Gymnázium Kroměříž
Masarykovo náměstí 496, 767 01 Kroměříž

Středoškolská odborná činnost

Obor SOČ: 18 informatika

Návrh řídicího systému CNC stroje se zaměřením na dynamiku pohybu

Design of a control system for a CNC machine with a focus on the dynamics of movement

Hulín 2013 Autor: Jan Mrázek

Prohlášení

Prohlašuji, že jsem svou práci vypracoval samostatně, použil jsem pouze podklady (literaturu, software, atd.) uvedené v této práci a postup při zpracování a dalším nakládání s prací je v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) v platném znění.

V Hulíně dne 28. března 2013

podpis

Poděkování

Chtěl bych poděkovat zejména mému otci, který mě v celém mém bádání podporuje a poskytuje mi pomoc v konstrukčních stránkách celého projektu. Také bych chtěl poděkovat mé matce a Evě Trojanové za provedenou jazykovou korekci této práce.

Abstrakt

Tato práce popisuje vývoj řídicího systému pro tříosý hobby CNC stroj (frézku). Tento systém se zaměřuje především na dynamiku pohybu stroje tak, aby byl co nejméně mechanicky namáhán. Toho dociluje použitím akceleračních S-křivek, které limituje maximálním použitým zrychlením a ryvem.

Uživatelské rozhraní je řešeno skrz klasický počítač, samotnou interpolaci pohybu však obstarává speciální hardwarový interpolátor postavený na STM32F4Discovery kitu. Tento interpolátor komunikuje s počítačem pomocí USB 2.0 a hlásí se jako vendor-defined zařízení.

Řídicí systém přijímá instrukce pro stroj v G-kódu. Z G-kódu podporuje většinu nejpoužívanějších funkcí a umí provádět korekci nástroje.

Klíčová slova

- CNC stroj
- řidicí systém
- dynamika pohybu
- S-křivky
- ryv
- USB komunikace
- G-kód
- korekce nástroje

Abstract

The work deals with the design of a control system for a 3-axis hobby CNC machine (mill). This system focuses on the dynamics of the motion and tries to minimize the dynamic stress on the mechanics of the machine. To achieve this goal it uses acceleration S-curves, which are limited by peak acceleration and jerk.

The user interface is realised via a personal computer, but a special hardware unit is used for intepolation of the movement. This unit is built on the STM32F4Discovery kit. It communicates with the PC via USB 2.0. The unit presents itself as a vendor-defined USB device in the host system.

The control system takes G-code as an input. The system can interpret all the commonly used G-functions. It is also able to perfom cutter compensation for a mill.

Keywords

- CNC machine
- control system
- movement dynamics
- S-curves
- jerk (physical quantity)
- USB communication
- G-code
- cutter compensation

Obsah

1 Úvod **1**
1.1 Cíle projektu . . . 1

2 Koncepce řídicího systému hobby CNC stroje **2**
2.1 Pohony . . . 2
2.2 Generování řídicích signálů . . . 3
2.3 G-kód . . . 3
2.4 Existující řešení řídicího sytému . . . 3
2.4.1 Mach 3 . . . 3
2.4.2 Linux CNC . . . 4
2.4.3 Intepolační jednotky Gravos . . . 4

3 Koncepce mého řídicího systému **6**

I Fyzikální model **8**

4 Pojem fyzikální model a jeho vlastnosti **9**
4.1 Vlastnosti ideálního fyzikálního modelu . . . 9
4.2 Fyzikální veličina ryv . . . 9
4.3 Akcelerační křivky . . . 10
4.4 Postulát pohybu . . . 11

5 Odvození fyzikálního modelu **12**
5.1 Limitující faktory pohybu . . . 12
5.2 Pohyb po úsečce . . . 12
5.2.1 Rozjezd a brzdění . . . 12
5.2.2 Omezení maximálního ryvu . . . 14
5.3 Pohyb po kruhovém oblouku . . . 16
5.3.1 Maximální zrychlení v čase $t = T$ . . . 18
5.3.2 Maximální zrychlení v čase $t = \frac{T}{2}$ . . . 19
5.3.3 Problém pohybu po kruhovém oblouku . . . 19
5.4 Pohyb po ideální křivce . . . 20

II Realizace **21**

6 Použité prostředky a hardware **22**
6.1 Interpolátor . . . 22

6.2 Aplikace pro počítač . . . 22
6.3 Testovací stroj . . . 23

7 Software pro interpolátor **25**
7.1 Implementace USB a komunikační protokol . . . 26
7.1.1 Komunikační protokol a komponenta Receive . . . 26
7.2 Komponenta CommandStack . . . 27
7.2.1 Struktura CommandStruct . . . 27
7.2.2 Zpracování zpráv . . . 27
7.3 Komponenta Axis . . . 28
7.3.1 Použití datového typu float . . . 28
7.3.2 Implementace funkce sinus . . . 29
7.3.3 Lineární interpolace . . . 29
7.3.4 Oblouková interpolace . . . 30
7.3.5 Interpolace ideální křivky . . . 32
7.4 Komponenty Movement a Stepper . . . 32
7.4.1 Komponenta Stepper . . . 32
7.4.2 Komponenta PulseGenerator . . . 33
7.5 Implementace ostatních komponent . . . 33
7.5.1 Komponenta GPIOControl . . . 34
7.5.2 Komponenty SystemComponents . . . 34

8 Software pro počítač **35**
8.1 Uživatelské rozhraní . . . 35
8.2 Intepretace G-kódu . . . 36
8.2.1 G-kód a jeho formát . . . 36
8.2.2 Interní formát a třída GCodeInterpreter . . . 38
8.2.3 Převod G-kódu . . . 38
8.3 Třída PathPart . . . 38
8.4 Převod GCodeLine na PathPart . . . 39
8.4.1 Funkce G00 . . . 40
8.4.2 Funkce G01 . . . 40
8.4.3 Funkce G02 a G03 . . . 40
8.4.4 Funkce G20 a G21 . . . 40
8.4.5 Funkce F . . . 41
8.4.6 Funkce G40, G41 a G42 . . . 41
8.4.7 Funkce G43, G44 a G49 . . . 41
8.5 Implementace korekce nástroje . . . 41
8.5.1 Korekce průměru . . . 41
8.5.2 Korekce délky . . . 47
8.6 Výpočet rychlosti . . . 48
8.6.1 Určení rozjezdové a brzdné dráhy při lineárním pohybu . . . 49
8.6.2 Určení rozjezdové a brzdné dráhy při pohybu po oblouku . . . 50
8.6.3 Určení nové rychlosti pomocí bisekce . . . 52
8.7 Komunikace s interpolátorem . . . 53
8.7.1 Tvorba balíčku ovladačů WinUSB . . . 53
8.7.2 Třída MyDevice . . . 54
8.7.3 Třída ReceiveFromDevice . . . 54

8.7.4 Třídy ProgramControl a ProgramRun 54

III Závěr **56**

9 Zhodnocení plynulosti pohybu **57**
9.1 Subjektivní hodnocení 57
9.2 Objektivní měření 57
9.2.1 Měření na základě polohy 58
9.2.2 Měření pomocí akcelerometru 58

10 Současný stav projektu a jeho budoucnost **62**

Literatura **64**

IV Přílohy **I**

A Dotaz na řešení akcelerace u interpolačních jednotek Gravos **II**
A.1 Můj dotaz II
A.2 Vyjádření společnosti Gravos II

B Obsah přiloženého CD **IV**

Kapitola 1

Úvod

Dnes lze najít spoustu řešení pro řízení doma postaveného CNC stroje. Většina z nich je postavena na osobním počítači, který vykonává veškeré nutné úkony – od zpracování vstupního G-kódu, přes výpočet rychlosti a generování řídicích signálů pro pohony, až po samotné uživatelské rozhraní. Tento koncept je velmi jednoduchý na realizaci (počítač poskytuje dostatek výpočetního výkonu a přívětivé rozhraní pro uživatele), zároveň je také velice levný – není třeba žádné speciální vybavení.

To s sebou přináší však i řadu nevýhod. Mezi největší řadím omezené možnosti generování výstupního řídicího signálu (zejména co se výstupní frekvence a doby odezvy týče). Klasický stolní počítač disponuje mnohonásobně větším výkonem než jednočipové mikrokontroléry, avšak narozdíl od jednočipu mu jeho koncepce neumožňuje provádět přesné výstupní operace, co se časování týče.

Také tyto systému často disponují nepropracovaným systémem řízení rychlosti, který většinou spoléhá na kvalitní a tuhou konstrukci stroje, která je často u hobby strojů právě nejslabším článkem.

1.1 Cíle projektu

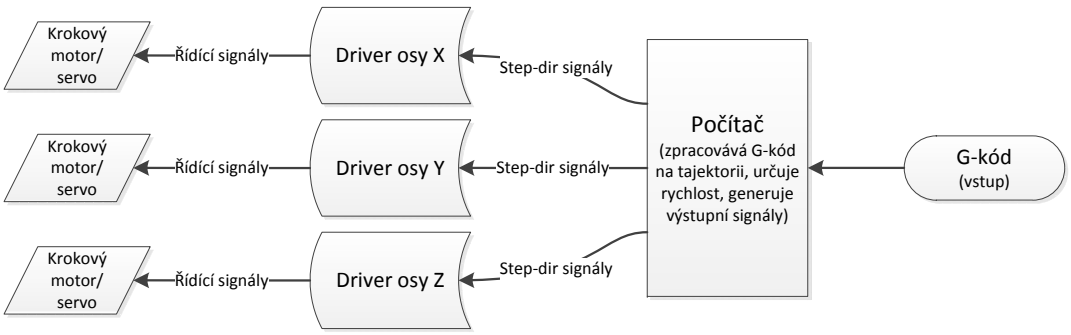
Výše uvedené důvody mě motivovaly k tomu pokusit se navrhnout a posléze i realizovat vlastní řídicí systém, který by tyto nedostatky měl odstranit. Můj systém by měl řídit tříosou frézku v klasickém pravoúhlém uspořádání, což je nejčastější a nejjednodušší možné uspořádání. Vstupní data by měl přebírat v G-kódu, který je nepsaným standardem pro programování CNC strojů, a měl by také podporovat funkce pro kompenzaci nástroje.

Kapitola 2

Koncepce řídicího systému hobby CNC stroje

V domácích podmínkách není možné dosáhnout na prostředky běžně používané v průmyslovém prostředí. Roli zde hrají nejen finance, ale také výrobní možnosti. Na hobby CNC stroj však nejsou kladeny takové nároky jako na průmyslové stroje. Proto se často přistupuje k jednodušším řešením, která však pro nenáročné hobby použití dostačují..

Na schématu 2.1 znázorňuji běžné uspořádání řídicího systému hobby CNC strojů. Toto schéma jsem sestavil na základě strojů ze sekce „Naše mašinky“ na diskusním fóru C-N-C.cz [2].



Obrázek 2.1: Schéma znázorňující uspořádání klasického řídicího systému hobby CNC stroje.

2.1 Pohony

Pro řízení hobby CNC strojů se používají krokové motory, zřídka servomotory. Zpravidla se nepoužívá žádná zpětná vazba do řídicí jednotky (i zpětná vazba z enkodérů u servomotorů je zpracovávána pouze v rámci pohonu). Na řízení krokových motorů a servomotorů se používají speciální drivery, které přijmají informaci ve formátu step-dir. Úroveň jednoho datového vodiče určuje směr otáčení, vstupní pulz na druhém pak „krok“ – pootočení motoru o daný úhel. Při řízení se spoléhá na to, že pohon je dostatečně nadimenzovaný a nedojde ke ztrátě kroků.

Krokové motory jsou velmi levnou variantou pohonu. Jejich výhodou je jednoduchý způsob řízení a velký přídržný moment. Pro jejich řízení existují i speciální itegrované obvody –

např. hojně používané Toshiba TB6560[21]. Nevýhodou krokových motorů je jejich malý moment ve vysokých otáčkách a relativně „tvrdý chod“ (motor se pohybuje po přesně daných krocích) – vznikají rezonance[15].

Servomotory v poloprofesionálním provedení se skládají ze stejnosměrného či bezkomutátorového motoru, ke kterému je připevněn rotační encoder[16]. Na základě požadované pozice a reálné pozice motoru získané z enkodéru se tvoří řídicí signál pro motor. Ačkoliv zde již zpětná vazba existuje, není posílána zpět do systému. Servomotory mají oproti krokovým motorům také lepší moment při vyšších otáčkách[16]. Avšak používaný systém řízení step-dir trochu degraduje jejich možnosti (při nízkém počtu kroků na otáčku je nutí pracovat v nespojitém režimu).

2.2 Generování řídicích signálů

Řídicí signály zpravidla generuje počítač pomocí pinů paralelního portu. Pro správnou funkci je třeba generovat pulzy v řádech stovek Hz až jednotek kHz. Zde spatřuji největší slabinu těchto řídicích systémů. Klasický počítač není stavěn na přesné generování výstupních signálů v „režimu GP I/O“. Částečně je možné přesné časování zajistit použitím real-timového operačního systému, avšak i přesto nelze dosahovat závratných výsledků.

Řešením může být použití interpolátoru postaveného na mikrokontroléru, kde není problém generovat přesně dané pulzy v přesně daném časovém rámci (až s přesností na nanosekundy), či použít speciální přídavné karty do PCI, popř. PCIe, slotu, která přidá do počítače podporu pro GP I/O.

2.3 G-kód

G-kód je označení jazyka pro zápis drah CNC stroje[23]. Tento jazyk se používá jak v průmyslovém prostředí, tak i v hobby. Dráhy stroje lze programovat ručně, dnes se však již i mezi hobby uživateli používá CAM program, který G-kód vygeneruje (téměř) automaticky. Standard tohoto jazyka však není většinou respektován a každý systém některé jeho části interpretuje různě.

V základu disponuje G-kód prostředky pro zapsání lineární a obloukové interpolace[20]. Objevily se však snahy zaimplementovat podporu pro obecné křivky (např. systém Sinumerik 810 T implementuje podporu pro spline[17]). Avšak tato řešení se neujala v praxi. Osobně se domnívám, že je to zapříčiněno nulovou podporou v CAM programech. Složitější křivky se tedy i dnes zapisují jako řada na sebe tečných kruhových oblouků, které danou křivku pouze aproximují.

2.4 Existující řešení řídicího sytému

Na závěr této kapitoly jsem zařadil stručný přehled již existujících řídicích systémů, které jsou běžně používány.

2.4.1 Mach 3

Mach 3 je řídicí systém od společnosti ArtSoft[10]. Tento software je určen pro stolní počítače s operačním systémem Windows. Je cílen na amatérské až poloprofesionální použití a je schopen ovládat až 6 os. Mach 3 využívá lineárních akceleračních křivek – používá konstantní zrychlení

a nijak nelimituje ryv[10]. Podporuje pouze stroje s pravoúhlým uspořádáním os. V základní verzi je dostupný zdarma, avšak omezuje maximální délku načítaného programu.

Veškeré vstupně-výstupní operace probíhají přes piny paralelního portu. Avšak díky absenci real-time kernelu ve Windows není schopen generovat vysoké řídicí frekvence. To potvrzují zkušenosti uživatelů z diskusního fóra C-N-C.cz[2]. Vesměs označují systém za nestabilní a fungující pouze na konkrétním hardwaru. Avšak lze najít spoustu spokojených uživatelů tohoto systému – většinou se však nesnaží dosáhnout vysokých obráběcích rychlostí.

Zajímavou vlastností Machu 3 je existence relativně velkého počtu různých tzv. hardwarových stepgenů[1]. Stepgeny jsou diskrétní kousky hardwaru postaveného na mikrokontroléru či FPGA, které se připojují k počítači pomocí komunikačního rozhraní (USB, paralelní či sériový port). Ze systému dostávají instrukce o aktuálním pohybu a až stepgen generuje řídicí signály pro pohony stroje. Tím je odstraněn problém přesného časování výstupních operací. V podstatě se jedná o stejnou formu řešení jako můj řídicí systém. Příkladem stepgenu může být SmoothStepper[2]. Tyto stepgeny však nejsou příliš rozšířeny a nepovedlo se mi zjistit žádné uživatelské zkušenosti.

2.4.2 Linux CNC

LinuxCNC, dříve nazývaný EMC (Endhanced Machine Controller), je open source projekt univerzálního řídicího systému postaveného na stolním počítači[6]. Narozdíl od Machu 3 běží LinuxCNC pod Linuxem s kernelem obohaceným o RT API (real-time API), které umožňuje spouštět určité operace v přesně daném čase (přesnost je omezena použitým hardwarem počítače).

LinuxCNC staví na univerzálnosti. Je plně konfigurovatelný a přizpůsobitelný. Osobně se mi na něm líbí koncept HAL (hardware abstraction layer)[3], který umožňuje zcela libovolně propojit vstupy a výstupy jednotlivých komponent nejen s hardwarovým výstupem, ale i mezi sebou. Systém je téměř libovolně upravitelný. Díky tomu je však komplikovaný, což může odradit spoustu uživatelů.

Z hlediska dynamiky používá LinuxCNC také lineární akcelerační křivku. Systém však zvládá výpočet rychlosti pro několik druhů kinematiky stroje – nejen pravoúhlé, ale např. i tzv. hexapodu.

Vstupně-výstupní operace jsou realizovány za pomoci paralelního portu. Avšak díky RT kernelu dosahuje vyšší spolehlivosti (dle zkušeností uživatelů C-N-C.cz fóra[2]). Je méně závislý na použitém hardwaru počítače, avšak i přesto existují konfigurace, na kterých neběží spolehlivě.

Pro tento systém existuje podpora MESA karet[4]. Jedná se o karty, nejčastěji do PCI či PCIe slotu, osazené FPGA, na kterých je přenecháno generování výstupních signálů. Při použití těchto karet dosahuje systém vyšší spolehlivosti, avšak stále není možné vypustit real-time kernel.

2.4.3 Intepolační jednotky Gravos

Česká společnost Gravos vyvíjí a vyrábí vlastní řídicí systém, resp. celý balík CAD, CAM programu a řídicího systému[12]. Tento systém využívá nezávislou řídicí jednotku postavenou na ARM mikrokontroléru. Tato jednotka s počítačem komunikuje pomocí sériového portu, z nějž získává data a realizuje uživatelský vstup a výstup.

[1] Přehled všech dostupných řešení lze nalézt na `<http://www.machsupport.com/plugins.php>`

[2] `<http://www.warp9td.com/>`

[3] Podrobnosti o systému lze najít v manuálu: `<http://www.linuxcnc.org/docs/HAL_User_Manual.pdf>`

[4] `<http://www.mesanet.com/>`

Koncepcí se mi tento systém líbí a na základě něj jsem se i trochu inspiroval pro koncepci mého řídicího systému. Řídící jednotka generuje step-dir signály. Nikde na stránkách společnosti jsem však nenašel informace o použité akcelerační křivce. Rozhodl jsem se proto společnost kontaktovat se žádostí o podrobnější informace. Obdržel jsem velmi vstřícnou odpověď (kopie odpovědi se nachází v příloze A).

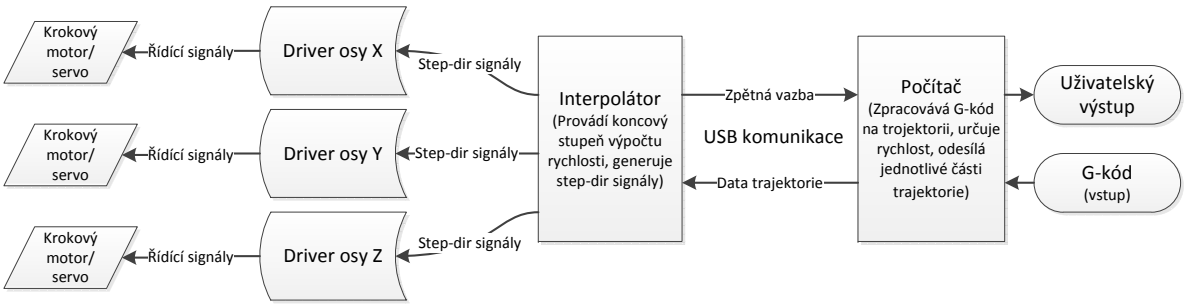
Pro obrábění se používají lineární akcelerační křivky, avšak pro rychloposuv je implementována i S-křivka. Na tomto systému mě překvapilo, jak elegantně byl vyřešen problém s dynamikou stroje pohybu po kruhovém oblouku. Do systému se zadávají dvě omezení maximálního zrychlení – jedno pro pohyb po přímce, druhé pro pohyb na oblouku o poloměru 1 mm, z nějž lze dopočítat zrychlení pro oblouk o libovolném poloměru. Ačkoliv se nejedná o fyzikálně podložené řešení, tak po experimentálním zjištění vhodného limitního zrychlení musí v praxi fungovat nadmíru dobře.

Zajímavý je také přístup k interpolaci kruhového oblouku. Veškeré křivky jsou rozloženy na sérii přímých úseků. Pokud se nastaví délka těchto úseků blízko, či dokonce pod rozlišovací schopnost pohonů, může systém naprosto přesně projet libovolnou křivku při zachování jednoduchosti interpolační jednotky.

Kapitola 3

Koncepce mého řídicího systému

Koncepci mého řídicího systému znázorňuje schéma 3.1. Řídicí systém se skládá ze dvou částí – počítače zajišťujícího uživatelský vstup, zpracování G-kódu na interní formát a výpočet mezních rychlostí. Druhou částí systému je nezávislý interpolátor, který přebírá již zpracované informace o jednotlivých úsecích a na základě nich generuje řídicí step-dir signály (provádí tedy samotnou interpolaci).



Obrázek 3.1: Schéma znázorňující uspořádání mého řídicího systému.

Použití interpolátoru postaveného na jednočipovém mikrokontroléru byla jasná volba – pouze tak lze docílit kvalitního výstupního signálu a spolehlivosti stroje. Na jednočipu je totiž možné narozdíl od počítače uhlídat přesné časování a generovat vyšší výstupní frekvence.

Jako komunikační rozhraní mezi počítačem a interpolátorem jsem zvolil USB, ani ne tak díky jeho rychlosti (nepřenáší se velká kvanta dat), ale díky jeho jednoduchosti, co se uživatele týče. Narozdíl od sériové linky nemusí uživatel vybírat správný port a má aplikace je schopna detekovat správné připojení zařízení. Navíc při použití integrované USB periférie v mikrokontroléru můžu na komunikaci pohlížet jako na „black box“ a nemusím se starat o integritu dat – data dostanu vždy správně napaketovaná tak, jak jsem je odeslal.

Narozdíl od řídicího systému Gravos by měl můj interpolátor umět generovat řídicí signály jak pro lineární, tak i pro obloukovou interpolaci. Z počítače jednotka obdrží pouze informace o typu úseku, jeho charakteristice (počáteční a koncový bod) a daných omezeních (počáteční a koncová rychlost, či maximální zrychlení).

Z výše uvedeného by se mohlo zdát, že role počítače je zde zbytečná – jednotka skoro vše obstarává sama. Počítač je zde zařazen hlavně kvůli komunikaci s uživatelem a jednoduchosti vývoje. Vyvinout uživatelské rozhraní pro počítač je mnohem jednodušší než se snažit připojit display k mikrokontroléru a vyvíjet od nuly. Navíc uživatelské rozhraní pro počítač může být

mnohem sofistikovanější (vzniká zde komfort při práci se soubory, či je velmi jednoduché provádět změny programu přímo u stroje).

Počítač zde navíc slouží jako výkonná výpočetní jednotka se spoustou paměti. Jelikož se chci přiblížit co nejideálnějšímu pohybu, je pro určení některých limitů třeba relativně složitých výpočtů. Díky počítači si můžu dovolit takový komfort, jako připravit si celou trajektorii stroje dopředu ještě před tím, než začne samotný pohyb stroje, a tak vyřešit všechny případné řetězové závislosti.

V následující části Fyzikální model (I) řeším teorii co nejideálnějšího pohybu a odvozuji vlastní fyzikální model pohybu. V části Realizace (II) se zaměřuji na implementaci odvozeného fyzikálního modelu, popisuji použitý hardware a návrh programu jak na straně počítače, tak i na straně interpolátoru.

Část I

Fyzikální model

Kapitola 4

Pojem fyzikální model a jeho vlastnosti

Fyzikálním modelem řídicího systému chápu seznam pravidel, či vzorců, dle kterého lze určit polohu stroje (jednotlivých os) v kterémkoliv okamžiku v čase t. Tudíž mou snahou je vyjádřit všechny potřebné veličiny v závisloti na čase t. G-kód definuje dva druhy pohybu[20] – pohyb přímočarý (po úsečce) a pohyb po kruhovém oblouku – pro každý je nutné vytvořit samostatný model.

Vstupem do fyzikálního modelu jsou limitující faktory a jednotlivé úseky (dílčí pohyby) zadané programem v G-kódu, po kterých se má stroj pohybovat. Každý dílčí pohyb se skládá ze tří částí – zrychlení z počáteční rychlosti v_0 na zadanou cílovou rychlost V, pohybu rovnoměrného přímočarého rychlostí V a zpomalení na danou brzdnou rychlost v_b. Ve speciálních případech může být vynechán rovnoměrný přímočarý pohyb, nebo rychlost v_0 (potažmo v_b) je rovna rychlosti V – tedy nedojde k zrychlování, respektive zpomalování.

Z pohledu fyzikálního modelu je nejdůležitější se zaměřit na zrychlování, resp. zpomalování, jelikož na pohybu rovnoměrném přímočarém není co řešit.

4.1 Vlastnosti ideálního fyzikálního modelu

Ideální fyzikální model by měl dosáhnout co nejplynulejšího a nejrychlejšího pohybu (v zájmu zrychlení obrábění). Zároveň však musí respektovat mechanické limity stroje. Jedním ze způsobů, jak toho docílit, je vhodně nastavit akcelerační křivky (průběh rychlosti v závisosti na čase) a tím omezit rázy. Rázem chápu prudkou změnu působícího zrychlení a tudíž i působící síly. Je tedy nutno určit, jak tyto mechanické limity definovat, což provádím v kapitole 5.1.

4.2 Fyzikální veličina ryv

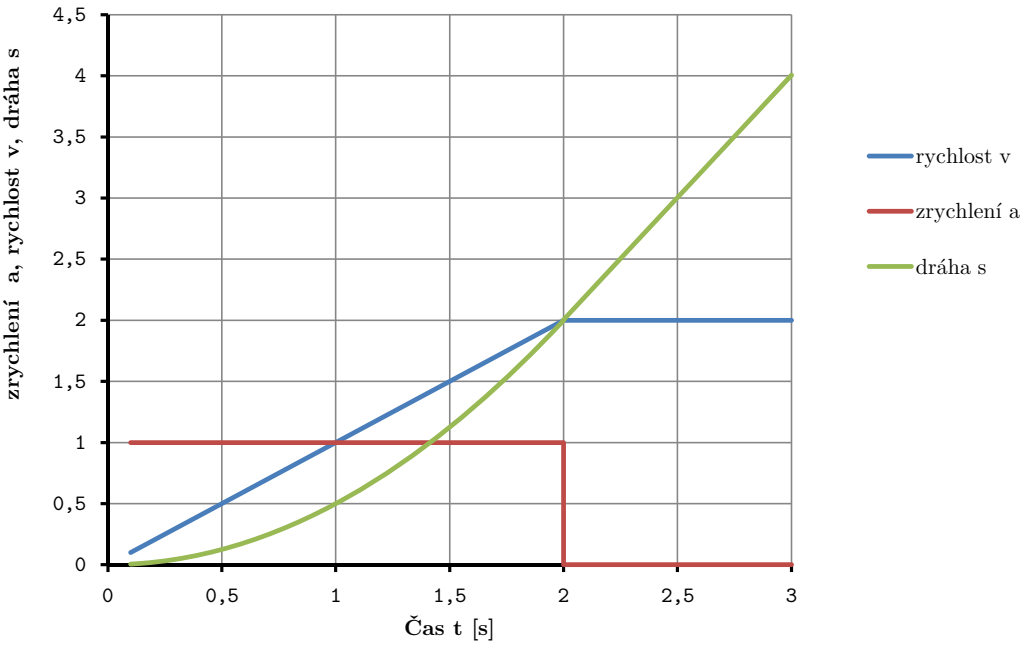
Běžně se pro charakteristiku pohybu používá rychlost a zrychlení. Pro popsání pohybu z hlediska dynamiky je však vhodné uvažovat ještě třetí veličinu – ryv (anglicky jerk), značený j[26]. Ryv je definován následovně[26]:

$$\mathbf{j} = \frac{\mathrm{d}\mathbf{a}}{\mathrm{d}t} = \frac{\mathrm{d}^2\,\mathbf{v}}{\mathrm{d}t^2} = \frac{\mathrm{d}^3\,\mathbf{d}}{\mathrm{d}t^3}, \tag{4.1}$$

kde $\mathbf{a}$ je zrychlení pohybu, $\mathbf{v}$ je rychlost pohybu a $\mathbf{d}$ je poloha tělesa v prostoru. Jednotkou ryvu je 1 $\mathrm{m{\cdot}s^{-3}}$. Ryv popisuje změnu zrychlení v čase. Název ryv byl odvozen od českého slova „poryv“[27] (anglického jerk – cukat[26]).

4.3 Akcelerační křivky

Akcelerační křivkou se běžně označuje graf závislosti rychlosti pohybu na čase. Na grafu 4.1 je znázorněna akcelerační křivka běžně používaného modelu zrychlení na hobby CNC strojích – za použití konstantního zrychlení.



Obrázek 4.1: Akcelerační křivka konstantního zrychlení.

Sklon křivky rychlosti (tedy derivace rychlosti) ukazuje aktuální zrychlení. Z Newtonova zákona síly vyplývá, že působící síla je přímo úměrná zrychlení. Pokud tedy zrychlení ustane (na grafu 4.1 v čase $t = 2\ s$), prudce ustane i působící síla a nastane ráz – prudké odlehčení zátěže ze stroje. Obdobná situace nastává i při nástupu zrychlení (zde v čase $t = 0\ s$). Tyto jevy nastávají při ostrém zlomu na grafu rychlosti v závislosti na čase.

Pokud se na situaci podíváme z pohledu ryvu, tak vidíme, že zrychlení je nespojitou funkcí času, tudíž jeho derivace v bodech nespojitosti není definována. To je však pouze teoretická situace. Ve skutečnosti je zrychlení stále spojitou funkcí, ovšem s velmi strmými přechody. Ryv zde tedy dosahuje obrovských hodnot (dokonce můžeme mluvit až o $j \to \infty$). Velikost ryvu charakterizuje tyto prudké změny zrychlení, a tedy i změnu zátěže.

Z výše uvedeného lze usoudit, že pro nejplynulejší pohyb je důležité, aby na grafu rychlosti v závislosti na čase nedocházelo ke zlomům. Tehdy bude docházet k postupnému nárůstu a úbytku zátěže na stroj. Takovýmto grafem rychlosti je tzv. S-křivka. Tato křivka je na počátku tečná k ose x a na svém konci tečná ke konstantní funkci

$$v(t) = V, \tag{4.2}$$

kde V je rychlost, které je třeba dosáhnout. Mnou sestrojená S-křivka je ukázána na grafu 5.1 v kapitole 5.2.1.

Jako vhodná S-křivka se ukazuje kosinusoida

$$y = 1 - \cos x, \tag{4.3}$$

která splňuje tečnost k funkcím $y = 0$ a $y = V$. Kosinusoida není jediná S-křivka. Obdobně můžou sloužit i některé polynomy. Pro mé účely je však kosinusoida výhodná – je matematicky snadno definovatelná, na celém svém průběhu je „hladká“ a symetrická; také její derivace libovolného stupně a integrály libovoného stupně jsou definovány.

Derivací této kosinusoidy je funkce

$$y' = \sin x. \tag{4.4}$$

Zrychlení pro docílení S-křivky ve tvaru kosinusoidy tedy musí mít sinusový průběh.

Je však také nutné omezit také strmost S-křivky – pokud by byla příliš strmá, lze ji považovat za podobnou s přímkou a její efekt by byl zanedbatelný. K tomuto omezení slouží právě ryv, který omezí strmost zrychlení, čímž omezí i strmost křivky rychlosti.

4.4 Postulát pohybu

Na základě zkušeností a úvahy v předcházející části 4.3 jsem si pro odvození fyzikálního modelu pro můj řídicí systém stanovil následující postulát:

„Pro ideální pohyb bez rázů musí mít zrychlení plynulý nástup a postupný pokles (v mém případě volím sinusový průběh). Průběh zrychlení je třeba limitovat jak amplitudou zrychlení, tak i maximálním přípustným ryvem.“

Kapitola 5

Odvození fyzikálního modelu

V následujícím odvození fyzikálního modelu pro můj systém vycházím z mého postulátu (kapitola 4.4). Pro přehlednost jsem si zavedl následující konvenci: okamžitý stav veličin značím malým příslušným písmenem (např. okamžité zrychlení a), maximální (omezující a obvykle vstupní) hodnoty označuji příslušným velkým písmenem (např. maximální amplituda zrychlení A).

5.1 Limitující faktory pohybu

Jak jsem zmínil v části 4.1, limitujícím faktorem dynamiky pohybu je kvalita (tuhost) mechanické konstrukce stroje. Většina systémů používá omezení maximálního zrychlení a maximální rychlosti. Do mého systému jsem přidal ještě omezení maximálního ryvu.

Můj systém používá sinusový průběh zrychlení, tudíž limituji amplitudu A tohoto zrychlení a špičkovou hodnotu ryvu J. U ryvu J se nesnažím docílit jeho přesného průběhu.

Narozdíl od běžných systémů jsem se rozhodl implementovat veškerá omezení pro celý stroj, nikoliv pro jednotlivé jeho osy. Jednak se v praxi u hobby strojů minimálně limity pro osy x a y nastavují zpravidla stejně, také se jedná o zjednodušení a také jde o jisté přiblížení k ideálnímu stavu (stroj by se ve směru všech os měl pohybovat stejně).

5.2 Pohyb po úsečce

Pohyb po úsečce je nejjednodušším možným pohybem na stroji. Z hlediska fyzikálního modelu se na každou část pohybu (zrychlování, pohyb rovnoměrný přímočarý a zpomalování) dívám jako na samostanou, tzn. každá začíná v čase $t = 0$. Pohyb rovnoměrný přímočarý není třeba popisovat, v následující sekci se proto zaměřím pouze na rozjezd a brzdění. V první části (5.2.1) odvozuji základní vztahy pro tento pohyb s omezením maximálního zrychlení. V druhé části (5.2.2) tyto vztahy rozšiřuji o omezení maximálního ryvu.

5.2.1 Rozjezd a brzdění

Rozjezd a brzdění jsou z fyzikálního pohledu naprosto shodné pohyby lišící se pouze směrem zrychlení. Lze na ně tedy uplatnit stejné vztahy.

Dle postulátu musí mít zrychlení pro tento pohyb sinusový průběh, resp. pro rozjezd je třeba kladná půlvlna, pro zpomalení naopak záporná půlvlna. Pokud označím celkovou dobu trvání tohoto rozjezdu, resp. zpomalení, T a maximální dosažené zrychlení A, dostanu následující vztah

pro okamžité zrychlení a v závislosti na čase t.

$$a = A \sin \frac{\pi t}{T} \tag{5.1}$$

Integrováním tohoto výrazu pro zrychlení dostanu vztah pro okamžitou rychlost v v závislosti na čase t:

$$v = \int a(t)\,\mathrm{d}t = -\frac{AT}{\pi} \cos \frac{\pi t}{T} + c \tag{5.2}$$

Nyní je však třeba dopočítat konstantu c (v čase $t = 0$ je rychlost $v = 0$) a vzorec doplnit o počáteční rychlost v_0:

$$\begin{aligned} \text{Pro } t = 0 \text{ platí } v = 0 \implies -\frac{AT}{\pi} + c &= 0 \\ c &= \frac{AT}{\pi} \\ v = \frac{AT}{\pi} \left(1 - \cos \frac{\pi t}{T}\right) &+ v_0 \end{aligned} \tag{5.3}$$

Zde stojí za povšimnutí, že pokud dosadíme zápornou amplitudu zrychlení A a místo počáteční rychlosti v_0 dosadíme brzdnou rychlost v_b, získáme platný vzorec pro zpomalený pohyb. Pro přehlednost jej však můžeme zapsat i následovně (s kladnou amplitudou):

$$v = v_b - \frac{AT}{\pi} \left(1 - \cos \frac{\pi t}{T}\right) \tag{5.4}$$

Tento získaný vztah pro rychlost (5.3) můžeme opět zintegrovat a získat tak vztah pro okamžitou dráhu pohybu s v závisloti na čase t, což je zároveň vzdálenost uražená od počátku pohybu.

$$\begin{aligned} s = \int v(t)\,\mathrm{d}t = \frac{AT}{\pi} t - \frac{AT^2 \sin \frac{\pi t}{T}}{\pi^2} &+ c \\ s = \frac{AT}{\pi^2} \left(\pi t - T \sin \frac{\pi t}{T}\right) & \end{aligned} \tag{5.5}$$

Dosazením $t = T$ do odvozeného vztahu pro okamžitou rychlost (5.3), dostanu vztah pro maximální dosažitelnou rychlost V za čas T při amplitudě zrychlení A (5.6), resp. celkovou dobu pohybu T při zadaném zrychlení A (5.7), resp. vztah pro potřebné zrychlení ke změně rychlosti z v_0 na V za čas T (5.8).

$$V = \frac{2AT}{\pi} + v_0 \tag{5.6}$$

$$T = \frac{\pi\,(V - v_0)}{2A} \tag{5.7}$$

$$A = \frac{\pi\,(V - v_0)}{2T} \tag{5.8}$$

Obdobně jako v předchozím případě lze za v_0 dosadit v_b a získat tak vztahy pro zpomalování.

Stejným dosazením ($t = T$) lze získat i vztah pro celkovou dráhu pohybu pro zrychlování S (5.9), resp. brzdnou dráhu S_b (5.10):

$$S = \frac{AT^2}{\pi} + v_0 T \tag{5.9}$$

$$S_b = VT - \frac{AT^2}{\pi} \tag{5.10}$$

Z výše uvedených vztahů můžu sestavit vztah pro celkovou dráhu S pohybu po přímce složeného z rozjezdu a zpomalení – bez rovnoměrného přímočarého pohybu rychlostí V:

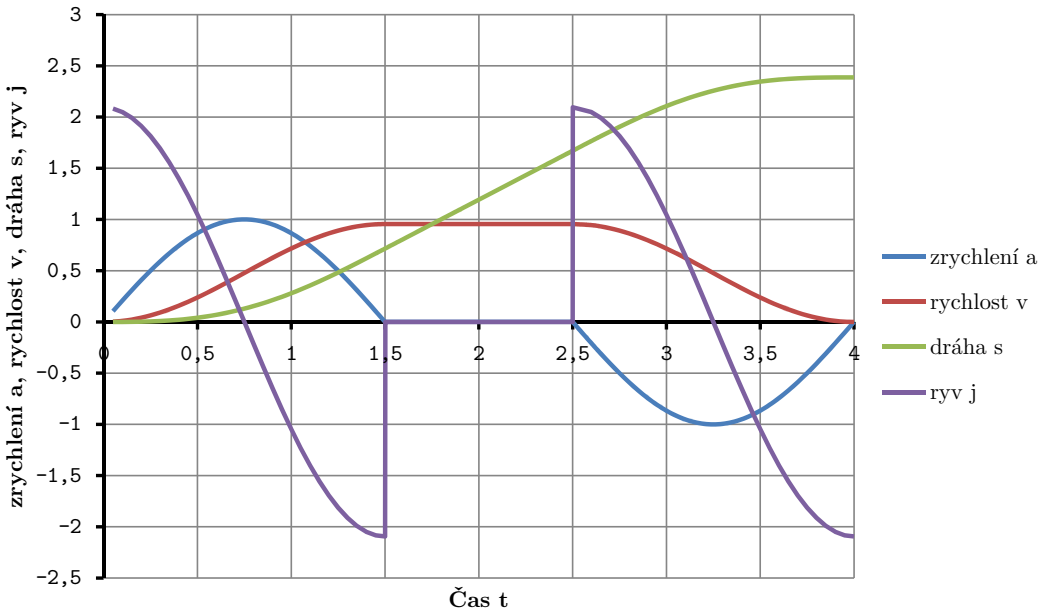
$$\begin{aligned} S &= \frac{AT^2}{\pi} + v_0 T + VT_b - \frac{AT_b^2}{\pi} \\ S &= \frac{\pi V\left(V - v_0\right)}{2A} + \frac{\pi\left(V - v_0\right)^2}{4A} + \frac{\pi V\left(V - v_b\right)}{2A} - \frac{\pi\left(V - v_b\right)^2}{4A} \end{aligned} \tag{5.11}$$

Z tohoto vztahu mohu vyjádřit rychlost V, což je vhodné pro nalezení maximální rychlosti, které lze dosáhnout na úsečce o délce S za omezení maximální amplitudou zrychlení A:

$$V = \frac{\sqrt{4AS + \pi\left(v_0^2 - v_b^2\right)}}{2\pi} \tag{5.12}$$

Tento vztah lze v praxi použít k nalezení nové maximální rychlosti, pokud požadované rychlosti nelze na daném úseku dosáhnout.

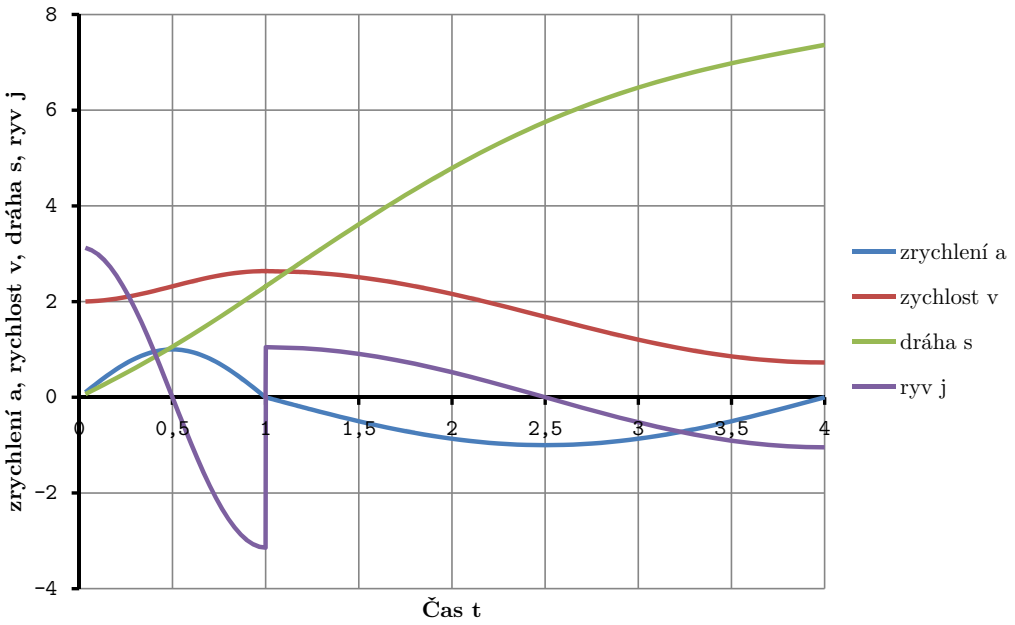
Na grafech 5.1 a 5.2 je zobrazen průběh jednotlivých veličin v závisloti na čase pro výše odvozené vztahy.



Obrázek 5.1: Graf znázorňující průběh jednotlivých veličin při použití sinusového zrychlení. Je složen z 1,5 sekundového rozjezdu z rychlosti $v_0 = 0$; 2 sekundového pohybu přímočarého a 1,5 sekundového brzdění na nulovou rychlost.

5.2.2 Omezení maximálního ryvu

Zrychlení není dle postulátu (kapitola 4.4) jediný omezující faktor. Pro ideální pohyb by měl být omezen i ryv. Ryv ve své aplikaci omezuji pouze jeho špičkovou hodnotou J, nikoliv jeho přesně daným průběhem.



Obrázek 5.2: Graf znázorňující průběh jednotlivých veličin při použití sinového zrychlení. Je složen ze sekundového rozjezdu z rychlost $v_0 = 1$ a 2 sekundového brzdění na brzdnou rychlost v_b

Jelikož ryv je definován následovně[26]:

$$j = \frac{\mathrm{d}a}{\mathrm{d}t}, \tag{5.13}$$

lze jeho závislost na čase t pro tento konkrétní příklad napsat následovně:

$$j = \frac{A\pi}{T} \cos \frac{\pi t}{T} \tag{5.14}$$

Průběh ryvu je znázorněn na grafech 5.1 a 5.2.

Maximum výrazu pro ryv (5.14) je v čase $t = 0$, resp. $t = T$ (zajímá nás i maximální výchylka ryvu do záporných hodnot). Maximální hodnora ryvu J je tedy potom rovna:

$$J = \frac{\pi A}{T} \tag{5.15}$$

Vyjádřením A z tohoto vztahu a jeho dosazením do vztahu pro dobu trvání pohybu T (5.7) získám omezení pohybu ryvem:

$$T = \frac{\pi}{2} \sqrt{\frac{2\,(V - v_o)}{J}} \tag{5.16}$$

Zde si je nutno povšimnout, že v tomto vztahu nevystupuje maximální zrychlení A, tudíž tento výpočet jím není omezen. Při výpočtu je tedy nutné prvně zkusit spočítat dobu trvání pohybu T za omezení ryvu a na základě ní dopočítat maximální zrychlení A. Pokud vyjde hodnota A větší než zadaná, je výpočet nutno provést znovu, tentokrát s použitím vztahu 5.7 a výpočet místo ryvu limitovat zrychlením.

Stejně jako v případě omezení pohybu zrychlením, i zde lze sestavit vztah pro celkovou dráhu pohybu složeného z rozjezdu a zpomalení:

$$S = \frac{\pi}{2\sqrt{2}} \left(v_b \sqrt{\frac{V - v_b}{J}} - v_0 \sqrt{\frac{V - v_0}{J}} + V \left(\sqrt{\frac{V - v_b}{J}} + 3\sqrt{\frac{V - v_0}{J}} \right) \right) \tag{5.17}$$

Z této rovnice však nelze získat obecné vyjádření maximální rychlosti V, jelikož se jedná o iracionální rovnici (postupným umocňováním získáme polynom 8. stupně). Je třeba ji řešit numericky.

V prvních verzích řídicího systému jsem tento vztah k nalezení nové rychlosti používal a řešil jsem jej Newtonovou metodou, avšak v poslední verzi jsem jej nahradil hledáním nové rychlosti pomocí bisekce bez použití tohoto vztahu (viz 8.6). U tohoto vztahu je nutné vytvořit jeho několik variant (kombinace rozjezd omezen ryvem, zpomalování zrychlením apod.), aby dával korektní výsledky. Jeho výpočetní náročnost je zhruba totožná s bisekcí (pokud nejsou rozdíly rychlostí přiliš velké, což v praxi nenastává). Bisekce je dle mě však také elegantnější řešení, které se v kódu lépe čte. Proto jsem se k ní nakonec přiklonil.

5.3 Pohyb po kruhovém oblouku

Pohyb pro kruhovém oblouku je z hlediska dynamiky na první pohled podobný úsečce. Je zde však nutné uvažovat i vznikající dostředivé zrychlení, které způsobuje změnu směru pohybu. Zadané omezení špičkového zrychlení tedy neomezuje tečné zrychlení ve směru pohybu, nýbrž velikost jeho vektorového součtu se vznikajícím dostředivým zrychlením.

Stejně jako v kapitole Pohyb po úsečce (5.2.1) i zde se výsledný pohyb skládá ze tří částí. Jelikož je však rozjezd a zpomalování totožné (liší se pouze směrem zrychlení) a pohyb konstantní rychlostí triviální, zaměřím se v této sekci pouze na rozjezd.

Pro zachování souladu s dříve odvozenými vztahy, kde A reprezentuje zadané omezení zrychlení a současně reprezentuje maximální velikost tečného zrychlení, budu i nadále označovat maximální velikost tečného zrychlení A. Avšak jako omezující faktor zde budu používat velikost celkového zrychlení A_k. A_k je tedy vstupním parametrem a zrychlení A závisí na charakteristice pohybu. Mým cílem je proto ze zadaných omezení J a A_k dopočítat hodnoty T a A. Není třeba se zajímat o uraženou dráhu, jelikož tento vztah je po dosazení přislušných hodnot A a T stejný jako u úsečky (vztah 5.9).

Jak jsem již zmínil, celkové zrychlení je rovno vektorovému součtu tečného a dostředivého zrychlení. Jelikož se jedná o pohyb po kružnici, jsou tato zrychlení na sebe navzájem kolmá, a proto platí:

$$\mathbf{a_k} = \mathbf{a} + \mathbf{a_d} \implies |a_k| = \sqrt{|a|^2 + |a_d|^2} \tag{5.18}$$

Dostředivé zrychlení pro pohyb na kružnici o poloměru r je rovno

$$a_d = \frac{v^2}{r}, \tag{5.19}$$

a proto po dosazení vztahů 5.1 a 5.3 do vztahu 5.18 dostanu vyjádření okamžité velikosti zrychlení a_k v závislosti na čase t:

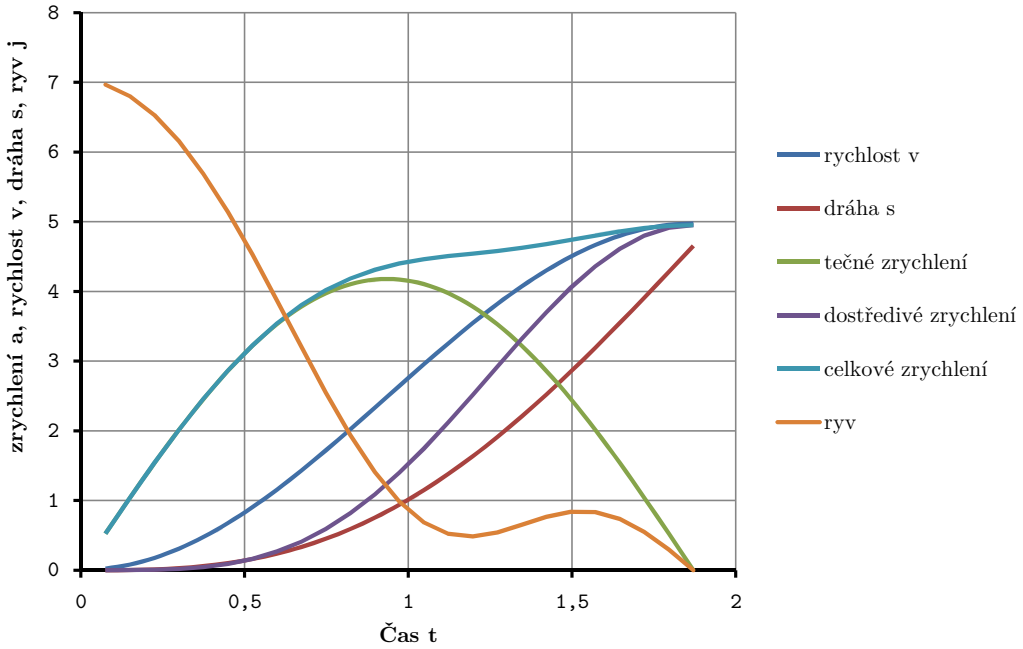
$$a_k = \sqrt{\frac{A^4 T^4 \left(\cos\frac{\pi t}{T} - 1\right)^4}{\pi^4 r^2} + A^2 \sin^2 \frac{\pi t}{T}} \tag{5.20}$$

Derivací tohoto výrazu podle času dostanu vztah pro okamžitý ryv v závisloti na čase t:

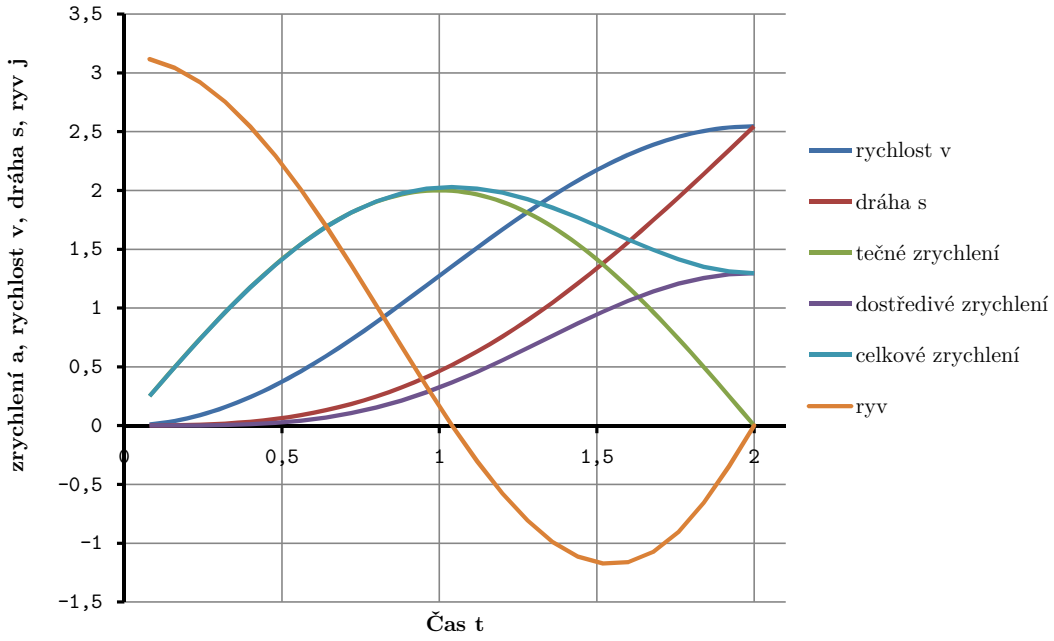
$$j = \frac{\mathrm{d}a_k}{\mathrm{d}t} = \frac{32A^4T^4 \sin^6 \frac{\pi t}{2T} \sin \frac{\pi t}{T} + A^2\pi^4 r^2 \sin \frac{2\pi t}{T}}{2\pi r^2 T \sqrt{\frac{A^4T^4(\cos \frac{\pi t}{T} - 1)^4}{r^2} + A^2\pi^4 \sin^2 \frac{\pi t}{T}}} \tag{5.21}$$

Pro další pokračování je nutno u těchto vztahů (5.20 a 5.18) nalézt jejich maxima. Porovnáním jejich derivací s nulou jsem získal složité rovnice, které jsou řešitelné pouze numericky. Tomuto řešení jsem se chtěl vyhnout, a proto jsem začal zkoumat průběhy pro různé kombinace A a T.

Jak lze vidět na grafech 5.3 a 5.4, mohou nastat dvě situace – maximum zrychlení se nachází v čase $t = T$, nebo přibližně v čase $t = \frac{T}{2}$. Oběma případům se věnuji v následujících sekcích 5.3.1 a 5.3.2.



Obrázek 5.3: Graf znázorňující průběh jednotlivých veličin při rozjezdu na oblouku. Maximální hodnota zrychlení se nachází v čase $t = T$.



Obrázek 5.4: Graf znázorňující průběh jednotlivých veličin při rozjezdu na oblouku. Maximální hodnota zrychlení se nachází přibližně v čase $t = \frac{T}{2}$.

5.3.1 Maximální zrychlení v čase $t = T$

Tato situace nastává, pokud je maximální dosažitelná rychlost V rovna:

$$A_k = \frac{V^2}{r} \implies V = \sqrt{A_k r} \tag{5.22}$$

Zároveň se však jedná i o podmínku – při pohybu po oblouku nelze dosáhnout vyšší rychlosti než výše uvedené – požadované dostředivé zrychlení by bylo větší než zadané maximální zrychlení.

Jelikož zde neznám hodnotu zrychlení A, je třeba pohyb limitovat ryvem. Stejně jako v případě celkového zrychlení a_k daného vztahem 5.20, je i v příadě vztahu pro ryv j (5.21) nemožné najít obecné řešení pro jeho maximum. Lze však vypozorovat, že maximum ryvu j se nachází buď v čase $t = 0$, anebo se nachází přibližně na intervalu $t = \langle 0{,}68T; 0{,}78T\rangle$ (tyto hodnoty byly experimentálně vypozorovány).

Pro maximum ryvu j v $t = 0$, lze použít stejné vztahy jako pro pohyb po úsečce – konkrétně vztah 5.16 pro výpočet doby pohybu T a pro výpočet A vztah

$$A = \frac{\pi\sqrt{A_k r}}{2T}, \tag{5.23}$$

který vychází z vyjádření zychlení 5.20 a dosazení $t = T$.

Pokud se maximum ryvu nachází na intervalu $t = \langle 0{,}68T; 0{,}78T\rangle$, rozhodl jsem se pro zjednodušení celého výpočtu použít hodnotu maxima pro $t = \frac{3T}{4}$, nikoliv průměr krajních hodnot $0{,}73T$ – získám elegatnější tvar koeficientů, které vzniknou z funkcí sinus a cosinus.

Dosazením $t = \frac{3T}{4}$ do vztahu pro okamžitý ryv j (5.21) dostanu vztah pro maximální ryv J. Pokud k němu přidám základní vztah pro dobu pohybu T (5.7), získám tuto soustavu rovnic:

$$\begin{cases} J = \frac{A^4T^4\left(7+5\sqrt{2}\right)-A^2\pi^4r^2}{\pi rT\sqrt{2A^2\pi^4r^2+A^4T^4\left(17+12\sqrt{2}\right)}} \\ T = \frac{\pi(V-v_0)}{2A} \text{ také lze zapsat jako: } T = \frac{\pi(\sqrt{A_k r}-v_0)}{2A} \end{cases} \tag{5.24}$$

Tato soustava je řešitelná pouze numericky. Na její řešení jsem použil Newtonovu metodu a přesný postup výpočtu uvádím v kapitole 8.6.2.

5.3.2 Maximální zrychlení v čase $t = \frac{T}{2}$

Tento případ nastává, pokud je požadovaná rychlost

$$V < \sqrt{A_k r}. \tag{5.25}$$

Nyní pokud dosadím do vyjádření celkového zrychlení (5.20) $t = \frac{T}{2}$ a přidám němu vyjádření času (5.7) získám soustavu:

$$\begin{cases} A_k = \frac{A\sqrt{\pi^4 r^2 + A^2 T^4}}{\pi^2 r} \\ T = \frac{\pi(V-o)}{2A} \end{cases}, \tag{5.26}$$

jejímž reálným řešením pro A a T jsou výrazy:

$$\begin{aligned} A &= \frac{16A^2r^2 - (V - v_0)^4}{4r} \\ T &= \frac{2\pi r\,(V - v_0)}{16A^2r^2 - (V - v_0)^4} \end{aligned} \tag{5.27}$$

Tím je vyřešeno omezení pohybu z pohledu maximálního dosažitelného zrychlení.

Pro omezení ryvu je třeba najít jeho maxima. První maximum se opět nachází v $t = 0$ a druhé maxímum opět přibližně v $t = \frac{3T}{4}$ – tentokrát však se zápornou hodnotou, protože zrychlení zde klesá (jak lze vidět na grafu 5.4).

Pokud se maximum nachází v čase $t = 0$, platí opět stejné vztahy, jako v případě pohybu po úsečce (vztahy 5.16).

Pokud se maximum nachází v $t = \frac{3T}{4}$, platí vztah 5.24, avšak s opačnou výchylkou ryvu, konkrétně tedy:

$$\begin{cases} -J = \frac{A^4T^4\left(7+5\sqrt{2}\right) - A^2\pi^4r^2}{\pi r T\sqrt{2A^2\pi^4r^2 + A^4T^4\left(17+12\sqrt{2}\right)}} \\ T = \frac{\pi(V-v_0)}{2A} \end{cases}$$

To jsou všechny vztahy nutné k popisu rychlosti při pohybu po kruhovém oblouku. Jejich použití ukazuji v kapitole 8.6.2.

5.3.3 Problém pohybu po kruhovém oblouku

Pohyb po kruhovém oblouku se v praxi používá k obrábění vnějších rohů obrobku (viz kapitola Implementace korekce nástroje 8.5). Cílem je zrychlit čas obrábění – oblouk navazuje na přislušné dráhy nástroje tečně, tudíž se běžně projíždí bez brzdění. Zde však nastává ráz – aby se nástroj pohyboval po kruhovém oblouku, musí na něj okamžitě začít působit dostředivé zrychlení. Oblouk bez rázu lze projet pouze pokud se těsně před ním zabrzdí na nulovou rychlost a až teprve na oblouku se stroj začne rozjíždět.

Jedná se však o natolik běžně používaný postup, že jej i můj řídící systém respektuje a tečně navazující oblouky projíždí bez brzdění i za cenu vzniklého rázu.

5.4 Pohyb po ideální křivce

Jak jsem zmínil v předcházející sekci, nelze kruhový oblouk projet bez rázu. Rozhodl jsem se proto najít křivku podobnou kruhovému oblouku, kterou lze projet bez brzdění za dodržení všech pravidel postulátu.

Jak vyplývá z postulátu, musím působit zrychlením ze sinovým průběhem. Také je vhodné, aby toto zrychlení neměnilo směr a nemusel jsem ještě uvažovat nutné normálové zrychlení. Po vyzkoušení několika variant jsem dospěl k uspořádání, které ukazuji na obrázku 5.5.

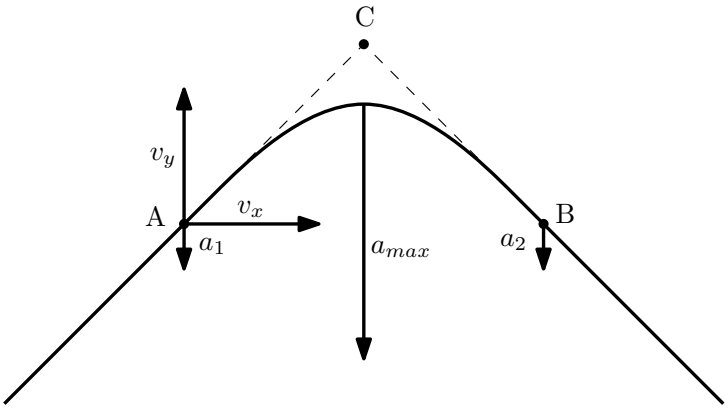
Zrychlení v tomto případě působí kolmo na spojnici A a B. Poté mohu rozložit rychlost pohybu na složky v_y a v_x (opět ve směru kolmém, respektive rovnoběžném se spojnicí AB). Rychlost v_x nebude ovlivněna zrychlením – v tomto směru se tedy stroj bude pohybovat pohybem rovnoměrným přímočarým. Naopak ve směru rychlosti v_y je stroj nejprve plynule zastaven a poté se rozjíždí. Výslednou opsanou křivkou je sinusoida. Jelikož je tato křivka symetrická, tak rychlost v bodě A je stejná jako v bodě B, pouze se změnil směr pohybu.

Pro určení maximálního zrychlení A lze použít vztah pro maximální zrychlení na přímce (5.8) – zajímá nás totiž pouze změna zrychlení ve směru rychlosti v_y. Dobu pohybu určíme z rychlosti v_x – stroj musí urazit vzdálenost $|A - B|$ rychlostí v_x, tedy:

$$T = \frac{|A - B|}{v_x} \tag{5.28}$$

Změna rychlosti, výraz $V - v_0$ v původním vztahu, je rovnen $2v_y$, jelikož rychlost v_y se změní na opačnou. Celkově tedy platí:

$$A = \frac{\pi v_x v_y}{|A - B|} \tag{5.29}$$



Obrázek 5.5: Obrázek znázorňující „ideální křivku“ – sinusoidu. Zrychleními a_1, a_{max} a a_2 je zde naznačen sinusový průběh zrychlení.

Tato křivka je však nevhodná pro obrábění – upřednostňuje totiž dynamiku pohybu před přesností obrábění. Původně jsem zamýšlel použít tuto křivku k „zakulacení“ rohů pohybu rychloposuvem, kde by nepřesnost nemusela vadit. Avšak nemůžu garantovat, že i při rychloposuvu by zůstal dostatek místa pro zkreslení trajektorie tak, aby stroj nenaboural do obrobku, jelikož CAM program toto zkreslení nemůže předpokládat – ač bývá zvykem při rychloposuvu nechávat co nejvíce místa mezi strojem a obrobkem. Proto jsem od myšlenky implementovat tuto křivku upustil a ani jsem ji nějak dále nerozvíjel – proto jsem ji nezkoumal z hlediska omezení ryvu a nalezení mezních rychlostí.

Část II

Realizace

Kapitola 6

Použité prostředky a hardware

6.1 Interpolátor

Jednotku interpolátoru jsem se rozhodl postavit na STM32F4 Dicovery kitu. Tento vývojový kit je postaven na STM32F407VGT6 ARM mikrokontroléru[3]. Kit obsahuje integrovaný debugger. Deska je osazena i dalšími zajímavými komponentami (akcelerometrem, audio DAC, či mikrofonem), ty však nejsou pro můj projekt využité.

Tento vývojový kit jsem se rozhodl použít pro jeho výkonný mikrokontrolér s bohatými perifériemi. Jádro mikrokontroléru je schopné běžet až na 168 MHz, což jej společně s integrovanou FPU (floating-point unit) a 192 kB RAM předurčuje pro aplikace vyžadující vysoký výkon. Tento kit také disponuje osazeným PHY pro USB FS aplikace využívající integrovanou USB-OTG periférii na mikrokontroléru, což byl jeden z mých požadavků. Jako poslední rozhodující faktor pro tuto desku byla její příznivá cena – 329 Kč[1].

Pro vývoj aplikací na kitu bylo použito IDE Atollic TrueSTUDIO Lite 2.2.0. (Nejedná se o nejnovější verzi tohoto IDE – při započetí vývoje byla nejnovější, avšak na novou verzi jsem neupgradoval, protože v ní přibylo omezení velikosti výsledného kódu). Toto IDE je přímo navrženo pro spolupráci s tímto vývojovým kitem. Ve verzi Lite (2.2.0), která je dostupná zdarma, podporuje pouze překladač jazyka C (nikoliv C++) a omezuje maximální počet breakpointů při debuggování na dva, avšak bez omezení velikosti výsledného kódu.

Celá aplikace pro interpolátor byla napsána v jazyce C s využítím StdPeriph knihovny od STMicroelectronics pro obsluhu periferií mikrokontroléru a knihovny USB-Host-Device pro obsluhu USB-OTG periférie.

6.2 Aplikace pro počítač

Aplikaci pro počítač jsem se rozhodl vyvíjet pro Windows. Na základě tohoto rozhodnutí jsem zvolil použité technologie.

Aplikaci jsem vyvíjel v jazyce C++ s použitím frameworku WxWidgets[2] a knihovny WinUSB[3] pro komunikaci s interpolátorem. Jazyk C++ jsem použil, jelikož s ním mám největší zkušenosti, navíc se v něm velice dobře implementoval relativně nízkoúrovňový protokol komunikace s interpolátorem (popsaný v kapitole 7.1.1). Knihovnu WxWidgets jsem použil, abych

[1] 15. 2. 2013 ve Farnell electronics <http://cz.farnell.com/stmicroelectronics/stm32f4discovery/stm32f407-usb-otg-discovery-kit/dp/2009276>

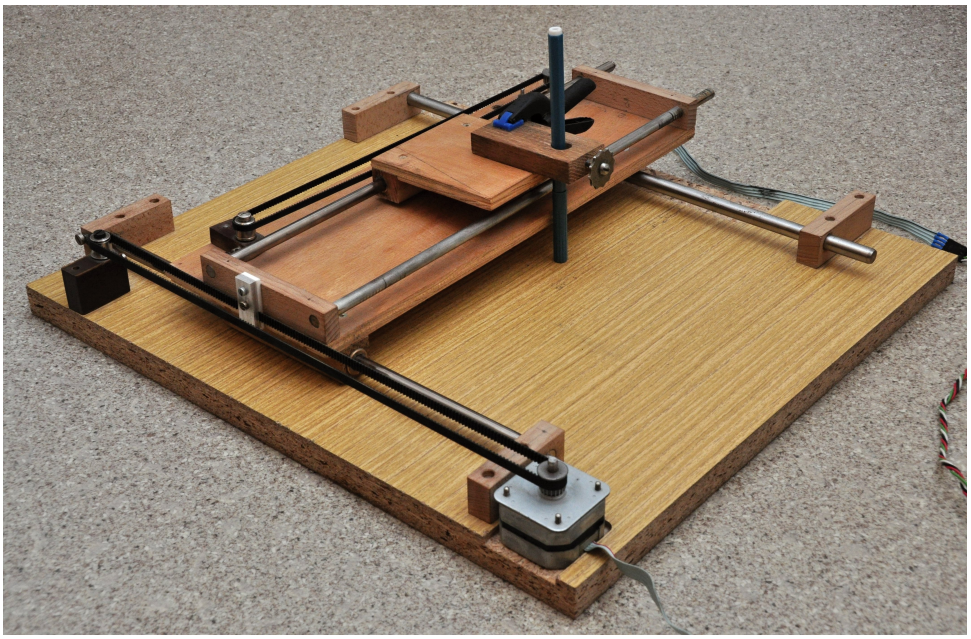
[2] <http://www.wxwidgets.org/>

[3] <http://msdn.microsoft.com/en-us/library/windows/hardware/ff540196(v=vs.85).aspx>

zachoval „look'n'feel“ mé platformy (Windows). Jelikož mým primárním cílem nebylo vyvíjet multiplatformní aplikaci, zvolil jsem knihovnu WinUSB před knihovnou LibUSB díky její nativní podpoře v systému Windows (od verze XP SP2[9]).

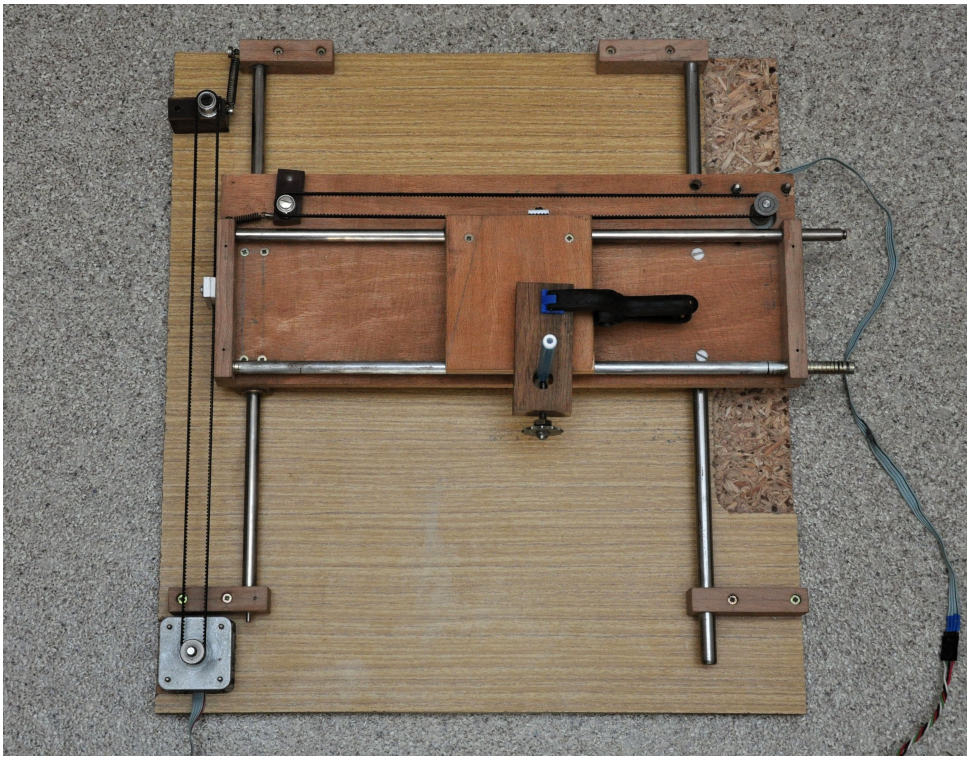
6.3 Testovací stroj

Aby bylo možné zařízení odzkoušet, rozhodl jsem si postavit jednoduchý „plotter“. Tento plotter (obrázek 6.1 a 6.2) je postaven na dřevotřískové desce. Jako vedení slouží broušené tyče ze staré inkoustové tiskárny, odkud byla použita i bronzová kluzná pouzdra na tyto tyče. K pohonu slouží dva bipolární krokové motory opět ze staré inkoustové tiskárny. Tyto motory pohybují strojem za pomoci ozubených řemenů. S návrhem a výrobou tohoto plotteru mi pomáhal můj otec, za což mu děkuji.

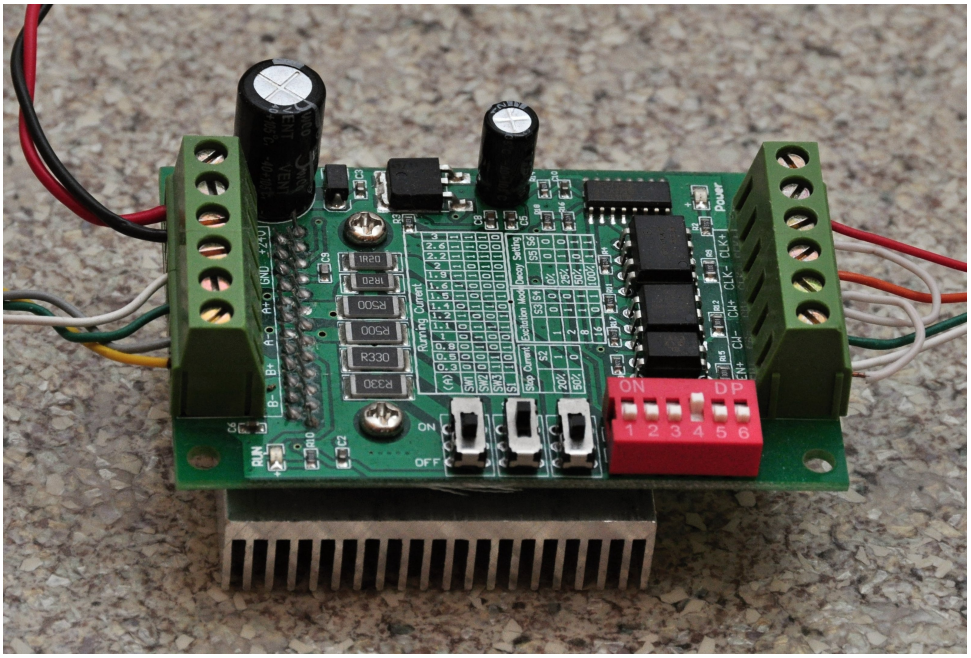


Obrázek 6.1: Fotografie plotteru, na kterém byl systém testován

Motory jsou řízeny čínskými „no-name“ jednoosými drivery postavenými na integrovaném obvodu Toshiba TB6560[21]. Mezi uživateli C-N-C.cz fóra se běžně označují jako „čínská zelená jednoosá deska“. Existuje více mutací tohoto driveru, pro jeho přesnou identifikaci přikládám fotografii (obrázek 6.3). Tyto drivery jsou řízeny step-dir signálem.



Obrázek 6.2: Fotografie plotteru, na kterém byl systém testován



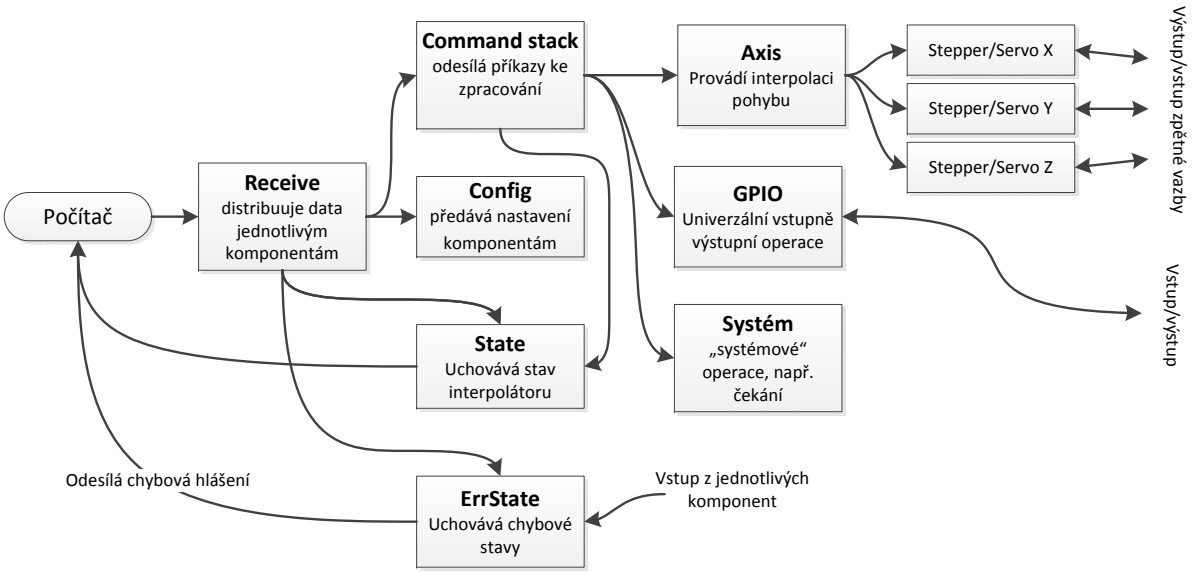
Obrázek 6.3: Fotografie použitých driverů krokových motorů

Kapitola 7

Software pro interpolátor

Software pro interpolátor je primárně závislý na příchozí USB komunikaci a interpolaci. Veškerou sérii úkonů jsem se rozhodl rozdělit do jednolivých komponent. Komponenty nejsou reprezentovány objekty, jelikož software je napsán v jazyce C. Jelikož existuje pouze jediná instance od každé komponenty, rozhodl jsem se každou komponentu reprezentovat jako soubor globálních funkcí pracujících nad svými globálními proměnnými.

Rozvržení a provázání jednotlivých komponent je znázorněno na schématu 7.1.



Obrázek 7.1: Schéma znázorňující rozvržení softwaru pro interpolátor

Základní komponentou celé aplikace je komponenta `Receive`. Tato komponenta zachytává přicházející pakety z USB, skládá je dohromady a nasledně tato přijatá data předává dále příslušným komponentám.

Komponenta `Config` přijímá zprávy týkající se nastavení a na základě přijatých dat volá funkce pro nastavení příšlušných komponent.

Komponenta `State` uchovává informace o současném stavu interpolátoru a podává zpětnou vazbu počítači. Obdobně komponenta `ErrState` uchovává informace o chybovém stavu jednotlivých komponent a podává zpětnou vazbu.

Jednou z nejdůležitějších komponent je `CommandStack`. Tato komponenta přebírá veškeré příkazy pro stroj a dle příznaku je buď řadí do fronty a postupně vykonává nebo je vykoná

přímo hned po přijetí. Příkazy pohybu zpracovává komponenta Axis, která provádí samotnou interpolaci. Zbývající dvě komponenty (`GPIO` a `Global`) zpracovávají ostatní druhy příkazů – např. prodlevu v programu a podobně.

7.1 Implementace USB a komunikační protokol

Pro komunikaci mezi počítačem a interpolátorem používám integrovanou periférii v mikrokontroléru. Pro její obsluhu jsem použil knihovnu dodávanou výrobcem – STMicroelectronics – USB Host-DeviceLibrary.

Jelikož počítač umí v USB komunikaci vystupovat pouze jako host, nakonfiguroval jsem kit jako device. Mým cílem bylo vytvořit vendor-defined device, které bude kromě control enpointu pro komunikaci využívat jeden vstupní a druhý výstupní bulk endpoint – tedy bude tvořit sériovou linku. Kit s počítačem komunikuje rychlostí Full Speed (pro rychlost High Speed není na kitu příslušné PHY). Délku paketu jsem použil nejdelší možnou pro Full Speed komunikaci – 64 bytů.

Jelikož jsem měl problémy s nastavením projektu tak, aby se knihovna pro obsluhu USB periférie zkompilovala, vyšel jsem pro mou aplikaci z projektu vytvořeného v článku „Začínáme s STM32F4Discovery 7“[7] od uživatele s přezdívkou Mard na serveru mcu.cz. Vstupní projekt jsem upravil tak, aby se zařízení tvářilo jako vendor-defined. O této úpravě a mém prvním sestrojeném demu USB komunikace byl napsán článek na témže serveru[1].

7.1.1 Komunikační protokol a komponenta Receive

Jak vyplývá z hiearchie jednotlivých komponent, komunikační protokol je vrstvený. V nejvyšší vrstvě, kterou zpracovává komponenta `Receive`, je ošetřeno rozdělení dat do více paketů (v případě, že celková délka dat přesáhne 64 bytů). Veškerá komunikace probíhá binárně – jednak abych ušetřil přenášená data, tak také, aby zpracování dat v interpolátoru proběhlo co nejrychleji. Pokud budu čísla posílat v binárním tvaru (což si můžu dovolit, jelikož platforma x86, resp. x64, je little-endian a platforma ARM umí pracovat v obou režimech – little i big-endian[22]), tak mi stačí pouhé ukládání dat a nemusím parsovat textový formát.

První byte prvního přijatého paketu obsahuje informaci o počtu paketů, do kterých je rozdělena právě přijímaná zpráva. Za tímto bytem následuje další byte, který definuje, které komponentě je zpráva určena. Seznam těchto kódů je definován v hlavičkovém souboru `CommunicationEnumerations.h`. Všechny pakety, které jsou přijaty, jsou předány funkci ProcessPacket. Tato funkce při přijetí prvního paketu vyčte délku zprávy n v paketech, nakopíruje ji do bufferu a dalších $n-1$ přijatých paketů nakopíruje do bufferu za sebou. Jakmile je přijato všech n paketů, zavolá funkci DistributeData, která předá přijatou zprávu dalším komponentám (na základě druhého bytu).

Každá komponenta definuje svou funkci ProcessMessage (tedy např. `CommandStackProcessMessage`), která zajistí zpracování zprávy dané komponenty. Tvar zprávy pro dané komponenty popisuji v následujících sekcích.

Pro komunikaci s počítačem (pro zpětnou vazbu) je použit stejný princip na rozdělení zprávy do paketů. Druhý byte zprávy však nyní neurčuje danou komponentu (tato hiearchie na straně počítače neexistuje), nýbrž identifikátor zprávy. Všechny možné zprávy pro počítač jsou definovány v hlavičkovém souboru `MessageCodeToPC.h`.

[1] `<http://mcu.cz/comment-n2904.html>`

7.2 Komponenta CommandStack

Komponenta `CommandStack` má za úkol spouštět přijaté příkazy. Název této komponenty jsem bohužel zvolil jako trochu zavádějící. CommandStack reprezentuje datový typ fronty – příkaz přijatý jako první se vykoná jako první.

Příkazy jsou dvojího druhu – jedny jsou určeny pro okamžité vykonání po jejich přijetí a druhé jsou prvně umístěny do bufferu a postupně se vykonávají. `CommandStack` je tedy implementován jako kruhový buffer. Do tohoto bufferu jsou ukládány prvky typu `CommandStruct`. Tato struktura reprezentuje jakýkoliv vykonatelný příkaz. `CommandStack` také udržuje informace o aktuálně vykonávaném a posledním vykonaném příkazu.

Rozhraním pro tuto komponentu je funkce StartPerforming. Tato funkce je po inicializaci všech komponent po startu interpolátoru volána v nekonečné smyčce. V této funkci je vždy vyjmut z bufferu první příkaz a je poslán ke zpracování.

7.2.1 Struktura CommandStruct

Struktura CommandStruct je definována následně:

```
typedef struct
{
    uint8_t receiver;//Receiver component for command
    uint8_t type;//Type of command
    CommandID ID;//Unique ID for command
    union
    {
        uint32_t delay;//Delay in ms for waiting
        uint8_t LED;//Led argument for GPIO
        AxisLine line;//line argument for Axis component
        AxisSine sine;//sine arguments for Axis component
        AxisCircle circle;//circle argument for Axis component
    };
} CommandStruct;
```

V první části struktury jsou deklarovány členy, které jsou společné pro všechny příkazy. Mezi ně patří cílová komponenta, typ příkazu (enumerace pro tyto členy jsou opět definovány v souboru `CommunicationEnumerations.h`) a ID každého příkazu. V druhé části jsou pak v unionu uloženy datové struktury pro jednotlivé typy příkazů.

`ID` je definováno jako 32-bitové neznaménkové číslo. Každý příkaz má jedinečný identifikátor přidělený počítačem – používá se např. když interpolátor posílá počítači zprávu, který příkaz je právě vykonáván. Identifikátory není třeba recyklovat – pokud by každou milisekundu byl vykonán jeden příkaz, k přetečení tohoto datového typu by došlo za 49 dní, což je dostatečná rezerva.

7.2.2 Zpracování zpráv

CommandStack příjmá následující zprávy:

COMSTACK_RET_SPACE se dotazuje na prázdné místo v bufferu. Nemá žádné parametry. Odpovědí je `COMSTACK_FREE_SPACE_MESS`, která vrací 16-bitové číslo vyjadřující počet prázdných míst v bufferu.

COMSTACK_ADD_COM přidává do bufferu příkazy. Zpráva je složena z 8-bitového čísla určující počet zpráv k přidání. Za ním následuje n příkazů v podobě `CommandStruct`. Pokud není dostatek volného místa v bufferu, nastaví se chybový stav `ERR_COMSTACKOWERFLOW` a odešle se zpráva počítači.

COMSTACK_LAST_COM_ID se dotazuje na ID posledního vloženého příkazu. Odpovědí je `COMSTACK_LAST_ID`, která vrací ID posledního příkazu.

COMSTACK_LAST_ITEM_PROC vrací identifikátor posledního úspěšně dokončeného příkazu. Odpovědí je `LAST_PROC_ITEM_STACK`, která vrací ID posledního příkazu.

COMSTACK_CURRENT_ITEM_PROC vrací identifikátor právě prováděného příkazu. Odpovědí je `CUR_PROC_ITEM_STACK`, která vrací ID posledního příkazu.

COMSTACK_CLEAR vyprázdní celý buffer.

COMSTACK_START spustí vykonávání příkazů v bufferu.

COMSTACK_PAUSE zastaví vykonávání příkazů v bufferu.

7.3 Komponenta Axis

Komponenta `Axis` vykonává provádění veškerých příkazů pro pohyb. V této komponentě jsou definovány buffery, které uchovávají aktuální „virtuální“ pozici všech os. Na základě obsahu těchto bufferů řídí komponenta `Stepper`, popř. `Servo`, drivery pohonů.

Interpolace probíhá v předem dané obnovovací frekvenci. To má na starosti timer `TIM14`, který na základě nastavení, které je posláno z počítače, pravidelně volá funkci `UpdatePosition` pro vykonání dalšího kroku interpolace.

Příkaz pro pohyb vyvolává komponenta `CommandStack`, která volá funkci `ProcessAxisCommand`. Tato funkce jako argument přebírá strukturu `CommandStruct`. Na základě jejího obsahu připraví funkce `ProcessAxisCommand` příslušný druh interpolace. Funkce poté před svým ukončením čeká, než funkce `UpdatePosition` nastaví příznak označující dokončení pohybu.

7.3.1 Použití datového typu float

Před implementací této komponenty bylo důležité vhodně vybrat datové typy, ve kterých budou probíhat interpolační výpočty. Ačkoliv jsem zvažoval použití celočíselných datových typů a následnou implementaci fixed-point aritmetiky, rozhodl jsem se nakonec použít čísla s plovoucí desetinnou čárkou – konkrétně datový typ `float`.

Čísla s plovoucí desetinnou čárkou se pro podobné výpočty zpravidla nepoužívají, jelikož při jejich nesprávném použití můžou vznikat velké chyby a práce s nimi na mikrokontrolérech bývá pomalá. Jelikož však mnou vybraný mikrokontrolér disponuje hardwarovou FPU, je práce s datovým typem float téměř stejně rychlá jako s celočíselným datovým typem. Navíc nemusím řešit změnu řádu při operacích jako je násobení a dělení, či operace se dvěma čísly v rozdílných řádech. Veškeré výpočty, které provádím, jsou počítány absolutně – jejich vstupem jsou předem daná přesná data, nikdy se nezakládají na dříve vypočtené hodnotě – výpočty tedy nejsou

iterativní. V případě interpolace nepřičítám k aktuální pozici změnu, ale do bufferu ukládám součet počáteční pozice a uražené dráhy.

Existují však případy, kdy se nelze vyhnout iteračnímu výpočtu – jedním z případů je postupné obnovování hodnoty aktuálního času, kdy se k proměnné přičítá velmi malá hodnota. V tomto případě uchovávám a pracuji s hodnotou času jako s celočíselným typem a až těsně před použitím ve výpočtu převedu celočíselný datový typ na float. Stejně tak pracuji s uloženou pozicí krokového motoru.

7.3.2 Implementace funkce sinus

Celý fyzikální model je založen na funkcích sinus a cosinus. Tyto funkce jsou ve standardní knihovně implementovány za pomoci řad. Tato implementace je přesná, avšak velmi pomalá. Rozhodl jsem se proto naimplementovat tyto funkce pomocí vyhledávací tabulky.

Mou prioritou bylo dosažení co nejrychlejšího výpočtu a rozumné přesnosti. Jelikož má mnou vybraný mikrokontrolér dostatek flash paměti, připravil jsem si tabulku obsahující hodnoty pro kladnou půlvlnu sinu po krocích $\frac{\pi}{4000}$. Tato tabulka zabírá v paměti necelých 16 kB. Funkce cosinus je realizována za pomoci posunu v této tabulce.

Touto tabulkou jsem získal velmi rychlý výpočet pro hodnoty sinu s dostatečnou přesností.

7.3.3 Lineární interpolace

Lineární interpolace je využita pro realizaci funkcí G00 a G01, popř. jako část cyklů (např. G73 vrtací cyklus)[20]. Tato interpolace je předávána jako prvek line struktury CommandStruct. Line je typ AxisLine, který je definován následovně:

```
typedef struct
{
    float axis[3];
    float v0, v, vb;
    float As, Ab;
} AxisLine;
```

Pole `axis` reprezentuje novou polohu os, neboli cílový bod interpolace ze současné pozice. Členy `v0, v, vb` reprezentují počáteční rychlost, respektive cílovou a brzdnou. `As a Ab` reprezentují zrychlení použité pro rozjezd a brzdění. Veškeré jednotky, které stroj používá, používají jako jednotku délky 1 mm, tedy rychlost je v jednotkách 1 $\mathrm{mm \cdot s^{-1}}$, zrychlení v jednotkách 1 $\mathrm{mm \cdot s^{-2}}$.

Tato struktura je zpracována funkcí `ProcessAxisLine` (voláno funkcí `ProcessAxisCommand`). Tato funkce připraví nezbytná data tak, aby interpolace každé osy mohla probíhat nezávisle na ostatních. V této funkci nejprve určím vektor ve směru pohybu jako rozdíl konečného a počátečního bodu. Tento vektor znormalizuji a jeho vynásobením se zadaným zrychlením dostanu maximální velikost zrychlení pro každou osu. Stejnou proceduru provedu i s jednotlivými rychlostmi.

Nyní dopočítám dle odvozených vztahů v kapitole 5.2 čas a dráhu nutnou k rozjezdu. Na základě těchto dvou údajů poté určuji v jaké fázi se pohyb nachází, abych mohl jeho polohu počítat dle správných vztahů.

Pro interpolaci využívám vztah pro dráhu (vztah 5.5), který mi vyjadřuje okamžitou vzdálenost od počátku pohybu. V tomto vztahu je spousta konstantních výrazů pro daný pohyb, proto si tyto konstanty před začátkem pohybu předpočítám, abych je nemusel při každém kroku interpolace znovu počítat.

Všechny předpočítané údaje jsou uloženy ve dvou strukturách. Struktura `AxisMovementLineCommon` reprezentuje data shodná pro všechny osy.

```
typedef struct
{
    float T, TB, t;
} AxisMovementLineCommon;
```

Zde `T` vyjadřuje čas nutný k rozjezdu, `TB` čas nutný ke zbrzdění a `t` čas, jak dlouho trvá pohyb konstantní rychlostí.

Struktura `AxisMovementBufferLine` je přiřazena ke každé ose a uchovává jejich specifická data.

```
typedef struct
{
    float* position;
    float ATpi, ATpiB;
    float start, constant, brake, end;
    float v, v0;
} AxisMovementBufferLine;
```

`position` je ukazatel na float uchovávající aktuální pozici osy, `ATpi` (`ATpiB`) je předpočítaná hodnota výrazu $\frac{AT}{\pi^2}$ (respektive jeho varianty pro brzdění). `start, constant, brake, end` znamenají postupně: pozici osy pro počátek; místo, odkud se osa pohybuje konstantní rychlostí; místo, odkud začíná osa brzdit; cílová pozice. `v` reprezentuje konstantní rychlost, `v0` počáteční.

Interpolace probíhá následovně. Prvně se pro všechny osy inkrementuje čas. Dále probíhá interpolace pro každou osu samostatně – na základě času se určí, v jaké fázi se pohyb nachází. Následně se spočte jeho dráha podle vztahu 5.5 a přičte se k ní počátek celého pohybu. Pokud na dané ose došlo ke změně polohy, je zavolána fukce NotifyMovement, která aktualizuje polohu pohonu.

Jakmile čas dosáhne zadané hranice, je nastaven příznak dokončeného pohybu a funkce `ProcessAxisLine` se ukončí.

7.3.4 Oblouková interpolace

Oblouková interpolace je využita pro realizaci funkcí G02, G03, G12 a G13[20], popř. je využita při cyklech a kompenzaci nástroje. Tato interpolace má podobnou mechaniku a hiearchii jako lineární, proto se v následujícím textu zaměřím pouze na rozdíly. Příkaz pro obloukovou interpolaci je předáván jako prvek `circle` struktury `CommandStruct`. `circle` je typ `AxisCircle`, který je definován následovně:

```
typedef struct
{
    float B[3], C[3];
    float v, v0, vb;
    float As, Ab;
} AxisCircle;
```

`B` a `C` zde označují body; B je koncový bod oblouku a C je bod ležící kdekoliv na něm (s podmínkou, že současná poloha stroje a body B, C nesmí ležet na jedné přímce). `v, v0, vb` opět označují příslušné rychlosti, `As` a `Ab` opět označují použitá zrychlení.

Zde je nutno podotknout, že současná podoba interpolace je v rozporu se standardem G-kódu. Můj kód je schopný projet kruhový oblouk ležící v obecné rovině (proto se zadává za pomoci tří bodů), zatímco G-kód definuje jako přípustné roviny pouze XY, XZ a YZ. Tato chyba vznikla mým prvopočátečním špatným pochopením standardu. Dále dle standardu je možné pomocí kruhové interpolace interpolovat šroubovici. Toho lze docílit zadáním z-ové (respektive y-ové nebo x-ové) souřadnice koncového bodu tak, aby ležel mimo danou rovinu. Tento druh interpolace můj systém zatím nezvládá a chybně se snaží ze zadaných údajů sestrojit kružnici v obecné rovině. Z tohoto důvodu probíhá samotná interpolace zbytečně složitě.

Výše uvedená struktura je zpracovávaná funkcí `ProcessAxisCircle` (volané funkcí `ProcessAxisCommand`). V této funkci jsou připravena veškerá data. Kružnice je interpolována společně pro všechny osy – výstupem interpolace je úhlová dráha (v kódu označována jako `alpha`), na základě které se dopočítává poloha jednotlivých os.

Funkce `ProcessAxisCircle` nejprve dopočte střed kružnice na základě jejích třech bodů (sestrojením normálového vektoru roviny kružnice, díky němuž je možné vytvořit osy tětiv ze tří bodů a nalezením průsečíků těchto os). Poté připraví data. Přijatá data ve struktuře `AxisCircle` vyjadřují obvodové hodnoty. Jak jsem napsal v předchozím odstavci, pro mou interpolaci potřebuji úhlové hodnoty. Proto všechny hodnoty rychlostí a zrychlení před použitím vydělím poloměrem oblouku, čímž získám úhlovou rychlost a úhlové zrychlení, na základě nichž dopočítám potřebné konstanty stejně jako v případě lineární interpolace (nyní však pouze pro jednu „osu“ – úhlovou).

Po dopočítání úhlové dráhy potřebuji tuto dráhu převést na kružnici v kartézských souřadnicích. Pro tento převod používám parametrickou rovnici kružnice v obecné rovině (vztah 7.1[5])

$$\mathbf{P} = R\,\mathbf{u}\cos(t) + R\sin(t)\;\;\mathbf{n}\times\mathbf{u} + \mathbf{c} \tag{7.1}$$

V tomto vztahu $\mathbf{P}$ je polohový vektor hledaného bodu, R poloměr oblouku, t parametr, $\mathbf{n}$ jednotkový vektor kolmý na rovinu kružnice, $\mathbf{u}$ jednotkový vektor směřující ze středu k počátečnímu bodu oblouku a $\mathbf{c}$ polohový vektor středu kružnice. Vektorovou rovnici jsem si pro potřeby převodu rozdělil na jednotlivé osy, jim předpočítal konstantní hodnoty a jako paramet předávám okamžitou úhlovou dráhu.

Potřebná data jsou uložena obdobně jako u lineární interpolace v následujících strukturách:

```
typedef struct
{
    float w, w0;
    float alpha;
    float constant, brake, end;
    float ATpi, ATpiB, T, TB;
} AxisMovementCircleCommon;

typedef struct
{
    float* position;
    float final;
    float crossProduct, center, R, u;
} AxisMovementBufferCircle;
```

Vynechané části průběhu interpolace probíhají podobně jako u lineární, proto je dále nepopisuji.

7.3.5 Interpolace ideální křivky

V kódu lze spatřit i odkazy na interpolaci ideální křivky z kapitoly 5.4. Jak jsem v ní zmínil, s křivkou jsem počítal pouze na počátku a v pozdější fázi vývoje jsem se ji rozhodl vypustit. Interpolace této křivky je tedy ve funkčním stavu, avšak na této křivce není omezen ryv a neexistuje možnost, jak křivku vložit z G-kódu (není podporovaná na straně počítače). V kódu interpolátoru jsem ji však do budoucna pro případnou plnou implementaci ponechal.

7.4 Komponenty Movement a Stepper

Komponenta `Movement` zajišťuje převod pozice jednotlivých os na výstupní signál pro drivery. V začátcích projektu jsem měl v plánu implementovat jak step-dir řízení v podobě komponenty `Stepper`, tak i PID smyčku se zpětnou vazbou pro přímé řízení servomotoru v podobě komponenty `Servo`. V současnosti je implementována pouze komponenta `Stepper` a interpolátor tedy umí generovat pouze step-dir signál. Úkolem komponenty `Movement` je zajistit jednotné rozhraní pro generování výstupních signálů pro komponentu `Axis`, která volá její funkci `NotifyMovement`.

Pro každou osu je vytvořen jeden prvek následující struktury:

```
typedef enum {STEPPER = 1, SERVO = 2, UNUSED = 0} MovementType;

typedef struct
{
    MovementType type;
    union
    {
        Stepper stepper;
        Servo servo;
    };
} MovementStruct;
```

Při zavolání funkce `NotifyMovement` (tedy když se změnila poloha osy), je na základě členu type rozhodnuto, zdali se bude volat funkce pro upozornění komponenty `Stepper` nebo `Servo`.

To, která osa bude obsahovat komponentu `Stepper` a která `Servo`, je určeno nastavením z počítače. Toto nastavení je provedeno pomocí příchozí zprávy `CONFIG_MOVEMENT_SETTYPE`. První byte této přijaté zprávy určuje index osy, pro kterou se nastavuje typ, a druhý byte určuje druh generátoru výstupního signálu (Stepper/Servo) pro tuto osu. Druhou zprávou, kterou je nutno poslat, je zpráva `CONFIG_MOVEMENT_INDIVIDUAL` předávající dané ose, respektive generátoru, jeho příslušná nastavení. První byte této zprávy určuje index osy a za ním následují konfigurační data (konfigurační zpráva), která jsou předána již příslušné komponentě ke zpracování.

7.4.1 Komponenta Stepper

Komponenta `Stepper` sloužící ke generování step-dir signálu je reprezentována následující strukturou:

```
typedef struct
{
    uint32_t mmPerStep;
    int32_t position;
```

```
    uint8_t pulsePin;
    GPIO_TypeDef* GPIO;
    uint16_t directionPin;
    bool inverted;
    bool lastDirection;
} Stepper;
```

Tato struktura uchovává veškerá data a nastavení. Člen `mmPerStep` vyjadřuje vzdálenost, o kterou je posunuta osa při jednom kroku. Tato hodnota je uložena v desetitisícinách milimetru (10^{-4} mm). Člen `position` vyjadřuje současnou pozici osy v desetitisícinách milimetru (tedy maximální délka osy je ±214 metrů). Člen `pulsePin` je index výstupního pinu pro komponentu `PulseGenerator`, na kterém probíhá generování signálu step. Členy `GPIO` a `directionPin` určují výstupní pin pro signál dir. Člen `inverted` určuje, zda-li je třeba prohodit směr otáčení osy. Člen `lastDirection` není nastavením, nýbrž ukládá směr pohybu při poslední aktualizaci pohybu.

Pokud nastane změna pohybu osy, je skrz univerzální rozhraní komponenty `Movement` zavolána funkce komponenty `Stepper NotifyStepper`. Tato funkce nejprve zjistí odchylku fyzické pozice osy (na základě členu `position` ve struktuře `Stepper`) od pozice osy vypočtené komponentou `Axis`. Pokud je tato odchylka větší než velikost jednoho kroku (člen `mmPerStep`), zjistí jednotka směr (tedy polaritu signálu dir) a za pomoci komponenty `PulseGenerator` vygeneruje signál step. Poté přičte, či odečte délku jednoho kroku od aktuální pozice.

7.4.2 Komponenta PulseGenerator

Komponenta `PulseGenerator` má za úkol generovat pulzy. Původní myšlenka byla, aby byla schopna generovat pulzy libovolné délky v daném rozsahu s největší možnou přesností. Toho jsem chtěl docílit použitím timerů `TIM3` a `TIM12` v PWM módu. V PWM módu generují timery na pinech `PA6, PA7, PB0, PB1` a `PB14, PB15` PWM signál[19], jehož střídu lze určit hodnotou příslušných compare registrů. Frekvence PWM je dána předděličkou k timeru. Procesor umí generovat přerušení při přetečení timeru[19]. Těchto vlastností jsem využil ke generování jednoho pulzu – nastavil jsem si do fronty délku jednoho pulzu, při přetečení timeru jsem nastavil správnou hodnotu šířky pulzu a PWM mód jsem spustil. Při opětovném přetečení jsem timer vypnul. Tím jsem docílil generování jednoho pulzu. V průběhu testování se však ukázalo, že vznikají parazitní pulzy – pohony dělaly kroky navíc. Podezříval jsem, že se díky vnořování přerušení nestíhá PWM mód v čas vypínat – tuto doměnku se mi však nepodařilo ověřit.

Rozhodl jsem se proto oželet možnost generovat proměnlivou délku pulzu a za pomoci timeru generovat pulz konstantní šířky. Bohužel při tomto generování pulzu vznikal stejný jev – objevovaly se pulzy navíc. Snažil jsem se chybu objevit, avšak neúspěšně.

Z nedostatku času a díky nutnosti funkčnosti této komponenty (bez ní nelze řídicí systém testovat), jsem se uchýlil k dočasnému hacku. Nyní generuji 100 ns pulz a čekání zajišťuji sérií instrukcí nop. Nyní mi již nevznikají žádné parazitní pulzy. Toto řešení je dočasné a mělo by být co nejdříve nahrazeno jiným.

7.5 Implementace ostatních komponent

V předchozích sekcích byly popsány nejdůležitější komponenty, které tvoří jádro celého interpolátoru. Tyto komponenty mají na starost pohyb stroje. G-kód však definuje další druhy příkazů, které se netýkají pohybu a proto bylo nutné do návrhu přidat ještě dvě komponenty.

7.5.1 Komponenta GPIOControl

Tato kompenta má za úkol obsluhovat vstupy a výstupy mikrokontroléru. Skrze ni by mělo být např. zapínáno vřeteno (funkce M03, M04[20]). Záměrně byla tato komponenta navžena úzce spjatá s hardwarem, aby si zachovala univerzálnost, a tak bylo možné v počítači co nejpodrobněji nakonfigurovat chování.

Tato komponenta není zatím dokončena a pro demonstrativní účely zatím umí ovládat jednotlivé LED na kitu, které jsem používal pro demonstraci funkčnosti.

7.5.2 Komponenty SystemComponents

`SystemComponents` zastřešuje příkazy spíše systémového rázu, které nesouvisí s výstupem. Příkladem takového příkazu (a zatím jediného implementovaného z této katogorie) je funkce prodlevy – P parametr příkazů G-kódu[20] realizován pomocí `DelayComponent`.

Prodleva je implementována pomocí přerušení `SysTick` [19]. Frekvence tohoto přerušení byla nastavena na 1 kHz – přerušení nastává každou milisekundu. `DelayComponent` implementuje seznam až 32 prodlev. Pokaždé, když nastane přerušení, je hodnota každé prodlevy právě o jednu milisekundu zmenšena. Jakmile dosáhne hodnota prodlevy nuly, je prodleva ze seznamu odstraněna a je zavolána případná callback funkce.

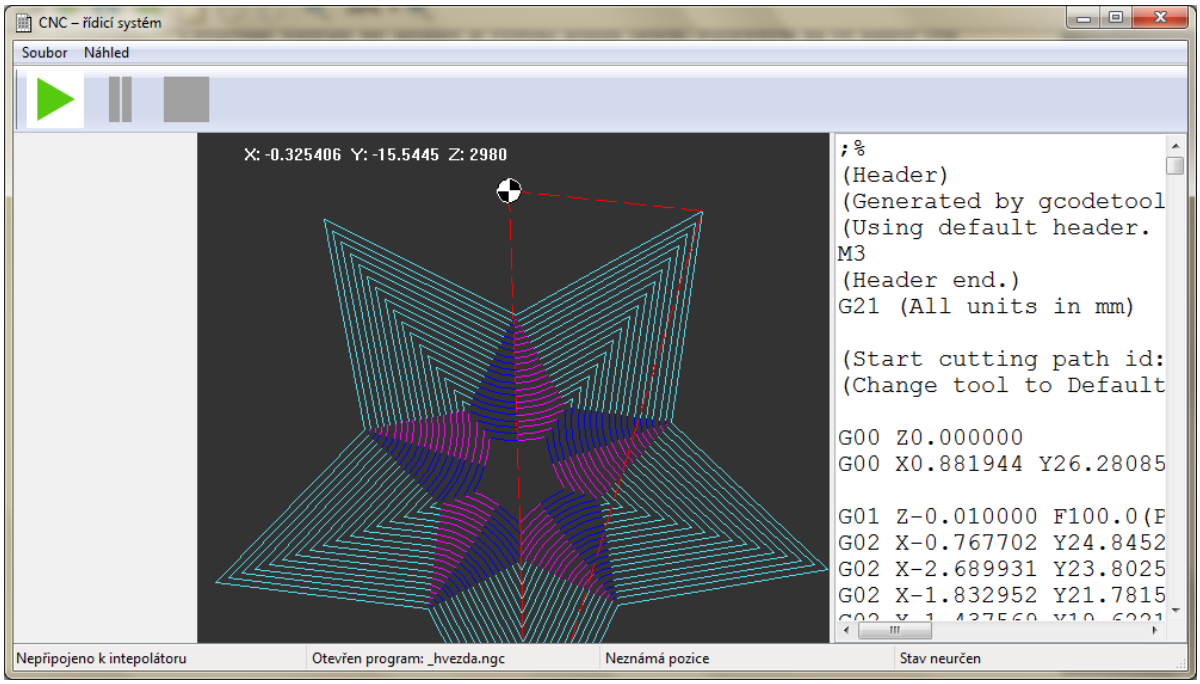
Kapitola 8

Software pro počítač

Jak jsem již zmínil v kapitole 6 (Použité prostředky a hardware), aplikace pro počítač je napsána v jazyce C++ s využitím prvků z nového standardu C++11. Pro GUI jsem použil framework WxWidgets. V následujícím textu záměrně vynechávám některé detaily implementace (především co se GUI a výstavby aplikace týče), jelikož se jedná vesměs o standardní postupy, a zaměřuji se na specifika řídicího systému – interpretaci G-kódu, výpočet rychlosti pro stroj a následnou komunikaci s interpolátorem.

8.1 Uživatelské rozhraní

Uživatelské rozhraní mé aplikace je tvořeno jedním oknem, rozděleným na tři panely (viz obrázek 8.1).



Obrázek 8.1: Screenshot uživatelského rozhraní

Pravý panel zobrazuje aktuálně načtený program v G-kódu a za jeho běhu zvýrazňuje aktuálně prováděný řádek. Úplně levý panel zatím není využit, do budoucna by však měl zobrazovat další ovládací prvky a informace.

Prostřední panel zobrazuje náhled drah stroje. V současné verzi zobrazuje pouze 2D náhled drah, které leží v rovině XY. Náhled je možné libovolně posouvat myší a kolečkem jej přibližovat/oddalovat. Do tohoto náhledu se také zobrazuje zpětná vazba z interpolátoru (zvýrazňují se již projeté dráhy). Dráhy zadané přímo G-kódem se zobrazují barevně odlišené podle druhu interpolace (rychloposuv je naznačen přerušovaně). Barvy jednotlivých typů interpolace lze navolit v konfiguračním souboru. Dráhy, které byly dopočteny (vznikly kompenzací nástroje), jsou zobrazeny bíle.

Dole v taskbaru aplikace jsou zobrazeny následující údaje: stav připojeno/odpojeno od interpolátoru, otevřený program, okamžitá pozice os a současný (chybový) stav interpolátoru.

V toolbaru se nacházejí tři tlačítka pro spuštění, pozastavení a úplné zastavení běhu programu. V menu aplikace Soubor lze otevřít program v G-kódu. V menu náhled je možné vypnout zobrazování již provedených pohybů strojem a obnovit zobrazení náhledu drah.

Uživatelské rzohraní je v současném stádiu vývoje velmi chudé a veškerá nastavení jak aplikace, tak interpolátoru jsou prováděna úpravou konfiguračního INI souboru. Aplikace využívá jediný konfigurační soubor `settings.ini`. V tomto souboru jsou definovány jednotlivé sekce, obsahující příslušná nastavení.

8.2 Intepretace G-kódu

Jednou z klíčových vlastností řídicího systému je parsování G-kódu na interní formát, se kterým lze uvnitř programu pracovat.

8.2.1 G-kód a jeho formát

G-kód je univerzální jazyk pro programování CNC strojů[23]. Syntaxe G-kódu je ralativně jednoduchá. Program je dělen na tzv. bloky. Blok je jeden řádek programu. V každém bloku jsou uvedeny hodnoty jednotlivých registrů (A–Z). Registr G může být uveden vícekrát, čímž mu je přiděleno více hodnot[23]. G-kód ignoruje mezery a obecně jakékoliv bílé znaky. Program zpravidla začíná a končí znakem procenta (%). Komentáře v G-kódu jsou umístěny do obyčejných jednoduchých závorek a nesmí přesáhnout hranici jednoho bloku.

Nejpoužívanějšími registry jsou registry G a M, které uchovávají typ funkce k vykonání. Ostatní registry slouží jako parametry těchto funkcí. Registry X, Y, Z uchovávají souřadnice lineárních os (koncové body). Vedle těchto registrů se používají i registry A, B, C pro polohu rotačních os, avšak jelikož můj interpolátor podporuje pouze 3 osy, nebudu tyto registry dále zmiňovat. Registry I, J, K a R slouží pro zadávání parametrů obloukové interpolace. Registr F určuje rychlost stroje (posuv). Ostatní registry jsou méně používané, jejich kompletní seznam lze nalézt na stránkách [23].

Seznam všech G-funkcí, které můj systém v současné verzi podporuje, a jejich popis je uveden v tabulce 8.1.

Funkce G zpravidla vykonávají pohyb stroje a nastavují parametry. Funkce M mají speciální význam, který je dán specificky pro každý stroj. Existují však některé ustálené M-funkce (např. M03 pro zapnutí vřetena[23]). Pokud je v jednom bloku uvedeno více funkcí, určuje pořadí jejich vykonání stroj.

G-kód podporuje i podprogramy, zápis matematických výrazů či polární zadávání souřadnic. Tyto funkce však dnes už nejsou (obvzláště v hobby sféře) příliš rozšířené. Dříve podprogramy

Funkce	Význam	Argumenty
G00	Rychloposuv. Funkce není určená pro obrábění. Stroj nemusí pohyb interpolovat lineárně.	X, Y, Z
G01	Lineární interpolace ze současné pozice na zadanou posuvem F	X, Y, Z, F
G02	Oblouková interpolace ve směru hodinových ručiček. Oblouk je zadán současnou polohou a koncovým bodem. Jeho střed je určen buď registry I, J, K, které určují odsazení středu od počáteční polohy, nebo prametrem R učujícím poloměr oblouku. Učebnice G-kódu doporučují zadávání pomocí odsazení.	X, Y, Z, I, J, K, R, F
G03	Oblouková interpolace proti směru hodinových ručiček. Zadáváno stejně jako G02	X, Y, Z, I, J, K, R, F
G20	Nastaví pracovní jednotky na palce	
G21	Nastaví pracovní jednotky na milimetry	
G40	Funkce zruší kompenzaci nástroje	
G41	Zapne kompenzaci nástroje doleva ve směru obrábění. Průměr nástroje je zadán jako D	D
G42	Zapne kompenzaci nástroje doprava ve směru obrábění. Průměr nástroje je zadán jako D	D
G43	Zapně kompenzaci délky nástroje kladného směru osy Z. Délka nástroje je zadána jako H	H
G44	Zapně kompenzaci délky nástroje do záporného směru osy Z. Délka nástroje je zadána jako H	H
G49	Vypne délkovou kompenzaci nástroje	

Tabulka 8.1: Tabulka G-funkcí podporovaných mým systémem. Definice funkcí převzaty z [20].

umožňovaly zkrácení programu a jeho snažší zápis pro člověka. Tyto funkce však již dnes s nástupem CAM programů nejsou zapotřebí. Proto jsem se tyto funkce rozhodl do mého systému neimplementovat.

Některé hodnoty některých registrů si systém pamatuje do doby, dokud nejsou znovu nastaveny. Příkladem můžou být zadané cílové souřadnice, či posuv. Není tedy nutné zadávat pokaždé znovu všechny 3 souřadnice, stačí pouze ty, které se změnily. Také pokud byla použita funkce G01 (lineární interpolace), tak lze pro sérii pohybů po úsečce napsat pouze do prvního bloku G01 a do dalších bloků uvést pouze souřadnice. Systém pak automaticky použije funkci G01.

Část programu v G-kódu může vypadat následovně:

```
G00 X1 Y31 Z50
G01 Z0.1 F100
    X0.19 Y 29.58
G02 X2.1 Y26.89 I6.64 J3.47
    X5.53 Y25.61 I4.26 J6.16
    X2.1 Y26.89 I6.64 J3.47
```

8.2.2 Interní formát a třída GCodeInterpreter

Abych mohl G-kód dále zpracovat (dopočítat rychlost, provést korekce, odeslat do interpolátoru), musím jej neprve převést na interní formát mého programu, se kterým lze dobře pracovat.

Veškeré zpracování G-kódu má na starosti třída `GCodeInterpreter`. Tato třída implementuje veškerou funkcionalitu pro převod vstupního řetězce obsahujícího G-kód na příkazy pro interpolátor. Tento proces v sobě zahrnuje i provedení korekcí nástroje a výpočet rychlosti. Převedení G-kódu na výsledek je spuštěno funkcí `ProcessString`. Tato metoda přebírá `std::string`, který obsahuje G-kód. Výsledkem je seznam objektů pro vykreslení na obrazovce a seznam připravených příkazů pro interpolátor. Třída `GCodeInterpreter` v sobě zahrnuje i načtení konfigurace týkající se zpracování z daných konfiguračních souborů a jejich následné předání příslušným metodám či objektům. Také uchovává stav jednotlivých parametrů v průběhu převodu.

8.2.3 Převod G-kódu

Parsování G-kódu probíhá v metodě `GCodeInterpreter::ParseCodeToPath` ve dvou krocích – v prvním kroku z G-kódu vyčtu hodnoty jednotlivých registrů pomocí metody `GCodeInterpreter::ParseGCodeLine`. Jelikož některé registry mohou obsahovat více hodnot (např. registr G), vrací tato funkce asociativní pole, kde je klíčem char označující registr a hodnotou je seznam čísel s plovoucí desetinnou čárkou (`std::map<char, vector<float>>`). V druhém kroku jsou z tohoto asociativního pole vybrány jednotlivé funkce a jejich argumenty podle prototypů. To provádí metoda `GCodeInterpreter::SeparateCommandsFromLine`. Zároveň tato metoda zajišťuje správné pořádí provádění funkcí v případě, kdy je jich v bloku uvedeno více.

Metoda `ParseGCodeLine` v cyklu postupně načítá jeden znak ze stringu. Pokud je tento znak bílý, tak jej přeskočí. Pokud narazí na znak „`(`“, začne načítat a zahazovat další znaky, dokud nenarazí na znak „`)`“. Pokud je načtený znak písmeno, načte pomocí třídy stringstream z následujících znaků číslo a uloží je do asociativního pole, které je na konci navráceno jako výsledek.

Metoda `SeparateCommandsFromLine` využívá seznam prototypů funkcí G-kódu uvedených v souboru `gcode.dat`. Tento soubor zároveň definuje pořadí vykonání funkcí, pokud by všechny byly uvedeny v jednom bloku. Toto pořadí jsem převzal z online učebnice G-kódu [11]. Prochází jeden prototyp funkce za druhým a zjišťuje, jestli pasuje na hodnoty registrů předané v tomto bloku. Pokud ano, zkopíruje tyto hodnoty do struktury `GCodeLine`, která vždy obsahuje přesně jednu funkci pouze s jejími parametry. Tato metoda zároveň zjišťuje, zda-li je možné převzít některý z předchozích registů G – např. pro řetězení funkcí G01 bez jejich opisování, jak bylo zmíněno v kapitole 8.2.1.

Soubor `gcode.dat` je textový soubor, který na každém řádku obsahuje definici jednoho prototypu. Je složen ze sloupců oddělených tabulátorem. První sloupec obsahuje druh registru – buď M, nebo G. Druhý sloupec obsahuje číslo funkce, popř. -1 pokud na tomto čísle nesejde (např. posuv je realizován jako funkce). V dalších sloupcích jsou vždy uloženy jednotlivé kombinace registrů, které fungují jako argumenty. Jednotlivé registry jsou od sebe odděleny mezerou a za posledním je umístěn středník.

8.3 Třída PathPart

G-kód je v mém systému převáděn na trasu (`path`). Trasa vyjadřuje všechny úseky pohybu proložené všemi modálními příkazy (např. změna posuvu, zapnutí vřetene, atd.). Jednotlivé úseky jsou poté převáděny na příkazy pro interpolátor.

Trasa je definována jako `list<unique_ptr<PathPart>>`. `list` (tedy řetězený seznam) jsem zvolil, abych mohl přidávat prvky uprostřed seznamu s malou časovou náročností. Cenou za to je ztráta nahodilého přístupu jako u kontejneru `vector`. Avšak trasu vždy potřebuji procházet pouze sekvenčně, takže zde pro mě `list` neskýtá žádné omezení.

Třída `PathPart` je abstraktní základní třída, od které dědí všechny ostatní části. Abych mohl využít polymorfického chování, je umístěna v kontejneru skrz třídu `unique_ptr` ze standardní knihovny C++11. Tato třída je deklarována v hlavičkovém souboru `PathPart.h`. Třída obsahuje deklaraci metod pro všechny úkony, které lze s úsekem trasy provést. Definuje funkce pro získání jednotlivých rychlostí, pro získání tečného vektoru na svém začátku či konci. Také definuje metodu `ComputeSpeeds`, která má za úkol provést výpočet rychlosti na daném úseku. Metoda `ProcessCompensationStarting`, respektive `ProcessCompensationEnding`, řeší provedení kompenzace nástroje na začátku tohoto úseku, respektive na jeho konci. Jelikož je při některých operacích nutné přistupovat k předcházejícímu či následujícímu prvku, obsahuje třída PathPart atribut `list<unique_ptr<PathPart>>::iterator item`. Tento atribut uchovává iterátor ukazující na tento objekt třídy `PathPart`. Jeho inkrementováním či dekrementováním můžu získat následující úsek, respektive předcházející. Toto řešení je založeno na tom, že iterátor kontejneru `list` zůstává po celou dobu existence daného prvku platným, nehledě na přidané či odebrané prvky.

Z třídy `PathPart` jsou děděny dvě třídy, které slouží jako základ pro implementaci konkrétních typů úseku. Jsou to třídy `PathPartMovable` a `PathPartModable`. Třída `PathPartMovable` je základem pro části trasy, které skutečně reprezentují pohyb stroje a je nutné pro ně počítat rychlost a provádět korekci nástroje. Naopak potomci třídy `PathPartModable` nereprezentují pohyb stroje, ale ostatní příkazy. Pro tyto úseky se rychlost nepočítá a ani se neprovádí korekce nástroje.

Obě výše uvedené třídy mají stejný základ a musí definovat všechny metody. Třída `PathPart` definuje metodu `IsMovable`, která vrací typ třídy. Tato metoda je definována v `PathPartMovable` tak, aby vracela `true`, v `PathPartModable` vrací `false`. Třída `PathPartMovable` také definuje všechny pro ni nesmyslené funkce (např. výpočet rychlosti) tak, že ve svém těle vyhodí výjimku, což usnadňuje debugování. Během práce s úseky tedy volám funkci `IsMovable`, abych věděl, jestli pro daný úsek mohu volat všechny metody. Toto řešení není nejelegantnější, ale použil jsem ho kvůli jednoduchosti implementace a také kvůli tomu, že drtivá většina úseků je typu `PathPartMovable` a jen mizivá část je typu `PathPartModable`.

Z třídy `PathPartMovable` jsou odvozeny třídy `PathPartLine` (reprezentuje lineání interpolaci), `PathPartRapid` (rychloposuv), `PathPartCircle` (reprezentuje obloukovou interpolaci) a třídy `PathPartOffsetLine` a `PathPartOffsetCircle` (reprezentují úseky trasy, které vznikly např. při kompenzaci nástroje a nemají odkaz na sebe v G-kódu).

Z třídy `PathPartModable` byla prozatím odvozena pouze jediná třída `PathPartUnproc`, která funguje jako forma návěstidla – označuje např. konec či začátek programu.

8.4 Převod GCodeLine na PathPart

Tento převod má na starosti metoda `GCodeInterpreter::ConvertCommandToPath`. Jako argument přebírá funkci G-kódu v podobě třídy `GCodeLine`. Podstata této metody spočívá v asociativním poli `processingFunctions`. Jako klíč zde vystupuje kód funkce `GFunction` a hodnotou je ukazatel na metodu třídy `GCodeInterpreter`, která vykonává převod. Toto pole je deklarováno následovně:

```
//Definice datových typů
```

```
typedef void (GCodeInterpreter::*GCodeProcessingFunction)(GCodeLine&);
typedef pair<char, float> GFunction;
//Definice pole
map<GFunction, GCodeProcessingFunction> processingFunctions;
```

V konstruktoru třídy `GCodeInterpreter` je toto pole naplněno příslušnými ukazateli na metody. Tyto metody pro převod jsem nazval stejně jako příslušné G-funkce – např. G00, G01 apod.

Metoda `ConvertCommandToPath` prvně zkusí najít příšlušnou funkci v asociativním poli. Pokud ji nenalezne, vyhodí výjimku. Pokud ji nalezne, zavolá ji a příslušná funkce se postará o převod a zařazení úseku do trasy.

8.4.1 Funkce G00

Tato funkce má za úkol vytvořit pohyb rychloposuvem. Ačkoliv standard tvrdí, že v tomto případě není nutné, aby interpolace probíhala lineárně (může probíhat maximální rychlostí pro každou osu), implementuji jej jako lineární interpolaci s rychlostí pro rychloposuv nastavenou v konfiguračním souboru. Jedině tak můžu zajistit, že rychloposuv probíhá na základě fyzikálního modelu a respektuje všechna omezení.

V prvním kroku funkce ještě na základě aktuálně nastaveného režimu souřadnic (absolutní a inkrementální) provede jejich přepočet tak, aby vždy byly v absolutních hodnotách.

Na základě těchto údajů se zavolá konstruktor třídy `PathPartRapid`, která je odvozena od třídy PathPartLine. Takto vytvořený úsek je následně vložen do trasy.

8.4.2 Funkce G01

Funkce G01 slouží pro lineární interpolaci. Tato interpolace je realizována stejně jako funkce G00, pouze se jako hodnota posuvu použije naposledy zadaná hodnota funkce F a jako úsek trasy je vytvořen objekt třídy `PathPartLine`.

8.4.3 Funkce G02 a G03

Funkce G02 a G03 slouží pro zadání obloukové interpolace. Navzájem se liší pouze ve směru. V mé konkrétní implementaci to tedy znamená, jestli výsledný úsek obdrží identifikátor `CIRCLE_CW`, nebo `CIRCLE_CCW`. Proto tento druh úseku vytváří funkce `CircleFunction`, která je volána jak ve funkci G02, tak i ve funkci G03 – pouze je jí předán jiný identifikátor.

`CircleFunction` projde předané argumenty v `GCodeLine` a buď zavolá konstruktor `PathPartCircle`, který přebírá poloměr, nebo konstruktor, který přebírá souřadnice středu kružnice. Zde se implementace G-kódu na různých strojích rozcházejí – některé stroje berou argumenty I, J a K jako odsazení od počátku oblouku, jiné je berou jako absolutní hodnotu. Zde jsem se inspiroval implementací, kterou používá LinuxCNC – standardně jsou tyto argumenty přebírány jako odsazení od počátku oblouku, avšak pomocí funkce G90.1 nebo změnou nastavení v konfiguračním souboru je lze přepnout na absolutní hodnoty.

8.4.4 Funkce G20 a G21

Funkce G20 nastavuje pracovní jednotky na palce, naopak G21 na milimetry. Můj systém standardně pracuje v milimetrech. Pro implementaci jsem zavedl do třídy `GCodeInterpreter` atribut `unitMultiply`, který uchovává aktuální jednotku. Tímto atributem jsou všechny souřadnice zadané v G-kódu násobeny před jejich použitím pro kontrukci úseků trasy. Funkce G20 mu tedy přiřazuje hodnotu 2,54; funkce G21 hodnotu 1.

8.4.5 Funkce F

Funkce F nastavuje aktuální rychlost. Činí tak natavení atributu `GCodeInterpreter::standardFeed`. Tato hodnota uchovává posuv – jednotkou je tedy buď $mm \cdot min^{-1}$ nebo $in \cdot min^{-1}$. Před jejich použitím je tedy nutné je převést na $mm \cdot s^{-1}$. Funkce F zároveň do trasy přidává návěstidlo pro debugovací účely. Toto návěstidlo je typu `PathPartUnproc`.

8.4.6 Funkce G40, G41 a G42

Funkce G40 ruší jakoukoliv aplikovanou korekci průměru nástroje. Funkce G41 zapíná kompenzaci průměru nástroje směrem doleva ve směru obrábění, naopak funkce G43 doprava.

Tyto funkce jsou relizovány nastavením příšlušných hodnot objektu třídy `PathPartOffset`. Objekt této třídy je předáván každému úseku typu `PathPartMovable` v jeho konstruktoru. Na základě něj potom úsek vytvoří příslušnou úpravou trasu, která respektuje kompenzaci nástroje.

8.4.7 Funkce G43, G44 a G49

Tyto funkce jsou analogické k funkcím uvedeným v předchozím odstavci – tyto funkce pouze nastavují délkovou korekci nástroje. Jsou i naprosto stejně implementované.

Funkce G49 délkovou kompenzaci vypíná, G43 ji zapíná do kladného směru osy Z a funkce G44 ji zapíná do záporného směru osy Z.

8.5 Implementace korekce nástroje

Poté, co je G-kód převeden na jednotlivé úseky trasy, je tato trasa dále zpracovávána. Prvním stupněm zpracování je vyřešešení korekce nástroje. Tato korekce je vyvolána v metodě `GCodeInterpreter::PostProcessPath`, kde je postupně pro každý úsek volána jeho metoda `PathPart::ProcessToolCompensation`.

Zpracování každého úseku se skládá ze dvou částí – prvně je vyřešen počáteční bod tohoto úseku a následně koncový bod. Během tohoto zpracování je třeba někdy vložit do trasy nové úseky. Jelikož však tyto úseky nemají přímý odkaz na G-kód, nevychází z něj a neobsahují informace o původní trase, jsou reprezentovány speciálními úseky – `PathPartOffsetLine` a `PathPartOffsetCircle`.

Standard G-kódu nijak podrobně nespecifikuje, jak má korekce nástroje probíhat a jak mají být řešeny jednotlivé přechody. Různé stroje implementují korekce mírně odlišně. Pro mou implementaci jsem vycházel z chování systému LinuxCNC (viz dokument [4]), systému Sinumerik[17] (i přesto, že se jedná o systém pro soustruh, bral jsem jej jako porovnání s novějšími systémy) a systému Heidenheim (viz manuál [13]).

V následujícím textu se odprostím od implementace tohoto řešení do kódu, která je relativně jednoduchá, a zaměřím se na teoretickou část problému – jak má korekce vypadat a jak dopočítat souřadnice korigované trasy.

8.5.1 Korekce průměru

Korekce průměru nástroje probíhá v rovině XY. Tedy veškeré úvahy se vztahují pouze k průmětu aktuální trasy do roviny XY.

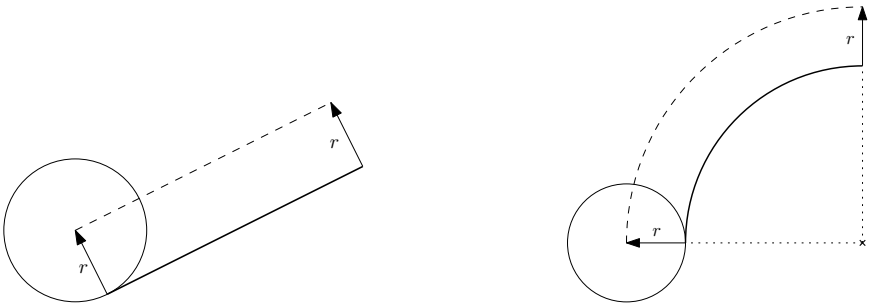
Základní korekce, tečné úseky

Nejjednodušším případem je korekce úseku, který se nachází uprostřed již korigované oblasti. Situace je zobrazena na obrázku 8.2.

Pro korekce jsem zavedl konvenci, že kladná hodnota korekce značí korekci doprava ve směru obrábění, záporná doleva. Pro provedení odsazení na úsečce mi tedy stačí pouhé posunutí jejího počátečního a koncového bodu. Vektor $\mathbf{n}$ je kolmý jednotkový vektor na daný úsek, k označuje velikost korekce a nabývá hodnot $\pm r$ (r je poloměr nástroje). Posun $\mathbf{A}$, lze tedy zapsat jako:

$$\mathbf{A}' = \mathbf{A} + k \cdot \mathbf{n} \tag{8.1}$$

Obdobně lze provést i korekci na kruhovém oblouku. Pouze vektor $\mathbf{n}$ nyní není stejný pro počáteční i koncový bod a je nutné jej pro každý bod určit sólo. Zkorigovaný oblouk je s původním soustředný.



(a) Korekce na úsečce (b) Korekce na oblouku

Obrázek 8.2: Znázornění základní korekce. Černou čarou je znázorněna naprogramovaná kontura (tedy trasa zadaná G-kódem), přerušovaně pak dopočtená korigovaná trasa.

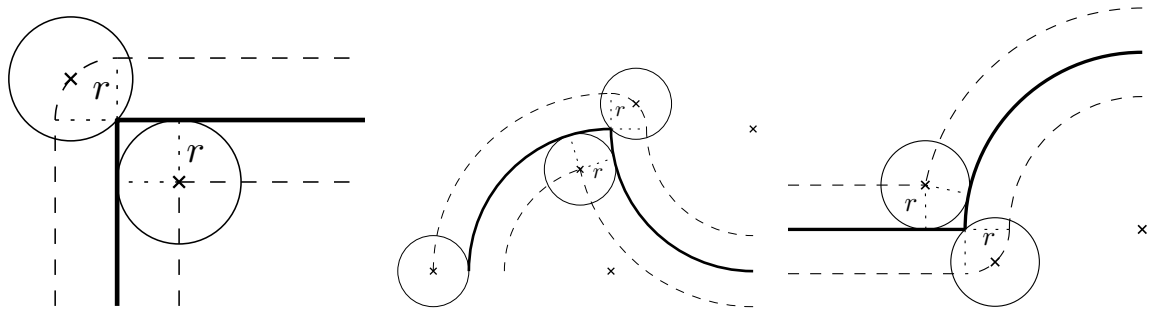
Tuto korekci lze provést i na tečně navazujících úsecích se stejnou hodnotou korekce – jelikož jsou tečné, jejich kolmý vektor v koncovém bodě prvního a počátečním bodě druhého úseku je stejný, tudíž zkorigované body splynou a po korekci dostávám spojitou trasu.

Korekce pro úseky nenavazující tečně

Pokud bych na úseky, které na sebe tečně nenavazují použil výše uvedenou korekci, dostal bych nespojitou zkorigovanou trasu. V tomto případě navázání je nutné rozlišit několik případů. Z pohledu korekce je nutné hlavně rozlišit, zdali se jedná o vnitřní nebo vnější roh. Z pohledu výpočtu je ještě nutné rozlišit, mezi jakými druhy úseků korekce nastává. Možné situace jsou znázorněny na obrázcích 8.3.

Provedení odsazení vnějšího rohu je relativně jednoduché. Pro zkrácení obráběcího času se neprojíždí plné odsazení, nýbrž se roh opíše obloukem[13]. Tím je dosaženo zkrácení trasy z $2r$ na $\frac{\pi}{2}r$. Zároveň se tím však zjednodušuje výpočet korigované trasy. Oba úseky stačí zpracovat stejně jako v základním případě podle vztahu 8.1 a mezi koncový odsazený bod prvního úseku a počáteční odsazený bod druhého úseku vložit oblouk se středem v koncovém neodsazeném bodu prvního úseku (popř. počátečním neodsazeném bodu druhého úseku, jelikož tyto body jsou totožné). Při obrábění vnějšího rohu nevznikají žádné nepřesnosti.

Pro zpracování vnitřního rohu je třeba nalézt průsečík již zkorigovaných úseků. Jak lze vidět na obrázcích 8.3, vzniká při obrábění vnitřního rohu nepřesnost, kterou nelze nijak korigovat. Lze ji pouze minimalizovat použitím menšího nástroje.



(a) Korekce na úsečkách (b) Korekce na obloucích (c) Kombinovaná korekce

Obrázek 8.3: Znázornění korekcí na tečně nenavazujících úsecích. Tlustou čarou je zde znázorněna naprogramovaná kontura (původní trasa zadaná G-kódem), přerušovaně pak dopočítaná korekce trasy.

Nejjednodušší situace nastává u dvou netečně navazujících úseček. Průsečík zde hledám tak, že si parametricky vyjádřím zkorigované úsečky (k parametrickému vyjádření původní trasy přičtu $k \cdot \mathbf{n}$). Tato soustava vypadá následovně:

$$\begin{cases} \mathbf{P} = \mathbf{A} + p \cdot \mathbf{u} + k \cdot \mathbf{n} \\ \mathbf{P} = \mathbf{B} + q \cdot \mathbf{v} + k \cdot \mathbf{m}, \end{cases} \tag{8.2}$$

kde p a q jsou parametry, u a v směrové vektory, n a m přislušné normálové vektory a k je hodnota korekce nástroje.

Porovnáním těchto dvou rovnic a následným vyřešením získám hodnotu parametru, ze kterého lze jednoduše dopočítat průsečík, což je koncový a počáteční bod příslušných zkorigovaných úseků.

$$p = \frac{-A_1 + A_2 + B_1 - B_2 + m_1 - m_2 - n_1 + n_2}{u_1 - u_2} \tag{8.3}$$

Obdobně lze naléz i průsečík dvou oblouků – pro zjednodušení výpočtu však všechny útvary posunu tak, aby střed jedné z kružnici ležel v bodě $[0;0]$ a po provedení výpočtu výsledek posunu opačným směrem (zpět). Tímto posunem se mi výrazně zjednodušší výpočet, jelikož dostanu jednoduší středovou rovnici jedné z kružnic:

$$\begin{cases} x^2 + y^2 = r_1^2 & \text{– rovnice zjednodušená posunem} \\ (x-m)^2 + (y-n)^2 = r_2^2, \end{cases} \tag{8.4}$$

kde x a y jsou souřadnice hledaného průsečíku $P[x;y]$, m a n jsou souřadnice středu druhé kružnice $S[m;n]$, r_1 a r_2 jsou zkorigované poloměry odsazených kružnic (k poloměru původních oblouků zadaných G-kódem je přičtena korekce k).

Řešením této soustavy je tento relativně složitý výraz:

$$P[x;y] \begin{cases} x = \frac{m^3+m\left(n^2+r_1^2-r_2^2\right) \pm \sqrt{n^2\left(-\left(m^2+n^2-r_1^2\right)^2+2r_2^2\left(m^2+n^2+r_1^2\right)-r_2^4\right)}}{2\left(m^2+n^2\right)} \\ y = \frac{n^4+n^2\left(m^2+r_1^2-r_2^2\right) \mp m\sqrt{n^2\left(-\left(m^2+n^2-r_1^2\right)^2+2r_2^2\left(m^2+n^2+r_1^2\right)-r_2^4\right)}}{2n\left(m^2+n^2\right)} \end{cases} \tag{8.5}$$

Existují dva průsečíky těchto kružnic, bohužel se mi nepodařilo zjistit, jak předem rozlišit, který průsečík je ten mnou hledaný. V mém kódu tedy počítám oba dva a za správný považuji ten, který leží blíže původnímu napojení nekorigovaných oblouků.

Poslední případ, který může nastat je vnitřní roh mezi obloukem a úsečkou. Zde opět využiji parametrického vyjádření úsečky, které následně dosadím do obecné rovnice kružnice. Opět v zájmu zjednodušení byl proveden posun tétou soustavy tak, aby střed oblouku ležel v bodě $[0; 0]$.

$$\begin{cases} \mathbf{P} = \mathbf{A} + p \cdot \mathbf{u} + k \cdot \mathbf{n} \\ x^2 + y^2 = r^2 \end{cases} \tag{8.6}$$

Pro zjednodušení výrazu sloučím v rovnici přímky konstantní výrazy v konstantu **k**:

$$\mathbf{A} + k \cdot \mathbf{n} = \mathbf{k} \tag{8.7}$$

Po rozdělení rovnice přímky na dvě (pro x-ové a y-ové souřadnice) a jejich dosazení do rovnice kružnice, dostávám následující vztah pro parametr p:

$$p = \frac{\pm k_1 u_1 \pm k_2 u_2 \mp \sqrt{u_1^2(r^2 - k_2^2) + 2k_1 k_2 u_1 u_2 + u_2^2(r^2 - k_1^2)}}{u_1^2 + u_2^2}, \tag{8.8}$$

z něhož snadno dopočítám oba průsečíky. Stejně jako v předchozím případě, i zde jsem nepřišel na způsob, jak předem určit, který z parametrů je mnou hledaný. Proto opět počítám oba dva a vybírám bod, který je blíže průsečíku původních nekorigovaných úseků.

Korekce pro úseky se změnou její hodnoty

To, jak se má korekce chovat, pokud je nastavena, je relativně logické. Standard však nedefinuje, jak se má systém zachovat po spuštění korekce. Rozhodl jsem se respektovat chování systémů LinuxCNC[4] a Heidenhein[13]. Tyto systémy považují první pohyb po aplikování korekce za „lead-in“[1]. Tento pohyb by neměl být určen k obrábění a jeho cílem je aplikovat korekci – na svém konci by měl být nástroj již ve správném odsazení od původní trasy. Jako vzor chování stroje během lead-in pohybu jsem si vzal chování systému LinuxCNC (popsáno v [4]). Toto chování se však ukázalo v jednom případě jako nevhodné (viz dále).

Výše uvedené systémy (LinuxCNC, Heidenhein, Sinumerik), akceptují jako lead-in pohyb pouze lineární pohyb (funkci G01), avšak já jsem se rozhodl do mého systému zaimplementovat i podporu pro obloukový lead-in pohyb (funkce G02, G03).

Celkem tedy mohou nastat čtyři případy: lead-in pohyb je lineární a navazuje na vnitřní roh; lead-in pohyb je lineární a navazuje na vnější roh; lead-in pohyb je obloukový a korekce se zmenšuje; lead-in pohyb je obloukový a korekce se zvětšuje.

Nejjednodušším případem je lineární lead-in pohyb navazující na vnitřní roh. Zde jsem se také po konzultaci na diskusním fóru C-N-C.cz[2] rozešel s implementací LinuxuCNC. Implementace systémem LinuxCNC je znázorněna na nákresu 8.4. Logikou tohoto systému je, že nástroj nesmí odebrat materiál mimo zadanou konturu, proto raději zanechá neobrobenou část na kontuře[4].

Po konzultaci na diskusním fóru jsem si stanovil pro můj systém stejnou prioritu jako systém Heidenheim[13] – systém musí, co nejpřesněji obrobit zadanou konturu. Na nákresech 8.5 je znázorněno chování mého systému. Na nákresu b si lze povšimnout, že systém porušuje konturu lead-in pohybu. Jak jsem však již dříve zmínil, lead-in pohyb by neměl obrábět. Proto porušení jeho kontury nevadí. Ve správně napsaném programu by v prostoru lead-in pohybu neměl být žádný materiál.

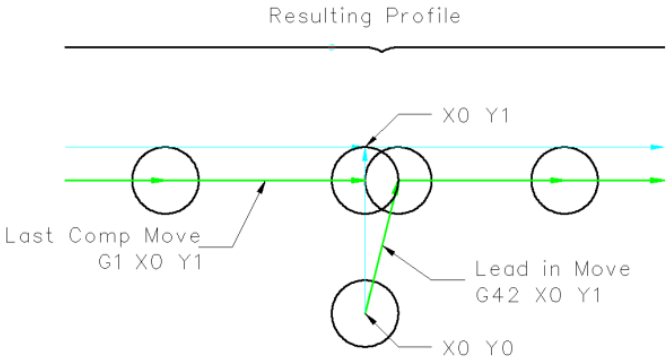
[1] termín používaný v dokumentaci LinuxuCNC[4], nenašel jsem adekvátní překlad, proto v textu používám anglický výraz

[2] <http://www.c-n-c.cz/viewtopic.php?f=47&t=9669>

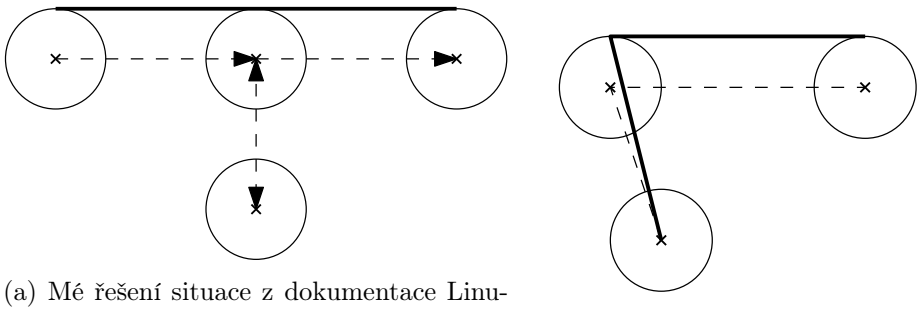
Z této úvahy vyplývá, že pro zpracování lineárního lead-in pohybu stačí počáteční bod zpracovat stejně, jako by se jednalo o klasickou konturu, a jeho koncový bod má souřadnice:

$$\mathbf{P} = \mathbf{B} + k \cdot \mathbf{n}, \tag{8.9}$$

kde $\mathbf{B}$ je počáteční bod navazujícího úseku, $\mathbf{n}$ normálový vektor navazujícího úseku a k je hodnota korekce.



Obrázek 8.4: Implementace chování lineárního lead-in pohybu v systému LinuxCNC. Převzato z <http://www.linuxcnc.org/docs/2.4/html/comp02.png>



(a) Mé řešení situace z dokumentace LinuxCNC

(b) Obecný případ

Obrázek 8.5: Implementace lineárního lead-in pohybu v mém systému. Tlustá čára znázorňuje trasu zadanou G-kódem, přerušovaná pak dopočtenou korekci.

Pokud lineární lead-in pohyb navazuje na vnější roh, je situace obdobná jako v případě klasického korigovaného pohybu – vnější roh je obroben vloženým obloukem, který jej opisuje. Pokud bych však na počátku a na konci lead-in pohybu použil jinou hodnotu korekce, nenavazoval by tento oblouk tečně, tudíž by těsně před ním musel stroj zabrzdit. Ideálně by měl oblouk navazovat tečně, jak znázorňuje obrázek 8.6.

Výpočet pozice bodu C jsem v tomto případě realizoval skrz vektor odsazení $\mathbf{o}$. Mé řešení vychází z pravoúhlého trojúhelníku ABC. Pokud budu uvažovat vektor mezi body A a B jako $\mathbf{u}$, mohu úhel α vyjádřit následovně:

$$\mathbf{u} \cdot \mathbf{o} = \cos \alpha \tag{8.10}$$

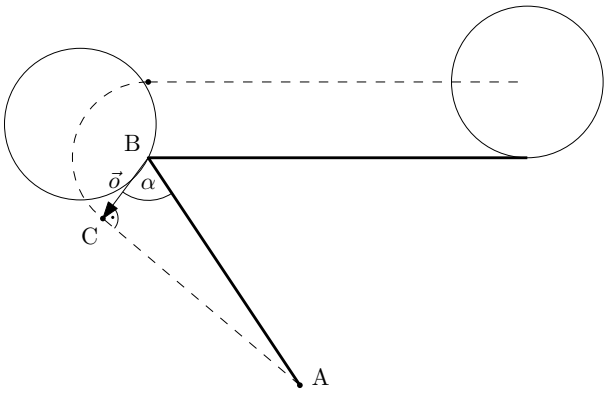
Jelikož je délka vektoru **o** rovna korekci $|k|$, tak můžu vyjářit $\cos\alpha$ i pomocí těchto délek. Získám tedy soustavu rovnic:

$$\begin{cases} \mathbf{u}\cdot\mathbf{o} = \frac{|k|}{|\mathbf{u}|} \\ |\mathbf{o}| = |k| \implies o_1^2 + o_2^2 = k^2 \end{cases} \tag{8.11}$$

Pokud označím $|\mathbf{u}| = d$, řešením této soustavy je

$$\mathbf{o}\begin{cases} o_1 = \frac{dku_1 \pm \sqrt{d^2k^2u_2^2\left(d^2\left(u_1^2+u_2^2\right)-1\right)}}{d^2\left(u_1^2+u_2^2\right)} \\ o_2 = \frac{dku_2^2 \mp u_1\sqrt{d^2k^2u_2^2\left(d^2\left(u_1^2+u_2^2\right)-1\right)}}{d^2u_2(u_1^2+u_2^2)} \end{cases} \tag{8.12}$$

Tento výraz je použitelný i pro situaci, kdy původní korekce nebyla nulová, či dokonce měla opačný směr. Pouze se jako bod A vezme již jeho zkorigovaná hodnota. Nadále zde zůstává pravoúhlý trojúhelník. U tohoto výrazu se mi podařilo určit znaménko ve vzorci pro určení požadovaného řešení. Pokud se nová korekce nachází na stejné straně od kontury jako předcházející, je ve výrazu pro o_1 plus a ve výrazu o_2 mínus. Pokud se původní korekce nacházela na opačné straně od kontury, jsou znaménka přesně opačná.



Obrázek 8.6: Znázornění implementace lead-in pohybu, který navazuje na vnější roh.

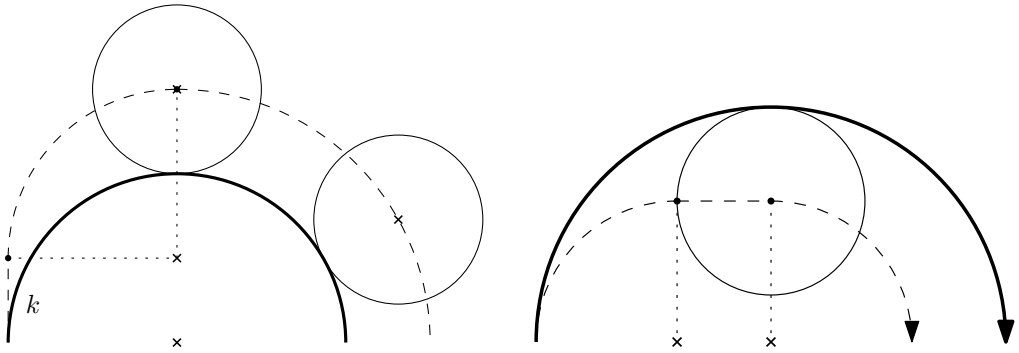
Jak jsem zmínil v úvodu této sekce, rozhodl jsem se do mého systému zaimplementovat podporu pro obloukový lead-in pohyb. Jsou dvě možnosti korekce – výsledkem první je oblouk o menším poloměru, výsledkem druhé je oblouk o větším poloměru. Obě situace znázorňují nákresy 8.7.

Při korekci na oblouk o větším poloměru přidám do trasy úsečku délky k ve směru tečném k počátečnímu bodu oblouku. Následně za tento lineární úsek přidám oblouk o původním poloměru a úhlové délce 90 °. V jeho koncovém bodě se již nástroj dostává do zkorigované pozice a následuje oblouk již o zkorigovaném poloměru.

Při korekci na oblouk o menším poloměru je postup opačný – pohyb začíná obloukem se zkorigovaným poloměrem a úhlové délce 90 °. Za něj vkládám lineární úsek o délce k. V tomto bodě se již nástroj nachází ve zkorigované pozici a následuje zkorigovaný původní oblouk kontury.

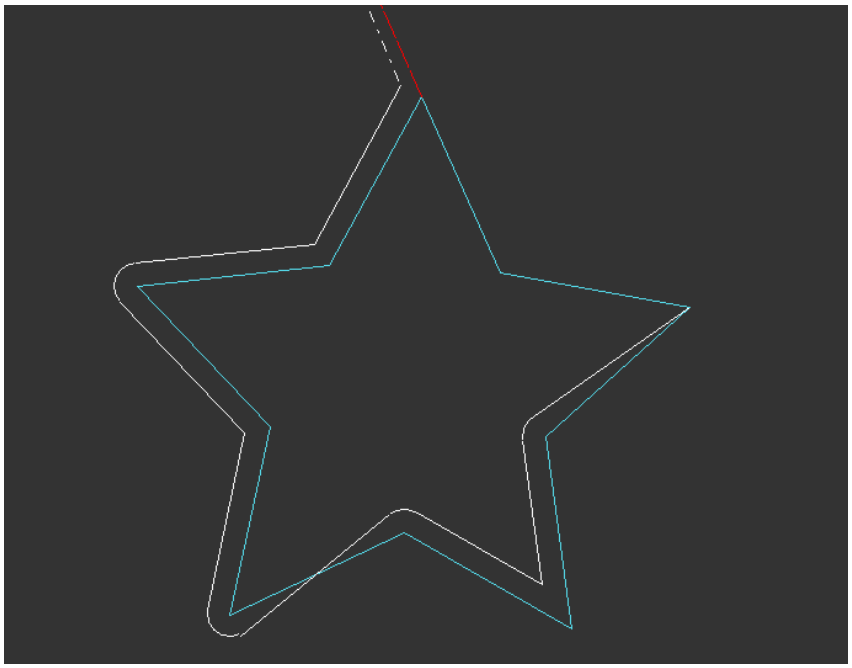
Z mnou použitého řešení, jak implementovat oblouk jako lead-in pohyb, vyplývá jedno omezení – jako lead-in pohyb nelze použít oblouk, který má úhlovou délku kratší než 90 °.

Na obrázku 8.8 si lze prohlédnout výstup již implementované korekce průměru v mém systému. Korekce je znázorněna bílou čarou. Jedná se o ukázkový příklad, kdy byly do jednoduchého programu v G-kódu na náhodná místa přidány funkce G40, G1 a G42.



(a) Korekce na větší poloměr (b) Korekce na menší poloměr

Obrázek 8.7: Znázornění implementace obloukového lead-in pohybu



Obrázek 8.8: Ukázka zpracování korekce průměru nástroje mým systémem. Na počátku kontury je nastavena korekce doprava ve směru obrábění, na třetím vrcholu hvězdy, je změněna korekce na opačnou (doleva). Za čtvrtým vrcholem je korekce vypnuta.

8.5.2 Korekce délky

Oproti korekci poloměru nástroje je implementace korekce délky nástroje jednoduchou záležitostí. Jedná se totiž pouze o korekci ve směru jedné osy (Z), narozdíl od dvou os (X a Y) v případě korekce poloměru nástroje.

Počáteční bod každého úseku je nutno zkorigovat o korekci nastavenou pro úsek předcházející a koncový bod každého úseku je nutno zkorigovat korekcí nastavenou pro aktuálně zpracovávaný úsek.

Zde je nutno podotknout, že tato korekce zatím nebyla v mém systému plně implementována a může způsobovat chyby. Na vině je implementace kruhové interpolace, která není v souladu se standardem G-kódu (viz kapitola 7.3.4). Pokud by nastala změna délkové korekce na kruhovém oblouku, měl by počáteční a koncový bod jiné Z-ové souřadnice. Dle standardu by se tato změna

měla interpolovat po šroubovici, avšak můj systém se pokusí nalézt kruhový oblouk v obecné rovině, který tomuto zadání vyhovuje.

8.6 Výpočet rychlosti

Poslední činností, kterou vykonává třída `GCodeInterpreter` před tím, než vytvoří seznam příkazů pro interpolátor, je dopočtení rychlostí pro každý úsek.

Rychlost na každém úseku je charakterizována počáteční, cílovou a brzdnou rychlostí a také maximálním zrychlením pro rozjezd a brzdění. Na základě těchto údajů je schopen interpolátor provést pohyb. `GCodeInterpreter` na základě posuvu v G-kódu, maximálního zrychlení a ryvu zadaného v konfiguračním souboru a tvaru úseku určuje vhodnou rychlost.

Výpočet rychlosti probíhá pro každý úsek samostatně. Pro vypočtení rychlosti celé trasy tedy procházím každý prvek trasy od konce po začátek. Při provádění výpočtu rychlosti od konce je totiž menší šance na vznik cyklické závislosti – může se stát, že vypočtená počáteční rychlost je pro daný prvek příliš velká, a musím se proto tedy vrátit na n předcházejících prvků a ty znovu spočítat. Při procházení trasy od konce však tato situace nastává pouze zřídka, např. při seskupení krátkých oblouků o malém poloměru za sebou. Výpočet rychlosti daného prvku má na starosti metoda `PathPart::ComputeSpeeds`.

V následujícím textu používám pojmy předcházející a následující úsek. Ačkoliv úseky zpracovávám v opačném pořadí (od posledního po první), pojmem následující úsek mám na mysli úsek, který následuje během pohybu (tedy pořadí od prvního po poslední), resp. pojmem předchozí úsek mám na mysli úsek, který předchází během pohybu.

Pro jednotlivé rychlosti platí, že počáteční a koncová rychlost je menší nebo rovna požadované. Tato podmínka vychází z logiky, že posuvem je dána maximální možná rychlost na daném úseku, která nesmí být překročena. A jelikož posuv určuje požadovanou rychlost, tak počáteční i koncová rychlost musejí být menší.

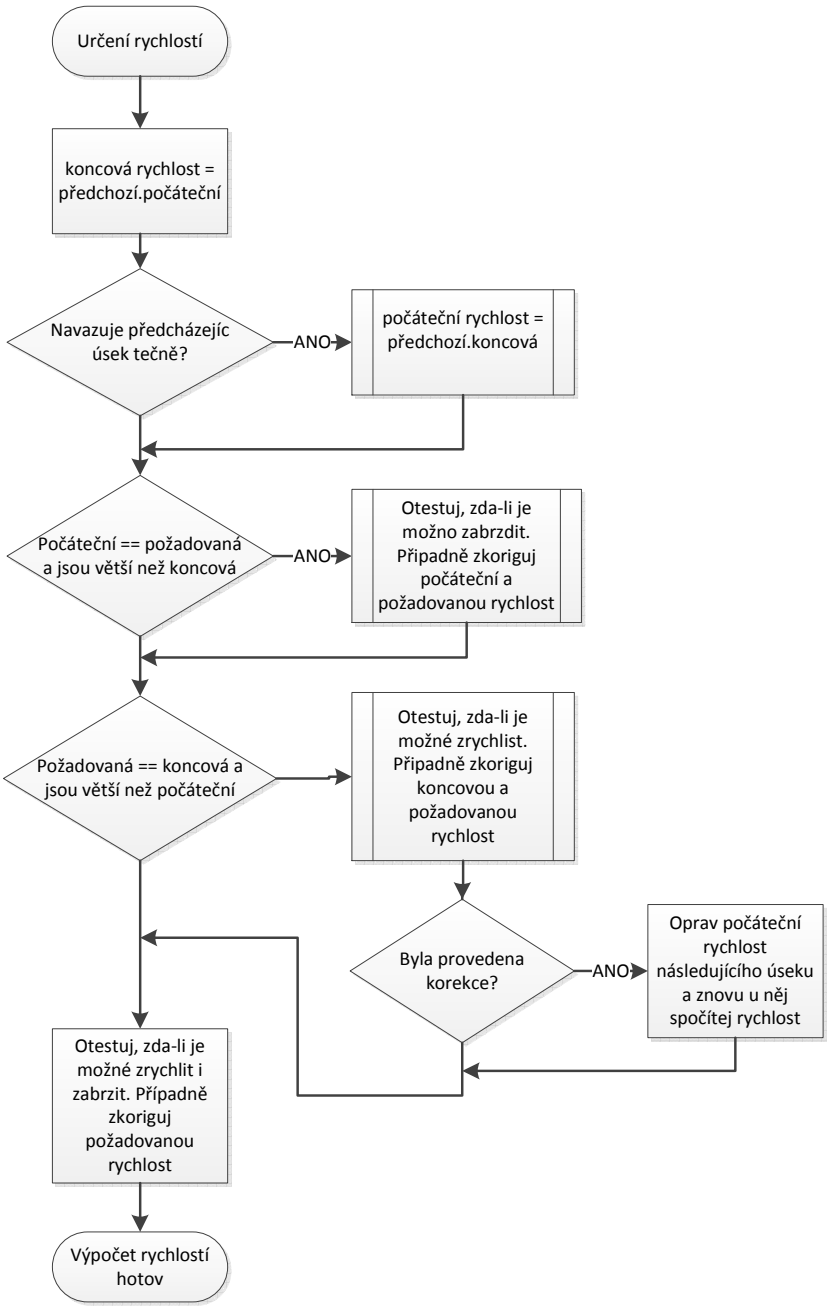
Postup výpočtu, který je popsán v následujících odstavcích je také znázorněn na schématu 8.9.

V každém úseku před zpracováním je počáteční a koncová rychlost nastavena na nulu a cílová rychlost je nastavena dle posuvu. Při výpočtu se koncová rychlost úseku nastaví na počáteční rychlost následujícího úseku. Poté se testuje, zda-li předcházející úsek navazuje tečně. Pokud ano, je současnému prvku nastavena počáteční rychlost rovna požadované rychlosti předchozího prvku (za podmínky, že je menší nebo rovna požadované rychlosti aktuálního prvku).

Nyní je třeba otestovat, zda-li je možno požadovaných rychlostí dosáhnout na aktuálním úseku (jeho délce). Pro tento test je nutno spočítat rozjezdovou a brzdnou dráhu (podrobněji popsáno v kapitolách 8.6.1 a 8.6.2).

Test rychlosti probíhá v několika úrovních. Pokud je počáteční rychlost rovna požadované a zároveň jsou tyto rychlosti větší než koncová, je proveden test navíc - je totiž nutné určit, zda-li je možno na zadaném úseku ze zadané počáteční rychlosti zbrzdit na koncovou. Pokud ne, je nutné vhodně upravit počáteční (a požadovanou) rychlost.

Následně se určí jak brzdná, tak rozjezdová dráha a zkoumají se jejich hodnoty. Pokud je brzdná dráha nulová a tedy požadovaná rychlost rovna koncové, zkoumá se, zda-li je možno požadované rychlosti dosáhnout. Pokud ne, je třeba snížit koncovou rychlost. Jelikož jsem změnil hodnotu koncové rychlosti, musím změnit hodnotu počáteční rychlosti následujícího úseku, který již byl zpracován, a znovu jej zpracovat. Tento případ však nenastává často (to je důvod, proč zpracovávám úseky od konce).



Obrázek 8.9: Znázornění postupu výpočtu rychlosti daného úseku.

Pokud brzdná dráha není nulová, nastává obecný případ. Pokud je součet obou drah větší než délka úseku, je nutné pomocí bisekce najít novou požadovanou rychlost. Tím je výpočet rychlosti na úseku vyřešen.

8.6.1 Určení rozjezdové a brzdné dráhy při lineárním pohybu

Určení délky rozjedové (brzné) dráhy při lineárním pohybu má na starosti metoda `PathPartLine::GetSFromStartMovement`, resp. `GetSFromEndMovement`. Tato metoda volá metodu

`PathPartLine::GetATFromMovement`, která na základě předané počáteční a požadované rychlosti dopočte hodnoty maximální amplitudy zrychlení A a doby pohybu T při zadaných omezeních. Poté z těchto hodnot podle vztahu 5.9 z kapitoly 5.2.1 (resp. vztahu 5.10 pro brzdění) dopočte dráhu. Zároveň uloží vypočtené zrychlení do interního atributu `Astart` (resp. `Aend`) třídy `PathPartMovable`, které je poté součástí příkazu pro interpolátor.

Metoda `GetATFromMovement` prvně testuje, zda-li nejsou předané rychlosti stejné. Pokud ano, vrátí nulový čas a maximální možné zrychlení. Pokud ne, pokusí se prvně dopočítat čas pohybu za omezení ryvem. Využívá k tomu vztah 5.16. Z vypočteného času poté určí potřebnou amplitudu zrychlení. Pokud je tato amplituda menší než konfigurací nastavená maximální hodnota, vrátí funkce již vypočtený čas a příslušné zrychlení. Pokud by při omezení ryvem bylo přesáhnuto maximální zrychlení, je dopočten čas pohybu za omezení zrychlením dle vztahu 5.7. Poté je navrácen tento nově vypočtený čas a maximální přípustná hodnota zrychlení.

8.6.2 Určení rozjezdové a brzdné dráhy při pohybu po oblouku

I u obloukového pohybu jsou použity pro výpočet drah metody `GetSFromStartMovement` a `GetSFromEndMovement`. Tyto metody volají přetíženou metodu `GetATFromMovement`, která vrací příslušné hodnoty A a T pro pohyb na oblouku pro předané rychlosti za daných omezení. Metoda `GetSFromStartMovement` (resp. `GetSFromEndMovement`) využívá pro určení dráhy stejné vztahy jako její obdoba pro lineární pohyb.

Metoda `GetATFromMovement` prvně zkontroluje, zdali je možno požadované rychlosti dosáhnout – podmínka vyplývající ze vztahu 5.22. Tato podmínka je zde pro debuggovací účely – již v dřívějších krocích by mělo být zajištěno, aby požadovaná rychlost nepřekračovala tuto mezní hodnotu.

Funkce dále samostatně řeší případ, kdy je požadovaná rychlost rovna mezní hodnotě. V tomto případě prvně počítám hodnoty A a T za omezení ryvem s jeho maximální hodnotou v $t = 0$, jelikož výpočet s maximem v $t = \frac{3T}{4}$ je výpočetně náročnější (je řešen numericky), proto jej počítám pouze v krajním případě. K výpočtu těchto hodnot jsou použity vztahy 5.16 a 5.23. Na základě spočítaného A a T dopočítám hodnotu ryvu v čase $t = \frac{3T}{4}$ (dle prvního vztahu ze soustavy 5.24). Pokud je tato hodnota vyšší než hodnota v čase $t = 0$, musím omezit pohyb ryvem s maximem v $t = \frac{3T}{4}$. Tento numerický výpočet provádí metoda `ComputeATLimitedByJerk` popsaná v sekci 8.6.2.

Pokud požadovaná rychlost nedosahuje mezní hodnoty, jsou hodnoty A a T určeny obdobně. Pouze je na závěr ještě nutno provést test, zda-li nebylo překročeno maximální zrychlení. Tento test není nutno v mezním případě provádět, protože maximum zrychlení automaticky nastává v čase $t = T$ a dosahuje přesně mezní hodnoty. Maximum celkového zrychlení zde nastává v čase $t = \frac{T}{2}$ a pro jeho výpočet je použit vztah 5.26. Pokud je překročeno, je nutné pohyb limitovat zrychlením. Příslušné hodnoty A a T v tomto případě určují vztahy 5.27.

Řešení soustavy rovnic Newtonovou metodou

Pro výpočet rychlosti, kdy maximum ryvu nastává v čase $t = \frac{3T}{4}$, je třeba vyřešit soustavu rovnic 5.24. Tato soustava však není řešitelná obecně, je třeba využít numerické metody. K jejímu řešení v mé implementaci používám Newtonovu metodu, jelikož se snadno aplikuje na soustavu rovnic a rychle konverguje[25]. Postup, jak řešit soustavu rovnic Newtonovou metodou jsem čerpal z dokumentu [8].

Princip řešení soustavy rovnic Newtonovou metodou spočívá v převedení hledaného kořene na vektor a nahrazení derivace příšlušné funkce maticí parciálních derivací daných funkcí[8].

Soustavu rovnic tedy musím převést na funkci[8]

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} f_1(x_1,x_2,\ \dots\ ,x_n) \\ f_2(x_1,x_2,\ \dots\ ,x_n) \\ \vdots \\ f_n(x_1,x_2,\ \dots\ ,x_n) \end{pmatrix}, \tag{8.13}$$

jejíž derivací je matice parciálních derivací[8]

$$D\mathbf{f}(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(x) & \frac{\partial f_1}{\partial x_2}(x) & \cdots & \frac{\partial f_1}{\partial x_n}(x) \\ \frac{\partial f_2}{\partial x_1}(x) & \frac{\partial f_2}{\partial x_2}(x) & \cdots & \frac{\partial f_2}{\partial x_n}(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(x) & \frac{\partial f_n}{\partial x_2}(x) & \cdots & \frac{\partial f_n}{\partial x_n}(x) \end{pmatrix} \tag{8.14}$$

Mnou hledaným kořenem soustavy je vektor $\mathbf{p}$, pro nějž platí[8]

$$\mathbf{f}(\mathbf{p}) = 0 \tag{8.15}$$

Rekurzivní vztah pro kořen soustavy tedy potom vypadá následovně[8]:

$$\mathbf{p_{n+1}} = \mathbf{p_n} - \left(D\mathbf{f}(\mathbf{p_n})\right)^{-1} \mathbf{f}(\mathbf{p_n}) \tag{8.16}$$

Implementace mého konkrétního problému Newtonovou metodou není složitá, jelikož mám pouze soustavu dvou rovnic, tudíž pracuji s dvojrozměrným vektorem a maticí 2×2.

Tím se mi výrazně zjednodušuje hledání inverzní matice parciálních derivací. Pokud mám matici 2×2

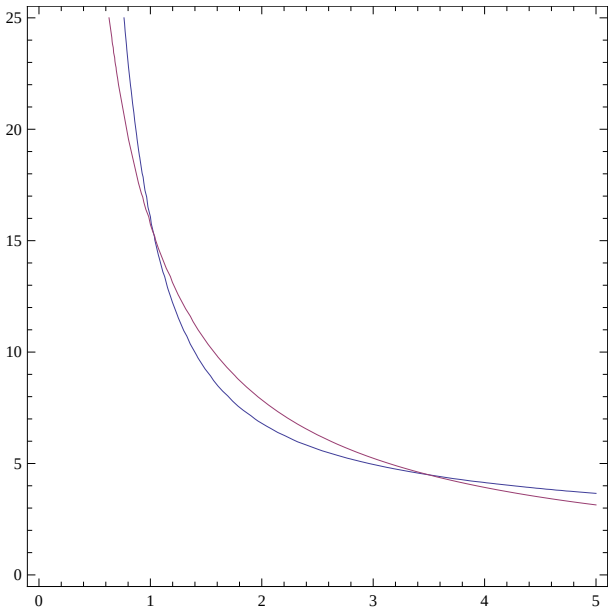
$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \tag{8.17}$$

tak její inverzní matici můžu vypočítat následovně[18]:

$$A^{-1} = \frac{1}{\det A} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \tag{8.18}$$

Následné přepsání do kódu už je pouze rutinní záležitost. Díky jednoduchosti hledání inverzní matice není nutno použít speciální knihovnu pro práci s maticemi a celý problém lze naimplementovat s použitím několika málo proměnných.

Soustava 5.24 reprezentuje v kladných hodnotách (jiné nemají pro můj problém smysl) dvě křivky se dvěma průsečíky, jak lze vidět na grafu 8.10. Rovnice má tedy dva reálné kořeny v kladných hodnotách (další reálné řešení se nachází v záporném čase). První řešení s nižší hodnotou času pohybu T, vždy přesahuje maximální zadanou hodnotu A (experimentálně ověřeno na vzorku reálných dat). Za relevatní tedy považuji druhé řešení s vyšší hodnotou periody T. Odtud se odvíjí můj odhad kořene pro Newtonovu metodu. Jako počáteční hodnotu volím dostatečně velkou hodnotu doby pohybu T – konkrétně stonásobek hodnoty vypočtené pomocí vztahu, který omezuje ryv v čase $t = 0$. Tím dosáhnu toho, že se k hodnotě blížím zprava a jako první naleznu pro mě relevantní řešení.



Obrázek 8.10: Graf znázorňující řešení soustavy rovnic 5.24. Vodorovná osa reprezentuje dobu pohybu T, svislá amplitudu zrychlení A.

8.6.3 Určení nové rychlosti pomocí bisekce

Jak jsem zmínil v předcházejícím textu, pro výpočet mezní rychlosti jsem použil metodu bisekce. Bisekce je též známá jako metoda půlení intervalů[1]. Tato metoda se zpravidla používá pro hledání kořene funkce mezi dvěmi zadanými mezemi. Metoda pracuje tak, že určuje, v které polovině daného intervalu se nachází hledaná hodnota. Jakmile určí správný interval, rekurzivně se opakuje a opět půlí tentokrát již menší interval[1]. Rekurze je ukončena, jakmile je buď nalezena správná hodnota, nebo je dosaženo požadované přesnosti.

V praxi však není hojně používána, jelikož konverguje relativně pomalu (např. oproti Newtonově metodě). Avšak pro mě má bisekce výhodu v tom, že ji lze použít nejen pro jednoduché hledání kořene, ale i k hledání parametru v určitém vztahu s relativně složitou podmínkou, což je právě můj případ pro hledání rychlosti.

Ačkoliv jsem v kapitole 5.2.2 uvedl vztah, ze kterého lze Newtonovou metodou určit maximální dosažitelnou rychlost pro lineární pohyb, nepoužívám jej v mém řídicím systému. Problémem tohoto řešení je, že je nutno vytvořit 4 varianty tohoto vztahu – pro různé kombinace omezení rozjezdu a brzdění omezeného ryvem či zrychlením. Také je nutné určit, který vztah je třeba použít. Výsledkem byl složitý kód, který měl podobnou časovou náročnost jako má současná implementace (avšak měl o něco lepší přesnost – zde jsem generoval rychlost s přesností 0,01 $\mathrm{mm{\cdot}s^{-1}}$, nyní generuji rychlost s přesností 0,1 $\mathrm{mm{\cdot}s^{-1}}$, což je více než dostatečné). Situace pro výpočet rychlosti na oblouku byla ještě složitější a počet variant zde prudce vzrostl. Proto jsem se rozhodl přiklonit k metodě bisekce.

Bisekci v kódu implementuji pomocí cyklu. Tento cyklus probíhá, dokud velikost intervalu je větší než 0,1 a nebylo provedeno více než 10 iterací. V tomto cyklu spočítám dráhu pro rychlost v polovině daného intervalu. Na základě této dráhy a podmínky rozhodnu, který interval budu dále půlit. Jako jedna hranice počátečního intervalu slouží nyní nevyhovující požadovaná rychlost a jako druhá slouží počáteční nebo koncová rychlost – podle toho, která z nich je větší. Na reálných programech je rychlost s přesností na 0,1 $\mathrm{mm{\cdot}s^{-1}}$ dosažena během 4 iterací.

8.7 Komunikace s interpolátorem

Pro komunikaci s interpolátorem přes USB používám knihovnu WinUSB. Tato knihovna obsahuje driver, který mi umožní z user space operačního systému otevřít jednotlivé endpointy a zapisovat na ně, popř. z nich číst. Odprostí mě tak od nutnosti psát vlastní driver v kernel space[9].

Ačkoliv je WinUSB dílem Microsoftu, musí tento ovladač projít stejným certifikačním procesem jako jakýkoliv jiný driver. To je pravděpodobně způsobeno tím, že pro použití musím upravit INF soubor ovladače, čímž poruším integritu dat a již dříve udělený certifikát by se stal neplatným. Problematika ovladačů a jejich certifikování na Windows je poměrně rozsáhlá a bohužel se mi do ní nepodařilo zcela proniknout. Je možné, že mnou vytvořený certifikát a vlastně i celý balíček ovladače nesplňuje všechny požadavky Microsoftu, avšak podařilo se mi jej nainstalovat na několika počítačích a různých systémech (Windows XP, Windows 7 x86 i Windows 7 x64).

8.7.1 Tvorba balíčku ovladačů WinUSB

Aby WinUSB spolupracoval s mým zařízením, je nutné upravit vzorový INF soubor[9]. Tento soubor slouží jako popis driveru, resp. zařízení.

V sekci `Version` je nutné vyplnit jméno poskytovatele driveru, uvést třídu zařízení, verzi a datum vydání driveru, jeho GUID a jako poslední také jméno souboru katologu. Katalog obsahuje hash všech souborů, které ovladač obsahuje a tím slouží k ověření, zda-li soubory nebyly modifikovány. K ověření pravosti katalogu slouží certifikát, jak zmiňuji níže.

GUID (Globally unique identifier) je 128 bitové číslo[24]. Toto číslo slouží jako identifikátor COM objektu, na základě něhož si je možné z mé aplikace tento objekt vyvolat a skrz něj komunikovat se zařízením. GUID se generuje náhodně – vzhledem k velikosti čísla je šance, že by se podařilo vygenerovat dvě stejná GUID velmi malá[9].

V sekci `Manufacturer` je třeba nastavit identifikátor, resp. cestu k zařízení, pro jednotlivé platformy. Tato cesta je ve tvaru `USB\VID_xxxx&PID_yyyy`, kde xxxx je příslušné vendor ID zařízení a yyyy je příslušné product ID zařízení.

Úpravou sekce `SourceDisksNames` je možné docílit vlastního pojmenování instalačních disků (toto jméno se zobrazuje během instalace ovladače). Poslední sekcí, kterou je třeba upravit, je sekce `Strings`. Touto sekcí lze nastavit jméno zařízení tak, jak se bude zobrazovat ve správci zařízení a specifikovat výrobce, kterého lze zobrazit ve vlastnostech daného zařízení.

Jakmile je INF soubor upraven, je třeba vytvořit katalog ovladače[14]. Katalog zajišťuje integritu celého ovladače a umožňuje systému před instalací ověřit, zda-li nebyl driver modifikován. Microsoft poskytuje utilitu zvanou Inf2Cat[14], která na základě předaného INF souboru vytvoří katalog všech souborů, na které INF soubor odkazuje. V případě problémů je možné také použít utilitu MakeCat[14], která vytvoří katalog na základě předaného seznamu souborů.

Následně je nutno katalog podepsat[3], čímž je zajištěna jeho pravost. Toto podepsání lze provést pomocí utility SingTool[14]. Pro podepsání je však třeba mít certifikát. Aby driver při instalaci do systému prošel bez chyby ověřovacím procesem, je třeba mít certifikát vydaný institucí schválenou Microsoftem[14]. Tato možnost je relativně drahá a hlavně pro fyzickou osobu nedostupná. Naštěstí Microsoft umožňuje pro vývojové účely vygenerovat si vlastní certifikát, který umožní katalog podepsat[14]. Avšak při instalaci ovladače se zobrazí, že certifikát pochází od nedůvěryhodné osoby. Tento certifikát je možné vygenerovat si pomocí utility MakeCert[14].

[3]Windows XP podepsání nevyžadují, avšak zobrazí varování o nedůvěryhodném driveru. Windows Vista a výše však již podepsání vyžadují, jinak odmítnou ovladač nainstalovat.

Mnou upravený balíček ovladačů WinUSB, společně s katalogem a příslušnými utilitami lze nalézt na přiloženém CD.

8.7.2 Třída MyDevice

Třída `MyDevice` reprezentuje rozhraní pro komunikaci s interpolátorem. Zaobaluje veškerou funkcionalitu okolo WinUSB – stará se o inicializaci zařízení (otevření endpointů), jeho správnou deinicializaci při ukončení aplikace a reaguje na jeho připojení či odpojení.

Posílání zpráv do interpolátoru je realizováno metodou `SendRawData`, která přebírá data k odeslání buď jako ukazatel na pole a jeho délku nebo jako `std::vector<char>`. Tato metoda zkontroluje, zda-li je zařízení připojeno a případně odešle data do interpolátoru pomocí funkce `WinUsb_WritePipe` přes příslušný endpoint. Jelikož je obrovské množství zpráv, které je možné do zařízení odeslat, nedefinuje tato třída pro každou zprávu speciální metodu, ale raději poskytuje obecnou metodu pro zápis libovolných dat. V mém návrhu existuje pouze několik málo speciálních zpráv (požadavků, na které je očekávána odpověď), pro které je definována speciální metoda. Tyto požadavky jsou popsány v následujícím odstavci.

Třída `MyDevice` také uchovává objekt třídy `ReceiveFromDevice`, který se stará o příchozí data. Tato třída je popsána v následující podkapitole. Ve spolupráci s ní definuje třída `MyDevice` speciální metody `GetFreeSpaceInQueue`, `GetLastProcessedItemStack`, `GetCurrentlyProcessedItemStack`. Tyto metody odešlou příslušnou zprávu pro intepolátor (dle svého názvu) a čekají na odpověď. Jsou tedy definovány jako blokující (proto by neměly být volány ve vláknu uživatelského rozhraní). Jako návratovou hodnotu mají již zpracovanou odpověď na danou zprávu.

8.7.3 Třída ReceiveFromDevice

Třída `ReceiveFromDevice` je potomkem třídy `wxThread` – reprezentuje tedy vlákno. Uvnitř tohoto vlákna je v nekonečném cyklu volána blokující funkce `WinUsb_ReadPipe`, která čte z daného endpointu. Tato třída skládá dohromady zprávy rozdělené na více paketů stejně jako to dělá interpolátor (popsáno v kapitole 7.1.1). Pro každou přijatou zprávu pak volá metodu `ProcessMessage`. V této metodě je definováno zpracování všech možných zpráv.

8.7.4 Třídy ProgramControl a ProgramRun

Nejdůležitějším typem komunikace je odesílání příkazů interpolátoru do jeho komponenty `CommandStack`, odkud jsou postupně vykonávány. Další důležitou funkcí je zpětná vazba o poloze stroje, která je následně zobrazována uživateli. Tyto operace má na starosti třída `ProgramRun`, která je ovládána třídou `ProgramControl`.

Třída `ProgramRun` je potomkem třídy `wxThread` – reprezentuje tedy vlákno. Proto jsem pro interakci tohoto vlákna se zbytkem aplikace implementoval třídu `ProgramControl`. Třída `ProgramControl` při spuštění vykonávání programu v G-kódu vytvoří novou instanci třídy `ProgramRun`. Tuto třídu, respektive vlákno, poté ovládá pomocí příznaků. Příznak může mít tři stavy – stav signalizující běžící program, stav signalizující pozastavení vykonávání a stav zastavení programu.

Během běhu programu třída `ProgramRun` se neustále dotazuje interpolátoru na volné místo v zásobníku a pokud nějaké volné místo je, odešle patřičný počet příkazů do interpolátoru. Zároveň také posílá dotaz na aktuální pozici stroje. Tento dotaz je posílán v neblokujícím režimu, jelikož jeho výsledek není nutný pro další běh vlákna (na rozdíl od dotazu na volné místo). Odpověď na pozici stroje zpracuje třída ReceiveFromDevice, která zajistí předání dat na správné místo tak, aby byla pozice stroje uživateli znázorněna.

Pokud je běh programu pozastaven, pošle vlákno interpolátoru příkaz pro zastavení aktuálně vykonávané činnosti a čeká na znovuobnovení běhu programu.

Pokud je program zastaven, odešle vlákno interpolátoru příkaz pro zastavení aktuálně vykonávané činnosti a příkaz pro vyprázdnění zásobníku příkazů. Poté se vlákno ukončí.

Část III

Závěr

Kapitola 9

Zhodnocení plynulosti pohybu

Nejen po dokončení, ale především i během debuggování řídícího systému jsem se snažil zhodnotit plynulost pohybu. Bohužel se mi nepodařilo dosáhnout očekávaných výsledků objektivních měření díky ne zcela vhodnému testovacímu stroji pro tato měření. Během debuggování jsem se tedy spoléhal na subjektivní hodnocení – především na pohled, ale také i na poslech. Krokové motory totiž vydávají charakteristický zvuk dle svých otáček a jakékoliv cuknutí v pohybu je tedy i slyšet.

V následujících podkapitolách popisuji způsob měření a výsledky. Veškeré výsledky se týkají nejnovější verze softwaru, která se nachází na přiloženém CD.

Pro generování programů v G-kódu jsem použil grafický program Inkscape s pluginem GCodeTools[1]. Tento jednoduchý CAM program umožňuje generovat G-kód pro 2D vyřezávání a gravírování, což pro mé potřeby debuggování plně dostačuje.

9.1 Subjektivní hodnocení

Pohyb testovacího plotteru řízeného mým interpolátorem vypadá na pohled plynule. O tom se lze přesvědčit na videích, která se nacházejí na přiloženém CD. I při dotyku rukou nejsou cítit žádné rázy. Jsou však cítit relativně silné vibrace. Tyto vibrace pocházejí od krokových motorů a jsou umocněny nepříliš tuhou konstrukcí a malou hmotností plotteru. Při umístění kilogramového závaží na pohyblivou plošinu plotteru jsou vibrace výrazně utlumeny.

Při umístění skleničky s vodou na plotter se hladina rozechvívá především díky vibracím. To usuzuji z toho, že i při dlouhých pojezdech relativně malou rychlostí se na hladině objevují soustředné kruhové vlnky. Tehdy by se objevovat neměly, jelikož se jedná o pohyb rovnoměrný přímočarý. Při projíždění „cik-cak čáry“ jeví hladina jen malou tendenci se rozpohybovat a stále jsou mnohem výraznější vibrace od krokových motorů.

9.2 Objektivní měření

Abych ověřil funkčnost výpočtu rychlosti nejen subjektivně, ale také objektivně, rozhodl jsem se provést měření, která by funkčnost dokázala.

[1] <http://www.cnc-club.ru/forum/viewtopic.php?t=35>

9.2.1 Měření na základě polohy

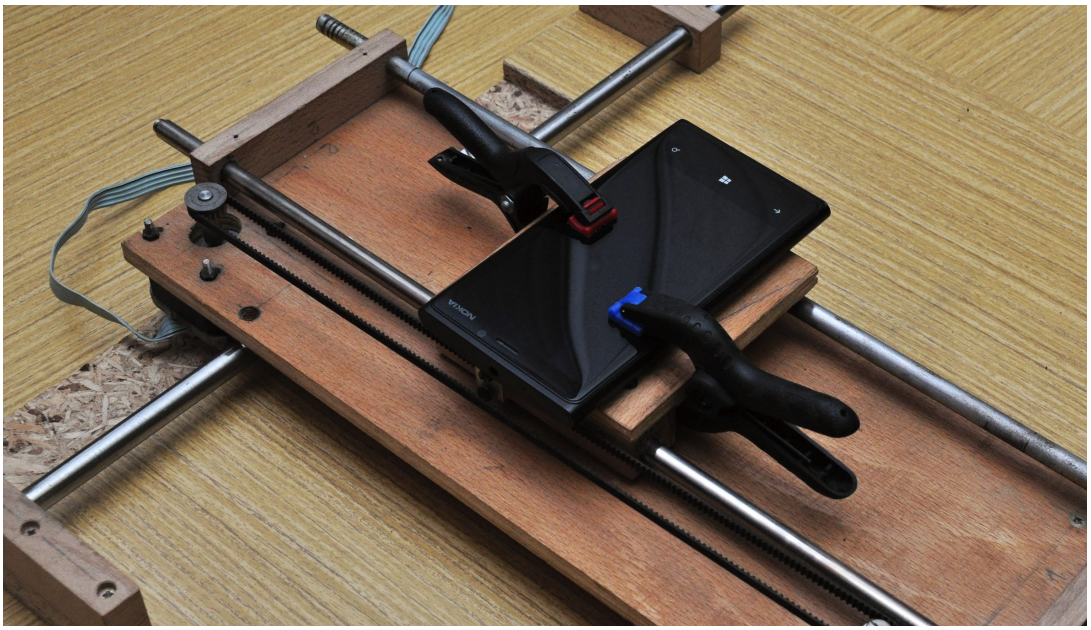
Jako první jsem chtěl z interpolátoru odesílat zpět do počítače aktuální pozici jednotlivých os v konstatním časovém intervalu. Na základě těchto dat jsem chtěl dopočítat aktuální rychlost, zrychlení a ryv. Běžně je pro grafické znázornění polohy na počítači použita zpráva `STATE_MESSAGE_POSITION`. Tato zpráva se však dotazuje v ne příliš přesném časovém rámci, což pro potřeby výše popsaného výpočtu nestačí.

Periodu odesílání dat jsem zvolil 10 ms. Ačkoliv jsem odesílal pouze pozici os x a y, tedy dohromady 8 bytů, narazil jsem na propustnost rozhraní USB, resp. limity integrované periférie v mikroprocesoru či způsobu vyčítání dat na počítači. Interpolátor přestal stíhat interpolovat a aplikace na počítači přestávala reagovat. Ani paketizace dat (kdy jsem posíl větší množství dat naráz) nepomohla. Příčinu této relativně malé propustnosti se mi nepodařilo zjistit, ačkoliv při dřívějších testech přenosová rychlost opačným směrem (z počítače do mikrokontroléru) dosahovala několika $\mathrm{Mbit \cdot s^{-1}}$

9.2.2 Měření pomocí akcelerometru

Dále jsem se pokusil změřit působící zrychlení na plotteru pomocí integrovaného MEMS akcelerometru v mobilním telefonu. Měření jsem prováděl na mobilní telefonu Nokia Lumia 920. Tento telefon jsem vybral, jelikož ze všech pro mě dostupných telefonů má nejnižší hladinu šumu výstupu z akcelerometru. Od tohoto měření jsem si nesliboval žádné výsledky, prováděl jsem je pro zajímavost. O to více mě překvapily výsledky tohoto měření.

Pro telefon jsem napsal jednoduchou aplikaci, která s nejvyšší možnou frekvencí (u mnou použitého telefonu to je 66 Hz) vyčítá data z akcelerometru. Tato zaznamenaná data jsem si díky relativně omezenému způsobu přístupu k paměti v systému Windows Phone 8 posílal pomocí e-mailu do počítače. Telefon byl k plotteru pevně připevněn pomocí modelářských svorek (obrázek 9.1).



Obrázek 9.1: Metodika měření zrychlení pomocí akcelerometru.

Pro vyčtení dat z akcelerometru jsem použil dva způsoby. První způsob využívá Motion API[2]. Toto API poskytuje informace o poloze telefonu založené na všech dostupných senzorech. Pro má měření jsem využíval člen `DeviceAcceleration` struktury `MotionReading`. Tento člen poskytuje trojrozměrný vektor vyjadřující veškerá zrychlení působící na telefon s odečtením tíhového.

Druhý způsob využíval třídu `Accelerometer` ze jmeného prostoru `Microsoft.Devices.Sensors`[3]. Tato třída poskytuje přímo data z akcelerometru. Abych získal informace o pohybu, uložil jsem si v klidovém stavu telefonu vektor působícího tíhového zrychlení, a následně jsem jej při každém měření odečítal od vektoru všech působících zrychlení. Tím jsem dostal pouze vektor zrychlení způsobující pohyb.

Ačkoliv jsem od druhého způsobu očekával přesnější výsledky, ukázala se data vyčtená oběma způsoby jako téměř rovnocená.

Data jsem získal z následujícího programu v G-kódu:

```
G00 X0 Y0 Z0.000000
G01 X100 F2500
G01 Y20
G01 X0 Y0
```

Tento program zobrazuje trojúhelník. První částí pohybu je 100 mm dlouhý posun pouze osou x rychlostí 2500 $\mathrm{mm{\cdot}min^{-1}}$ (přibližně 42 $\mathrm{mm{\cdot}s^{-1}}$). Druhá část je 20 mm dlouhý posun pouze osou y. Následně se plotter vrátí do výchozích souřadnic – nastvá pohyb obou os. Maximální amplituda zrychlení byla nastavena na 4000 $\mathrm{mm{\cdot}s^{-2}}$ a maximální ryv na 8000 $\mathrm{mm{\cdot}s^{-3}}$.

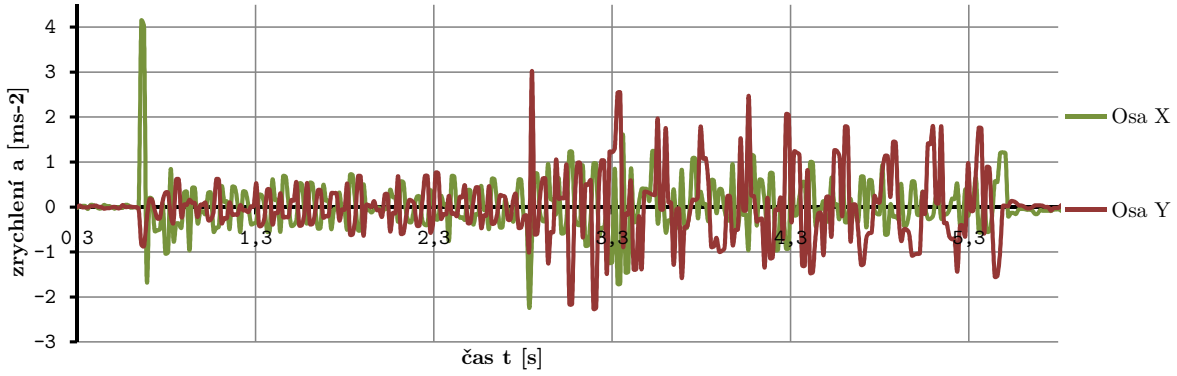
Mnou naměřená data jsou zobrazena na grafu 9.2. Jelikož jsou tato data nepřehledná, zkusil jsem vyfiltrovat – každou hodnotu jsem vzal jako průměr dvou předcházejících a dvou nadcházejících prvků. Tato vyfiltrovaná data jsou zobrazena na grafu 9.3. Na těchto datech jsou silně patrné vibrace od krokových motorů. Na datech lze spatřit vykonávaný pohyb, ale nelze je prohlásit za 100% průkazná.

Mezi časy $t = 0{,}6$ s a $t = 2{,}8$ s lze vidět 100mm pohyb osou x. V čase 0,6 sekundy se nachází výrazný peak zrychlení na ose x – to je zrychlení nutné pro dosažení požadované rychlosti. Tento peak je jasně vidět, jelikož plotter byl v klidu a nenastaly na něm žádné vibrace. V čase 2,8 sekundy lze vidět peak na ose x s opačnou amlitudou. To je zpomalování na konci pohybu. Výchylka pro zrychlení odpovídá svou velikostí nastaveným hodnotám. Výchylka pro zpomalení je o něco menší – to si vysvětluji jako nepřenost měření způsobenou vibracemi. K těmto peakům na ose x lze najít i stejně široké malé peaky na ose y. Jejich existenci si vysvětluji nepřesným upevněním telefonu, kdy měřící osa x v akcelerometru nebyla uložena zcela rovnoběžně s osou x na plotteru. Mezi těmito dvěma peaky probíhá pohyb rovnoměrný přímočarý a naměřené zrychlení je způsobeno vibracemi od plotteru.

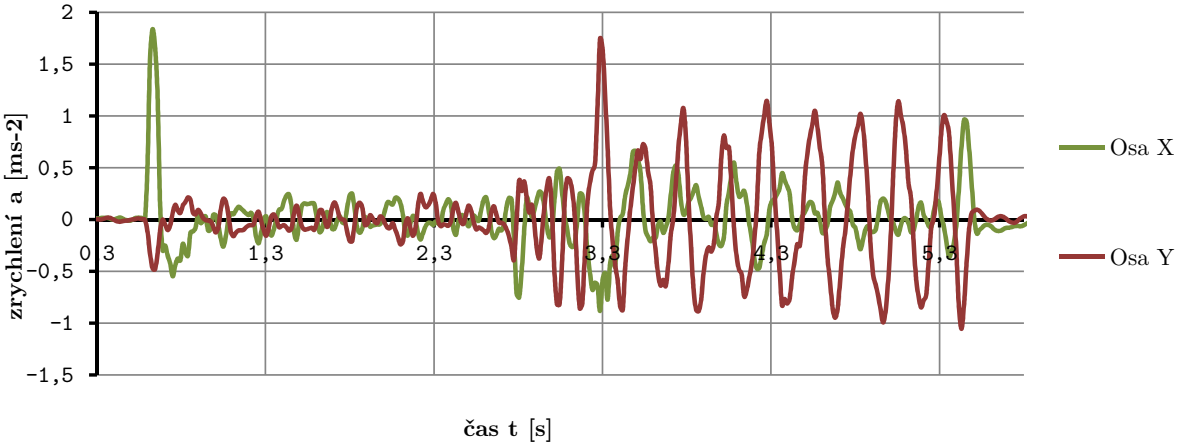
V grafech již nelze vypozorovat pohyb podél osy y. Zde předpokládám, že vyčtení znemožnily silné vibrace. Osa y se totiž nachází na portálu, v němž je upevněn její motor. Tento portál poté s motorem rozonuje. Portál je také pevně uložen v pozdrech na vodicí tyči pouze na jedné straně – na druhé straně po vodicí tyči klouže (kdyby byl uložen pevně i na druhé straně, musela by být naprosto přesně seřízena rovnoběžnost vodicích tyčí). Motor je navíc uložen na klouzající straně, čímž se celá situace ještě zhoršuje. Na ose x nejsou tyto vibrace tak silné, jelikož je pevně spojena se základnou a je celkově tužší.

Jelikož první měření poskytla relativně slibné výsledky, rozhodl jsem se upravit metodiku měření a pokusit se získat průkaznější data. Vibrace konstrukce jsou obrovským problémem, a tak jsem se rozhodl pohyblivou část plotteru zatížit 1,5kg závažím, na které jsem upevnil

[2] <http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh202984(v=vs.105).aspx>
[3] <http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff431810(v=vs.105).aspx>



Obrázek 9.2: Naměřená data z akcelerometru.



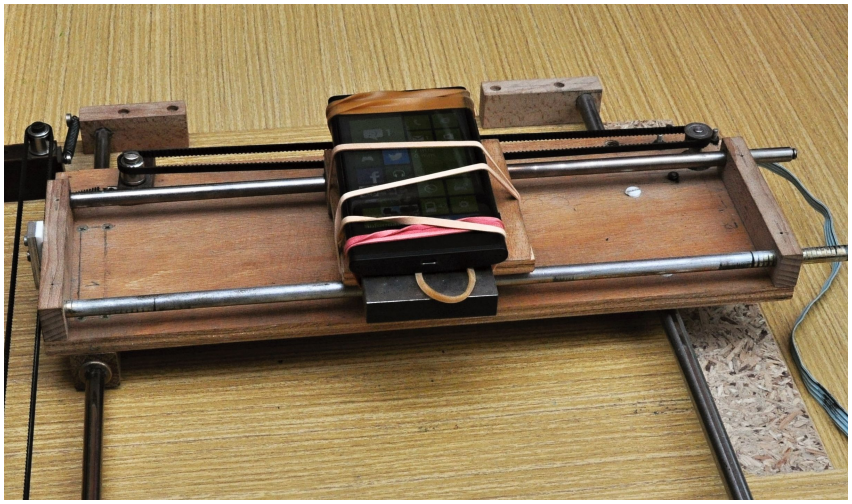
Obrázek 9.3: Filtrovaná data z grafu 9.2

telefon. Telefon jsem tentokrát společně se závažím připevnil k plotteru pomocí několika pevně utažených gumiček. Od závaží jsem očekával utlumení vibrací a připevnění gumičkou jsem zvolil, abych zamezil přímému přenosu vibrací (obrázek 9.4).

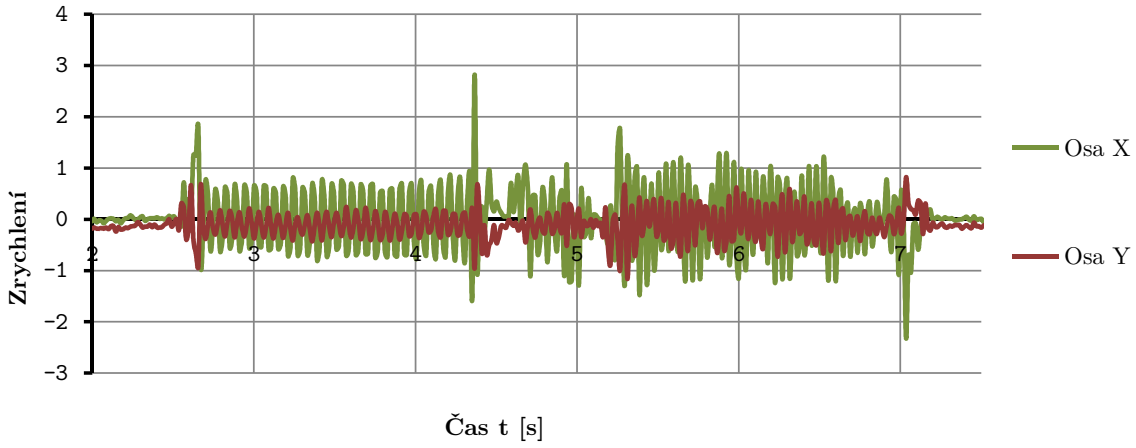
Výsledky mých měření jsou zobrazeny na grafech 9.5 a 9.6. Data byla vyfiltrována stejně jako v prvním případě. Má očekávání se naplnila. Vibrace byly znatelně utlumeny, jak lze vidět na vyfiltrovaném grafu. Jasně je rozpoznatelné počáteční zychlení na začátku pohybu ve směru osy x (v čase 2,5 s), dobře je vidět zrychlení i při pohybu ve směru osy y a také je vidět dobře konečné brzdění (v čase 7 s). Toto brzdění má zápornou amplitudu o téměř stejné velikosti jako rozjezd.

V naměřených datech je však jeden úsek, který si neumím vysvětlit – mezi 4. a 5. sekundou by mělo probíhat brzdění při pohybu ve směru osy x. Jenže zde se nenachází žádné zrychlení se zápornou amplitudou. Nevím, čím si tuto chybu v měření vysvětlit. Pravděpodobně se jedná o nějaké propružení použitého upevnění telefonu. Na grafu je opět vidět vibrace při pohybu na ose y, avšak jejich amplituda je mnohem menší.

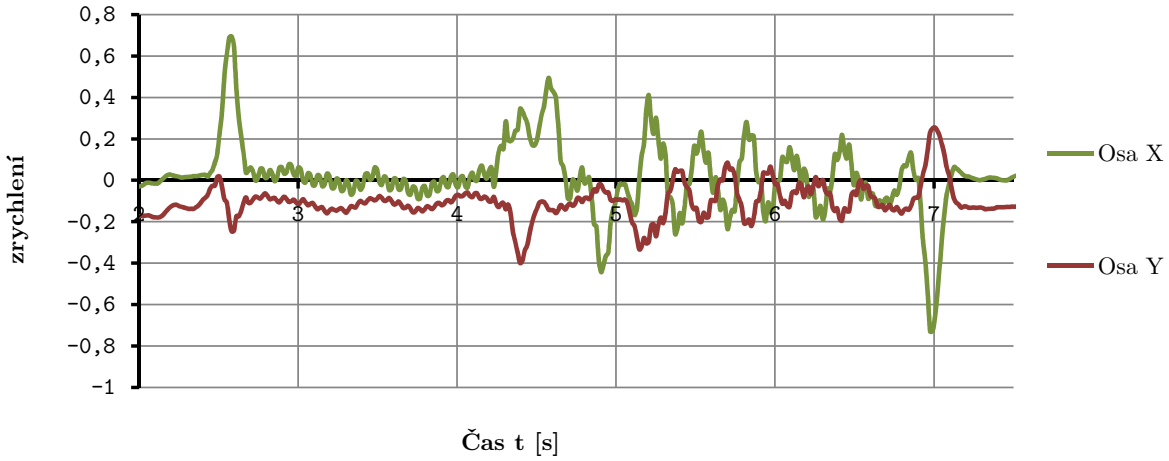
Z tohoto druhého měření je vidět, že systém funguje tak, jak má. I přesto by bylo zajímavé jej přeměřit pomocí lepších prostředků.



Obrázek 9.4: Nové uchycení telefonu pro měření zrychlení.



Obrázek 9.5: Naměřená data z akcelerometru.



Obrázek 9.6: Filtrovaná data z grafu 9.5

Kapitola 10

Současný stav projektu a jeho budoucnost

Můj řídicí systém ve své současné verzi umí načítat G-kód s podporou pro většinu běžně používaných G-kódů (jejich seznam je uveden v tabulce 8.1). Dále je schopen provádět kompenzaci nástroje. Systém zvládá určit a poté i následně interpolovat pohyb po akceleračních S-křivkách s omezením jak maximálního přípustného zrychlení, tak i maximálního ryvu. Tento výpočet zvládá jak pro lineární, tak i obloukový pohyb. Interpolační výkon jednotky je dostatečný na to, aby byla schopna interpolovat pohyb posuvy i kolem 10–15 $\mathrm{m{\cdot}min^{-1}}$.

Můj systém se však nyní nachází spíše v podobě technického dema než skutečného projektu/produktu. Ačkoliv zvládá ty nejpodstatnější úlohy, postrádá spoustu drobné funkcionality, která by mu umožnila být nasazen přímo v praxi. Z této drobné funkcionality můžu jmenovat podporu limitní snímačů, tabulku nástrojů, pokročilé ovládání programu (navazování po přerušení), podpora pro vřeteno a vlastně i samotné provedení interpolátoru. Můj systém ve své současné podobě slouží jako demonstrace možností řízení pohybu CNC stroje z pohledu dynamiky. Ukazuje funkčnost mnou odvozeného fyzikálního modelu, ukazuje jeho přednosti, ale není vhodý pro nasazení do praxe.

Dříve jsem byl optimističtější a plánoval jsem např. nahrazení počítače tabletem, což by umožnilo pohodlnější zabudování systému do stroje (aplikace pro tablet existuje, avšak je ve stádiu pokusů – zatím je schopna pouze zpracovávat jednoduchý G-kód bez podpory korekce nástroje). Avšak s postupem času se ukázalo, jak časově náročné by bylo doplnění veškeré nutné funkcionality. Rozhodl jsem se tedy projekt ponechat ve stádiu technického dema.

Z původních cílů tohoto projektu se mi povedlo splnit téměř všechny. Podařilo se mi navrhnout a odvodit vhodný fyzikální model pro pohyb CNC stroje. Také se mi povedlo realizovat interpolační jednotku, která obstarává úkony náročné na časování a počítač používá pouze pro uživatelský vstup a výkonově náročné operace. Dále se mi povedlo realizovat USB komunikaci mezi jednoutkou a počítačem. Jediný cíl, kterého jsem nedosáhl, je nasazení jednotky do praxe.

Ačkoliv jsem se rozhodl opustit můj projekt ve stádiu technického dema, zvažuji, zda-li by nestálo za to zaimplementovat můj fyzikální model do již existujícího řídícího systému. Tento systém by poskytl robustní základ pokrývající veškerou nutnou funkcionalitu. Jako vhodný se jeví řídicí systém LinuxCNC, který je opensource[6], ale chybí mu propacovanější model práce s dynamikou stroje. Nezkoumal jsem však možnosti této úpravy a už vůbec ne složitost tohoto úkonu. Tato myšlenka je pouze ve stádiu úvahy.

Literatura

[1] *Metoda bisekce a Newtonova metoda* [online]. [cit. 20. 2. 2013]. Dostupné z: <http://math.feld.cvut.cz/mt/txta/3/txc3aa3c.htm>.

[2] *www.C-N-C.cz - české diskusní fórum* [online]. 2013. [cit. 15. 2. 2013]. Dostupné z: <http://http://www.c-n-c.cz/>.

[3] STM32F4DISCOVERY - Product page. Technical report, STMicroelectronics, year. Dostupné z: <http://www.st.com/internet/evalboard/product/252419.jsp>.

[4] *Tool Compensation* [online]. 2 2013. [cit. 16. 2. 2013]. součást dokumentace systému EMC. Dostupné z: <http://www.linuxcnc.org/docs/html/gcode/tool_compensation.html#sec:cutter-compensation>.

[5] *Diskusní fórum Physics Forum, téma General equation of a circle in 3D?* [online]. 2006. [cit. 16. 2. 2013]. Příspěvek číslo 6. Dostupné z: <http://www.physicsforums.com/showpost.php?p=1007562&postcount=6>.

[6] *Linux CNC* [online]. 2013. [cit. 7. 2. 2013]. Dostupné z: <http://www.linuxcnc.org/>.

[7] *Začínáme s STM32F4Discovery 7* [online]. 2013. [cit. 1. 2. 2013]. Dostupné z: <http://mcu.cz/comment-n2848.html>.

[8] *Nonlinear Systems - Newton's Method* [online]. [cit. 20. 2. 2013]. Dostupné z: <http://www.math.ohiou.edu/courses/math3600/lecture13.pdf>.

[9] WinUSB (Windows drivers). Technical report, Microsoft, 1 2013. Dostupné z: <http://msdn.microsoft.com/en-us/library/windows/hardware/ff540196(v=vs.85).aspx>.

[10] ARTSOFT. *Mach 3 - oficiální stránky produktu* [online]. 2013. [cit. 7. 2. 2013]. Dostupné z: <http://www.machsupport.com/>.

[11] CNCCOOKBOOK, I. *Basic G-Code Program Structure* [online]. 2013. [cit. 12. 2. 2013]. Dostupné z: <http://www.cnccookbook.com/CCCNCGCodeBlocks.htm>.

[12] GRAVOS. *Řídící jednotka GVE84* [online]. 2013. [cit. 7. 2. 2013]. Dostupné z: <http://www.gravos.cz/gve84.htm>.

[13] *Příručka uživatele programování podle DIN/ISO*. Heidenhain, 12 2002. pro systémy TNC 410, TNC 426, TNC 430.

[14] MICROSOFT. *Kernel-Mode Code Signing Walkthrough* [online]. 2007. [cit. 25. 2. 2013]. Dostupné z: <http://msdn.microsoft.com/en-us/windows/hardware/gg487328>.

[15] POHONNATECHNIKA.CZ. *Krokový motor* [online]. 2013. [cit. 7. 2. 2013]. Dostupné z: <http://www.pohonnatechnika.cz/skola/motory/krokovy-motor>.

[16] POHONNATECHNIKA.CZ. *Servo* [online]. 2013. [cit. 7. 2. 2013]. Dostupné z: <http://www.pohonnatechnika.cz/skola/servo>.

[17] *Sinumerik 810 T – návod na programování.* Siemens, 9 1991.

[18] SIMMONS, B. *Multiplicative Inverse of a Matrix* [online]. 2012. [cit. 20. 2. 2013]. Dostupné z: <http://www.mathwords.com/i/inverse_of_a_matrix.htm>.

[19] *RM0090 – Reference manual.* STMicroelectronics, 9 2011. pro STM32F405xx, STM32F407xx, STM32F415xx a STM32F417xx.

[20] STUDICA, I. *G-codes* [online]. 2013. [cit. 12. 2. 2013]. Dostupné z: <http://www.cncezpro.com/gcodes.cfm>.

[21] *Toshiba TB6560 datasheet.* Toshiba, 7 2009. Dostupné z: <http://www.toshiba.com/taec/components2/Datasheet_Sync/382/27885.pdf>.

[22] WIKIPEDIA. *Endianness* [online]. 2013. [cit. 15. 2. 2013]. Dostupné z: <http://en.wikipedia.org/wiki/Endianness>.

[23] WIKIPEDIA. *G-code* [online]. 2013. [cit. 12. 2. 2013]. Dostupné z: <http://en.wikipedia.org/wiki/G-code>.

[24] WIKIPEDIA. *Globally unique identifier* [online]. 2013. [cit. 25. 2. 2013]. Dostupné z: <http://en.wikipedia.org/wiki/Globally_unique_identifier>.

[25] WIKIPEDIA. *Metoda tečen* [online]. 2013. [cit. 10. 2. 2013]. Dostupné z: <http://cs.wikipedia.org/wiki/Metoda_tečen>.

[26] WIKIPEDIA. *Jerk* [online]. 2013. [cit. 10. 2. 2013]. Dostupné z: <http://en.wikipedia.org/wiki/Jerk_(physics)>.

[27] WIKIPEDIA. *Ryv* [online]. 2013. [cit. 10. 2. 2013]. Dostupné z: <http://cs.wikipedia.org/wiki/Ryv>.

Část IV

Přílohy

Příloha A

Dotaz na řešení akcelerace u interpolačních jednotek Gravos

A.1 Můj dotaz

Dobrý den,

píši práci, která se tématicky týká řídicích systému pro hobby CNC stroje. Součástí práce je i stručný přehled dostupných řídicích systému. Vaše řídicí jednotky se mi líbí a rád bych je v práci zmínil. Nikde jsem však na Vašich stránkách nenašel některé informace. Mohl byste mi je prosím doplnit (pokud to samozřejmě budete pokládat za odhalení určitého know-how, odpovídat nemusíte).

- jaké máte stanoveny limity pro dynamiku pohybu stroje? Předpokládám, že omezujete maximální zrychlení. Omezujete také maximální ryv?
- jaký druh akceleračních ramp vaše jednotky používají? Lineární, přibližně navolené S-křivky, či S-křivky založené na daném matematicko-fyzikálním modelu?
- ačkoliv řídicí program podporuje kruhovou interpolaci, v dokumentaci k řídicí jednotce popisuje pouze vektory. Mám to chápat tak, že řídicí systému rozloží kruhový oblouk do spousty malých úseček, které řídicí jenotka následně projede? Jak v tomto případě je ošetřeno celkové zrychlení?

Předem děkuji za odpověď
Jan Mrázek

A.2 Vyjádření společnosti Gravos

Dobrý den,

pokusím se odpovědět na otázky ohledně našeho cnc řízení, pokud něco nebude jasné nebo budete mít další dotazy, ptejte se dál na helpdesk@gravos.cz.

Z hlediska samotné jednotky dynamika stroje se do jednotky zadává jako akcelerace výsledného pohybu, brzdná rychlost (rychlost na kterou se přibrzdí na konci vektoru a délka tzv krátkého vektoru, vektory kratší než nastavená délka krátkého vektoru jednotka akceleruje pouze na brzdnou rychlost.

Pak se nastavuje rychlost výsledného vektoru a limity rychlosti jednotlivých os. Změna rychlosti za pohybu se provádí procentuálně. Pokud by přišel jednotce vektor kde by jedna osa jela

rychleji než nastavený limit nebo přišel požadavek na procentuální změnu rychlosti, sníží se rychlost tak aby rychlostní limit nebyl překročen. Pro rychlosti se ještě nastavuje jestli má jednotka provádět rychlostní korekce (zadaná rychlost je rychlost výsledného pohybu) nebo jestli se rychlosti nechají na nadřízeném SW (zadaná rychlost je rychlost osy která má ujet nejvíce kroků). Druhá možnost se používá v sw Armote pro kontinuální 4 osé obrábění, kde je nutné rychlost upravovat podle směrů os a vzdálenosti od osy rotace, tak aby zadaná rychlost v g-kódu F odpovídala rychlosti špičky nástroje vůči materiálu.

Z hlediska sw Armote to jsou max rychlosti jednotlivých os, akcelerace, max změna rychlosti jednotlivých os, rychlost oblouku o poloměru 1 mm a mezní úhel do kterého se považuje další vektor ještě za navazující. Má-li stroj pak vykonat oblouk, dobrzdí se na začátek oblouku na rychlost pro R1·poloměr oblouku, pokud by rychlost byla vyšší než povolená změna rychlosti jednotlivých os, omezí se podle max změny rychlosti jednotlivých os.

Rampy se používají lineární $mm \cdot s^{-2}$, jednotka GVE94 (zatím není v nabídce) má implementovány S křivky ve formě dalšího typu vektoru pro který se akcelerace nastavuje zvlášť v $mm \cdot s^{-2}$ pro lineární část a v $mm \cdot s^{-3}$ pro začátek a konec akcelerace (omezení ryvu), tento typ vektoru má omezení, že během jeho chodu nelze měnit rychlost. Proto je v Armote použit jen pro rychloposuvy. Všechny jednotky přijímají vektory typu L pro nenavazující vektory, typu C pro navazující vektory (počáteční a koncová rychlost může být různá) a GVE94 má ještě typ vektoru LJ, jako typ L ale s akcelerační S křivkou. Jaký typ vektoru se jednotce pošle určuje sw Armote podle nastavených parametrů.

Oblouky jsou rozloženy na úsečky a odesílány do jednotky jako vektory typu C, jednotka má pro ně buffer (GVE64 může mít v bufferu 420 těchto vektorů) a každý může mít nastavenou jinou brzdnou rychlost. Armote tedy rozloží oblouk na úsečky a pokud by někde mělo dojít ke změně rychlosti (např se změnou poloměru) pošle jednotce k vektoru i jeho brzdnou rychlost. Jednotka pak vektory provádí postupně a když se uvolní místo v bufferu, Armote pošle další vektory. Tímto způsobem může stroj projet libovolnou prostorovou křivku plynule a je jedno jestli přijde v g kódu jako oblouk G2/G3 nebo už na úsečky rozložené z CAMu jako G1 což je zásadní pro rychlé 3D obrábění kde jsou prakticky veškeré dráhy z CAMu jako úsečky. Jediné omezení je v komunikační rychlosti (kromě GVE84 umí všechny jednotky až 921 000 Bd), tedy musí se stíhat plnit buffer, jednotka akceleruje maximálně na takovou rychlost, aby bylo možné na konci všech dosud přijatých vektorů v bufferu zabrzdit po nastavené rampě do nulové rychlosti. Rozklad na úsečky se provádí podle nastavené odchylky od sečny v Armote, defaultně nastaveno na 0,001 mm.

pozdravem
Jan Vostárek – GRAVOS

(Poznámka: Vyjádření bylo mírně typograficky upraveno)

Příloha B

Obsah přiloženého CD

Přiložené CD obsahuje následující adresářovou strukturu:

- `Ukázky`

  Tento adresář obsahuje videa ukazující funkci plotteru. Jednotlivá videa obsahují popis použitých nastavení atd.

- `Zdrojové soubory`

  Tento adresář obsahuje veškeré zdrojové soubory celého interpolátoru. Pro přehlednost jsou rozděleny do složek:

  - `Aplikace pro počítač`

    Adresář `src` obsahuje celý projekt aplikace pro počítač včetně všech nezbytných knihoven. Vytvořený projekt pochází z IDE Microsoft Visual C++ 2010. Uvnitř projektu jsou uloženy i zdrojové soubory a utility nutné pro vytvoření balíčku ovladačů WinUSB.

    Adresář `bin` obsahuje již sestavenou aplikaci.

  - `Firmware pro interpolátor`

    Adresář `src` obsahuje projekt firmware pro interlátor včetně všech nezbytných knihoven. Projekt pochází z IDE Atollic TRUEStudio Lite verze 2.2.0.

    Adresář `bin` obsahuje již sestavený firmware ve formátu `hex`.