FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Bakalářská práce

Graficko-uživatelské rozhraní pro ovládání zařízení pro polohování měřicí sondy ve dvou souřadných osách.

Studijní program: Mechatronika
Studijní obor: B0714A270001 Mechatronika
Autor práce: **Jan Hlaváč**
Vedoucí práce: Ing. Martin Černík, Ph.D.

Ústav mechatroniky a technické informatiky

Liberec 2023

FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Zadání bakalářské práce

Graficko-uživatelské rozhraní pro ovládání zařízení pro polohování měřicí sondy ve dvou souřadných osách.

Jméno a příjmení:	**Jan Hlaváč**
Osobní číslo:	M21000023
Studijní program:	Mechatronika
Studijní obor (specializace):	B0714A270001 Mechatronika
Zadávající katedra:	Ústav mechatroniky a technické informatiky
Akademický rok:	2023/2024

Zásady pro vypracování:

1. Seznamte se s původním zařízením a jeho původním programovým vybavením
2. Navrhněte a proveďte hardwarové úpravy pro zajištění větší spolehlivosti zařízení
3. Při využití dostupných knihoven na základě původního vybavení vytvořte GUI ve zvoleném programovacím jazyku
4. Změřte 2D pole vybrané fyzikální veličiny.

Rozsah grafických prací:	Dle potřeby dokumentace
Rozsah pracovní zprávy:	30–40 stran
Forma zpracování práce:	Tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] Pollak, Zbyněk.: Návrh a realizace dvouosého polohovacího zařízení pro měření zvukových polí. DP TUL 2008.

[2] Němeček, P.: Hluk v technické praxi I. Skriptum, FS TUL, 1998, ISBN 80-7083-285-1

[3] Rydlo P.: Řízení elektrických střídavých pohonů. Skriptum, FM TUL, 2007, ISBN 978-80-7372-223-4

Vedoucí práce:	Ing. Martin Černík, Ph.D. Ústav mechatroniky a technické informatiky
Datum zadání práce:	12. října 2023
Předpokládaný termín odevzdání:	14. května 2024

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Josef Černohorský, Ph.D.
vedoucí ústavu

V Liberci 12. října 2023

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval/a samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom/a toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

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

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom/a povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom/a následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

13. května 2024 Jan Hlaváč

Graficko-uživatelské rozhraní pro ovládání zařízení pro polohování měřicí sondy ve dvou souřadných osách.

Abstrakt

Tato bakalářská práce se zabývá vytvořením ovládacího programu pro polohovací zařízení určené k měření 2D polí. Zařízení je poháněno motory Maxon, které jsou řízeny ovládacími jednotkami EPOS 24/5. K programování ovládacích jednotek je použita knihovna od výrobce Maxon, která je volně dostupná na jeho stránkách. Program je psán jazyce C#. Součástí práce je také zvolení vhodného měřicího přístroje a jeho komunikace s počítačem a programovaní. V naší práci byl zvolen multimetr značky Metex. Nakonec následuje ověření funkčnosti programu změřením rychlosti proudění vzduchu.

Klíčová slova

EPOS, Polohování, 2D pole, Měření, C#

Graphical user interface for handling of an equipment for positioning of a measuring probe in two coordinate axes

Abstract

This bachelor thesis is about the creation of a control program for a positioning device designed to measure 2D fields. The device is powered by Maxon motors, which are controlled by EPOS 24/5 control units. For programing the control units, a library from the manufacturer Maxon is used, which is freely available on its website. The program is written in the C# language. The thesis also includes choosing a suitable measuring device and its communication with the computer and its programming. In our project, a Metex multimeter was chosen. For the final part of the thesis is verifying functionality of the program by measuring the speed of air flow.

Keywords

EPOS, Positioning, 2D fields, Measuring, C#

Poděkování

Rád bych poděkoval svému vedoucímu práce Ing. Martinu Černíkovi, Ph.D. za jeho užitečné rady a pomoc při tvoření této práce. Dále bych rád poděkoval doc. Ing. Martinu Pustkovi, Ph.D. za důvěru a podporu při tvoření programu.

Obsah

Seznam obrázků

Seznam grafů

Seznam zdrojových kódů

Seznam zkratek

VCS	Virtual Command Set
ID	Identifier
MM	Multimetr
COM	Communication Port
CAN	Controller Area Network
ASCII	American Standard Code for Information Interchange
VÚTS	Výzkumný Ústav Textilních Strojů
CSV	Comma Separated Values

Úvod

V roce 2008 vzniklo ve VÚTS zařízení pro polohování měřicí sondy ve dvou souřadných osách v rámci diplomové práce Pollak, Zbyněk.: Návrh a realizace dvouosého polohovacího zařízení pro měření zvukových polí. Cílem této bakalářské práce je vytvořit ovládací program v novějším a více podporovaném programovacím jazyce, a zvýšit spolehlivost zařízení.

Měření dvourozměrných polí je důležité pro vizualizaci měřené veličiny v prostoru. Tato měření jsou často velmi časově náročné a je nutné, aby bylo měření řízeno počítačem. K zajištění co nejlepšího měření musí být dbáno na kvalitu použitých prostředků pro polohování a spolehlivost programu.

V teoretické části si popíšeme teorii měření ve dvou osách, možné varianty a jejich klady a zápory. V druhé části je představeno programování ovládacích jednotek zařízení. Řídicí jednotky EPOS disponují spoustou funkcí a módů. Podrobně je zde popsáno jejich použití a příklady kódu v jazyce C#.

V praktické části bakalářské práce si nejprve přestavíme zařízení, jeho poruchovost a následné hardwarové vylepšení pro větší spolehlivost. Dále následuje zvolení měřicí techniky a její propojení s počítačem.

Poté se ve čtvrté části budeme zabývat programováním. Je zde popsáno vytváření programu, od zvolení vhodného jazyka až po konkrétní strukturu. Dále jsou představeny funkce, které zajišťují ovládání polohovacího zařízení. A nakonec jsou představeny funkce, které zařizují sběr a ukládání dat.

V páté části bakalářské práce je představen vzhled uživatelského rozhraní a popis jeho ovládání. Dále jsou zde popsány veškeré parametry měření, které lze přenastavit a jejich konkrétní účel.

Nakonec v poslední šesté části je demonstrována funkce programu změřením pole rychlosti vzduchu.

Teoretická část

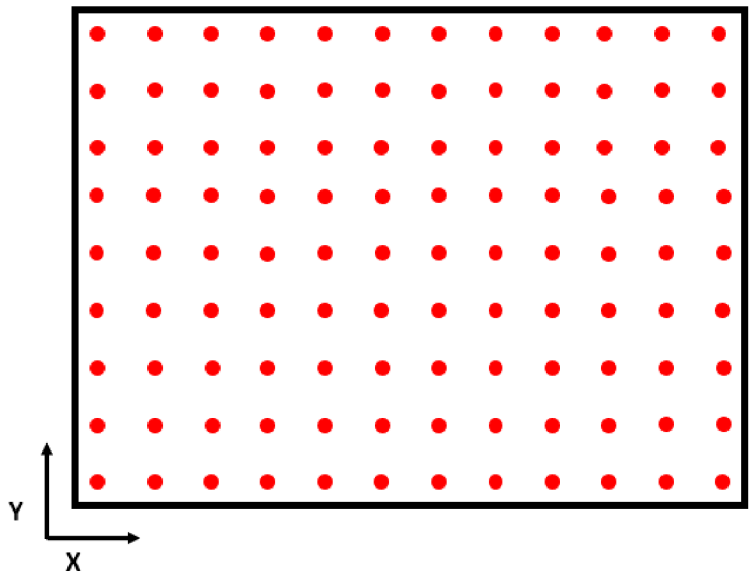
1 Měření

V této kapitole je představeno měření dvourozměrných polí. Jak takové měření vypadá, a jaké jsou možnosti. Dále je představena teorie měření, kterým se ověří funkčnost vytvářeného programu.

1.1 Měření ve dvou osách

Měření ve dvou osách je měření, při kterém se kromě naměřené hodnoty ukládá souřadnice, na které byla hodnota naměřena.

Měření ve dvou osách se využívá při měření jakýchkoliv 2D polí. Například hlukové pole, pole proudění vzduchu, a mnohé další. Je možné i měřit vzdálenost mezi měřicím rámem a libovolným tělesem, a sestavit tak jeho obrys. Princip měření spočívá v najíždění na předem zadaný rastr souřadnic a sejmutí náměru na každé pozici. Pohyb měřicí sondy je většinou realizován pomocí dvou nebo více motorů, které jsou uchyceny na nějakém rámu.



Obrázek 1.1 Příklad rastru měřících bodů

Tento způsob měření umožňuje vzít spoustu blízkých náměrů. Avšak je pomalý a nevhodný pro měnící se jevy. Jako alternativa je více nepohyblivých sond, které vezmou náměr najednou. Při velkém počtu měřících sond je tento způsob dražší, a je zde také menší počet měřicích bodů. Jeho výhodou je však rychlost, a je vhodný pro rychle se měnící jevy.

1.1.1 Doba měření

Doba měření má mnoho složek. První složkou je čas, po který se sonda pohybuje na měřicí bod (t_p). Tento čas ovlivňuje nastavená maximální rychlost a zrychlení zařízení. Po nájezdu na požadovanou souřadnici dochází k měření. Čas, po který je měřeno na jednom bodě je závislý na mnoha faktorech:

- Doba čekání na ustálení sondy (t_u)
- Počet jednotlivých náměrů v jednom bodě (n)
- Rychlost sejmutí náměru a komunikace měřicího přístroje s počítačem (t_k)

Po sečtení těchto časů je výsledek vynásoben počtem měřicích bodů (rastr o velikosti $a\ x\ b$). Celkový čas měření může být klidně až desítky hodin.

$$t = \left(t_p + t_u + n * t_k\right) * (a * b) \tag{1.1}$$

Rovnice (1.1) vyjadřuje teoretický výpočet doby měření.

1.2 Měření pole rychlosti proudění vzduchu

V naší práci pro ověření funkčnosti programu bylo zvoleno měření rychlosti proudění vzduchu z ventilátoru. Rychlost proudění je velmi proměnlivá a je doporučeno sejmout více hodnot a ty následně zprůměrovat. V našem měření jsou vhodné termické anemometry

1.2.1 Termické anemometry

Termoanemometry patří mezi nejjednodušší způsoby měření rychlosti proudění vzduchu. Měření spočívá v ohřívání odporového tělíska na konstantní teplotu, která je znatelně vyšší než teplota okolí. Odporové tělísko je následně ochlazováno proudícím vzduchem a k dosažení nastavené teploty je třeba vyššího napájecího výkonu. Snížení nebo zvýšení napájecího výkonu je pak přímo úměrné snížení nebo zvýšení rychlosti proudění vzduchu.

Druhý typ anemometrů vyhřívá odporové tělísko konstantním proudem a měří se jeho teplota, která je nepřímo úměrná rychlosti proudění.

2 Programování ovládacích jednotek

Pro programování ovládacích jednotek EPOS je nutné stáhnout příslušnou knihovnu. Všechny knihovny jsou volně dostupné ke stažení na stránkách výrobce Maxon.

2.1 Popis módů

Ovládací jednotky EPOS disponují nastavením do mnoha operačních módů. Pro použití určitých funkcí je třeba předem jednotky nastavit do požadovaného módu. Zde jsou popsány všechny operační módy a jejich použití.

2.1.1 Position mode

Jako první je popsán asi ten nejjednodušší, Position mode. Zařízení je ihned připraveno na polohování. Zadává se přímo požadovaná poloha, není tu žádný generátor trajektorie. Parametry pohybu, jako je rychlost a zrychlení, se berou ze základního nastavení jednotek.

2.1.2 Profile position mode

Podobný mód je Profile position mode, který umožňuje větší kontrolu nad vykonávaným pohybem. Zadává se rychlost, zrychlení a zpomalení. Přičemž jejich průběhy nemusí být jen lineární. Funkce také dovoluje nastavit maximální dovolenou odchylku od požadované polohy, kde se při dosažení pozice, která je v rámci maximální odchylky považuje za úspěšné. Je také možnost nastavit časové okno ve kterém musí být požadovaná poloha dosažena a maximální povolenou rychlost pohybu. Největší výhodou tohoto módu je však možnost nastavení trajektorie, po které se má zařízení pohybovat. Máme na výběr mezi lineárním pohybem a pohybem po sinusoidě. Všechny požadavky na pohyb jsou zpracovány generátorem trajektorie, který vypočítá trajektorii pohybu.

2.1.3 Velocity mode

Při nastavení zařízení do Velocity modu není třeba znát cílovou polohu zařízení, ale jen směr a rychlost, kterou se má pohybovat. Nastavuje se požadovaná rychlost a ovládací jednotky EPOS se ji snaží dodržet. Pokud by bylo potřeba nastavit více parametrů, musí se použít Profile velocity mode.

2.1.4 Profile velocity mode

Jako u Velocity módu není třeba znát cílovou polohu, ale pouze rychlost, s jakou se pohybujeme. Tento mód, jako u Profile position módu, umožňuje nastavit více parametrů pohybu. Nastavuje se požadovaná rychlost, která může mít buď lineární nebo sinusový průběh, zrychlení a zpomalení. Tyto parametry jsou poté zpracovány generátorem trajektorie.

2.1.5 Homing mode

Tento mód zajišťuje jednu z nejdůležitějších funkcí, a to je kalibrace domácí (nulové) polohy. Je spousta možností, jak domácí polohu určit, tím se však budeme zabývat později. Jako obvykle je možnost nastavit rychlost a zrychlení. Je také možnost posunout nalezenou nulovou polohu pomocí parametru offset. Jak bylo zmíněno dříve u Profile velocity módu, je zde možnost nastavení trajektorie pohybu po sinusoidě nebo po přímce.

2.1.6 MasterEncoder Mode

Tento mód se požívá při polohování s použitím externího enkodéru. Jednotka EPOS 24/5 je pro použití s externím enkodérem vybavena dvěma digitálními vstupy DigIN 2 a DigIN 3.

2.1.7 Step/Direction Mode

V tomto módu je požadovaná poloha zadávána externím signálem, který je přiveden na digitální vstupy DigIN 2 a DigIN 3. Ovládací Jednotka EPOS se chová podobně jako řídicí prostředek krokového motoru.

2.1.8 Current mode

Current mode (nebo také řízení proudu) nabízí jednoduché řízení motorů založené na sledování proudu. Základní parametry pohybu jako je maximální rychlost se berou ze základního nastavení. Při správném nastavení modelu motoru omezí EPOS výstupní proud. Jednotka EPOS pracuje s parametry omezení trvalého proudu, omezení výstupního proudu a tepelné časové konstanty vinutí. Jednotka předpokládá, že je motor poháněn při okolní teplotě [25 °C]. Když tato podmínka není splněna, je nutné snížit hodnoty parametrů. Konkrétně omezení trvalého proudu, omezení výstupního proudu a tepelné časové konstanty vinutí podle teploty nového prostředí. Řízení proudu se používá také pro ostatní provozní režimy, aby nedošlo k přehřátí vinutí.

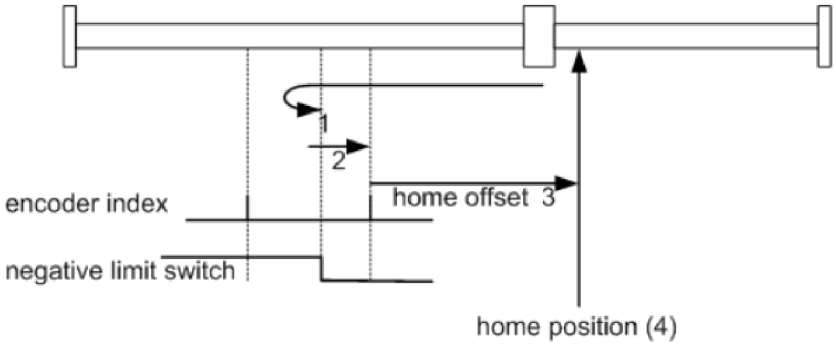
2.2 Metody hledání domácí polohy

Zde jsou popsány jednotlivé metody hledání domácí (nulové) polohy a jejich plusy a mínusy. Jednotlivé metody je možné rozdělit na tři skupiny. První skupina metody 1 až 27 potřebují ke své funkci přídavné snímače. Druhá skupina metody 33 až 35 jsou přímo závislé na předchozí poloze. A třetí skupina metody -1 až -4 hledají domácí polohu za pomocí mechanické překážky.

Metoda 1

Tato metoda začíná negativním pohybem (pohybem do záporných poloh), dokud nenarazí na koncový senzor, poté se rozjede opačným směrem, dokud jednotka nedostane pulz z enkodéru. Poté už se jen posune ve směru o požadovaný offset a výslednou polohy označí jako nulovou.

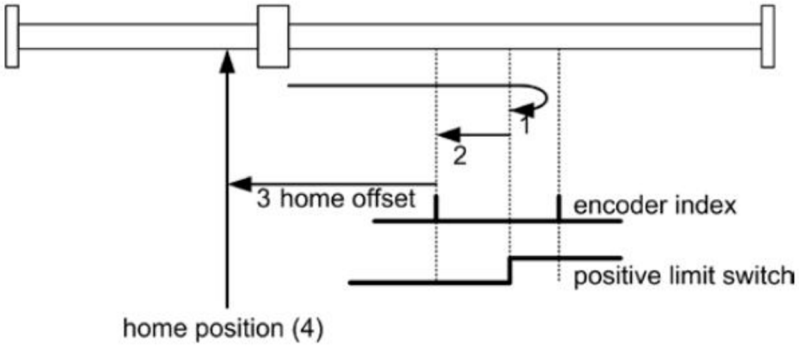
Tato metoda je přesná a jednoduchá. K její správné funkci je však potřeba koncový spínač.



Obrázek 2.1 Homing metoda 1 [8]

Metoda 2

Tato metoda je velmi podobná metodě č.1. Rozdíl leží v tom, že se začíná pozitivním pohybem (pohybem do kladných poloh).

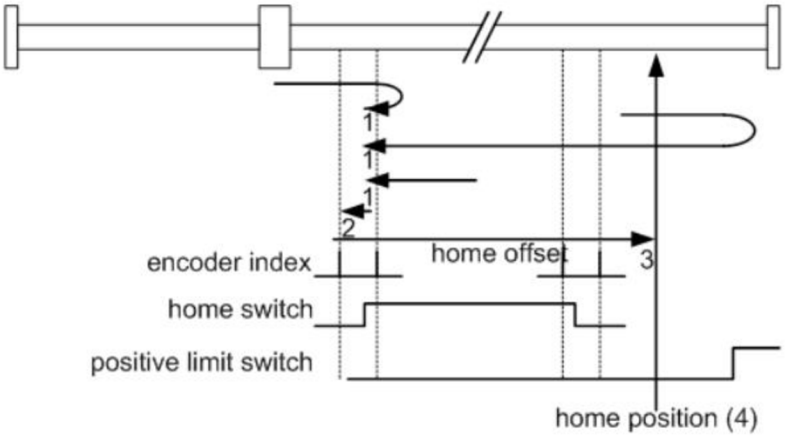


Obrázek 2.2 Homing metoda 2 [8]

Metoda 7

Tato metoda využívá ke své funkci tzv. home switch, který je aktivní pouze pro nějaký úsek. Začíná se pohybem do pozitivního směru s výjimkou, pokud je už home switch aktivní. Narazí se na pulz enkodéru, který se překrývá s aktivním home switchem a zahájí se opačný pohyb, dokud se nenarazí na pulz enkodéru, kde je home switch neaktivní. Poté už jen dojde k posunu o požadovaný offset a výsledná poloha se označí jako nulová.

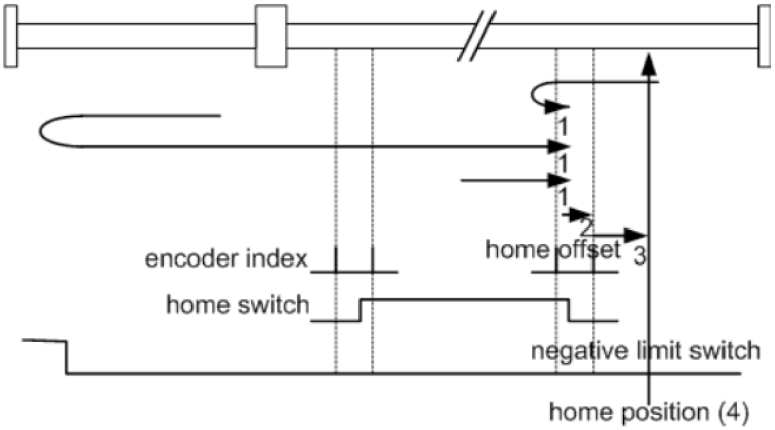
Metoda je vhodná, pokud máme za cíl mít domovskou polohu uprostřed pohybové dráhy. Opět je zapotřebí dalšího spínače.



Obrázek 2.3 Homing metoda 7 [8]

Metoda 11

Tato metoda je velmi podobná metodě č.7. Rozdíl leží v tom, že se začíná negativním pohybem (pohybem do záporných poloh).



Obrázek 2.4 Homing metoda 11 [8]

Metoda 17

Tato metoda je podobná metodě č.1 s rozdílem, že výsledná poloha není závislá na pulzu z enkodéru, ale pouze na koncovém senzoru. Kvůli tomu je menší přesnost při odkazovaní na ni.

Metoda 18

Stejně jako metoda č.17 je tato metoda je podobná metodě č.2 s rozdílem, že výsledná poloha není závislá na pulzu z enkodéru, ale pouze na koncovém senzoru. Kvůli tomu je menší přesnost při odkazovaní na ni.

Metoda 23

Metoda č.23 je podobná metodě č.7, ale jako předchozí dvě metody výsledná poloha není závislá na impulzu z enkodéru a přesnost domácí polohy je také horší.

Metoda 27

Stejně jako předchozí metoda je tato metoda velmi podobná metodě č.11 s rozdílem, že výsledná poloha není závislá na impulzu z enkodéru. Kvůli tomu je menší přesnost při odkazovaní na ni.

Metoda 33 a 34

Tato metoda je velmi jednoduchá, spočívá v pohybu k nejbližšímu indexovému pulzu enkodéru. Metoda č.33 hledá impulz negativním pohybem, zatímco metoda č.34 pozitivním pohybem.

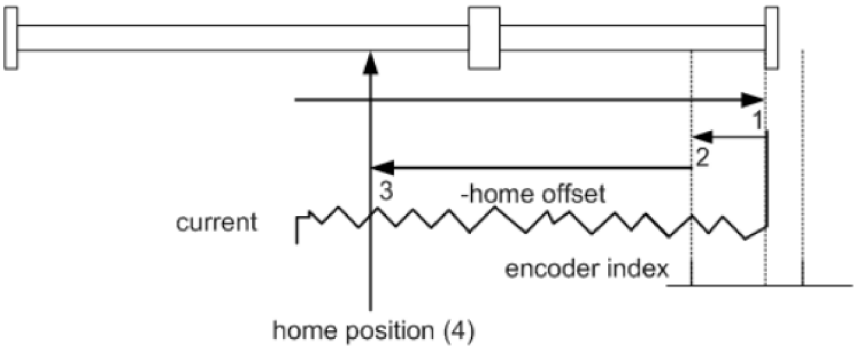
Tyto metody jsou velmi jednoduché a jejich výsledek je přímo závislý na předchozí poloze.

Metoda 35

Tato metoda neslouží přímo k nalezení domácí polohy ale k označení aktuální polohy jako nulové.

Metoda -1

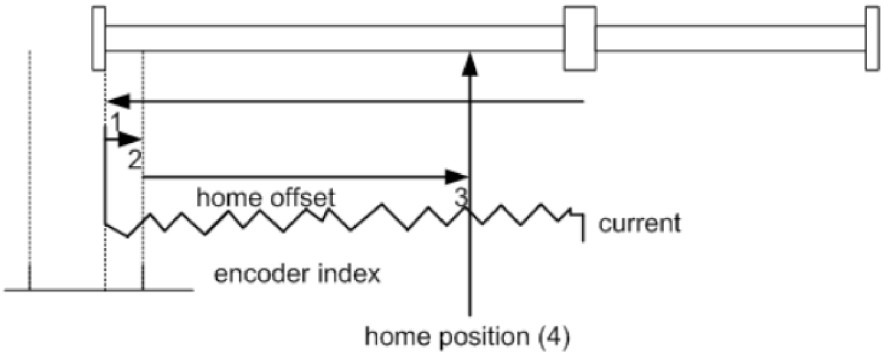
Na rozdíl od ostatních metod při použití není potřeba přídavných snímačů. Začíná se negativním pohybem (pohybem do záporných poloh). Cílem pohybu je narazit na překážku, která se stane referencí k nulovému bodu. S tím souvisí parametr, který se zadává při volání metody, a to je maximální proud. Tím je myšleno proud, který prochází vinutím motoru a tím také udává i maximální moment, který je motor schopen vyvinout. Pokud zařízení narazí na překážku, dojde k překročení maximálního proudu, motor se zastaví a zahájí pohyb na druhou stranu, dokud jednotka nezaznamená impulz z enkodéru. Poté se posune ve směru o případný offset a poloha je označena jako nulová poloha.



Obrázek 2.5 Homing metoda -1 [8]

Metoda -2

Tato metoda je velmi podobná metodě -1 s rozdílem, že počáteční pohyb k překážce je pozitivní (pohyb do kladných poloh)



Obrázek 2.6 Homing metoda -2 [8]

Metoda -3

Tato metoda je podobná metodě -1 s rozdílem, že po nárazu na překážku se nehledá nejbližší impulz z enkodéru, ale rovnou se posune o případný offset.

Metoda -4

Tato metoda je podobná metodě -2 s rozdílem, že po nárazu na překážku se nehledá nejbližší impulz z enkodéru, ale rovnou se posune o případný offset.

2.3 Komunikace

Komunikace mezi polohovacími jednotkami a programem je realizovaná přes sériový port RS232, který je přes redukci na USB připojen k počítači. Pro správnou funkci redukce musí být nainstalovány příslušné drivery. Ovládací jednotky EPOS 24/5 pro řízení motorů jsou zapojeny do sebe, přičemž jednotka s identifikačním číslem (ID) 1 funguje jako master a zajišťuje veš-kerou komunikaci s počítačem. Druhá jednotka s identifikačním číslem 3 je zapojená jako "slave". Chování jednotek je ale stejné a musí se adresovat obě zvlášť.

Nejprve je třeba inicializovat třídu *Device*, pod kterou se skrývají všechny potřebné funkce. To se provede pomocí funkce *Device.Init()*, tuto funkci je nutné volat už při načítání programu a nic nevrací. Všechny VCS funkce ale vrací nějakou hodnotu typu int. Tato hodnota říká, jestli byl příkaz úspěšný, pokud nebyl, tak funkce vrátí nulu.

Pro připojení zařízení je nutno zahájit komunikaci pomocí funkce *VcsOpenDevice*, která naváže komunikaci s hlavní řídicí jednotkou. Funkce musí znát jméno zařízení, což je v našem případě EPOS, způsob komunikace, v našem případě MAXON_RS232, jméno rozhraní, přes které se komunikuje, což je RS232, a port na kterém je zařízení připojeno. Také se zadává odkaz na proměnnou typu int (*ref int*), do které se zapíše případný error. Většina funkcí odkaz na tuto proměnnou také vyžaduje. Funkce vrací klíč ke komunikaci (dále jako keyHandle), který je potřeba při volání dalších funkcí, a zapisuje do již zmíněné proměnné případné errory.

```
using EposCmd.Net.VcsWrapper;

Device.Init();
keyHandle = Device.VcsOpenDevice("EPOS", "MAXON_RS232", "RS232", portName, ref
err);
Device.VcsSetProtocolStackSettings(keyHandle, 38400, 100, ref err);
```

Zdrojový kód 2.1 Příklad připojení k jednotce EPOS

Po připojení je třeba nastavit parametry komunikace jako je baudrate a timeout, Opět zadáváme keyHandle a odkaz na proměnnou error. Tyto příkazy jsou zpracovávány pouze jednou jednotkou EPOS, která má funkci master.

2.4 Popis základních funkcí

DLL od firmy Maxon disponuje mnoha předdefinovanými funkcemi, které si nyní popíšeme. Budeme se soustředit především na ty, které jsou nutné pro vytvoření našeho programu. Knihovna pro programovací jazyk C# je označena jako EposCmd.Net. Veškeré funkce použité v programu pro ovládání jednotek jsou ze skupiny VCS (Virtual Command Set). Tyto funkce se schovávají pod třídou *Device* (*EposCmd.Net.VcsWrapper.Device*).

2.4.1 Homing a nastavení operačního módu

Po každém spuštění a navázání komunikace je potřeba, aby zařízení našlo domácí polohu. K tomu slouží funkce *VcsFindHome*. Před zavoláním funkce je však potřeba připravit zařízení do Homing módu pomocí funkce *VcsSetOperationMode*. Tato funkce umožňuje nastavit zařízení do jakéhokoliv módu. Zadáváme keyHandle, ID karty, odkaz na požadovaný mód z enumerace *EOperationMode* (součást dll), který chceme nastavit. A samozřejmě odkaz na proměnnou, do které se zapíše error. Každá funkce vrací integer, kde nula znamená neuspěl a kladná hodnota znamená uspěl. Funkce se volají pro každou ovládací kartu zvlášť.

Místo funkce *VcsSetOperationMode* je možné použít přímo funkci *VcsActivateHomingMode*, zde se zadává pouze parametr komunikace keyHandle a ID ovládací karty, pro kterou se funkce volá, a také odkaz na proměnnou uchovávající error. Funkce také vrací integer s klasickou logikou, kde nula znamená neuspěl. Volá se pro každou kartu zvlášť.

Pokud není známo, ve kterém stavu zařízení nachází, může se použít funkce *VcsGetOperationMode*. Jako vždy je třeba zadat keyHandle a ID karty, která je dotazována, a odkaz na proměnnou, do které se následně informace uloží. Proměnná musí být typu *EOperationMode*, což jak bylo dříve řečeno je vyčíslení, které je součástí dll.

Jako další je potřeba nastavit parametry Homing módu pomocí funkce *VcsSetHomingParameter*. Opět se zadává keyHandle a ID. Jako další do se funkce zadává požadované zrychlení motoru, rychlost při najíždění na switch, rychlost při najíždění na index signál. Funkce také umožňuje nastavit offset home pozice od naleznutého bodu switche nebo signálu. A jako poslední se zadává maximální zatížení při hledání home pozice najížděním na překážku. A samozřejmě se taky zadává odkaz na proměnnou error.

Správnost parametrů je možné ověřit funkcí *VcsGetHomingParameter* do které se zadává keyHandle a ID zařízení, na které je cílena, a odkaz na proměnné do kterých se parametry uloží.

```
using EposCmd.Net.VcsWrapper;

Device.VcsActivateHomingMode(keyHandle, nodeId, ref err);
Device.VcsSetHomingParameter(keyHandle, nodeId, 100, 300, 0, 100, 1500, 0, ref
err);
Device.VcsFindHome(keyHandle, nodeId,
EposCmd.Net.EHomingMethod.HmCurrentThresholdNegativeSpeed, ref err);
Device.VcsWaitForHomingAttained(keyHandle, nodeId, 100_000, ref err);
```

Zdrojový kód 2.2 Příklad hledání domácí polohy

V příkladě se nejprve nastaví ovládací jednotka (řádky 3 a 4). Poté se zařízení uvede do pohybu funkcí *VcsFindHome*. A nakonec se počká na nalezení domácí polohy pomocí funkce *VcsWaitForHomingAttained*, kde třetí parametr funkce je timeout. Pokud bude tento čas překročen, nalezení domácí polohy bylo neúspěšné a funkce vrátí nulu.

2.4.2 Polohování

Zde se podíváme na funkce použité v programu, které se týkají polohování, od nastavení polohovacích jednotek přes najetí do požadované polohy, a nakonec zjištění, ve které poloze se nacházíme.

Nejprve je třeba nastavit obě jednotky EPOS do správného módu. Je na výběr mezi Position módem, Profile position módem a Interpolated position módem. Pro náš program je důležitý Profile position mód, který nabízí možnost nastavení více parametrů pohybu.

Nastavení jednotek do požadovaného módu se provádí, jak už bylo dříve přiblíženo pomocí funkce *VcsSetOperationMode* nebo se může použít přímo příkaz *VcsActivateProfilePositionMode* pro nastavení do Profile position módu, *VcsActivateInterpolatedPositionMode* pro nastavení do Interpolated position módu, a *VcsActivatePositionMode* pro nastavení do Position módu. Každá funkce se volá pro každou kartu zvlášť a zadávají se obvyklé parametry keyHandle, ID karty a odkaz na proměnnou uchovávající error. Nastavení parametrů Profile position módu se provádí pomocí funkce *VcsSetPositionProfile*. Jako vždy se funkce volá pro každou kartu zvlášť a zadávají se obvyklé parametry jako je keyHandle, ID karty a odkaz na proměnnou uchovávající error. Nastavuje se rychlost, zrychlení a zpomalení.

Najíždění do polohy se provádí u každého módu jinak. U Profile position módu se volá funkce *VcsMoveToPosition*. Zadávají se obvyklé parametry keyHandle, ID karty. Lze nastavit, jestli má pohyb začít ihned, nebo jestli má jednotka nejdříve počkat na dokonání posledního pohybu. Dále se nastavuje, jestli má být pohyb relativní nebo absolutní, a nakonec požadovaná poloha. Samozřejmě příkaz vykoná jen jedna karta s identifikačním číslem (ID), které se zadává do parametrů funkce.

```
using EposCmd.Net.VcsWrapper;

Device.VcsActivateProfilePositionMode(keyHandle, nodeId, ref err);
Device.VcsSetPositionProfile(keyHandle, nodeId, rychlost, zrychleni, zpomaleni,
ref err);
Device.VcsMoveToPosition(keyHandle, nodeId, pozice, 1, 0, ref err);
```

Zdrojový kód 2.3 Ukázka najetí do polohy (Profile position mode)

Pro najetí do požadované polohy v obyčejném Position módu se použije funkce *VcsSetPositionMust*. Jako vždy se zadává keyHandle, ID karty. Nastavuje se pouze hodnota požadované polohy. V případě Profile position módu příkaz nejprve prochází profilovým generátorem, v případě Position módu profil generuje CANopen Master.

```
using EposCmd.Net.VcsWrapper;

Device.VcsActivatePositionMode(keyHandle, nodeId, ref err);
Device.VcsSetPositionMust(keyHandle, nodeId, pozice, ref err);
```

Zdrojový kód 2.4 Ukázka najetí do polohy (Position mode)

Poloha se zjišťuje funkcí ze skupiny *VcsGet*. Konktrétně se jedná o funkci *VcsGetPositionIs*. Jako vždy se zadává ID karty, od které je požadována poloha, keyHandle, odkaz na proměnnou uchovávající error a odkaz na proměnnou do které se uloží získaná informace o poloze.

```
using EposCmd.Net.VcsWrapper;

Device.VcsGetPositionIs(keyHandle, nodeId, ref poloha, ref err);
```

Zdrojový kód 2.5 Ukázka získání polohy

2.4.3 Manuální pohyb

Zde si popíšeme jednodušší pohyby, kde není třeba znát polohu. Konktrétně se jedná o rychlostní řízení. Rychlostní řízení se hodí například při manuálním pojíždění.

Pro manuální pohyb je nejvíce vhodný Profile velocity mode. Nejprve je třeba nastavit ovládací jednotku do pomocí funkce *VcsSetOperationMode* do správného módu. Jako další se musí nastavit parametry pohybu funkcí *VcsSetVelocityProfile*. Do funkce se zadává jako vždy keyHandle, ID jednotky, pro kterou funkci voláme, odkaz na proměnnou s errorem, a nastavujeme zrychlení a zpomalení. Po úspěšném nastavení se může ovládat rychlost motorů pomocí funkce *VcsMoveWithVelocity*. Zadává se do ní keyHandle, referenci na error, a samozřejmě se nastavuje požadovaná rychlost.

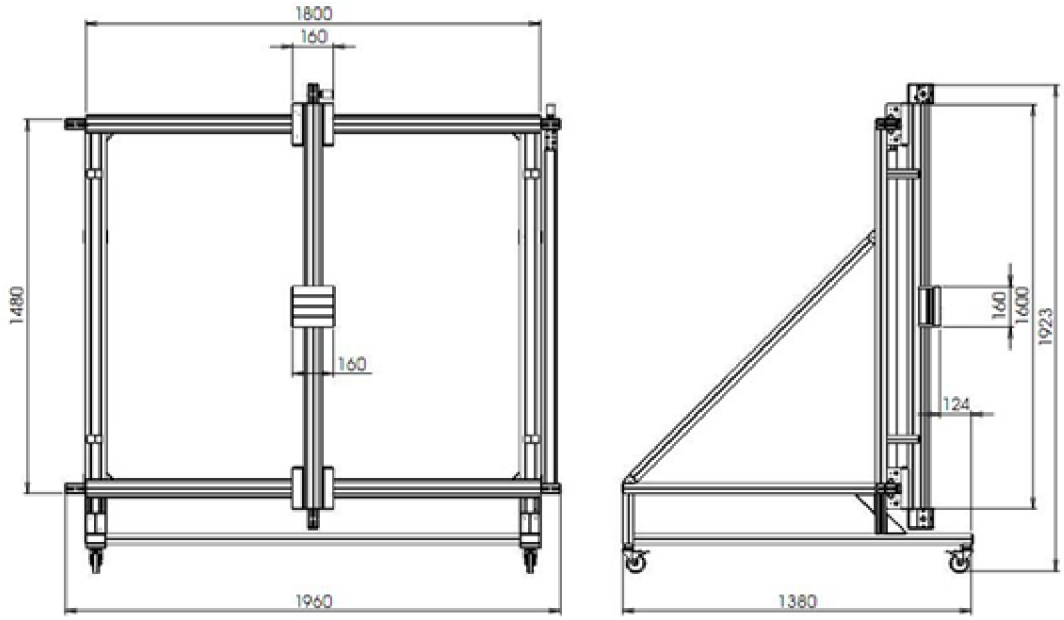
```
using EposCmd.Net.VcsWrapper;

Device.VcsSetOperationMode(keyHandle, nodeId,
EposCmd.Net.EOperationMode.OmdProfileVelocityMode, ref err);
Device.VcsSetVelocityProfile(keyHandle, nodeId, zrychleni, zpomaleni, ref err);
Device.VcsMoveWithVelocity(keyHandle, nodeId, rychlost, ref err);
```

Zdrojový kód 2.6 Příklad rychlostního řízení

Praktická část

3 Popis zařízení

Zařízení vzniklo ve VÚTS konkrétně pro měření hlukových polí. Měřicí zařízení je umístěno na pojízdném vozíku, který je uchycen na dvouosém lineárním vedení. Jeho vertikální pohyb zařizuje řemenice s ozubeným řemenem, která je k rámu přichycena dvojicí kladek. Horizontální pohyb tvoří soustava dvou řemenic, které jsou propojeny ocelovou kalenou broušenou tyčí, aby nedocházelo k případnému kroucení a nepřesnostem. Dvojice řemenic potom hýbe celou soustavou pro vertikální pohyb. Rám má rozměry 1800 x 1480 mm a je smontován z hliníkových profilů. Rám je postaven na podstavě, která má hloubku 1380 mm pro větší stabilitu. K podstavě jsou připevněná kola pro lepší manipulaci. Samotný pojízdný vozík je snadno rozšiřitelný.



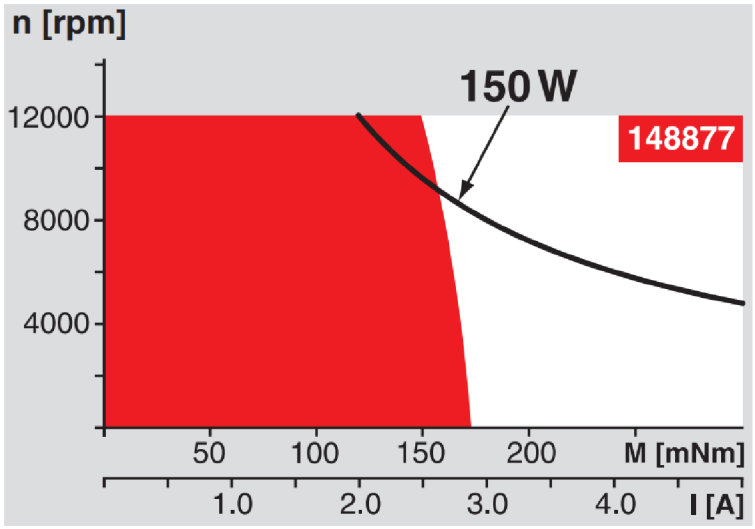
Obrázek 3.1 Měřicí rám [11]

3.1 Pohonné jednotky

Veškerý použitý hardware je od firmy Maxon. Pojízdný vozík je poháněn dvěma motory. Jeden pro vertikální pohyb a druhý pro horizontální pohyb. Ovládání motorů zařizují dvě ovládací jednotky EPOS 24/5, pro každý motor jedna.

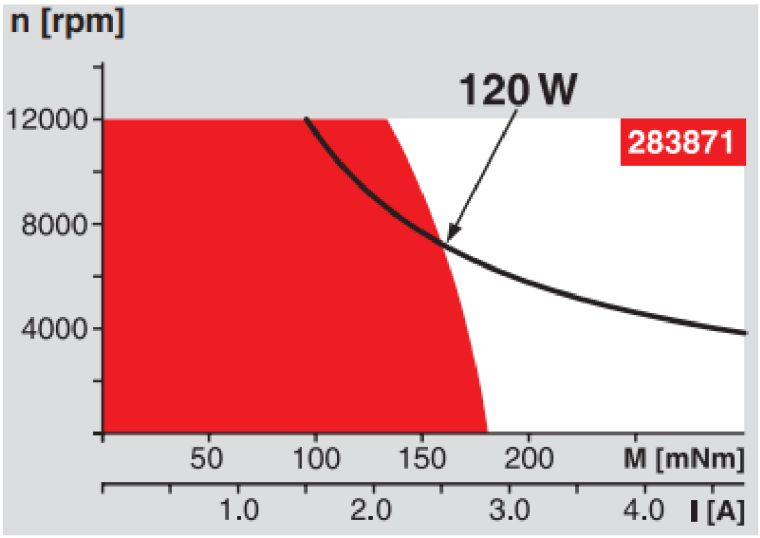
Motory

Pro vertikální pohyb je použit motor Maxon RE 40. Je to stejnosměrný motor s grafitovými kartáči. Maximální rychlost otáček je 12000 rpm a maximální výkon 150 W.



Graf 3.1 Momentová charakteristika motoru Maxon RE 40 [5]

Pro horizontální pohyb je použit motor Maxon EC-max 40, který je elektricky komutovaný a má výkon 120 W. Převodovka a enkodér se shodují se soustavou pro vertikální pohyb.



Graf 3.2 Momentová charakteristika motoru Maxon EC-max 40 [4]

Převodovky

Na každý motor je připevněná planetová převodovka také od firmy Maxon, která zařizuje převod do pomala v poměru 21:1. Konkrétně se jedná o převodovku GP 42 C. Maximální přípustná vstupní rychlost je pouze 8000 rpm.

Enkodéry

Poloha každého motoru se zjišťuje pomocí enkodéru MR, Type L také od firmy Maxon. Každý enkodér má 2000 pulzů na otáčku.

3.1.1 Počet pulzů na mm

V praxi nás nejvíce zajímá, jaký je počet pulzů enkodéru na milimetr (E).

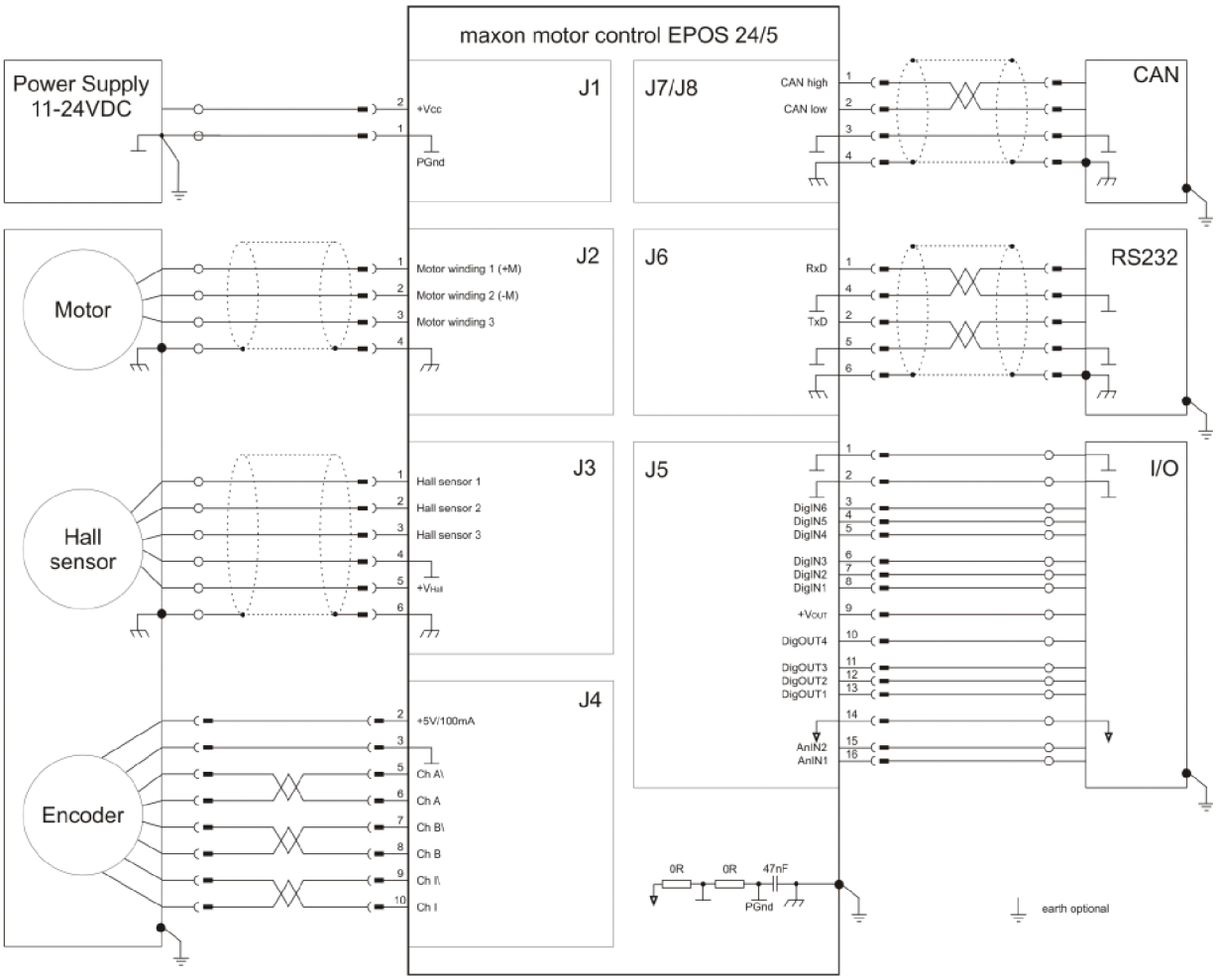
- 150 mm na otáčku převodovky (l)
- 2000 pulzů enkodéru na otáčku motoru (e)
- 21:1 převod (p)

$$E = \frac{(e * p)}{l} \tag{3.1}$$

Rovnice (3.1) vyjadřuje počet pulzů na milimetr. Po dosazení do vzorce nám vyjde, že počet pulzů na milimetr je 280 puls/mm.

3.2 Řízení

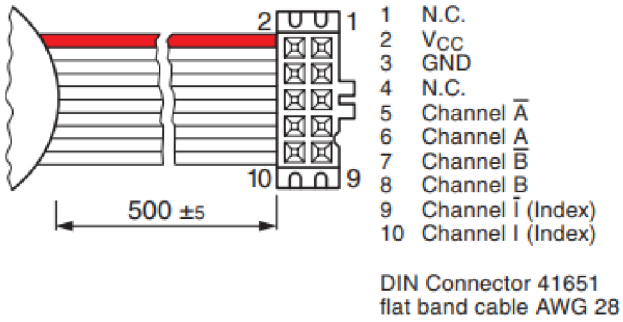
Řízení je prováděno pomocí řídicích jednotek EPOS 24/5 (pro každý motor jedna). EPOS 24/5 je plně digitální jednotka řízení polohy. Může být použita s kartáčovými DC motory s inkrementálním snímačem stejně jako s bezkartáčovými EC motory s Hallovými sondami a inkrementálním snímačem. Komutace EC motorů je prováděna sinusovým průběhem, což nabízí řízení motorů s minimálním zvlněním momentu a nízkou hlučností. Jednotka EPOS disponuje integrovaným řízením polohy, rychlosti a proudu, a tím umožňuje složitější aplikace při polohování. Jednotka je navržena tak, aby byla ovládána a řízena jako slave uzel v síti CANopen. Navíc může být jednotka ovládána i přes port RS232.



Obrázek 3.2 Schéma zapojení ovládací jednotky EPOS 24/5 [8]

3.3 Úpravy zařízení

Zařízení mělo tendenci se při pohybu v ose X zasekávat a často docházelo k přerušení pohybu při prudším zrychlení motorů. Všechny spoje byly proměřeny a byl zjištěn špatný kontakt na konektoru enkodéru. Špatný konektor byl přepájen na klasický sériový konektor a zařízení od té doby funguje bez problémů.



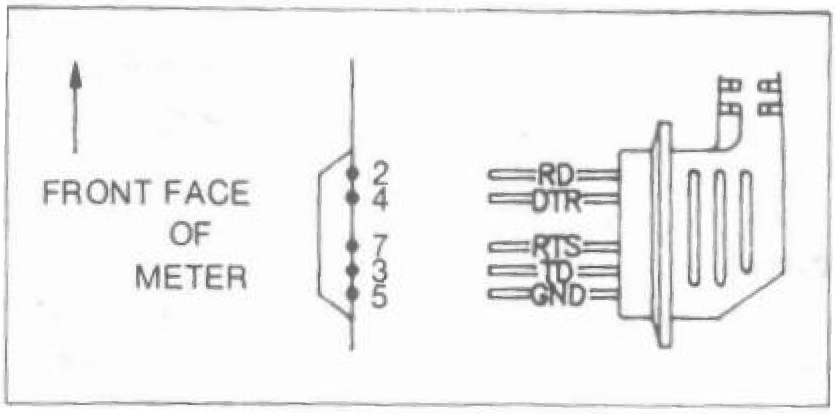
Obrázek 3.3 Schéma konektoru enkodéru [7]

3.4 Měření

Při návrhu způsobu měření byl kladen důraz na univerzálnost. Každá veličina lze převést na napětí nebo proud. S touto myšlenkou jsme se řídili při zvolení způsobu měření. Další požadavek byl na jednoduchou komunikaci s počítačem. Z těchto důvodů byl k měření zvolen multimetr Metex M-3850D.

Obrázek 3.4 Multimetr Metex

Tento multimetr disponuje automatickým nastavováním rozsahu. Komunikace s počítačem probíhá pomocí sériového portu RS232. Pro získání náměru je z počítače poslán znak “D“, Po přijetí multimetr společně s dalším obnovením hodnoty pošle informaci s naměřenou hodnotou. Velikost poslané informace je 14 bajtů, kde každý bajt je písmeno kódované 7-bit ASCII kódem. Parametry komunikace jsou 1200 baud, žádná parita a dva stop bity.



Obrázek 3.5 Schéma konektoru multimetru [9]

4 Vytváření programu

Tato kapitola popisuje vytváření našeho programu. Požadavky na program a jeho ovladatelnost. Je zde popsána jeho struktura a obsažené funkce.

4.1 Požadavky na program

V této části je popsán myšlenkový postup při volbě vhodného programovacího jazyka a základní požadavky na ovládací interface.

4.1.1 Zvolení programovacího jazyku

Při volbě programovacího jazyka bylo několik možností, které však byly limitovány dostupností knihoven pro ovládací jednotky EPOS. K programu takového obsahu by byl vhodný python, avšak pro tento jazyk nebyla vytvořena knihovna. Kvůli stáří ovládacích jednotek nejsou vytvářeny nové knihovny pro nové a více oblíbené programovací jazyky. Na výběr bylo tedy mezi Borland C++ a Delphi, a Visual Basic, C# nebo C++. Vzhledem k požadavkům na funkčnost programu a mým předchozím zkušenostem s jazykem C# byl zvolen tento jazyk. Program byl vytvářen ve Visual Studiu s použitím šablony Windows Forms.

4.1.2 Požadavky na ovladatelnost programu

Základní požadavky pro vytváření programu byla jednoduchá ovladatelnost a možnost přenastavení parametrů měření.

Manuální pohyb

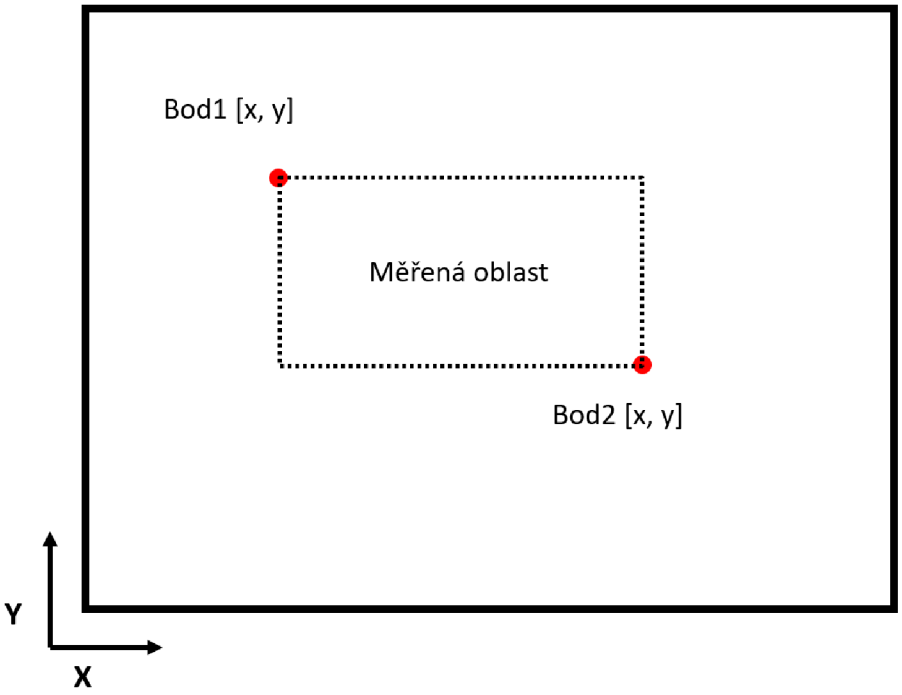
Jako základní pohyb zařízení je manuální pohyb pomocí šipek. Pro volnější ovladatelnost byla vytvořena funkce, která uvolní motory a s pojízdným vozíkem lze hýbat rukou.

Parametry měření

- Vzdálenost měřicích bodů
- Rychlost a zrychlení motorů
- Počet náměrů na jednom bodě (průměr)
- Čekání na pozici před prvním náměrem

Libovolné nastavení velikosti a pozice měřicí oblasti

Měřicí oblast má tvar obdélníka a jeho velikost a poloha se nastavuje pomocí zadání dvou bodů, které jsou jeho protější rohy.



Obrázek 4.1 Příklad měřené oblasti

4.2 Struktura programu

V této části je popsaná struktura programu, vytváření jednotlivých tříd a jejich funkce.

4.2.1 Knihovny

Pro náš program bylo použito hned několik knihoven. Pro ovládání řídících jednotek EPOS byla přidána knihovna *EposCmd.Net*. Další přidaná knihovna je *CsvHelper* pro zápis do CSV souborů.

4.2.2 Třídy

V programu se kromě hlavní třídy *Program* nachází celkem čtyři třídy, zde si popíšeme obsah a funkci každé z nich.

Parametry

Začněme malou třídou v názvem *Parametry*. Jak už jméno napovídá, v této třídě se nachází proměnné uchovávající parametry pohybu. Všechny uvedené parametry je možné přenastavit ve zvláštním formu. Kromě konstant *Xmax* a *Ymax*, tyto konstanty představují maximální polohy v jednotlivých osách. Třída také obsahuje funkci na dopočítání startovací a konečné pozice v závislosti na zadaném posunu.

```
    public class Parametry
    {
        public uint rychlost = 300;
        public uint zrychleni = 300;
        public char osa = 'X';  //udává v jaké ose se bude jezdit
        public int posun = 500; //vzdálenost mezi měřicími body
        public int Xstart = 0;
        public int Ystart = 0;
        public int Xkonec = 412000;
        public int Ykonec = 350000;
        public int wait = 0;    //čekání na pozici před naměřením hodnoty
        public int pocet = 1;   //počet náměrů na pozici

        const int Xmax = 412000;
        const int Ymax = 350000;

        public void vypocet_posunu(double konecX, double konecY, double startX,
double startY, double posun){...}
      }
```

Zdrojový kód 4.1 Třída Parametry

Data

Tato třída obsahuje pouze naměřenou hodnotu a souřadnice. Z této třídy se později v programu dělá list, do kterého se ukládají veškerá naměřená data. Kromě toho soubor také obsahuje další třídu *DataMap*, která slouží jako mapa při zapisování dat do CSV souboru.

```
    public class Data
    {
        public int X;
        public int Y;
        public float namer;
    }

    public class DataMap : ClassMap<Data>
    {
        public DataMap(){...}
    }
```

Zdrojový kód 4.2 Třída Data

Multimetr

Ve třídě *Multimetr* se nachází veškeré funkce zařizující komunikaci s multimetrem. Vytvoření speciální třídy pro ovládání multimetru zlepšuje čitelnost programu. Umožňuje také snadnější úpravu programu pro případ nutnosti předělání pro použití jiné měřící techniky. Komunikace s MM probíhá pomocí sériového portu. Proto je zde také inicializovaná privátní třída *SerialPort*.

```
public class Multimetr
{
      private SerialPort port = new SerialPort();

       public int openPort(string name){...}

       public string Zmer(int format){...}

       public double ZmerDouble(){...}
}
```

Zdrojový kód 4.3 Třída Multimetr

HlavniOvladani

Nyní se konečně podíváme na nejdůležitější třídu s názvem *HlavniOvladani*. V této třídě jsou veškeré funkce pro ovládání řídících jednotek EPOS. Nachází se zde i funkce pro zápis dat do souboru. Používají se zde instance všech již zmíněných tříd. Ve třídě je také vytvořen list ze třídy *Data* pro uchování všech naměřených dat.

```
public class HlavniOvladani
{
public Parametry parametry = new Parametry();
      public Multimetr MM = new Multimetr();
      public List<Data> databaze = new List<Data>();  //sem se zapisují naměřená
data

      public bool stop = false;           //stop proměnná, používá se při
zastavení měření
      public bool motory_enabled = true;  //proměnná, která nám říká jestli jsou
aktivovány motory

      int keyHandle = 0;  //klíč komunikace
      public uint err;    //proměná uchovávající errory od jednotek EPOS

      const ushort nodeIdX = 1;   //ID ovládacích karet
      const ushort nodeIdY = 3;
      const int remen = 150;                //konstanta remenice (pocet mm na
otacku)
      const int prevod = 21;                //konstanta prevodovky(prevodovy
pomer 21:1)
      const int encoder = 2000;             //konstanta encoderu(pocet kroku na
otacku)


      /// ////////////////////////////////////////////////////////////////////
      /// NASTAVENÍ A SPUSTENI MERENI
      public async Task<int> SpustitMereniAsync(){...}

      /// ////////////////////////////////////////////////////////////////////
      /// FUNKCE ZAJISTUJICI POHYB MERICI SONDY
      private int Mereni(ushort nodeId1, ushort nodeId2, int start1, int start2,
int posun1, int posun2, int pKroku1, int pKroku2){...}
```

Zdrojový kód 4.4 Třída HlavniOvladani část 1

```
        /// ////////////////////////////////////////////////////////////////////
        /// SEJMUTI NAMERU Z MULTIMETRU
        private string VezmiNamer(){...}

        /// ////////////////////////////////////////////////////////////////////
        /// SPUSTENI HLEDANI HOME
        public async Task<int> FindHomeAsync(){...}

        /// ////////////////////////////////////////////////////////////////////
        /// HLEDANI HOME POZICE
        private int Homing(ushort nodeId){...}

        /// ////////////////////////////////////////////////////////////////////
        /// ZJISTENI AKTUALNI POLOHY
        public int[] ZiskatPolohu(){...}

        /// ////////////////////////////////////////////////////////////////////
        /// PRIPOJENI K HLAVNI RIDICI JEDNOTCE
        public int Connect(string portName){...}

        /// ////////////////////////////////////////////////////////////////////
        /// INICIALIZACE TRIDY DEVICE-nutno provadet jako prvni pri spusteni
programu
          public void Inicializovat(){...}
        /// ////////////////////////////////////////////////////////////////////
        /// MANUALNI POHYB DO STRAN
        public void ManualniPohyb(int smer){...}

        /// ////////////////////////////////////////////////////////////////////
        /// UVOLNENI MOTORU
        public void UvolnitMotory(){...}

        /// ////////////////////////////////////////////////////////////////////
        /// ULOZENI DAT DO CSV
        public void Zapis(string path){...}

        /// ////////////////////////////////////////////////////////////////////
        /// ZAJETI DO SPODNI POLOHY
        public void BezpecneUkoncit(){...}
}
```

Zdrojový kód 4.5 Třída HlavniOvladani část 2

4.2.3 Formy

Program obsahuje dva formy. Hlavní form má název *Form1* a slouží k veškerému ovládání řídících jednotek EPOS a měřícího multimetru. Druhý form má název *FormParametry* a slouží pouze pro úpravu parametrů měření jako je startovací pozice, rychlost, vzdálenost měřících bodů atd. Více o jednotlivých formech co se týče jejich ovládání a funkce je popsáno v samostatné kapitole.

4.3 Programování ovládacích jednotek v praxi

4.3.1 Připojení

Připojení k ovládacím jednotkám zařizuje funkce *Connect(string portName)*. Zadává se do ní příslušný COM, na kterém je zařízení připojeno. Nejprve se zahájí komunikace a uloží se komunikační klíč do globální proměnné keyHandle. Pokud zařízení odpovídá, tak se nastaví parametry komunikace. Pokud nastavení bylo úspěšné, tak se vymažou chyby a aktivují se motory. Funkce vrací hodnotu int, jehož hodnota odpovídá úspěšnosti připojení.

```
public int Connect(string portName)
{
      keyHandle = Device.VcsOpenDevice("EPOS", "MAXON_RS232", "RS232", portName,
ref err);    //zahájení komunikace
      if (keyHandle != 0) //pokud zařízení odpovídá
      {
            int connect = Device.VcsSetProtocolStackSettings(keyHandle, 38400,
100, ref err);   //nastavení parametrů komunikace

            if (connect != 0)
            {

                  Device.VcsClearFault(keyHandle, nodeIdY, ref err);
//vymazání všech chybových hlášek u obou jednotek EPOS
                  Device.VcsClearFault(keyHandle, nodeIdX, ref err);
                  Device.VcsSetEnableState(keyHandle, nodeIdY, ref err);
//nastavení motorů na nepohyblivé
                  Device.VcsSetEnableState(keyHandle, nodeIdX, ref err);

                  return 1;
            }
            else return 0;
      }
      else return 2;
}
```

Zdrojový kód 4.6 Připojení k ovládacím jednotkám

4.3.2 Získání polohy

V programu je opakovaně nutno získávat polohu. K tomu slouží funkce *ZiskatPolohu()*. Funkce vrací pole int, kde první dvě hodnoty jsou souřadnice X a Y. Jako třetí hodnota je, jestli získání polohy bylo úspěšné a čtvrté místo nám říká, jestli jsou souřadnice kladné.

```
public int[] ZiskatPolohu()
{
	int[] poloha = { 0, 0, 0, 0};

	int a = Device.VcsGetPositionIs(keyHandle, nodeIdX, ref poloha[0], ref err);
//získání polohy X
	int b = Device.VcsGetPositionIs(keyHandle, nodeIdY, ref poloha[1], ref err);
//získání polohy Y

	if (a != 0 && b != 0)   //pokud úspěšné
	{
		poloha[2] = 1;
		if (poloha[0] >= 0 && poloha[1] >= 0) poloha[3] = 1;    //pokud je
poloha nenulová
	}
	return poloha;
}
```

Zdrojový kód 4.7 Získání polohy

4.3.3 Manuální pohyb

Zařízení je možné polohovat ručně pomocí tlačítek (nahoru, dolů, doprava, doleva). Pohyb je realizován použitím Profile Velocity módu. Funkce pro manuální pohyb se nachází ve třídě *HlavniOvladani* a jmenuje se *ManualniPohyb(int smer)*. Pro zkrácení a větší přehlednost programu jsou všechny směry pohybů volány stejnou funkcí s parametrem, který udává požadovaný směr pohybu. Parametr je typu int. Zastavení se provádí stejnou funkcí, kde parametr směr je roven pěti.

Při stisknutí a držení tlačítka označující požadovaný směr se zavolá funkce *ManualniPohyb* s parametrem s příslušnou hodnotou. Funkce nejprve přenastaví ovládací jednotky do správného módu a nastaví parametry pohybu, zrychlení a zpomalení. Tyto parametry není třeba přenastavit, protože narozdíl od měření nám nevadí drobné záchvěvy měřicího aparátu. Poté se podle zadaného parametru *smer* vybere příslušná ovládací jednotka a příslušné znaménko rychlosti. Zavolá se funkce z knihovny *EposCmd.Net*, která zadá příslušné ovládací jednotce požadovanou rychlost. Hodnota rychlosti se bere z parametrů pohybu, které je možno přenastavit. K zastavení dojde po uvolnění tlačítka nastavením požadované rychlosti na nulu. Požadavek se posílá oběma jednotkám. Protože není třeba znovu nastavovat ovládací jednotky, požadavek na zastavení se ověřuje před přenastavením módu a parametrů pohybu. Díky tomu je také lepší odezva.

```
public void ManualniPohyb(int smer)
{
	int r = (int)parametry.rychlost;    //cílová rychlost

	if (smer == 5)    //zastavení
	{
		Device.VcsMoveWithVelocity(keyHandle, nodeIdY, 0, ref err);
		Device.VcsMoveWithVelocity(keyHandle, nodeIdX, 0, ref err);
		return;
	}
```

Zdrojový kód 4.8 Manuální pohyb část 1

```
      Device.VcsSetOperationMode(keyHandle, nodeIdX,
EposCmd.Net.EOperationMode.OmdProfileVelocityMode, ref err);
      Device.VcsSetOperationMode(keyHandle, nodeIdY,
EposCmd.Net.EOperationMode.OmdProfileVelocityMode, ref err); //nastaveni jednotek
do správného módu
      Device.VcsSetVelocityProfile(keyHandle, nodeIdX, 1000, 1000, ref err);
//nastavení zrychlení a zpomalení
      Device.VcsSetVelocityProfile(keyHandle, nodeIdY, 1000, 1000, ref err);

      if (smer == 1) Device.VcsMoveWithVelocity(keyHandle, nodeIdY, - r, ref err);
//nahoru
      if (smer == 2) Device.VcsMoveWithVelocity(keyHandle, nodeIdY, r, ref err);
//dolu
      if (smer == 3) Device.VcsMoveWithVelocity(keyHandle, nodeIdX, - r, ref err);
//vpravo
      if (smer == 4) Device.VcsMoveWithVelocity(keyHandle, nodeIdX, r, ref err);
//vlevo
}
```

Zdrojový kód 4.9 Manuální pohyb část 2

Uvolnění motorů

Pro potřeby ručního posouvání pojízdného vozíku byla naprogramovaná funkce *UvolnitMotory()*, která, jak už její název napovídá, uvolní motory a umožní tak jejich volné otáčení. Tato funkce je velmi nebezpečná, protože při vertikálním postavení rámu může dojít k pádu pojízdného vozíku. Při uvolnění motorů se také znemožní veškeré pohybové a polohovací funkce, dokud se motory opět neaktivují.

Funkce je pod třídou *HlavniOvladani* a pracuje s veřejnou proměnnou *bool motory_enabled*, která nám říká, v jakém stavu se motory momentálně nachází. Funkce zjistí, v jakém stavu se motory aktuálně nachází a následně ho změní.

```
public void UvolnitMotory()
{
      if (motory_enabled)
      {
            Device.VcsSetDisableState(keyHandle, nodeIdX, ref err);
            Device.VcsSetDisableState(keyHandle, nodeIdY, ref err); //uvolnění
motorů
            motory_enabled = false;
      }
      else
      {
            Device.VcsSetEnableState(keyHandle,nodeIdY, ref err);
            Device.VcsSetEnableState(keyHandle,nodeIdX, ref err);   //aktivování
motorů
            motory_enabled = true;
      }
}
```

Zdrojový kód 4.10 Uvolnění motorů

4.3.4 Homing

Po každém spuštění zařízení je nutno provést homing, který zařizuje funkce *Homing(ushort nodeId)*. Funkce se nachází pod třídou *HlavniOvladani* a jako parametr zadáváme ID karty, pro kterou chceme provést homing.

Nejprve nastavíme ovládací jednotky do Homing módu a nastavíme parametry. Pro nalezení domácí polohy (home) je použitá metoda -4, která plně vyhovuje našim požadavkům na přesnost a nabízí jednoduché řešení hledání domácí polohy bez použití dalšího hardwaru. Na dorazových plochách rámu jsou umístěny pěnové zarážky, které se při dojíždění pojízdného vozíku lehce stlačí. Aby pokaždé, když by byl pojízdný vozík v nulové pozici, byla tato zarážka stlačená motory, je nastavený offset o 2000 bodů. Parametr offset je součástí funkce nastavující parametry hledání domácí polohy.

Pomocí poslední funkce se počká, až bude domácí poloha nalezena. Může dojít k chybě a nemusí se podařit home najít. Proto se do této funkce zadává timeout, do kdy má být domácí poloha nalezena. Když dojde k překročení tohoto času nebo k jiné chybě, funkce vrátí nulovou hodnotu. Tuto hodnotu pak vrací naše funkce *Homing*.

```
private int Homing(ushort nodeId)
{
      Device.VcsActivateHomingMode(keyHandle, nodeId, ref err);
//nejprve se nastaví zařízeni na homing mode
      Device.VcsSetHomingParameter(keyHandle, nodeId, 100, 300, 0, 100, 2000, 0,
ref err);     //zadájí se příslušné parametry

      Device.VcsFindHome(keyHandle, nodeId,
EposCmd.Net.EHomingMethod.HmCurrentThresholdNegativeSpeed, ref err); //zahájí
homing

      int konec = Device.VcsWaitForHomingAttained(keyHandle, nodeId, 100_000, ref
err);   //počká se na nalezení home, funkce vrací nenulovou hodnotu, když homing
proběhl úspěšně
      return konec;
}
```

Zdrojový kód 4.11 Homing část 1

Při požadavku na homing se ve skutečnosti volá funkce *FindHomeAsync()*, která volá předešlou funkci *Homing*. Tato funkce je asynchronní, aby nedošlo k zamrznutí formu při vykonávání homingu, a vrací kladnou hodnotu typu int, jestli byl homing úspěšný. Nejprve se zavolá funkce *Homing* pro osu x. Funkce se volá jako *Task* s použitím operátoru *await*. Díky tomu dojde k uvolnění vlákna a můžou se vykonávat další operace. Jakmile doběhne homing pro osu x, volá se identicky homing pro osu y. Jako poslední se ověří, jestli byl homing obou os úspěšný.

```
public async Task<int> FindHomeAsync()
{
      int konecX = await Task.Run(() => Homing(nodeIdX)); //homing pro osu X
      int konecY = await Task.Run(() => Homing(nodeIdY)); //homing pro osu Y
      if (konecX != 0 && konecY != 0) return 1;       //ověření jestli byla
operace úspěšná
      else return 0;
}
```

Zdrojový kód 4.12 Homing část 2

4.3.5 Bezpečné ukončení

Po vypnutí zařízení se uvolní motory, a pokud je zařízení postaveno vertikálně, může dojít k pádu pojízdného vozíku. Při pádu může dojít k poškození měřícího zařízení a převodovky motoru. K ošetření možného pádu je nutné zajet s vozíkem do nejspodnější pozice v ose Y. Pro tento účel byla v programu použita funkce *BezpecneUkoncit()*. Funkce zajede pojízdným vozíkem do spodní polohy použitím Homing modu. Princip je stejný jako při hledání domácí polohy, avšak pohyb je na druhou stranu.

```
public void BezpecneUkoncit()
{
      Device.VcsActivateHomingMode(keyHandle, nodeIdY, ref err);
      Device.VcsSetHomingParameter(keyHandle, nodeIdY, 100, 200, 0, 0, 1500, 0,
ref err);
      Device.VcsFindHome(keyHandle, nodeIdY,
EposCmd.Net.EHomingMethod.HmCurrentThresholdPositiveSpeed, ref err);
}
```

Zdrojový kód 4.13 Bezpečné ukončení

4.3.6 Měření

Funkce zajišťující měření je opět pro větší přehlednost rozdělena do několika menších funkcí a ty si zde popíšeme.

Spouštění měření

Začněme první funkcí s názvem *SpustitMereniAsync()*. Tato funkce je opět asynchronní. Nejprve funkce smaže veřejný *List databaze* obsahující případné předchozí měření. Dále nastavuje ovládací jednotky do Profile Position módu a nastavuje parametry pohybu, rychlost, zrychlení, zpomalení, přičemž zpomalení má stejnou hodnotu jako zrychlení. Tyto parametry lze přenastavit, například kdyby byl měřící aparát citlivý na prudké pohyby.

```
public async Task<int> SpustitMereniAsync()
{
      databaze.Clear();

      Device.VcsActivateProfilePositionMode(keyHandle, nodeIdX, ref err);
//změna modu na position mode
      Device.VcsActivateProfilePositionMode(keyHandle, nodeIdY, ref err);
      Device.VcsSetPositionProfile(keyHandle, nodeIdX, parametry.rychlost,
parametry.zrychleni, parametry.zrychleni, ref err);     //nastavení parametrů pro
position mode (rychlost, zrychleni, zpomalení)
      Device.VcsSetPositionProfile(keyHandle, nodeIdY, parametry.rychlost,
parametry.zrychleni, parametry.zrychleni, ref err);
```

Zdrojový kód 4.14 Spouštění měření část 1

Pro velkou volnost, co se týče zadávání oblasti měření je nutno přepočítat parametry, aby zůstala univerzalita funkce vykonávající měření. Nejprve se uloží parametry pro pohyb po různých osách. Parametry s označením jedna se váží k ose, po které se bude pojízdný vozík pohybovat. Parametry s označením dva se váží k ose, na které bude docházet k posunu na konci každého řádku. Konkrétně přiřazujeme startovací a konečné pozice a ID příslušných ovládacích karet.

```
            ushort nodeId1 = 1;
            ushort nodeId2 = 3;
            int konec1 = 0;
            int konec2 = 0;
            int start1 = 0;
            int start2 = 0;
            int posun1 = 0;
            int posun2 = 0;
            int pKroku1 = 0;
            int pKroku2 = 0;
            //parametry s oznacenim 1 se váží k ose po které se bude pojizdny
vozik pohybovat
            //níže probíhá přiřazení

            if (parametry.osa == 'X')
            {
                nodeId1 = nodeIdX;
                nodeId2 = nodeIdY;
                konec1 = parametry.Xkonec;
                konec2 = parametry.Ykonec;
                start1 = parametry.Xstart;
                start2 = parametry.Ystart;
            }
            if (parametry.osa == 'Y')
            {
                nodeId1 = nodeIdY;
                nodeId2 = nodeIdX;
                konec1 = parametry.Ykonec;
                konec2 = parametry.Xkonec;
                start1 = parametry.Ystart;
                start2 = parametry.Xstart;
            }
```

Zdrojový kód 4.15 Spouštění měření část 2

Následuje výpočet počtu kroků a přiřazení vhodného znaménka k parametru posun. Například pokud by souřadnice startovaní polohy byla větší než konečné polohy, musí se posun místo přičítání odečítat.

```
            //výpočet počtu kroků
            pKroku1 = Math.Abs(start1 - konec1) / parametry.posun;
            pKroku2 = Math.Abs(start2 - konec2) / parametry.posun;

            //Přiřazení vhodného znaménka
            if (start1 < konec1) posun1 = parametry.posun;
            else posun1 = -parametry.posun;

            if (start2 < konec2) posun2 = parametry.posun;
            else posun2 = -parametry.posun;
```

Zdrojový kód 4.16 Spouštění měření část 3

Po vykonání všech přepočtů dochází na volání funkce *Mereni*, která se volá jako *Task* s použitím operátoru *await*. Počká se na hodnotu int, kterou vrací již zmíněná funkce a stejnou hodnotu pak vrací i funkce *SpustitMereniAsync*. Ještě před vrácením této hodnoty dochází k nulování stop proměnné, která umožňuje ukončit měření uprostřed procesu. Více o zastavení měření je popsáno v další části.

```
      //spuštění měření
      int konec = await Task.Run(() => Mereni(nodeId1, nodeId2, start1, start2,
posun1, posun2, pKroku1, pKroku2));
      stop = false;
      return konec;
}
```

Zdrojový kód 4.17 Spouštění měření část 4

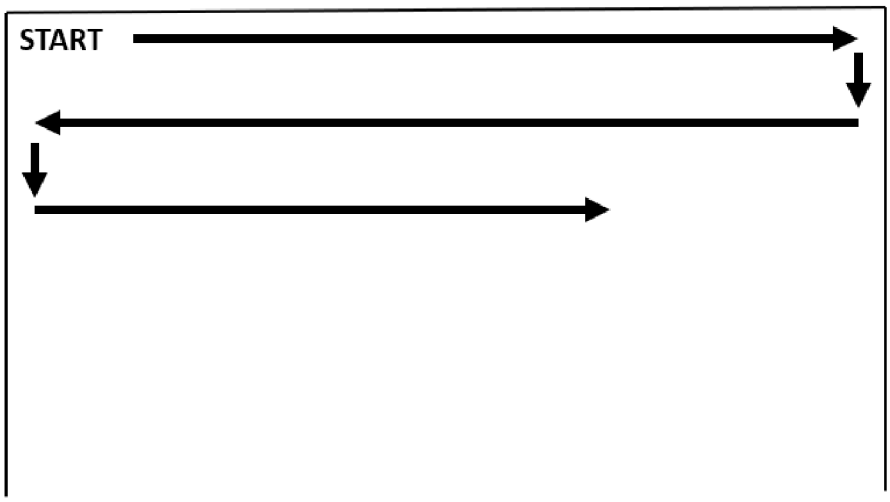
Automatické Měření

Zde je podrobně popsáno, jak funguje samotné měření. Funkce *Mereni* zajišťuje automatické polohování a měření. Celá funkce je postavena kolem dvou smyček *for*. Nejprve je třeba vytvořit instanci třídy *Data*, do které se uloží naměřená data. Poté se najede do startovací pozice a uloží startovací pozice do pracovních proměnných, pomocí kterých budou počítány další pozice. A inicializují se proměnné *X* a *Y*, do kterých se ukládá informace o poloze.

```
        private int Mereni(ushort nodeId1, ushort nodeId2, int start1, int start2,
int posun1, int posun2, int pKroku1, int pKroku2)
        {
            Data data = new Data();     //vytvoření proměnné do které se zapisují
data o poloze a náměry

            Device.VcsMoveToPosition(keyHandle, nodeId1, start1, 1, 0, ref err);
//nájezd do startovací pozice
            Device.VcsMoveToPosition(keyHandle, nodeId2, start2, 1, 0, ref err);

            Device.VcsWaitForTargetReached(keyHandle, nodeId1, 100_000, ref err);
//čekání na dosažení požadované pozice
            Device.VcsWaitForTargetReached(keyHandle, nodeId2, 100_000, ref err);

            int o2 = start2;
            int o1 = start1;
            int X = 0;
            int Y = 0;
```

Zdrojový kód 4.18 Automatické měření část 1

Poté se funkce dostane na dvojici *for*, které postupně projedou všechny body podle zadaných souřadnic a posunu. První *for* zařizuje pohyb a měření ve sloupci. Druhý *for*, který se nachází uvnitř prvního obstarává pohyb a měření v řádcích. Když zařízení dojede na požadovanou pozici, tak se nejprve hodnota polohy přepočte na mm a zapíše se do *data*. Poté se vezme náměr pomocí funkce *VezmiNamer()*, která je popsaná níže. Jakmile je měření dokončeno, tak se *data* uloží do listu *databaze* a vynulují se. V prvním *for* se při každém cyklu prohazuje znaménko posunu v řadě, díky tomu se v řádku může jet na opačnou stranu a měření je tím rychlejší.



Obrázek 4.2 Příklad pohybu při automatickém měření

Při každém cyklu každého *for* se testuje proměnná stop, jestli nedošlo k požadavku na přerušení měření. Tato proměnná je veřejná, a tak jí lze měnit přímo při stisku tlačítka. Pokud dojde k přerušení, tak se zruší smyčka a funkce se nechá doběhnout. Po dokončení cyklů se pouze vrací hodnota jedna, že funkce skončila.

```
            for (int i = 0; i <= pKroku2; i++)
            {
                Device.VcsGetPositionIs(keyHandle, nodeIdX, ref data.X, ref err);
                Device.VcsGetPositionIs(keyHandle, nodeIdY, ref data.Y, ref err);

                data.namer = VezmiNamer();

                databaze.Add(data);
                data = new Data();

                for (int j = 0; j < pKroku1; j++)
                {

                    o1 = o1 + posun1;
                    Device.VcsMoveToPosition(keyHandle, nodeId1, o1, 1, 0, ref
err);
                    Device.VcsWaitForTargetReached(keyHandle, nodeId1, 600_000,
ref err);

                    Device.VcsGetPositionIs(keyHandle, nodeIdX, ref data.X, ref
err);
                    Device.VcsGetPositionIs(keyHandle, nodeIdY, ref data.Y, ref
err);

                    data.namer = VezmiNamer();

                    databaze.Add(data);
                    data = new Data();

                    if (stop)
                    {
                        break;
                    }
                }
                o2 = o2 + posun2;
                Device.VcsMoveToPosition(keyHandle, nodeId2, o2, 1, 0, ref err);
                Device.VcsWaitForTargetReached(keyHandle, nodeId2, 600_000, ref
err);
                posun1 = -posun1;

                if (stop)   //zastaveni mereni
                {
                    break;
                }
            }
            return 1;
}
```

Zdrojový kód 4.19 Automatické měření část 2

Sejmutí náměru

Pro větší přehlednost předchozí funkce byla část, která se opakuje přesunuta do samostatné funkce s názvem *VezmiNamer()*. Tato funkce se stará o sejmutí a zprůměrování správného počtu náměrů, a o čekací dobu před prvním náměrem.

```
private double VezmiNamer()
{
	Task.Delay(parametry.wait).Wait();  //cekani pred prvnim namerem

	double h = 0;
	for (int p = 0; p < parametry.pocet; p++)   //sejmuti nastaveneho poctu
nameru
	{
		h = h + MM.ZmerDouble();
	}
	return (float)h / parametry.pocet;   //zprumerovani a vraceni hodnoty
}
```

Zdrojový kód 4.20 Sejmutí náměru podle zadaných parametrů

4.4 Programování zápisu dat

V této části si popíšeme způsob zápisu dat, jeho provedení a myšlenkový postup za výběrem vhodného formátu.

4.4.1 Volba vhodného formátu

Při zvolení vhodného formátu musíme brát v potaz, jak se data budou dále zpracovávat. Jestli budou zpracovávána plně automaticky nebo jestli je bude zpracovávat člověk. V našem případě je požadováno, aby byla výstupní data dobře čitelná jak pro počítač, tak pro lidskou obsluhu. V úvaze byly dva formáty, a to textový soubor a soubor CSV. Soubory CSV se od textových souborů liší tím, jak jsou jednotlivá data oddělovány. Zatímco textové soubory mají mnoho způsobů, jak data oddělovat (čárky, středníky, mezery), soubory CSV jsou více standardizované a data se zde oddělují čárkou. Tato standardizace umožňuje jasné zpracování dat dalšími aplikacemi. Není třeba žádného převodu, ani speciálního přístupu. Díky jednoduchosti a jasnosti CSV souborů existuje spousta aplikací, které podporují tento formát. Po zvážení našich požadavků bylo jasné použít formát CSV.

4.4.2 Programování zápisu dat do souboru CSV

Jak bylo dříve zmíněno soubory CSV mají velkou podporu jak externích programů, tak různých knihoven. Pro náš program byla použita knihovna *CsvHelper*, která nabízí veškeré funkce, které bychom mohli potřebovat pro práci s CSV soubory. Pro zápis dat byla v programu vytvořena vlastní funkce. To nám případně zjednoduší práci, kdyby bylo potřeba formát souboru změnit. Vytvořená funkce je zobrazena níže. Vytvoření souboru a zápis dat probíhá až po ukončení měření (data se ukládají jako celek, ne každá hodnota zvlášť). Ukládá se poloha měřícího sondy a hodnota napětí na voltmetru, tudíž celkem tři hodnoty. Před uložením dat je třeba vybrat složku, kam chceme soubor uložit. Stisknutím tlačítka Uložit se spustí *FolderBrowser* dialog. Funkce poté pracuje s adresou složky, kterou jsme vybrali.

```
public void Zapis(string path)
{

      for(int i = 0; i < databaze.Count; i++) //prepocitani souradnic na mm, a tak
aby start byl 0
      {
            databaze.ElementAt(i).X = (databaze.ElementAt(i).X -
parametry.Xstart)/280;
            databaze.ElementAt(i).Y = (databaze.ElementAt(i).Y -
parametry.Ystart)/280;
      }

      var csvPath = Path.Combine(path, $"Mereni-{DateTime.Now.ToString("yyyy-MM-
dd_HH.mm.ss")}.csv"); //cesta kam se uloží náš soubor a jeho název
      using (StreamWriter streamWriter = new StreamWriter(csvPath))   //vytvoření
csv souboru
      using (CsvWriter csvWriter = new CsvWriter(streamWriter,
CultureInfo.InvariantCulture)) //nástroj pro zápis do csv souboru
      {
            csvWriter.Context.RegisterClassMap<DataMap>();  //formát dat, v jakém
je chceme zapisovat
            csvWriter.WriteRecords(databaze);   //zápis
      }
}
```

Zdrojový kód 4.21 Funkce pro zápis dat

Tuto funkci můžeme najít pod třídou *HlavniOvladani*. Jak bylo dříve řečeno, funkci se předává zvolená cesta do složky, kde se následně vytvoří nový CSV soubor s naměřenými daty. Soubor má automaticky generovaný název, který se skládá ze slova Měření a následovaným aktuálním datem a časem. Použitím *CsvWriteru* poté zapíšeme data podle mapy, která je definovaná ve třídě *DataMap*.

```
public class DataMap : ClassMap<Data>
    {
        public DataMap()
        {
            Map(r => r.X).Name("X");
            Map(r => r.Y).Name("Y");
            Map(r => r.namer).Name("hodnota");

        }
    }
```

Zdrojový kód 4.22 Mapa pro zápis dat

4.5 Programování čtení dat

V této části je představeno, jak vypadá připojení a komunikace multimetru Metex s počítačem. A následně je popsáno její programové provedení.

4.5.1 Připojení a komunikace s multimetrem

Multimetr je připojen k počítači prostřednictvím sériového portu RS232. Parametry komunikace jsou 1200 baud, parita žádná a dva stop bity. Data jsou kódována do 7-bit ASCII kódu. Vzhledem k velké energetické spotřebě sériového portu nejsou data posílána pořád, ale jsou poslána pouze na požadavek. Tento požadavek je ve formě velkého tiskacího D, který pošleme z počítače. Multimetr následně pošle data s dalším jeho obnovením. Poslaný řetězec je velký 14 bajtů (14 znaků). Vzhledem k malé přenosové rychlosti a dlouhé obnovovací frekvenci multimetru je měření zdlouhavé a je třeba čekat na příchozí data několik stovek milisekund.

4.5.2 Programování multimetru

Veškeré funkce jsou napsány pod třídou *Multimetr* pro větší přehlednost a snadnou úpravu, kdyby byla nutná změna procesu měření. Instance této třídy je vytvořená v naší hlavní třídě *HlavniOvladani* se jménem *MM*. Při vytváření instance třídy se nic nepředává.

Připojení

K připojení multimetru slouží funkce *openPort*. Při volání funkce se zadává jeden parametr, a to je jméno portu na kterém je multimetr připojen. Funkce vrací integer, který říká úspěšnost připojení. Nejprve se vytvoří instance třídy *SerialPort* z knihovny *System.IO.Ports* se jménem portu a parametry komunikace. Pro správnou funkci multimetru se pin DTR nastaví na hodnotu jedna a pin RTS na hodnotu nula. Nastaví se také čtecí timeout. Následně se pomocí funkce *port.Open()* započne komunikace. Pro velké riziko se funkce volá ve fragmentu *try*. Po otevření portu se ještě pro jistotu smaže buffer.

Poté je třeba ověřit, zda multimetr pracuje správně. Pošleme příkaz „D“ a čekáme, jestli se vrátí správný počet bajtů. Doba, po které multimetr odešle data, se může pokaždé lišit, jak bylo dříve řečeno. A proto je ověřovací část obklopena funkcí *while*. Pokud multimetr nevrátí správný počet bajtů, počká se 20ms. Celkem může program na multimetr počkat až 1400ms. Pokud počet bajtů stále nesedí, vrátí se hodnota dva. Pokud bylo ověření úspěšné vrací se hodnota jedna a pokud pokus skončil výjimkou vrací se hodnota nula.

```
public int openPort(string name)
        {
            port = new SerialPort(name, 1200, Parity.None, 7, StopBits.Two);

            port.RtsEnable = false;
            port.DtrEnable = true;
            port.ReadTimeout = 1000;

            try
            {
                port.Open();
                port.DiscardInBuffer();
                port.Write("D");    //Pošleme požadavek na poslání náměru

                int i = 0;
                while (i < 70)  //multimetr pošle náměr až s dalším obnovením,
musíme počkat až k obnovení dojde
                {
                   int b = port.BytesToRead;
                    if (b == 14)
                    {
                        return 1;

                    }
                    else
                    {
                        i++;
                        Task.Delay(20).Wait();
                    }
                }
                port.Close();
                return 2;

            }
            catch (Exception)
            {
                return 0;
            }
        }
```

Zdrojový kód 4.23 Připojení multimetru

Získání náměru

Pro získání náměru byla vytvořena funkce *Zmer(int format)*, která se také nachází pod třídou *Multimetr*. Vstupní proměnná *format* udává, v jakém formátu se přečtená hodnota vrátí. Vrátí se celý čtrnáctiznakový string, pokud se hodnota rovná nule. Pokud hodnota bude nabývat jedné, vrátí se pouze oříznuté číslo.

Nejprve je třeba zjistit, zda je náš port otevřený a může probíhat komunikace. Poté se smaže buffer a vyšle se požadavek na zaslání náměru. Jako u připojení multimetru se čeká, jestli multimetr pošle všechny bajty, až poté se přečte buffer. Následuje převedení bajtů do stringu a případné oddělení číselné hodnoty.

```
        public string Zmer(int format)
        {
            if (port.IsOpen)    //zjistí, jestli je port otevřený
            {
                string data = "";
                string pattern = @"[-+]?\d*\.\d+|\d+";
                string oddeleno = "";
                try
                {
                    port.DiscardInBuffer(); //smaže se buffer
                    port.Write("D");    //Pošle se požadavek na poslání náměru

                    int i = 0;          //pomocná proměnná
                    int b = port.BytesToRead; //zjistí kolik bajtů bylo přijato

                    while (i < 70)  //multimetr pošle náměr až s dalším obnovením,
musíme počkat až k obnovení dojde
                    {
                        b = port.BytesToRead;
                        if (b == 14)    //ověřujeme, jestli byl přijat správný
počet bajtů pokud ne, čekáme 20ms a opakujeme
                        {
                            break;
                        }
                        else
                        {
                            i++;
                            Task.Delay(20).Wait();
                        }
                    }

                    byte[] buffer = new byte[b];    //promenná kam se uloží
přečtená hodnota
                    port.Read(buffer, 0, buffer.Length);    //čtení

                    foreach (byte r in buffer)  //převedení bajtů do stringu
                    {
                        data += (char)r;
                    }

                    if (format == 1)    //oddělení čísla od zbytku stringu
                    {
                        MatchCollection matches = Regex.Matches(data, pattern);

                        foreach (Match match in matches)
                        {
                            oddeleno += match.Value;
                        }
                        return oddeleno;
                    }
                    else return data;
```

Zdrojový kód 4.24 Získání náměru (text)

Pro případ potřeby číselné hodnoty byla vytvořena funkce *ZmerDouble()*, která místo stringu vrací proměnnou typu double. Tato funkce nejprve získá náměr pomocí předchozí funkce. Pro správnou funkci *ToDouble()* musí mít string místo desetinné tečky desetinnou čárku. Pro případ, že by funkce *Zmer()* nevrátila číslo, je převod na double obklopen fragmentem *try*.

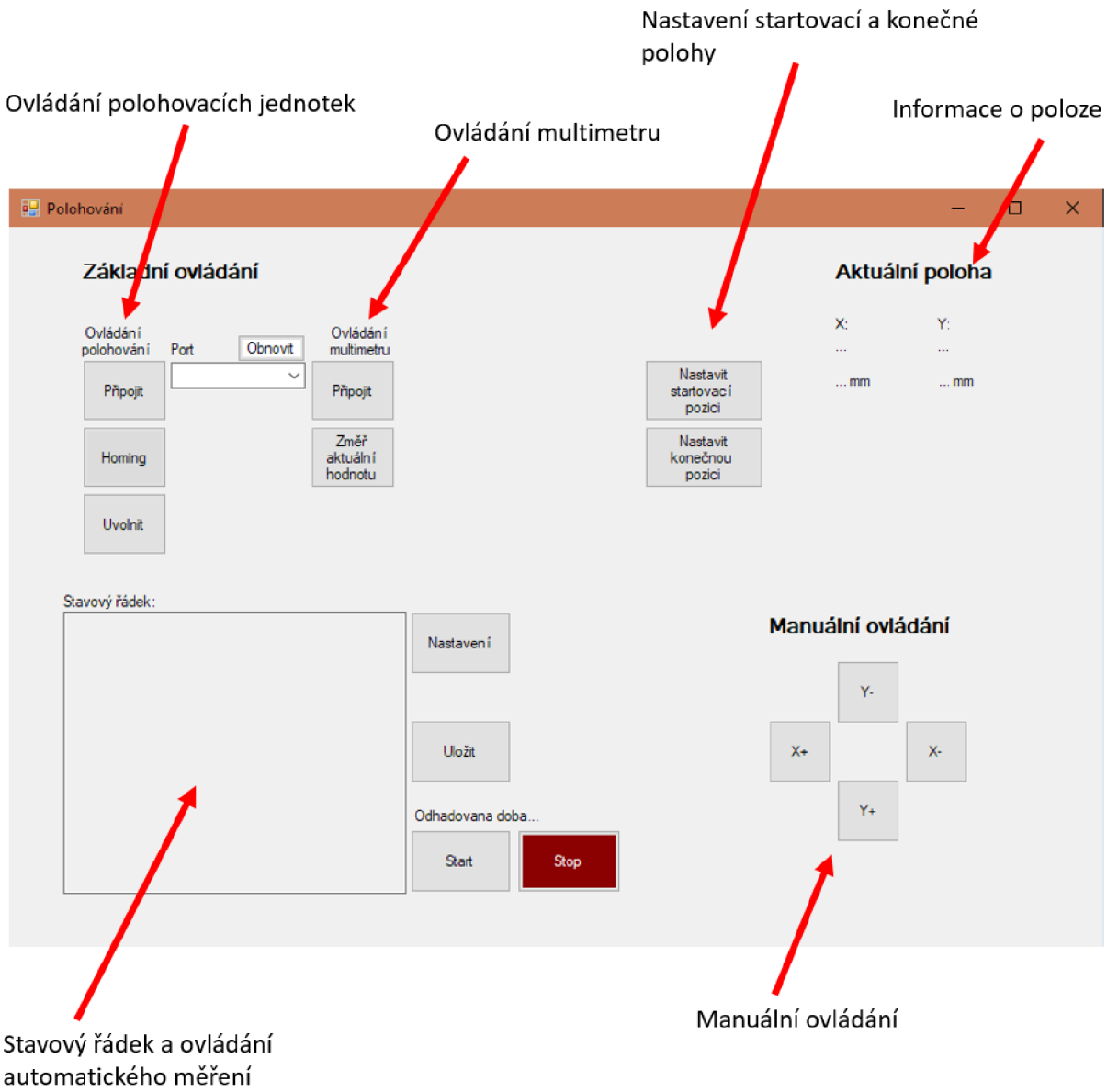
```
public double ZmerDouble()
{
	string namer = Zmer(1); //ziskani nameru

	try
	{
		string dnamer = namer.Replace('.', ',');    //funkce ToDouble pracuje
s desetinnou čárkou, ne tečkou
		return Convert.ToDouble(dnamer);
	}
	catch   //může se stát, že MM nic nevrátí a fce Zmer vrátí "err"
	{
		return 0;   //V tom případě tato funkce vrátí 0
	}
}
```

Zdrojový kód 4.25 Získání náměru (číslo)

5 Ovládání programu

V této kapitole je podrobně popsáno ovládání programu. Program má dvě okna. První okno je hlavní a slouží k ovládání polohovacího zařízení a multimetru. Druhé okno je pro nastavení parametrů měření.

5.1 Hlavní okno



Obrázek 5.1 Hlavní okno programu

5.1.1 Ovládání polohovacích jednotek

Nejprve je třeba zařízení připojit. Pro připojení se musí vybrat správný port a stisknout tlačítko *Připojit*. Po úspěšném připojení tlačítko zezelená.

Po připojení je třeba zařízení nahomovat, to se provede stiskem tlačítka *Homing*. Po úspěšném homování tlačítko také zezelená.

Kdyby bylo třeba pojízdným vozíkem volně pohybovat, stiskem tlačítka *Uvolnit* se uvolní motory. Pro nebezpečí pádu při vertikálním postavení zařízení vyskočí před provedením operace okénko, jestli jsme si požadavkem jisti. Po potvrzení tlačítko změní barvu na tyrkysovou. Pro další práci se zařízením je třeba motory opět zabrzdit. To se provede opětovným stiskem tohoto tlačítka.

5.1.2 Ovládání multimetru

Nejprve je třeba multimetr připojit podobně jako připojení polohovacích jednotek. Pro připojení se musí vybrat správný port a stisknout tlačítko *Připojit*. Po úspěšném připojení tlačítko zezelená.

Pro zjištění aktuální hodnoty, která se na multimetru nachází stiskneme tlačítko *Změř aktuální hodnotu*. Do stavového řádku se poté vypíše naměřená hodnota.

5.1.3 Ovládání automatického měření

Nastavení parametrů měření probíhá stiskem tlačítka *Nastavení*. Po stisku se otevře speciální okno, ve kterém je možno přenastavit všechny parametry.

Podmínka spuštění automatického měření je, že zařízení bylo nahomováno. Měření se spustí stiskem tlačítka *Start*. Po stisku tohoto tlačítka se objeví další tlačítko *Stop*, které v případě potřeby zastaví měření.

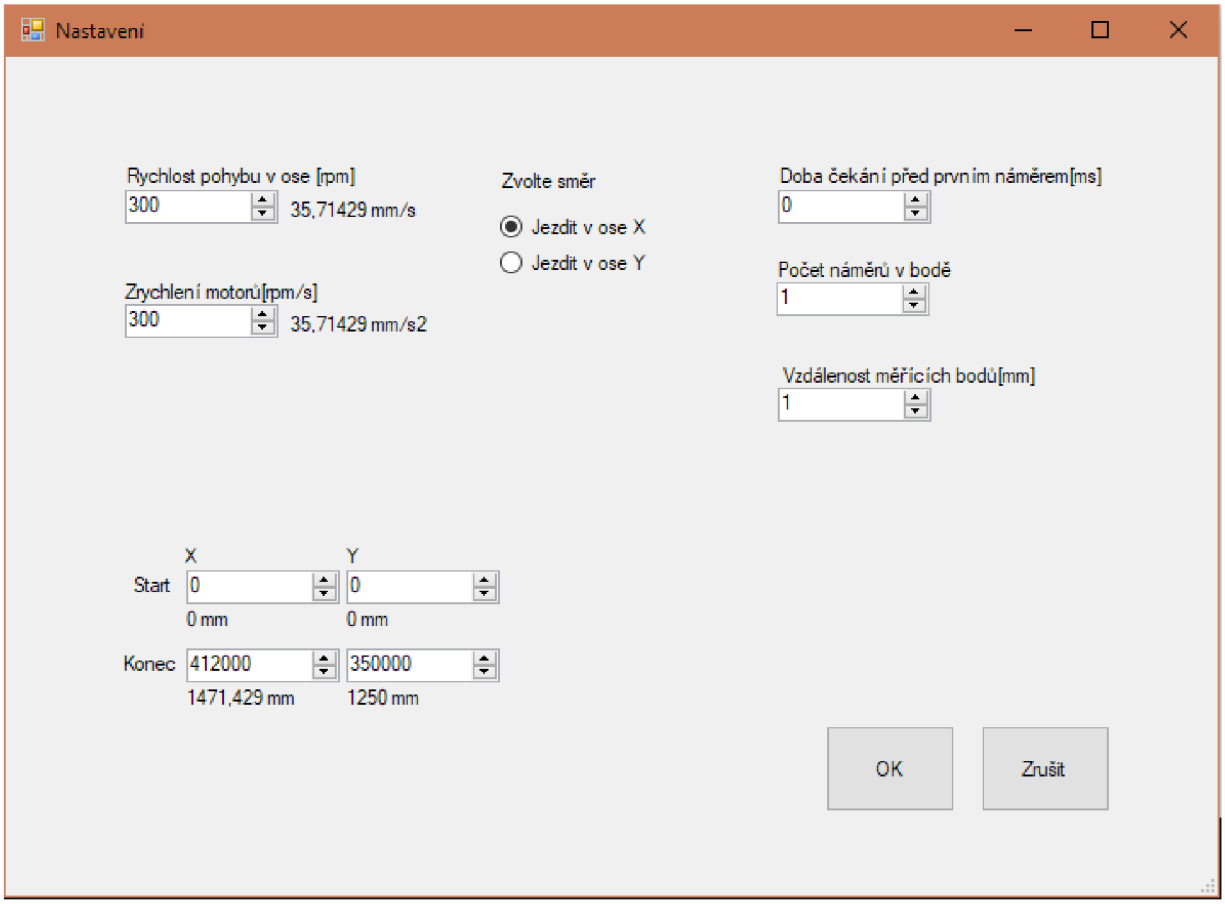
Po skončení měření, ať už stiskem tlačítka *Stop* nebo doběhnutím programu, je třeba naměřené hodnoty uložit. Uložení naměřených hodnot se realizuje stiskem tlačítka *Uložit*. Po stisku se objeví dialog, ve kterém vybereme složku, do které chceme data uložit.

5.1.4 Aktuální poloha

Aktuální poloha pojízdného se zobrazuje vpravo nahoře. Stiskem tlačítek *Nastavit startovací pozici* a *Nastavit konečnou pozici* lze přímo nastavit oblast, kterou chceme měřit bez nutnosti otevírat nastavení. Při nastavování těchto poloh je nutné mít na paměti směry os.

5.2 Okno nastavení

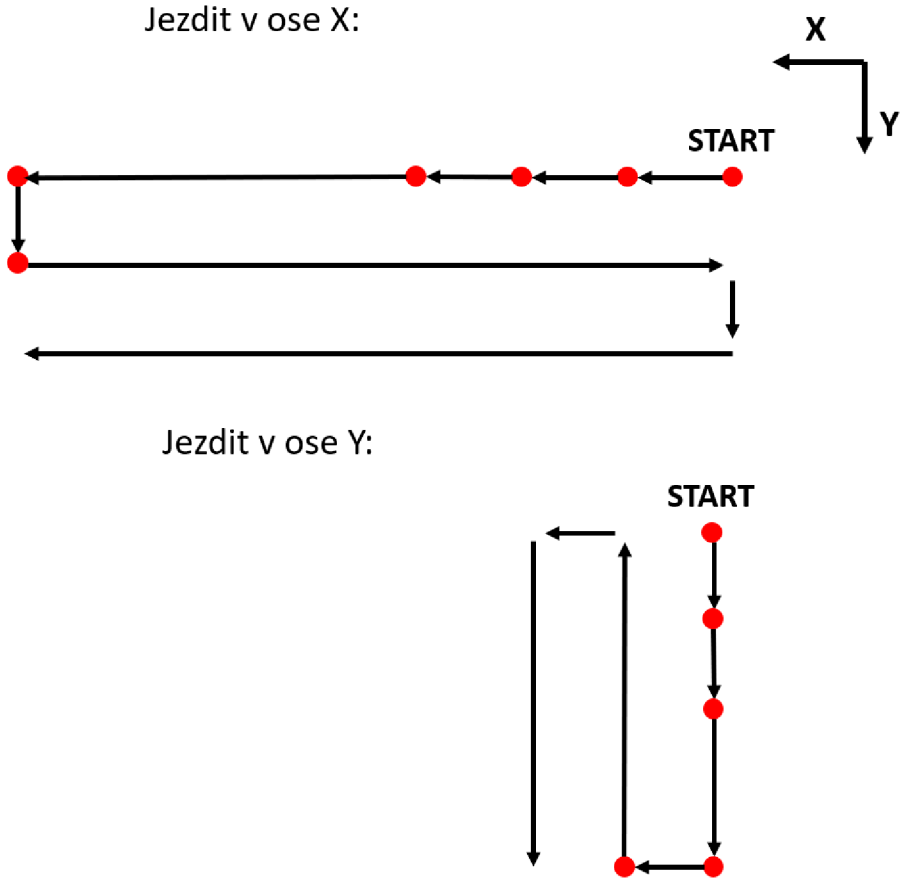
Zde je možno přenastavit veškeré parametry automatického řízení. Parametr rychlost se také aplikuje na manuální pohyb. Pro potvrzení parametrů se stiskne tlačítko *OK*, pro zahození přepsaných parametrů se stiskne tlačítko *Zrušit*.



Obrázek 5.2 Okno nastavení

5.2.1 Zvolení směru

Zvolení, jestli zařízení bude jezdit po sloupcích nebo po řádcích.



Obrázek 5.3 Vizualizace nastavovaných směrů měření

5.2.2 Doba čekání

Doba čekání před prvním náměrem nám umožňuje nastavit čas, po který bude zařízení čekat na měřícím bodě před sejmutím náměru. Pokud je požadováno více náměrů, tak se čeká pouze před prvním. Tento parametr se hodí například při použití senzoru s velkou setrvačností.

5.2.3 Počet náměrů v bodě

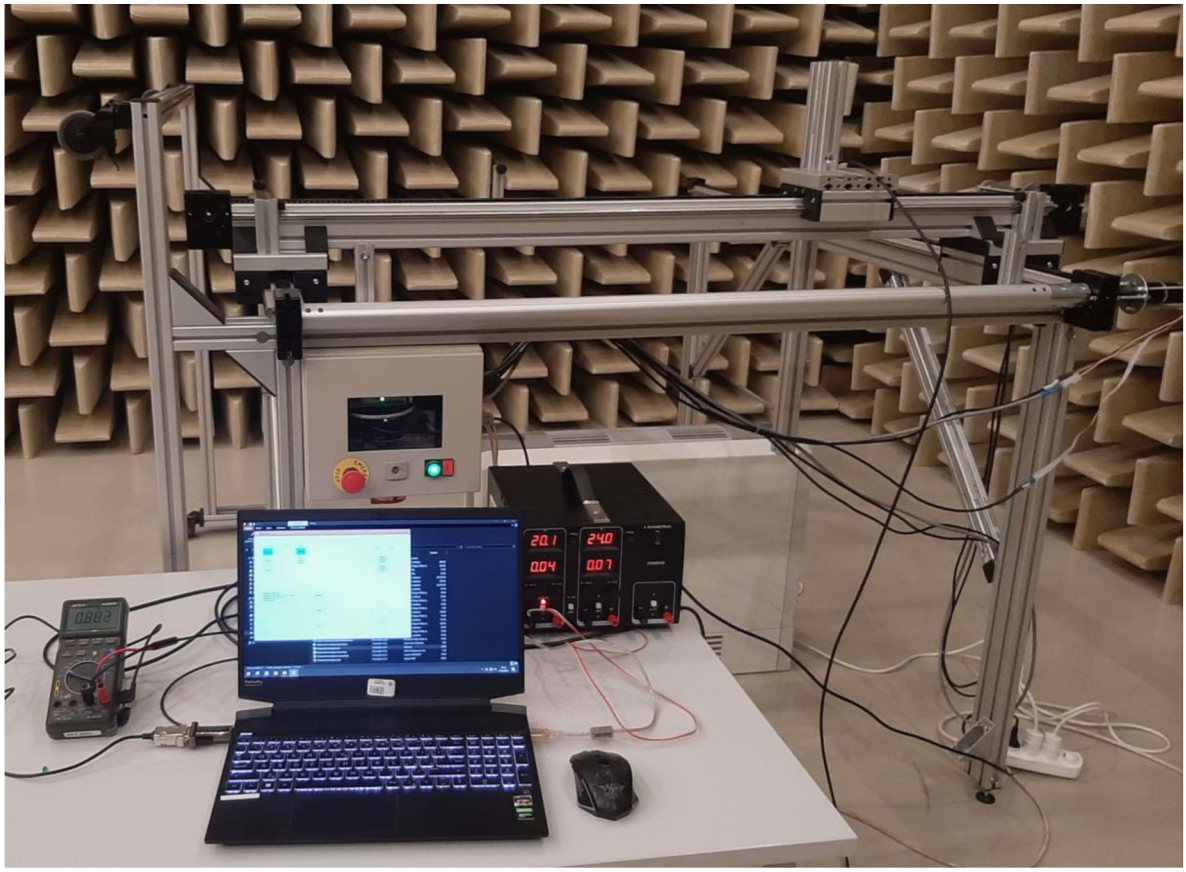
Počet náměrů v bodě nám říká, kolik náměrů se vezme na jednom měřícím bodě. Z těchto náměrů se automaticky dělá průměr. Tento parametr se hodí při velkém kolísání hodnot.

5.2.4 Vzdálenost měřicích bodů

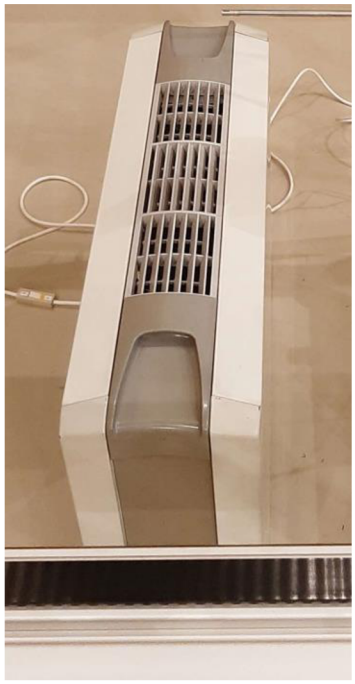
Vzdálenost měřicích bodů je vzdálenost mezi jednotlivými měřicími body v řádku a zároveň vzdálenost mezi body ve sloupci

6 Měření rychlosti proudění vzduchu

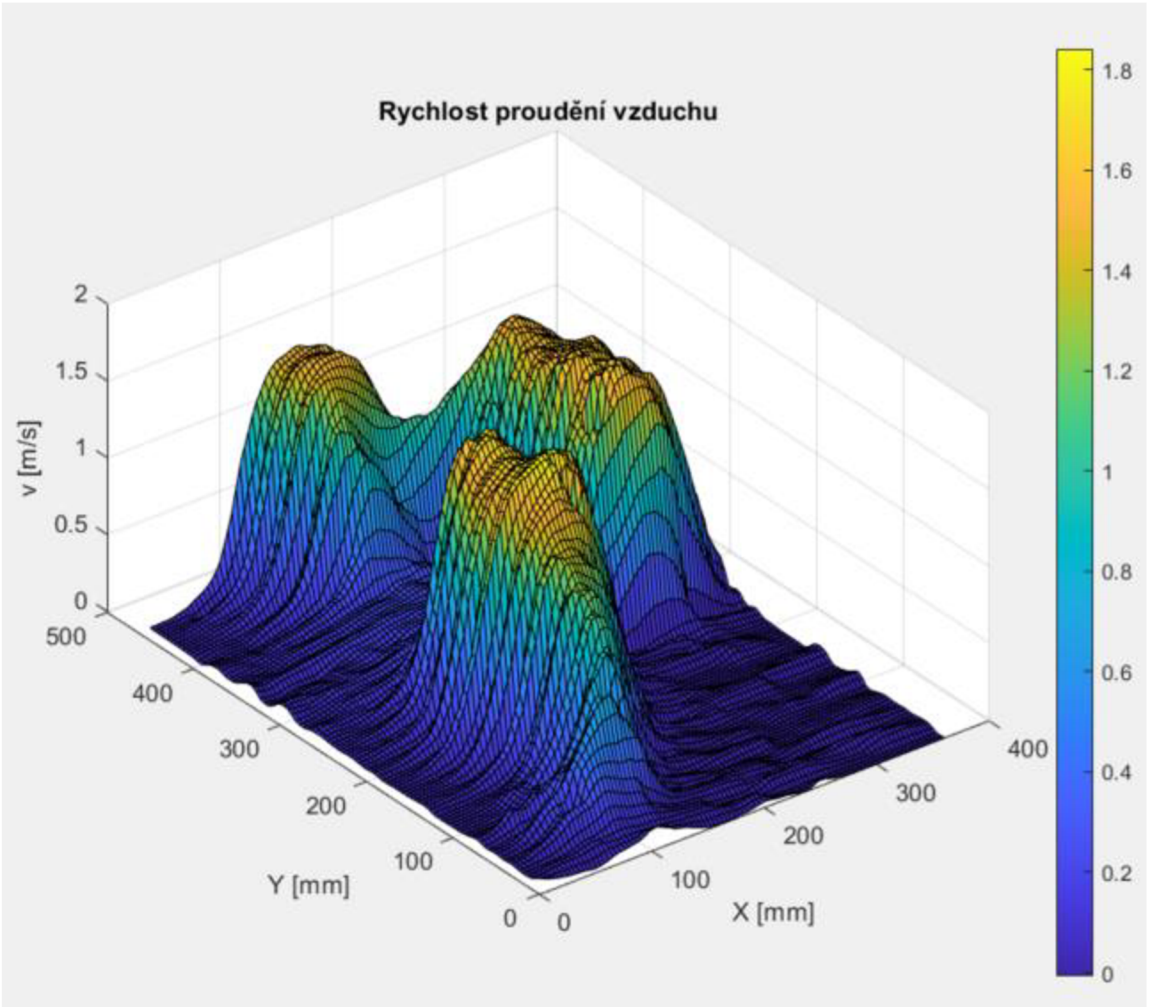
K ověření funkčnosti programu bylo vybráno měření rychlosti proudění vzduchu. Předmětem měření byly ventilační průduchy zařízení VUTS. Na pojízdný vozík byl namontován termo-elektrický anemometr od firmy Höntzsch, který je napájen laboratorním zdrojem s napětím 20 V. Následně bylo na sondě měřeno napětí v rozsahu 0–10 V, které odpovídá rychlosti 0–20 m/s. V programu byla nastavena vzdálenost měřících bodů 10 mm a pro velkou kolísavost hodnot bylo nastaveno na každém bodě vzít tři náměry, které nám program automaticky zprůměruje. Celkem bylo měřeno na cca 1700 bodech a měření trvalo kolem dvou hodin.



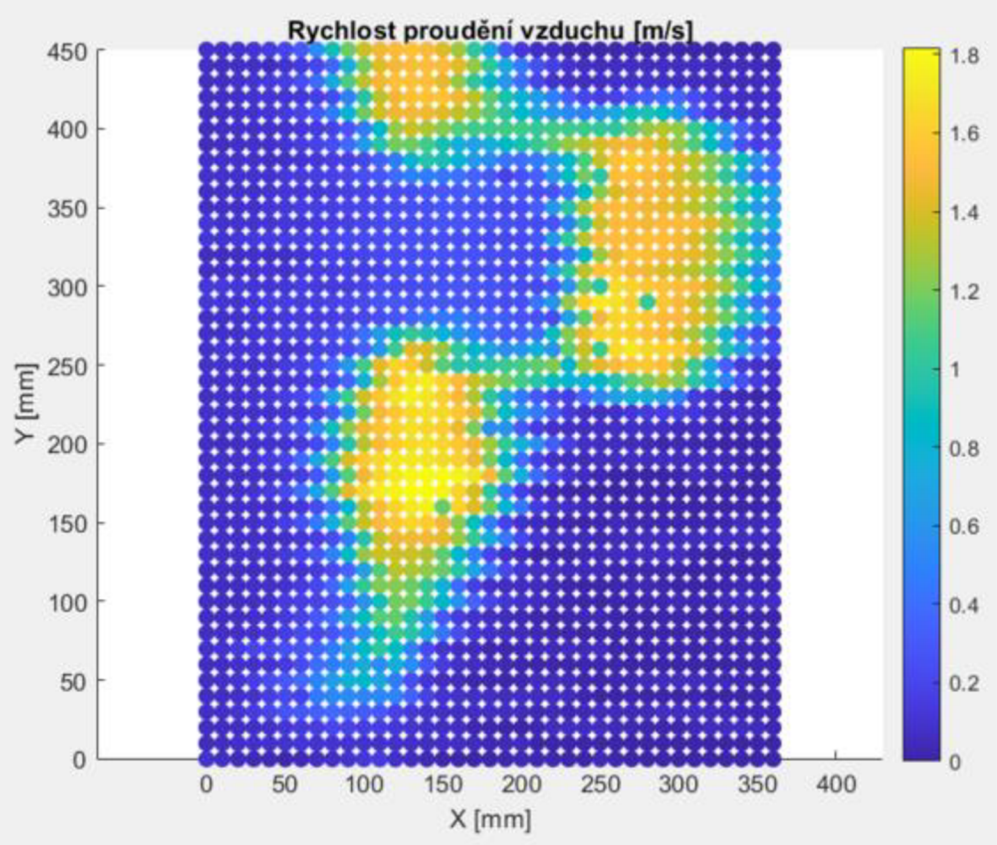
Obrázek 6.1 Foto měření



Obrázek 6.2 Měřené zařízení



Graf 6.1 Rychlost proudění vzduchu 3D



Graf 6.2 Rychlost proudění vzduchu 2D

Měření proběhlo bez problémů a naměřené hodnoty odpovídají očekávání. Z grafů je možné vyčíst, že senzor má velkou setrvačnost. Grafy byly vytvořeny v Matlabu.

Závěr

Firma VÚTS získává univerzální zařízení pro měření širokého spektra fyzikálních veličin v 2D poli. Program umožňuje nastavení výřezu z plochy, která je pro měření aktuální. Výstup naměřených hodnot je poskytnut v univerzálním textovém formátu, který lze zpracovat v jiném vhodném programu.

Zařízení bylo nejdříve vyzkoušeno starým ovládacím programem. Zjistila se chyba při rychlém pohybu v ose X. Měřením jednotlivých částí se přišel na špatný kontakt v konektoru enkodéru. Oprava představovala přepájení zmíněného konektoru na sériový konektor.

K polohovacímu zařízení byl vytvořen program na jeho ovládání. Program byl napsán v prostředí Visual Studia v jazyce C#. Pro ovládání řídících jednotek EPOS 24/5 byla stažena příslušná knihovna od výrobce Maxon. Program disponuje všemi potřebnými funkcemi a rozsáhlou nastavitelností parametrů automatického měření. Samotné měření veličin je zajištěno multimetrem Metex. Tento způsob umožňuje měřit jakoukoliv veličinu, kterou lze převést na napětí nebo proud.

Graficko-uživatelské rozhraní bylo vytvořeno použitím šablony Windows Forms. Při vytváření byl kladen důraz na jednoduchost a snadnou ovladatelnost. Program byl bez problémů převeden na jiný počítač a bylo provedeno kontrolní měření i na tomto počítači.

K ověření funkce a spolehlivosti zařízení byla změřena rychlost proudění vzduchu ventilačních otvorů. Měření trvalo několik hodin a proběhlo bez problémů. Naměřená data odpovídají předpokladu.

Všechny body zadání bakalářské práce byly splněny. V práci je možné pokračovat například rozšířením měřicích přístrojů o novější typy ovládané přes jiný typ rozhraní, než je RS232.

Použitá literatura

[1] MAXON MOTOR, *EPOS Command Library Documentation* [online] [cit. 2024-03-24]. Dostupné z: https://www.maxongroup.com/medias/sys_master/root/9157360353310/EPOS-Command-Library-En.pdf

[2] MAXON MOTOR, *EPOS 24/5 Jednotka řízení polohy: Dokumentace* [vydání duben 2004]

[3] MAXON MOTOR, *EPOS 24/5 Positioning Controller Documentation: Cable Starting Set* [vydání říjen 2005]

[4] MAXON MOTOR, *EC–max 40* [online] [cit. 2023-11-07]. Dostupné z: https://www.maxon-group.com/medias/sys_master/root/8882556567582/EN-21-253.pdf

[5] MAXON MOTOR, *RE 40* [online] [cit. 2023-11-07]. Dostupné z: https://www.maxongroup.com/medias/sys_master/root/8992314425374/EN-22-159.pdf

[6] MAXON MOTOR, *Planetary Gearhead GP 42 C* [online] [cit. 2023-11-07]. Dostupné z: https://www.maxongroup.com/medias/sys_master/root/8883965689886/EN-21-479.pdf

[7] MAXON MOTOR, *Encoder MR Type L* [online] [cit. 2023-11-07]. Dostupné z: https://www.maxon-group.com/medias/sys_master/root/8882781224990/EN-21-405-406-407.pdf

[8] MAXON MOTOR, *EPOS Firmware Specification* [online] [cit. 2024-03-24]. Dostupné z: https://www.maxongroup.com/medias/sys_master/root/8834321186846/EPOS2-Firmware-Specification-En.pdf

[9] METEX, *M-3850D Owner's manual* [online] [cit. 2024-02-18]. Dostupné také z: https://data.kemt.fei.tuke.sk/Meranie_BC/_materialy/Pristroje/Metex_M-3850D-1.pdf

[10] VŠCHT, *Měření počítačem pomocí multimetru Metex* [online] [cit. 2024-02-19]. Dostupné z: https://uprt.vscht.cz/ucebnice/LO/download/MT1-navod.pdf

[11] VÚTS, *Dvouosé lineární vedení.* Model, 2007

[12] PAPOUCH, *Sériový port RS232* [online] [cit. 2024-02-09]. Dostupné z: https://papouch.com/seriovy-port-rs232-p3740/

[13] WIKIPEDIA, *RS232* [online] [cit. 2024-02-09]. Dostupné z: https://cs.wikipedia.org/wiki/RS-232

[14] TZB–INFO, *Měření průtoku tekutin – principy průtokoměrů, Tepelný hmotnostní průtokoměr* [online] [cit. 2024-04-28]. Dostupné z: https://voda.tzb-info.cz/teorie-voda-kanalizace/4624-mereni-prutoku-tekutin-principy-prutokomeru

[15] PCE INSTRUMENTS, *Thermo–Anemometr* [online] [cit. 2024-04-29]. Dostupné z: https://www.pce-instruments.com/eu/measuring-instruments/test-meters/thermo-anemometer-kat_151876.htm

[16] POLLAK, ZBYNĚK, *Návrh a realizace dvouosého polohovacího zařízení pro měření zvukových polí*. DP TUL 2008.

[17] MICROSOFT LEARN, *SerialPort Class* [online] [cit. 2024-02-23]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.io.ports.serialport?view=net-8.0

[18] MICROSOFT LEARN, *Task Class* [online] [cit. 2023-12-10]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/api/system.threading.tasks.task?view=net-8.0

[19] IAMTIMCOREY, *C# Async / Await* [online] [cit. 2023-12-10]. Dostupné z: https://www.youtube.com/watch?v=2moh18sh5p4&list=LL&index=23&t=1129s&ab_channel=IAmTimCorey

[20] MATHWORKS, Surface plot [online] [cit. 2024-04-15]. Dostupné z: https://www.mathworks.com/help/matlab/ref/surf.html

[21] KOCOUREK, Petr, a kolektiv. *Číslicové měřicí systémy.* Vydavatelství ČVUT 1994. ISBN 80-01-01109-7