 1/min
    ENDIF
    IF NkDeltaMax = 0 THEN
        NkDeltaMax = 2760                      ;max zmena otaèek 1/min
    ENDIF

    Ak          = 138                          ;konstanata generátoru otacek
    Ck          = 73                           ;konstanata generátoru otacek
    Tk          = 6                            ;perioda generatoru otacek
    NkDeltaOld  = NkPoc                        ;

    No          = 240                          ;otacky odpalu 1/min
    DistanceO   = 1000                         ;draha jednoho odpalu - jedna otacka
    IF ToMin = 0 THEN
        ToMin     = 2                          ;minimalni perida odpalu
    ENDIF
    IF ToDeltaMax = 0 THEN
        ToDeltaMax = 10                        ;max zmena periody odpalu
    ENDIF

    Ao          = 13                           ;konstanata generátoru periody
    Co          = 7                            ;konstanata generátoru periody
    Too             = 2                        ;perioda generatoru periody
    ToDeltaOld  = ToMin

END_PROGRAM


PROGRAM _CYCLIC
    ErrorPoh = BOOL(BasicControl.Status.ErrorID <> 0) OR BOOL(GearControl.Status.ErrorID
<> 0) OR BOOL(CamControl.Status.ErrorID <> 0)

    SELECT StavAut
        STATE PORUCHA               ;0
            StartP1        = 0
            WHEN NOT(ErrorPoh)
                NEXT VYPNUTO

        STATE VYPNUTO               ;1

            WHEN StartP1 OR StartP2 OR StartP3
                NEXT PRIPRAVA
            WHEN ErrorPoh
                NEXT PORUCHA
```

```
        STATE PRIPRAVA              ;2
            BasicControl.Command.Power          = 1
            GearControl.Command.Power           = 1
            CamControl.Command.Power            = 1
            BasicControl.Command.Home           = 1
            GearControl.Command.Home            = 1
            CamControl.Command.Home             = 1
            WHEN BasicControl.Status.DriveStatus.ControllerStatus AND
GearControl.Status.DriveStatus.ControllerStatus AND
CamControl.Status.DriveStatus.ControllerStatus AND
BasicControl.Status.DriveStatus.HomingOk AND GearControl.Status.DriveStatus.HomingOk
AND CamControl.Status.DriveStatus.HomingOk
                NEXT PRIPRAVAOK
            WHEN NOT(StartP1) AND NOT(StartP2) AND NOT(StartP3)
                BasicControl.Command.Power      = 0
                GearControl.Command.Power       = 0
                CamControl.Command.Power        = 0
                NEXT VYPNUTO
            WHEN ErrorPoh
                BasicControl.Command.Power      = 0
                GearControl.Command.Power       = 0
                CamControl.Command.Power        = 0
                NEXT PORUCHA

        STATE PRIPRAVAOK          ;3
            GearControl.Command.StartSlave      = 1
            WHEN StartP1
                NEXT PARAMETRY1
            WHEN StartP2
                NEXT PARAMETRY2
            WHEN StartP3
                NEXT PARAMETRY3
            WHEN NOT(StartP1) AND NOT(StartP2) AND NOT(StartP3)
                BasicControl.Command.Power      = 0
                GearControl.Command.Power       = 0
                CamControl.Command.Power        = 0
                NEXT VYPNUTO
            WHEN ErrorPoh
                BasicControl.Command.Power      = 0
                GearControl.Command.Power       = 0
                CamControl.Command.Power        = 0
                NEXT PORUCHA

        STATE PARAMETRY1                                        ;4
            BasicControl.Parameter.Velocity     = (NkPoc + NkDelta)*1000/60
        ;prevod ot/min na unit/s
            BasicControl.Command.MoveVelocity = 1
            CamControl.Parameter.Velocity       = No*1000/60
            CamControl.Parameter.Distance       = DistanceO
            WHEN StartP1
                BasicControl.Command.MoveVelocity       = 1
                NEXT CHODP1
            WHEN NOT(StartP1)
                BasicControl.Command.Power      = 0
                GearControl.Command.Power       = 0
                CamControl.Command.Power        = 0
                NEXT VYPNUTO
```

```
        WHEN ErrorPoh
            BasicControl.Command.Power      = 0
            GearControl.Command.Power       = 0
            CamControl.Command.Power        = 0
            NEXT PORUCHA

    STATE CHODP1                ;5
        BasicControl.Parameter.Velocity      = (NkPoc + NkDelta)*1000/60
        BasicControl.Command.MoveVelocity = 1
        IF EDGEPOS(TikSek) THEN
            Perioda = Perioda + 1
            IF Perioda >= (ToMin + ToDelta) THEN
                CamControl.Command.MoveAdditive = 1
                Perioda = 0
            ENDIF
        ENDIF

        WHEN NOT(StartP1) AND NOT(CamControl.AxisState.DiscreteMotion)
            BasicControl.Command.Stop       = 1
            NEXT VYPINANI
        WHEN ErrorPoh
            BasicControl.Command.Power      = 0
            GearControl.Command.Power       = 0
            CamControl.Command.Power        = 0
            NEXT PORUCHA

    STATE PARAMETRY2                                        ;6
        BasicControl.Parameter.Velocity      = (NkPoc + NkDelta)*1000/60
    ;prevod ot/min na unit/s
        CamControl.Parameter.Velocity        = No*1000/60
        CamControl.Parameter.Distance        = DistanceO
        WHEN StartP2
            BasicControl.Command.MoveVelocity       = 1
            NEXT CHODP2
        WHEN NOT(StartP2)
            BasicControl.Command.Power      = 0
            GearControl.Command.Power       = 0
            CamControl.Command.Power        = 0
            NEXT VYPNUTO
        WHEN ErrorPoh
            BasicControl.Command.Power      = 0
            GearControl.Command.Power       = 0
            CamControl.Command.Power        = 0
            NEXT PORUCHA

    STATE CHODP2                                            ;7
        BasicControl.Parameter.Velocity      = (NkPoc + NkDelta)*1000/60
        IF EDGEPOS(TikSek) THEN
            Perioda = Perioda + 1
            IF Perioda >= ToMin THEN
                CamControl.Command.MoveAdditive = 1
                Perioda = 0
            ENDIF
        ENDIF

        WHEN NOT(StartP2) AND NOT(CamControl.AxisState.DiscreteMotion)
            BasicControl.Command.Stop       = 1
            NEXT VYPINANI
```

```
        WHEN ErrorPoh
            BasicControl.Command.Power      = 0
            GearControl.Command.Power       = 0
            CamControl.Command.Power        = 0
            NEXT PORUCHA

    STATE PARAMETRY3                        ;8
        BasicControl.Parameter.Velocity     = (NkPoc + NkDelta)*1000/60
    ;prevod ot/min na unit/s
        CamControl.Parameter.Velocity       = No*1000/60
        CamControl.Parameter.Distance       = DistanceO
        WHEN StartP3
            BasicControl.Command.MoveVelocity       = 1
            NEXT CHODP3
        WHEN NOT(StartP3)
            BasicControl.Command.Power      = 0
            GearControl.Command.Power       = 0
            CamControl.Command.Power        = 0
            NEXT VYPNUTO
        WHEN ErrorPoh
            BasicControl.Command.Power      = 0
            GearControl.Command.Power       = 0
            CamControl.Command.Power        = 0
            NEXT PORUCHA

    STATE CHODP3                    ;9
        BasicControl.Parameter.Velocity     = NkPoc*1000/60
        IF EDGEPOS(TikSek) THEN
            Perioda = Perioda + 1
            IF Perioda >= ToMin THEN
                CamControl.Command.MoveAdditive = 1
                Perioda = 0
            ENDIF
        ENDIF

        WHEN NOT(StartP3) AND NOT(CamControl.AxisState.DiscreteMotion)
            BasicControl.Command.Stop       = 1
            NEXT VYPINANI
        WHEN ErrorPoh
            BasicControl.Command.Power      = 0
            GearControl.Command.Power       = 0
            CamControl.Command.Power        = 0
            NEXT PORUCHA

    STATE VYPINANI                  ;10
        WHEN BasicControl.Status.ActVelocity < 10
            BasicControl.Command.Power      = 0
            GearControl.Command.Power       = 0
            CamControl.Command.Power        = 0
            NEXT VYPNUTO
        WHEN ErrorPoh
            BasicControl.Command.Power      = 0
            GearControl.Command.Power       = 0
            CamControl.Command.Power        = 0
            NEXT PORUCHA

ENDSELECT
```

```
LCCounter_0 FUB LCCounter()
    Tikk        = ((LCCounter_0.ms100cnt MOD (Tk*10)) < (Tk*5))

    IF EDGEPOS(Tikk) THEN               ;generuje nahodné cislo v rozsahu 0 az NkDeltaMax s
periodou Tk

        NkDelta   = (Ak*NkDeltaOld +Ck) MOD NkDeltaMax
        NkDeltaOld  = NkDelta
    ENDIF

    Tiko        = ((LCCounter_0.ms100cnt MOD (Too*10)) < (Too*5))

    IF EDGEPOS(Tiko) THEN               ;generuje nahodné cislo v rozsahu 0 az NkDeltaMax s
periodou Tk

        ToDelta   = (Ao*ToDeltaOld +Co) MOD ToDeltaMax
        ToDeltaOld  = ToDelta
    ENDIF

END_PROGRAM
```

```
(*********************************************************************
 * COPYRIGHT -- OM
 *********************************************************************
 * Program: Kontrol
 * File: Kontrol.st
 * Author: OM
 * Created: March 23, 2014
 *********************************************************************
 * Implementation of program Kontrol
 *********************************************************************)

PROGRAM _INIT


(* TODO : Add your code here *)


END_PROGRAM


PROGRAM _CYCLIC
    LCCounter_0();
    TikSek := ((LCCounter_0.ms100cnt MOD 10) < 4);

    IF EDGEPOS(StartM) THEN
        BasicControl.Command.MoveVelocity := 1;
    END_IF;
    IF EDGENEG(StartM) THEN
        BasicControl.Command.Stop := 1;
    END_IF;


    CASE Stav OF
        0:
            ColorMaster := 0;
            ColorSlave := 0;
            ColorShut := 0;
            IF Power THEN
                Stav := 1;
            END_IF;
        1:
            BasicControl.Command.Power := 1;
            GearControl.Command.Power := 1;
            CamControl.Command.Power := 1;
            ColorMaster := 2;
            ColorSlave := 2;
            ColorShut := 2;
            IF NOT(Power) THEN
                Stav := 0;
                BasicControl.Command.Power := 0;
                GearControl.Command.Power := 0;
                CamControl.Command.Power := 0;
            ELSE
                Stav := 2;
            END_IF;

        2:
            BasicControl.Command.Home := 1;
```

```
            GearControl.Command.Home := 1;
            CamControl.Command.Home := 1;
            ColorMaster := 2;
            ColorSlave := 2;
            ColorShut := 2;
            IF NOT(Power) THEN
                Stav := 0;
                BasicControl.Command.Power := 0;
                GearControl.Command.Power := 0;
                CamControl.Command.Power := 0;
            ELSE
                Stav := 3;
                ColorMaster := 3;
                ColorSlave := 3;
                ColorShut := 3;
            END_IF;

        3:
            GearControl.Command.StartSlave := 1;

            IF NOT(Power) THEN
                GearControl.Command.StartSlave := 0;
                GearControl.Command.DisengageSlave := 1;
                Stav := 0;
                BasicControl.Command.Power := 0;
                GearControl.Command.Power := 0;
                CamControl.Command.Power := 0;
            END_IF;

            IF (EDGEPOS(TikSek) AND (BasicControl.Status.ActVelocity > 0)) THEN
                IF ColorMaster = 10 THEN
                    ColorMaster := 3;
                ELSE
                    ColorMaster := ColorMaster + 1;
                END_IF
            END_IF;

            IF (EDGEPOS(TikSek) AND (GearControl.Status.ActVelocity < 0)) THEN
                IF ColorSlave = 3 THEN
                    ColorSlave:= 10;
                ELSE
                    ColorSlave := ColorSlave - 1;
                END_IF
            END_IF;

        (*  IF (EDGEPOS(TikSek) AND (CamControl.Status.ActVelocity > 0)) THEN
                IF ColorShut = 10 THEN
                    ColorShut := 3;
                ELSE
                    ColorShut := ColorShut + 1;
                END_IF
            END_IF;
*)
            CamMod := REAL_TO_UDINT(CamControl.Status.ActPosition) MOD 1000;
            IF CamMod > 875 THEN
                ColorShut := 10;
            ELSE
                IF  CamMod > 750 THEN
```

```
                ColorShut := 9;
            ELSE
                IF  CamMod > 625 THEN
                    ColorShut := 8;
                ELSE
                    IF  CamMod > 500 THEN
                        ColorShut := 7;
                    ELSE
                        IF  CamMod > 375 THEN
                            ColorShut := 6;
                        ELSE
                            IF  CamMod > 250 THEN
                                    ColorShut := 5;
                            ELSE
                                    IF  CamMod > 125 THEN
                                            ColorShut := 4;
                                    ELSE
                                            IF  CamMod >= 0 THEN
                                        ColorShut := 3;
                                            END_IF
                                    END_IF
                            END_IF
                        END_IF
                    END_IF
                END_IF
            END_IF
        END_IF

    END_CASE;

    a := b MOD c;

END_PROGRAM
```