VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV VÝROBNÍCH STROJŮ, SYSTÉMŮ A ROBOTIKY

INSTITUTE OF PRODUCTION MACHINES, SYSTEMS AND ROBOTICS

MONITOROVÁNÍ VÝROBNÍ PŘESNOSTI

MONITORING OF PRODUCTION ACCURACY

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. ADAM JELÍNEK**

AUTHOR

VEDOUCÍ PRÁCE — **Ing., Dipl.-Ing MICHAL HOLUB, Ph.D.**

SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav výrobních strojů, systémů a robotiky
Student:	**Bc. Adam Jelínek**
Studijní program:	Strojní inženýrství
Studijní obor:	Výrobní stroje, systémy a roboty
Vedoucí práce:	**Ing., Dipl.-Ing Michal Holub, Ph.D.**
Akademický rok:	2019/20

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Monitorování výrobní přesnosti

Stručná charakteristika problematiky úkolu:

Monitorování a posuzování výrobní přesnosti je dnes nezbytnou součástí přesné a vysoce přesné výroby. U některých výrobních strojů jsou již nástroje pro monitorování součástí dodávky, ale v celé řadě případů vzniká požadavek až po dodávce zařízení a to se vznikem problému ve výrobním procesu. Pro vybraný CNC obráběcí stroj a výrobu navrhnou vhodný způsob monitorování a posouzení výrobní přesnosti.

Cíle diplomové práce:

Analýza současného stavu pro monitorování obráběcích centrech s řídicím systémem Siemens.
Systémový rozbor problematiky, návrh a zdůvodnění zvoleného způsobu řešení zadaného úkolu.
Plánování experimentu na vybraném CNC obráběcím stroji.
Statistické zpracování a vyhodnocení výsledků experimentu.
Vlastní závěry z oblasti navrženého postupu pro monitorování a vyhodnocení výrobní přesnosti.

Seznam doporučené literatury:

MAREK, J. a kol., MM Průmyslové spektrum: Konstrukce CNC obráběcích strojů IV. 2018. Speciální vydání. Dostupný z WWW: . ISBN 978-80-906310-8-3.

WECK, M., Brecher, Ch. Werkzeugmaschinen : Konstruktion und Berechnung. 2006. überarb. Auflage. Verlag Berlin Heidelberg : Springer, 2006. 701 s. ISBN 3-540-22502-1.

HOLUB, M. Geometric Accuracy of Machine Tools. In Measurement in Machining and Tribology. Springer, Cham, 2019. p. 89-112. ISBN: 978-3-030-03821-2.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2019/20

V Brně, dne

L. S.

doc. Ing. Petr Blecha, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Předložená závěrečná práce se zabývá implementací cloudových systémů pro monitorování a zvyšování výrobní přesnosti CNC výrobních strojů v kontextu průmyslu 4.0. Byla vytvořena demonstrační aplikace běžící v cloudu MindSphere firmy Siemens.

SUMMARY

This master thesis is about implementation of cloud systems for the purpose of monitoring and advancing production accuracy of CNC machine tools within context of Industry 4.0. A demonstration application was created. It is hosted in MindSphere cloud from the Siemens company.

KLÍČOVÁ SLOVA

Cloud, průmysl 4.0, monitorování, výrobní přesnost, CNC

KEYWORDS

Cloud, industry 4.0, monitoring, production accuracy, CNC

BIBLIOGRAFICKÁ CITACE

JELÍNEK, A. *Monitorování výrobní přesnosti*. Brno, 2020. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/125219. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav výrobních strojů, systémů a robotiky. Vedoucí práce Michal Holub.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, zpracoval jsem ji samostatně pod vedením Ing., Dipl.-Ing M. Holuba, Ph.D. a s použitím literatury uvedené v seznamu.

Bc. Adam Jelínek

PODĚKOVÁNÍ

Rád bych poděkoval svému vedoucímu Ing., Dipl.-Ing M. Holubovi, Ph.D. za jeho trpělivost, podporu a odborné rady.

Bc. Adam Jelínek

Obsah

1 Úvod

”Nowadays mechanical engineering means software development.”

—Bosh Rexroth [25]

Pro strojírenství je v posledních letech charakteristické zvyšování výrobní přesnosti pro účely vysoce přesných aplikací, které vychází z požadavků průmyslových oborů jako jsou automotive, energetika, letecký průmysl, aj. Ve všech těchto oborech jsou kladeny vysoké nároky na spolehlivost a účinnost a to vyžaduje velmi přesnou výrobu. K tomu přispívá i rozvoj konkurenčních ekonomik a s tím spojený tlak na efektivnost výroby českých, potažmo evropských podniků. Pokroky v konstrukci výrobních strojů způsobily, že se dnes pohybujeme na samotné hranici vyrobitelnosti a dalšího zpřesňování nelze dosáhnout bez nových nástrojů pro monitorování v kontextu digitálních technologií.

Tato práce se věnuje studiu cloudových systémů a jejich možnému využití pro monitorování výrobních strojů s cílem využít analyzovaná data pro softwarovou kalibraci strojů a tím zpřesnění výroby a také pro využití při návrhu nových verzí strojů.

Cloudové systémy, distribuované služby a webové aplikace získávají stále rychleji na důležitosti a jsou již dnes nasazovány v nejrůznějších odvětvích. Ve strojírenství se již dlouhou dobu hovoří o Industry 4.0, ale aplikace pro výrobní stroje mají ještě daleko k praktickému využití. Mají ovšem potenciál přispět k efektivnější a konkurenceschopnější výrobě.

V rámci této práce budou posouzeny možnosti monitorování výrobních strojů pomocí moderních technologií a vhodnost jejich využití. Bude navržena aplikace pro použití při měření geometrické přesnosti.

V první části se práce zabývá rešerší technologií v oblasti výrobní přesnosti a technologií relevantních pro digitální systémy. V další části je provedení měření geometrické přesnosti a zhodnocení postupu včetně návrhu na zlepšení. Třetí část se věnuje návrhu cloudové aplikace. V závěru je provedeno zhodnocení a návrh na další vývoj a výzkum.

2 Motivace

Ústředními tématy této práce jsou výrobní přesnost CNC strojů a cloudové technologie. Motivací pro toto téma je aktuální snaha o využití těchto technologií a to i na ÚVSSR, který je domovským ústavem autora. Na ústavu je v současné době v řešení několik projektů, kdy každý v určité míře buď již využívá cloudové technologie, nebo je jejich využití v plánu. Motivací pro autora je podílet se na rozvoji těchto technologií a přispět k řešení probíhajících projektů.

Cloudové systémy zažívají v posledních několika letech rychlý rozvoj a jejich použití se začíná pomalu rozšiřovat i do pole působnosti výrobních strojů. Průkopníkem cloudových systémů ve strojírenství je firma Siemens se svou platformou MindSphere. Ve velké míře je uvedená platforma vyvíjena a testována ve spolupráci s výzkumným ústavem Fraunhofer. Dílčí motivací je, aby ÚVSSR hrál významnou roli při aplikování zmíněných technologií.

Autor se hlásí na doktorské studium na témž ústavu a má záměr pokračovat na rozvoji řešení, navrženého a rozpracovaného v této závěrečné práci. Téma cloudových systémů je rozsáhlé a vytvoření komplexního řešení pro výrobní stroje je nad rámec této práce. Avšak cílem a motivací je zahájit činnost v této oblasti, provést počáteční průzkum terénu a nastínit vhodný směr pro pokračování v dalším výzkumu.

3 Přehled současného stavu poznání

”As a general rule the most successful man in life is the man, who has the best information.”

—Benjamin Disraeli [9]

3.1 Definice a pojmy

Název této práce je ”Monitorování výrobní přesnosti”. Je vhodné si pojmy z názvu rozebrat a definovat. To pomůže k pochopení cílů práce a návrhu řešení.

3.1.1 Monitorování

Význam slova z akademického slovníku: ”několikastupňový víceúčelový informační systém, činnost, kt. na základě systematického pozorování, měření a analýz současného stavu objektu (např. přírodního prostředí znečištěného exhalacemi) předpovídá jeho budoucí vývoj” [12].

Z této definice v akademickém slovníku vyplývají podstatné charakteristiky monitorování. Těmi jsou informační systém, systematické pozorování a predikce budoucího vývoje. Informační systém je v této definici chápán šířeji. Není to systém informační ve smyslu IT systému, ale jakýkoli systém, který je:

1. schopen získávat informace,
2. požadovaným způsobem je zpracovat / transformovat a
3. je zaznamenat.

To jsou tři hlavní stupně informačního systému — proto je definován jako několikastupňový. Technická, obecněji fyzická povaha získání informace závisí na povaze požadované informace. Za příklad uveďme lidskou řeč. Jedná se o zvuk, tedy o mechanické vlnění přenášené prostředím, většinou vzduchem. K získání (odposlechnutí) zvukové informace slouží mikrofony. Zařízení, jehož membrána se pohybuje ve smyslu vlnění a svým pohybem cívkou indukuje elektrický proud. Membrána je v tomto systému úrovní získání informace. Soustava s cívkou už informaci transformuje na elektrický analogový signál. Zbývá úroveň zaznamenání informace. Mikrofon se dá připojit na digitálně-analogový převodník (další transformace) a dále do počítače, kde informaci teď už v digitální podobě uložíme. Za jiný příklad lze uvést člověka samotného. Ten zvukovou informaci získává pomocí sluchového ústrojí, pomocí kterého ji také transformuje na elektrický signál a zaznamená je do mozkové paměti nebo třeba na papír písmem.

Poznámka k definici v normě ISO 9000:2015. Ta definuje pojem monitorování takto: ”určování stavu systému, procesu, produktu, služby nebo činnosti”. Zdá se, že tato norma vymezuje monitorování jako zjišťování v jediném časovém okamžiku — zjišťuje, určuje současný

stav systému. To je v rozporu s definicí přijatou výše a autor je toho názoru, že definice z normy proto není vhodná.

3.1.2 Systém

Odbočka k pojmům systém a soustava. Autor považuje za vhodné krátce se vyjádřit k těmto dvěma pojmům, neboť jsou oba v této práci často používány a přitom pro jejich definici není všeobecná shoda.

Když nahlédneme na definice ve slovnících, zjistíme, že oba pojmy jsou synonyma. Alespoň tedy v širokém rámci pravidel českého jazyka. Definice ve všech těchto slovnících pojmy zaměňují, přičemž soustavu považují za řídčeji používanou. Definice z akademického slovníku: ”soubor jednotlivin navzájem spojených urč. strukturou, sítí vztahů v uspořádaný celek” [13]. Poměrně obecná definice. V tomtéž slovníku dokonce následující definice: ”komplex dílčích prvků tvořících urč. (technické aj.) zařízení”. A právě tady leží častá námitka.

Někdy je mezi těmito pojmy rozlišováno tak, že soustava je množina fyzických prvků, obšírně řečeno takových, na které si lze sáhnout a systém je potom formální popis této soustavy, který na ni není fyzicky nijak vázaný. Obvykle je přitom formálním popisem myšlen matematický model.

Dalším vodítkem budiž norma ISO 9000:2015, která se zabývá vymezením pojmů pro (systém) managementu kvality. V odstavci 3.5.1 je systém definován takto: ”soubor vzájemně provázaných nebo vzájemně působících prvků”. Definice zavádí pojem obecně a implicitně z ní vyplývá, že pod pojmem systém zahrnuje i potenciálně odlišný pojem soustava. Není to však možná úplně patrné. Pomůže následující definice 3.5.2, která definuje pojem infrastruktura takto: ”systém vybavení, zařízení a služeb potřebný pro provoz organizace” [16]. Na této definice lze vidět, že norma skutečně definuje systém jak pro množinu formálních prvků, tak pro fyzické soustavy.

Dalším vodítkem může být vztah k zahraniční literatuře. V anglickém jazyce se totiž tyto pojmy nerozlišují a systém (anglicky system) vystupuje v obou významech. Při citování, překládání, nebo zkrátka jen odkazování se na zahraniční zdroj tedy taktéž není vhodné pojmy rozlišovat. Pro přesnost, v anglickém jazyce je možné povahu systému rozlišit pomocí přídavného jména a užitím pojmů abstract system a real system. V praxi se to však téměř nepoužívá.

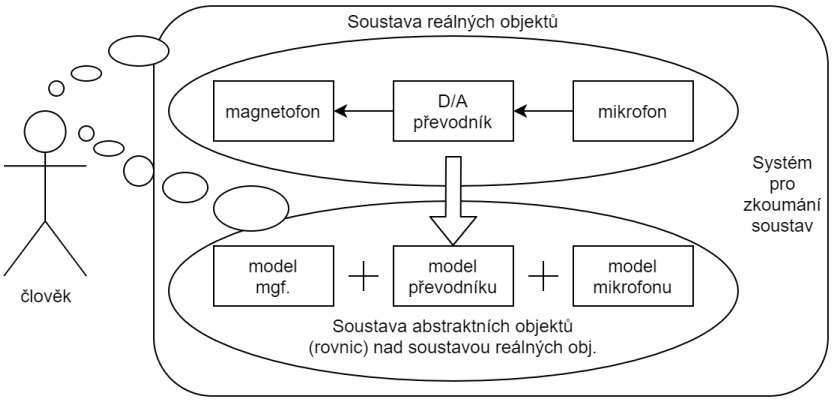
Systémová metodologie
Na ÚVSSR při FSI na VUT je v čele s prof. Dr. Ing. Markem Ph.D., DBA rozvíjena teorie systémové metodologie. Komplexně je tato problematika vysvětlena v knize Expertní inženýrství v systémovém pojetí [17]. V kontextu systémové metodologie je pro abstraktní i reálnou množinu objektů použit termín soustava. Rozlišení je potom v případě potřeby možné takto: soustava reálných objektů a soustava abstraktních objektů. Systém je potom označení pro způsob zkoumání obou soustav. Můžeme říci, že systém je vlastně metodologie toho, jak pracovat se soustavami.

Lze potom říkat: Existuje systém pro zkoumání této soustavy. Tento stroj, jakožto soustava reálných objektů, byl popsán soustavou rovnic a byl vytvořen systém pro analýzu zkoumaného problému.

Systém je tedy množina úvah, postupů a pravidel pro práci se soustavami, které mohou být abstraktní i reálné povahy. Souvisí to s tzv. systémovými vlastnostmi, které jsou definovány ve stejné literatuře. Systémovost je vlastnost, která vyjadřuje, zjednodušeně řečeno, možnost být zahrnut do systému, tedy být zkoumán systémově.

Provázanost soustav a systému je ukázána na následujícím obrázku, který byl vytvořen s laskavým svolením prof. Marka po konzultaci s ním.

Obrázek 3.1: Vztah mezi soustavami a systémem



Autor se přiklání k používání pojmů způsobem systémové metodologie. Avšak s výjimkami, kde by to způsobovalo nesoulad s původními zdroji. Příkladem takových zdrojů jsou průvodní materiály výrobců. Pokud výrobce nazývá své zařízení jako systém, potom bude v textu toto označení zachováno. V případech, kdy bude účelné rozlišit mezi abstraktními a reálnými soustavami, bude tak učiněno právě tímto explicitním způsobem. Pro strojní konstrukční uzly jako i další běžné strojní soustavy bude používáno výlučně označení soustava. Jako příklad: soustava šroubu a matice, soustava lineárního vedení, atd.

3.2 Výrobní přesnost

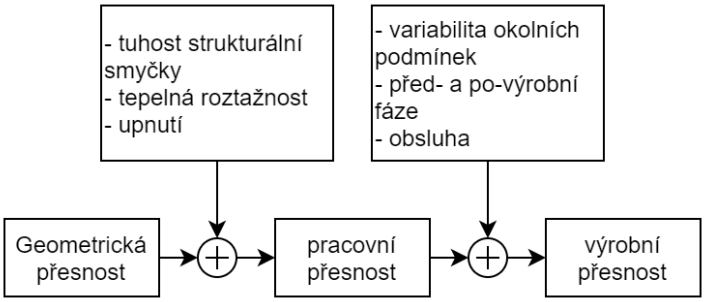
3.2.1 Definice

Schopnost dlouhodobě dosahovat shody mezi jmenovitou hodnotou rozměru resp. jmenovitým tvarem součásti a skutečnou hodnotou rozměru resp. skutečným tvarem dané součásti. Je to charakteristika výroby (nejen výrobního stroje) a je posuzována dlouhodobě [10]. Je charakteristikou výroby na daném stroji za rozsahu podmínek za určité období. Lze jí popsat, jakých kvalit dosáhla výroba za určené období, například čtvrtletí, při působení podmínek okolního prostředí jako jsou teplota, tlak, vlhkost. Není (inherentní) charakteristikou stroje. Neslouží k určení aktuální kondice výroby ani stroje.

3.2.2 Způsoby hodnocení výrobní přesnosti

Rozlišují se dva způsoby zjišťování stavu přesnosti, resp. rozlišujeme dva metrologické přístupy. Jsou to měření přímé a nepřímé. Mezi metody přímé se řadí měření geometrické přesnosti po-

Obrázek 3.2: Návaznost přesností



mocí laser-interferometru, ballbaru, aj. Nepřímé metody aproximují přesnost pomocí měření na vyrobených dílech.

Výrobní přesnost je charakteristika výroby a dá se proto hodnotit pouze metodami nepřímými. Nelze dělat závěry o výrobní přesnosti pouze na základě přímých metod. Je to z toho důvodu, že do výroby dílu vždy spadají dodatečné vlivy, které na samotnou geometrickou přesnost vliv nemají. Tyto vlivy jsou vysvětleny níže.

Nelze také hodnotit výrobní přesnost pouze na základě výroby jediného kusu a dokonce ani jedné výrobní dávky. Měřením jediné dávky se určuje pracovní přesnost.

3.2.3 Faktory ovlivňující výrobní přesnost

Mezi vlivy ovlivňující výrobní přesnost se řadí všechny procesy, které probíhají v interakci s vyráběnou součástí v průběhu výrobního procesu a po něm v rámci navazujících procesů před hodnocením výrobní přesnosti mimo výrobu na stroji. Kromě samotné výroby na stroji, jsou to fáze před výrobou a po výrobě.

Fáze před strojní výrobou zahrnuje všechny procesy v podniku, které vedou k zahájení strojní výroby na polotovaru. Jsou to především vstupní kvalita dílu. Logistika a uskladnění dílu před zahájením strojní výroby, kdy například délka uskladnění před zahájením obráběcích operací může způsobit změny v materiálu v důsledku stárnutí odlitku, při kterém dochází k pomalému uvolňování vnitřního pnutí. Tyto chyby mají systematický charakter 3.7.1.

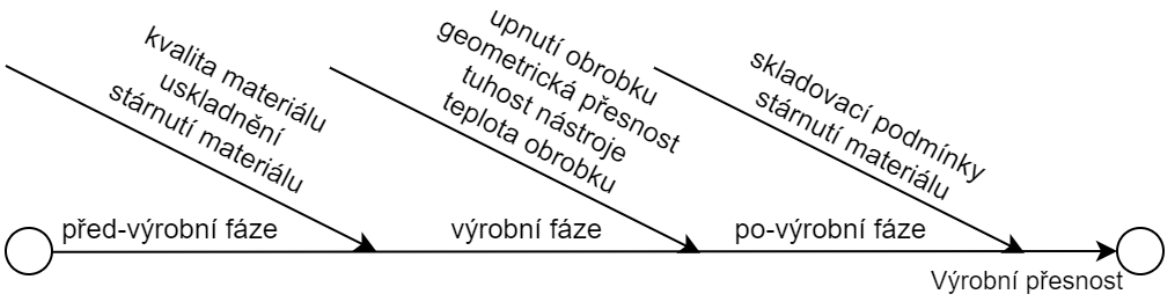
Výrobní fáze má na celkovou výrobní přesnost zásadní vliv. Výrobní fáze začíná okamžikem upnutí obrobku na pracovní stůl stroje a končí po dokončení všech technologických operací. Přesnost dílu je v této fázi ovlivněná několika významnými faktory. Jedním je samotné upnutí obrobku, které jej vždy v nějaké míře deformuje. Technologické operace potom probíhají na deformovaném obrobku a po jeho odejmutí ze stolu dochází k odeznění účinků deformace a tím se mění tvarové i rozměrové charakteristiky vyrobeného dílu. Upnutí má tedy nezanedbatelný vliv. Charakter chyb od upnutí je systematický v případě použití automatických upínacích soustav, ale náhodný při upínání obsluhou 3.7.1. Dalším faktorem je samotná geometrická přesnost stroje, která zásadním způsobem ovlivňuje výslednou přesnost. Dalším vlivem je tuhost soustavy stroje-nástroje-obrobku. Při obrábění dochází k deformaci všech tří objektů v důsledku sil od obrábění. Tuhost této strukturální smyčky určuje, v jaké míře bude výsledná přesnost ovlivněna od těchto sil. Dalším významným vlivem jsou okolní podmínky, především teplo, které způsobuje roztažnost a v důsledku deformaci všech prvků strukturální smyčky. Teplo vstupuje jako faktor jak do výrobní přesnosti (teplo od ročního období na halách bez klimatizace, které působí na materiál ve všech fázích výroby), tak i do geometrické přesnosti, protože se v jeho

důsledku deformuje samotný výrobní stroj a mění se tak jeho geometrická přesnost. V případě, že upínání probíhá ručně obsluhou, i obsluha se počítá jako samostatný faktor ovlivňující výrobní přesnost. Souvisí to i dlouhodobou povahou výrobní přesnosti, protože může například dojít k personální změně na pozici obsluhy a tím bude ovlivněna výrobní přesnost.

Fáze po strojní výrobě zahrnuje všechny procesy v podniku před tím, než dojde k hodnocení přesnosti výroby. Například obrobky mohou být opět ponechány přirozenému stárnutí, v důsledku čehož dojde ke změnám geometrických a tvarových charakteristik. Může tedy být rozdíl, zda se obrobky měří ihned po odebrání ze stroje, nebo až po nějaké době uskladnění.

Výrobní přesnost zahrnuje všechny tři fáze, neboť je hodnocením schopnosti podniku vyrábět v požadované kvalitě a na tu mají vliv procesy ve všech třech fázích.

Obrázek 3.3: Faktory ovlivňující výrobní přesnost



3.2.4 Normalizovaná výrobní přesnost

Cílem podniku je zvyšovat výrobní přesnost a snižovat její variabilitu v čase a vůči různým podmínkám. V ideálním případě jsou zcela eliminovány chyby přesnosti ve před-výrobní a po-výrobní fázi a dále jsou zcela eliminovány všechny chyby od obsluhy při výrobní fázi.

Ze statistického hlediska jde o snahu eliminovat chyby náhodného charakteru (změnou systému na chyby systematické) a snižovat velikost systematických chyb.

Příkladem jak se lze limitně blížit tomuto stavu je několik předpokladů pro výrobu. Obecně je snaha o vysokou opakovatelnost ve všech činnostech a dále snaha o eliminaci obsluhy z výrobního procesu. Konkrétně se může jednat o nahrazení obsluhy manipulátory, roboty, automatickými upínacími soustavami. Zajištění stabilního okolního prostředí zatemněním okem, použitím klimatizací, eliminací zdrojů tepla. V takovém případě bude opakovatelnost výroby vysoká. V ideálním případě 100%.

S využitím následující definice z akademického slovníku: ”Normalizace je technická činnost směřující ke sjednocení, zjednodušení a zhospodárnění konstrukce a výroby, popř. k dodržení jakosti výrobků.” [14] autor definoval takto omezenou výrobní přesnost jako normalizovanou výrobní přesnost.

Normalizovaná výrobní přesnost je tedy taková výrobní přesnost, kdy již byly vyčerpány všechny možnosti pro snížení vlivů od obsluhy a od okolí na tuto výrobní přesnost a současně bylo dosaženo vysoké opakovatelnosti výrobní přesnosti. Pak tedy normalizovaná výrobní

přesnost je (limitně se blíží) integrálem geometrické přesnosti, vlivů od statické a dynamické poddajnosti strukturální smyčky a vlivů od působení tepla.

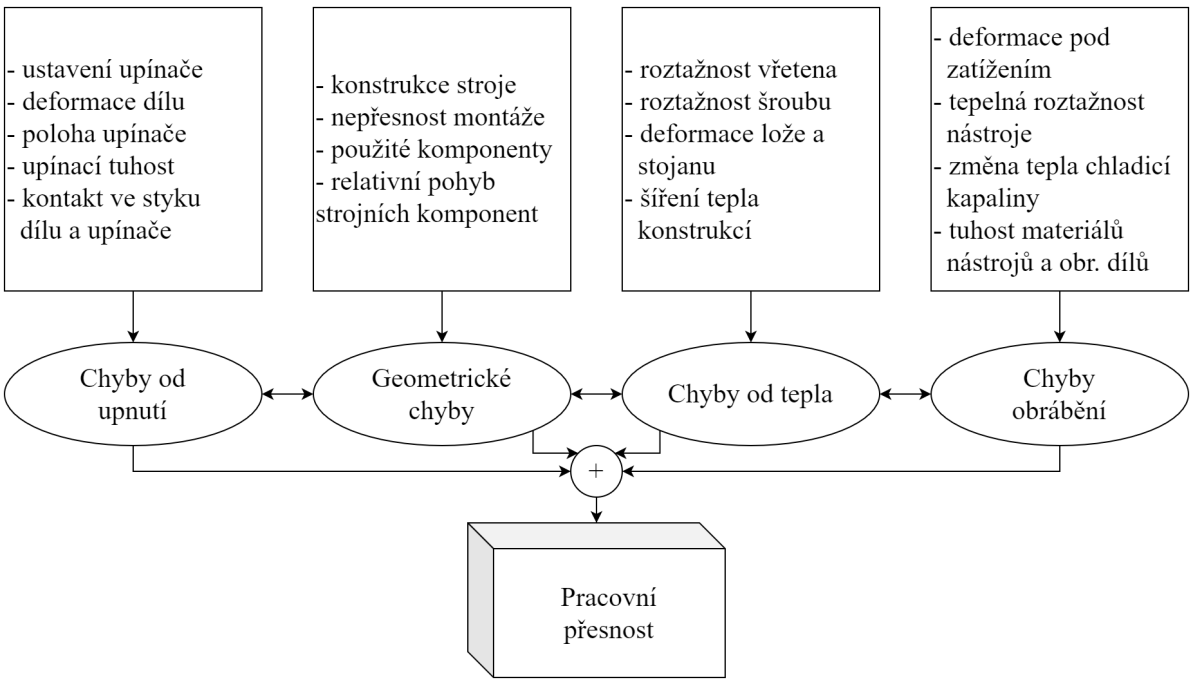
Užití této definice se může jevit jako nadbytečné, umožňuje však omezení se na ty podniky, příp. výroby, kde již byly aplikovány kroky, činnosti předpokládané v dalším textu.

3.3 Pracovní přesnost

Pracovní přesnost je, podobně jako výrobní přesnost, charakteristika výroby na stroji, nikoli stroje samotného. Posuzuje se tedy také nepřímými metodami na vyrobených obrobcích. Nicméně na rozdíl od výrobní přesnosti se neposuzuje dlouhodobě, ale na základě jediné série, někdy označované jako prototypové. Při této sérii se zpravidla nastaví podmínky blízké jako při následné sériové výrobě.

Pracovní přesnost v sobě tedy zahrnuje přesnost geometrickou a všechny chyby charakteristické pro technologické operace, nezohledňuje však variaci dalších vlivů jako je změna okolní teploty v důsledku jiného ročního období, aj. Její hodnota je tedy spolehlivá v případě, že je dlouhodobá opakovatelnost výroby vysoká, jinak řečeno, když je výrobní přesnost normalizovaná. V opačném případě však nemá adekvátní vypovídací hodnotu pro dlouhodobý charakter výroby.

Obrázek 3.4: Faktory ovlivňující normalizovanou výrobní přesnost [23]



3.4 Geometrická přesnost

3.4.1 Definice

Geometrická přesnost je tvarová a rozměrová přesnost stroje, popisuje jeho geometrickou strukturu. Je charakterizována těsností shody mezi funkčním bodem (nástrojem) a obrobkem při pohybu strojních os v nezatíženém stavu (tj. bez silových účinků od technologických operací, příp. podle normy ISO 230–2 za podmínek dokončovacích operací). Pokud není geometrická přesnost 100 %, pak při pohybu jedné, nebo více strojních os, dochází k relativnímu pohybu mezi funkčním bodem a obrobkem. Jinak formulováno: Geometrická přesnost vyjadřuje schopnost stroje neodchýlit se od jmenovité polohy funkčního bodu, tj. při uvažování ideálního tvaru a geometrie stroje [10] [18].

Geometrická přesnost je charakteristikou stroje samotného a nikoli výroby na tomto stroji. Je obecně proměnná v čase a tudíž je okamžitou charakteristikou. Nelze ji tedy změřit a prohlásit tuto hodnotu za trvale platnou, nýbrž je potřeba ji monitorovat.

Ověřuje se přímými metodami, tj. měřicími zařízeními na stroji. Nikoli nepřímo měřením na obrobcích. Nicméně lze ji nepřímým měřením za určitých podmínek částečně aproximovat.

3.4.2 Faktory ovlivňující geometrickou přesnost

Celková geometrická přesnost je součtem vlivů od těchto chyb:

- Chyby lineárního polohování
- Chyby přímosti
- Úhlové chyby
- Chyby kolmosti rovin

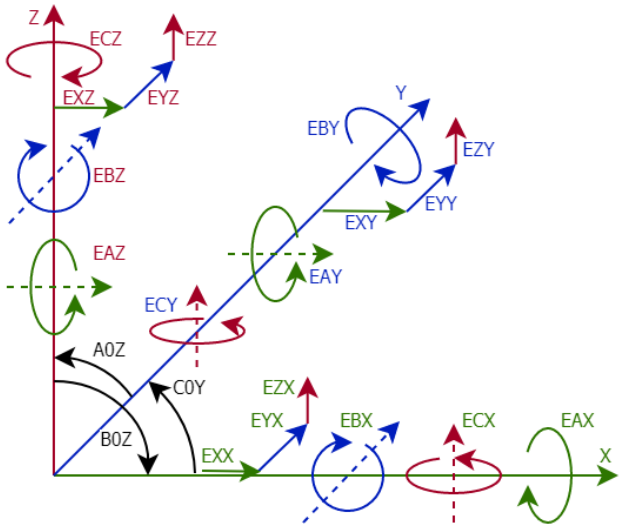
Znázornění geometrických chyb
Pro popis geometrických chyb se používá pravoúhlý souřadný systém.

Bod v prostoru má 6 stupňů volnosti (translace ve třech osách a rotace kolem tří os). Taktéž osa má v prostoru 6 stupňů volnosti. Tři osy potom dohromady 18 st. v. K tomu se přičítají ještě vzájemné kolmosti rovin. Dohromady tedy celkově 21 geometrických chyb.

Značky jednotlivých chyb se získají následujícím způsobem: Označení se skládá ze tří, nebo čtyř písmen. Chyby os začínají písmenem E (Error). Poslední písmeno značí, vůči které ose je popisována chyba. Prostřední písmeno značí směr odchylky. Například E_{CY} je chyba ve směru C (tedy rotační) vůči ose Y, ”ohnutí osy Y kolem osy Z”. V případě chyb kolmostí se mezi písmena značící referenční osu a směr úchylky přidává ještě číslice 0. Označení potom vypadá například takto: EB0X. To je tedy chyba natočení osy Y ve směru osy X. V případě kolmostí se pak počáteční písmeno konvenčně vynechává. Chyba se potom zapíše pouze B0X.

Chyby lineárního polohování
Jsou to chyby E_{XX}, E_{YY} a E_{ZZ}. Jsou zahrnuty v definici geometrických chyb, protože mají vliv na těsnost shody mezi polohou nástroje a obrobku. Avšak mají jiný mechanický původ než ostatní geometrické chyby. Mají proto zvláštní význam a budou blíže popsány v další kapitole.

Obrázek 3.5: Znázornění geometrických chyb



Chyby přímosti
E_{YX}, E_{ZX}, E_{XY}, E_{ZY}, E_{XZ}, E_{YZ}. Chyby přímosti jsou způsobeny převážně nedokonalým tvarem pohybových os a deformací rámu. Lze je softwarově kompenzovat.

Úhlové chyby
E_{AX}, E_{AY}, E_{AZ}, E_{BX}, E_{BY}, E_{BZ}, E_{CX}, E_{CY}, E_{CZ}. Jsou chyby natočení. Převládající chyby a jejich zdroji odvisí od konstrukčního provedení stroje. U řešeného CNC jsou převládající chyby od naklopení vřetena, tedy E_{AZ} a E_{BZ} (E_{CZ} nemá na vřetenu význam). Tyto chyby se nedají softwarově kompenzovat, pouze mechanicky seřídit.

Chyby kolmosti rovin
E_{AOZ}, E_{BOZ}, E_{COZ}. Chyby kolmostí mohou být způsobeny špatným tvarem rámu stroje. Často se jedná o chybu způsobenou nekolmostí stojanu vůči loži.

3.4.3 Chyby lineárního polohování

Chyby typu E_{XX}, E_{YY}, E_{ZZ} jsou chyby lineárního polohování. Je to chyba osy ve vlastním směru. Pokud je tato chyba nulová, znamená to, že požadovaného bodu na ose lze dosáhnout beze zbytku. Pokud je nenulová, znamená to, že se do daného bodu nelze dostat. Pozice zůstane o velikost chyby polohování před, nebo za tímto bodem. Chyby polohování tedy charakterizují schopnost stroje najet do zvolené polohy. Najíždění do poloh je úlohou posuvových soustav. Tím se tyto chyby liší od ostatních geometrických, které nejsou výsledkem činnosti stroje při jeho provozu, ale jsou dány strukturou stroje jako takovou.

Zdroj chyb lineárního polohování
Zkráceně jen polohování je výsledkem činnosti posuvových soustav. Ty jsou na stroji zdrojem

pohybu lineárních os. Skládají se z náhonu, vedení, odměřování, krytování a mazání. Posuvové soustavy jsou řešeny jako zpětnovazební smyčky. Chyby polohování jsou dány tím, že servomotor řízený přes PID regulátor, má omezenou přesnost natočení šroubu. Současně odměřovací soustava má omezené rozlišení a tedy informace o poloze není přesná.

Odměřovací soustavy se skládají z pravítka a enkodéru. Enkodér je na řešeném stroji fotoelektrický a odměřování je inkrementální. Pravítko, které je zdrojem informací o poloze má obecně jinou tepelnou roztažnost než lineární vedení, na které je uchyceno a než rám stroje. V důsledku toho dochází při tepelném působení na stroj ke zkreslení (v ideálním případě ke škálování) měřítka. Resp. dochází k neshodě mezi skutečnou délkou osy a délkou pravítka a tedy odečítanou délkou osy.

Z toho však plyne jedna výhoda a totiž, pokud je při tepelném namáhání zjištěna skutečná délka osy, dá se chyba polohování kompenzovat touto hodnotou. Tohoto principu se používá ke kalibrování pomocí systému měření laser-interferometrem.

Z výzkumu Ing., Dipl.-Ing M. Holuba, Ph.D. i jiných vyplývá, že chyby polohování jsou v největší míře způsobeny tepelným namáháním stroje [10].

Především u strojů pro velké obrobky, které mohou mít délku až 30 m, jsou chyby polohování významnější než u malých strojů. Protože na ÚVSSR při FSI na VUT jsou v projektovém řešení i tyto velké stroje a autor je součástí projektu Level-up [33], který se na velké stroje přímo zaměřuje, a současně protože chyby polohování lze efektivně kompenzovat, zvolil autor lineární polohování jako primární charakteristiku pro řešení v rámci této práce. Navrhovaný monitorovací systém tedy bude otestován právě na monitorování lineárního polohování. Další geometrické chyby a navazující charakteristiky výrobní přesnosti mohou být do řešení implementovány v průběhu dalšího výzkumu a činností na ústavu.

3.4.4 Kvazi statické podmínky

Norma ISO 230–1 připouští podmínky pro měření geometrické přesnosti za ”kvazi statických podmínek”, přičemž se tím myslí za podmínek obrábění dokončovacích operací, pro které jsou charakteristické statické silové účinky, tj. neměnné v čase. Příkladem je statické zatížení vřetena a stojanu velmi hmotným nástrojem. Nicméně v normě není přesně vymezeno, jaký je silový rozsah pro tyto podmínky. Jinak řečeno, není přesně stanoveno, do jaké velikosti výsledné síly, jsou podmínky považovány ještě za kvazi statické. V této podkapitole je na tuto otázku hledána odpověď.

V práci z roku 1995 Spaan uvádí, že síly od dokončovacích operací, resp. jimi indukované chyby pracovní přesnosti, jsou zanedbatelné: ”During the finishing process the static component of the cutting forces on the deflection of the structural loop is negligible due to the relatively high static stiffness of modern machine tools.” [27]. Odkazuje se přitom na další dobové práce. Lze dovozovat, že tehdejší stav techniky výrobních strojů byl na takové úrovni, že tyto síly skutečně bylo možné považovat za zanedbatelné.

V dnešní době lze dohledat mnoho prací, které se zabývají výzkumem odezvy na síly při dokončovacích operacích. Zaměřují se přitom na zjišťování tuhosti nástroje, obrobku a upínání obrobku, případně vlivu na tuhost celého stroje. Např. Szipka K. a spol. použili při svém experimentu takové podmínky, kdy výsledná síla při dokončovacích operacích se pohybovala mezi 400 a 700 N [31]. Zjistili přitom statickou tuhost stroje cca 10 N/µm.

Spíše než definovat konkrétní rozsah výsledných sil pro dokončovací operace, jeví se jako vhodnější provést experiment na konkrétním stroji s cílem určit velikost odezvy tuhosti a zjistit tak maximální podmínky, které lze na daném stroji považovat za kvazi statické.

3.5 Technické prostředky pro hodnocení geometrické přesnosti

Geometrická přesnost ze zjišťuje pomocí specializovaných přístrojů, přičemž každý z nich je zaměřen vždy jen na určitou podmnožinu geometrických chyb. Žádný přístroj neumí změřit všechny chyby.

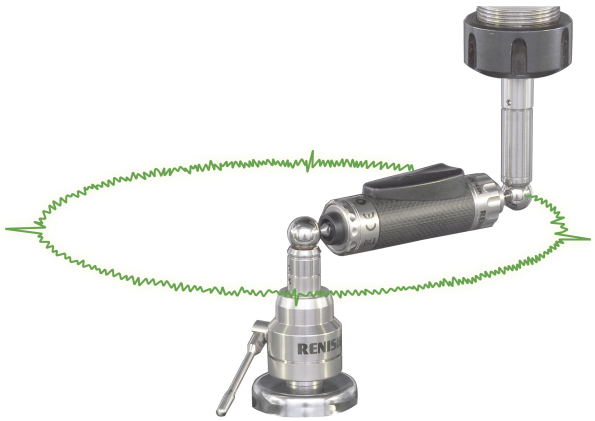
Přístroje rozlišujeme mechanické, elektronické a optické. Mezi mechanické patří libely, vodováhy (mohou být i elektronické). Práci s daty z mechanických přístrojů nelze snadno automatizovat, proto se jimi v této práci nebudeme zabývat.

Protože vybavení školní laboratoře zahrnuje přístroje převážně od firmy Renishaw, budou rozebrány přístroje od této firmy.

3.5.1 Ballbar QC20-W (Renishaw)

Ballbar je systém (pozn. označení systém používá výrobce) ke zjišťování některých geometrických chyb, především chyb kolmosti. Systém je tvořen několika součástmi. Jsou to především: lineární snímač, dva magnetické držáky a software.

Obrázek 3.6: Ballbar QC20 — Renishaw



Pricip činnosti
Měření lineární interpolace dle ISO 230–4. Jeden magnetický držák je umístěn do vřetena, druhý na pracovní stůl. Držáky jsou od sebe axiálně vzdáleny o zvolený průměr měření, tedy například o 150 mm. Mezi držáky je umístěn lineární snímač. Snímač komunikuje s měřicím softwarem přes bezdrátovou technologii Bluetooth. Je naprogramován takový posuv CNC stroje, aby držáky vůči sobě konaly relativní kruhový posuv. Snímač při tomto pohybu měří radiální odchylky od jmenovitého poloměru. Výsledky jsou vyhodnoceny dle normy ISO 230–4.

Tabulka 3.1: Parametry měřidla Ballbar QC20-W

Rozlišení senzoru	0,1 µmm
Přesnost měření	± (0,7 + 3e-3·L) µmm
Rozsah měření	± 1 mm
Rozsah měřidla	— 1,25 až 1,75 mm
Max. vzorkovací frekvence	1000 Hz
Rozsah operační teploty	0 až 40 °C

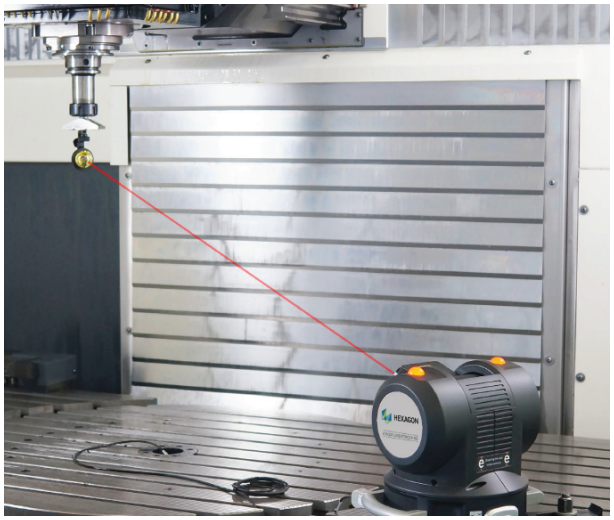
3.5.2 Laser-interferometer XL-80 (Renishaw)

Systém XL-80 (pozn. opět jde o označení výrobce) je laserový interferometr. Jedná se o soustavu optických dílů, interferometru a softwaru, která se používá k měření chyb lineárního polohování. Detailní popis je uveden v kapitole 4.1.2

3.5.3 Laser-Tracer NG (Etalon)

Laser-Tracer je řešení pro kompenzaci, kalibraci a verifikaci geometrické přesnosti CNC a CMM strojů. Jedná se o řešení od německého výrobce Etalon GmbH, který je součástí koncernu Hexagon AB.

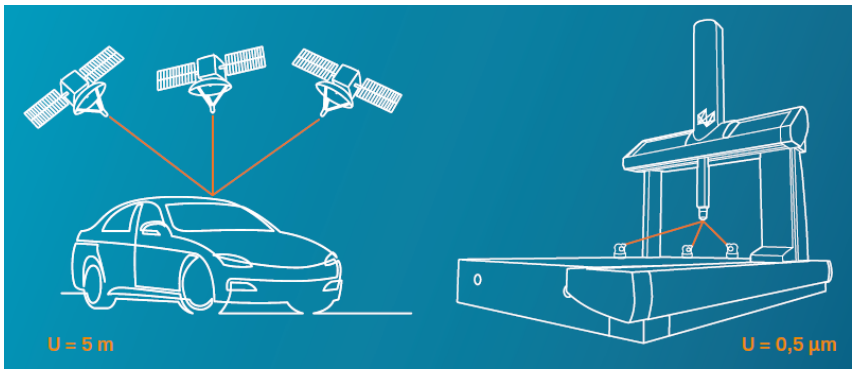
Obrázek 3.7: Laser-Tracer NG[8]



Princip činnosti Laser-Tracer pracuje obdobně jako XL-80 na principu laserové interferometrie. Emituje tedy laserový paprsek, který se na odražeči vrací do detektoru a pomocí interference se počítá relativní vzdálenost. Kromě toho pracuje ještě s jedním základním principem a to je princip triangulace, který se používá například u geo-pozičních služeb jako je Galileo nebo americké GPS. V případě geo-pozičních služeb se detekuje pozice objektu pomocí alespoň tří

různých satelitů. V případě měřidla Laser-Tracer se měřidlo postupně umisťuje na alespoň tři pozice. Na každé pozici se měří zvolený bod a výpočtem přes data ze všech pozic se extrapoluje informace o poloze, resp. chybách daného bodu. Měří se však pochopitelně ne pouze jediný bod, ale množství bodů rovnoměrně rozložených v pracovním prostoru stroje.

Obrázek 3.8: Princip triangulace[8]



Použití
Laser-Tracer se používá pro volumetrickou kompenzaci. Jedná se o komplexní měření v celém rozsahu pracovního prostoru. Toto měřidlo je však výrazně dražší a jeho použití je tedy pro většinu podniků dostupné pouze jako placená služba.

3.6 Způsoby monitorování výrobních strojů

V této kapitole je uveden rozbor problematiky a možností monitorování v obecném pojetí. Vychází se přitom z definice monitorování tak jak je uvedená výše 3.1.1. Před samotným rozborem možností monitorování je vhodné zabývat se několika pojmy a ukázat systém, který je váže dohromady.

Pozn. k pojmům informace a data
Data (z lat. data -> to, co je dáno; synonymum k údaje) jsou popisem objektů, procesů nebo prostředí a mají různé formy. V technické praxi jsou data vymezena úžeji jako popis inherentních vlastnosti objektů, nebo jevů. Příkladem je výška, teplota, barva, rychlost. Jsou tím, co je dáno přímo na objektu. Jsou pojmenovanou inherentní vlastností. V tomto smyslu nelze vždy pracovat přímo s daty, ale s informacemi.

Informace (z lat. in-formatio -> ve-tvaru, utváření, ztvárnění) jdou kódovaná data, vnímané údaje. Výška objektu je údaj / jsou data o tomto objektu, nelze mu je odejmout rozhodnutím. V okamžiku, kdy data nějakým způsobem zapíšeme, už jsou kódovaná. Například písmem — arabskými číslicemi v desítkové soustavě, nebo třeba binárně v počítači. Informace vznikají interpretací dat. Odtud i tzv. data science — datová věda (český překlad se moc nepoužívá) — je věda zabývající se interpretací dat.

Monitorování
Pro pochopení zcela zobecněného systému pro monitorování jsou zavedeny následující pojmy. Předmět monitorování, informační brány (interfaces), informační procesory a informační agregátory.

Předmět monitorování je soustava jednoho, nebo více prvků, které mohou být reálné i abstraktní povahy, která je cílem zájmu monitorování. Tato soustava je nositelem dat, která jsou významná pro cíle monitorování.

Rozhraní / informační brána (interface) je soustava, zajišťující správnou komunikaci a přenos dat. Zprostředkovává přístup k datům a v tomto smyslu je branou k datům. Například teploměr je rozhraním ke čtení teploty, protože bez něj teplotu nelze určit, přestože jako charakteristika objektu existuje. Komunikace mězi dvěma stranami je v tomto obecném pojetí možná vždy a výhradně s alespoň jedním rozhraním. Pro ilustraci lze uvést ústní komunikaci dvou osob. Ta není možná bez sluchového a ústního rozhraní — sluchové ústrojí a ústa. (Filosofická poznámka: Myslet-přemýšlet znamená rozmlouvat-komunikovat sám se sebou bez použití jakéhokoli rozhraní.)

Informační procesor je prvek, který provádí s daty nebo informacemi transformace. Transformací se v tomto smyslu rozumí změna obsahu nebo formátu dat či informací. Může se tedy jednat například o filtraci dat, změnu vnitřní struktury — kódování analogového signálu na digitální, nebo jen změnu formátu — převod z csv do json. Například DAC převodník, který transformuje analogová data na digitální.

Informační agregátor je soustava, která je přechodným, nebo konečným úložištěm dat nebo informací. Její role spočívá pouze v uchovávání. Jakékoli jiné procesy, například interpretace, jsou prováděny procesorem. Například tedy člověk odečítající informaci z teploměru, kdy v roli agregátoru je na vyšší rozlišovací úrovni člověk, na nižší úrovni potom přímo paměť jakožto část mozku. Jiným příkladem jsou počítačové entity hard-disk, paměť RAM, databáze, soubor, atd.

Při použití takto zobecněného systému lze monitorovací systémy rozdělit na několik úrovní složitosti.

3.6.1 Úroveň 1: Obsluha

Nejjednodušším způsobem monitorování je taková podoba systému monitorování, kdy je použito minimální množství technických rozhraní. Prakticky jediná rozhraní jsou lidská smyslová. To znamená, že člověk — obsluha nepoužívá žádné technické prostředky pro extrapolaci informací. Informační procesory nejsou taktéž žádné a informačním agregátorem je samotná obsluha, případně např. sešit.

To znamená, že obsluha používá své smyslové orgány (především zrak a sluch) pro získávání informací přímo ze stroje. Příkladem může být naslouchání procesu třískového obrábění, kdy podle frekvence zvuku lze poznat, ze nástroj nevibruje, nebo se dokonce nezlomil. Obsluha si dlouhodobým nasloucháním vytváří znalostní bázi, které se říká zkušenost. Zkušený operátor je skutečně schopen podle sluchu monitorovat, zda výrobní proces probíhá v pořádku.

Problém tohoto způsobu monitorování je, že znalostní báze se jen velmi obtížně přenáší do jiných agregátorů. V praxi tuto znalostní bázi nelze přenést ani na nového kolegu, natož třeba do počítačové databáze.

Poskytuje tak informace s jen velmi omezenými možnostmi dalšího využití.

3.6.2 Úroveň 2: Měřicí přístroje

Další, vyšší úrovní monitorovacího systému je přidání dodatečných informačních rozhraní, která umožňují zprostředkování dalších dat a nebo jejich efektivnější zpracování. Na této úrovni jsou to měřicí přístroje pro měření fyzikálních charakteristik stroje.

CNC stroje jsou vybaveny několika obslužnými agregáty. Patří mězi ně agregát fluidní, elektřina, dopravník třísek. Mechanické přístroje se používají pro měření veličin na fluidních agregátech. Fluidní agregáty jsou:

- průmyslový olej: hydraulické funkce, mazání, chlazení,
- průmyslový (stlačený) vzduch: ofuk a čistění, kontrolní funkce, chlazení,
- procesní kapaliny: chlazení řezného procesu, oplach a čistění.

Pro měření fluidních médií se používají průtokoměry, tlakoměry a teploměry. Všechny tyto typy přístrojů jsou informačními branami, které mohou sloužit pro monitorování strojů. Opět ale platí, že automatizace práce s informacemi z těchto přístrojů je složitá, nebo zcela vyloučena. V praxi slouží převážně pro okamžitou kontrolu stavu těchto médií.

3.6.3 Úroveň 3: Digitální technologie

Od určité doby je možné stroj řídit pomocí digitálních technologií, tedy programaticky. K tomu je potřeba, aby byl stroj vybaven tzv. číslicovým řízením.

Číslicové řízení strojů hraje v kontextu informačního systému z pohledu monitorování roli informační brány, přičemž umožňuje práci s elektrickými veličinami a jejich digitální interpretací. Zároveň hraje roli procesoru, protože s těmito veličinami pracuje.

Číslicové řízení
Je ve zcela obecném pojetí automatizace pohybu os. Historicky vzniklo potřebou opakovaného najíždění do netriviálních poloh. Tedy evoluce od výroby podle šablon k výrobě s opakovatelným najížděním do poloh. První provedení bylo pomocí diskretizace lineární osy na množinu bodů a ručním najížděním do těchto bodů podle instrukcí [18].

V moderním pojetí číslicové řízení strojů znamená, že jsou vybaveny řídicím systémem a na něj napojenými aktuátory. Mezi hlavní úkoly řídicích systémů patří:

- řízení pohybu,
- výpočet geometrie pohybu,
- vykonávání programů a
- síťová komunikace s okolím.

Řízení pohybu
Je prováděno ve zpětnovazební regulační smyčce, kdy se vyhodnocují informace ze soustavy odměřování polohy a přepočítává se požadovaná změna polohy dle naprogramované dráhy nástroje a následně se regulují náhony [18].

Výpočet geometrie pohybu
Původní, zcela základní a v tomto ohledu hlavní úlohou řídicího systému, je přepočítávat trajektorii pohybu tak, aby bylo dosaženo požadované relativní polohy obrobku a nástroje. Na vstupu je programaticky (v NC kódu) dána požadovaná geometrie, například kružnice o poloměru r a úlohou řídicího systému je přepočítat tuto geometrie na pohyb jednotlivých os. V tomto případě tedy na souběžný (synchronní) pohyb dvou na sebe kolmých lineárních os.

Vykonávání programů
Řídicí systémy je možné instruovat krok za krokem přímo z ovládacího panelu, avšak obvykle se pro řízení používá předem nachystaný program. Takový program se nahraje do paměti stroje a spustí se. Program obsahuje uspořádaný seznam instrukcí k vykonání. Program je počítačový textový soubor s příponou md. Obsahuje instrukce v programovacím jazyce RS-274, což je programovací jazyk pro číslicové řízení, obvykle označovaný jako G-kód, normovaný dle ISO 4342:1985. Soubory obsahující G-kód se označují jako NC programy (Numerical Control Program).

Síťová komunikace s okolím
Nepatří mezi obvyklý výčet hlavních úkolů řídicího systému a to z toho důvodu, že zajištění síťové komunikace není nezbytné pro chod stroje. Ovšem s přechodem společnosti do informačního věku na přelomu milénia a s v poslední dekádě intenzivně probíhající transformací průmyslových odvětví do informační doby, se stala síťová konektivita zcela zásadní a pro moderní stroje nezbytnou [7]. Síťovou komunikací s okolím je myšlena možnost připojit stroj přes běžně dostupné protokoly do počítačové sítě.

Závěr
Číslicové řízení vystupuje v roli procesoru, kdy všechny tyto informace transformuje na digitální data, ke kterým je možné přistupovat. Konkrétní způsoby, jak k datům přistupovat, vyplývají ze specifikací konkrétních řídicích systémů. Obecně je možné k datům přistupovat manuálně, tedy číst je obsluhou stroje přímo na řídicím panelu, nebo programaticky pomocí počítačových technologií. Tzn. připojením stroje do počítačové sítě (např. přes ethernet) a následné zpracování dat již libovolným způsobem.

Číslicové řízení umožňuje automatizovanou práci s daty, která lze snadno přes počítačovou síť přenášet na libovolné koncové body, kde je s nimi možné dále pracovat.

3.7 Zpracování monitorovaných dat

Tato kapitola se zabývá systémy pro zpracovávání dat. Jsou uvedeny základy statistického zpracování numerických dat. A dále některé v praxi používané metody.

Ve stavu kdy stroj generuje data v digitální podobě, je možné tato data dále zpracovávat. Přístupů pro toto další zpracování je velké množství. Lze například využít tabulkový procesor — LibreOffice-Calc příp. Microsoft Office Excel ve kterém lze provádět základní matematické operace, vytvářet grafy, aj. Práci v tabulkových procesorech lze do určité míry automatizovat pomocí skriptování. Calc lze skriptovat poměrně efektivně pomocí Pythonu, v případě MS Excel je bohužel nutné využít těžkopádný VBA.

Další možností je využít software třetí strany, většinou úzce zaměřený. Příkladem může být statistický software Statistica. Jak v případě tabulkových procesorů, tak v případě dalších softwaru však platí, že možnosti automatizace a propojení v rámci rozsáhlého řetězce nástrojů (toolchain) jsou značně omezené. Flexibilnější variantou je použití databází a programatického zpracování (vytvoření softwaru na míru).

3.7.1 Základy statistického zpracování

V této kapitole jsou uvedeny základy statistiky pro práci s numerickými daty, které jsou využity v rámci zpracování dat z monitorování.

Rozdělení na náhodné a systematické chyby
Každé měření je zatíženo chybou měření. To je chyba mezi změřenou hodnotou veličiny a její skutečnou hodnotou. Chybám měření se nedá vyhnout. Vyplývá to z teorie metrologie, kdy platí, že skutečná hodnota veličiny je nepoznatelná a lze se jí pouze libovolně těsně přiblížit. Je to tíha reality, kterou je potřeba nést a unést. Chyby měření jsou dvojího druhy — chyby systematické a náhodné.

Náhodné chyby
Jsou chyby, které způsobují variabilitu výsledků při opakovaném měření za stejných podmínek. Při měření jsou vždy přítomny. Jsou způsobeny nahodilými fluktuacemi při odečítání hodnot v měřicím přístroji nebo obsluhou z měřicího přístroje. Mají náhodný charakter a jejich hodnota není předvídatelná. Částečně vycházejí z interakce měřicího zařízení s okolím. Jsou inherentní vlastností procesu měření. Dají se snižovat statistickými metodami, například průměrováním vícero měření, nikdy je však nelze zcela eliminovat. Snahou proto je, snížit je na hodnotu pod rozlišovací úrovní daného problému.

Pojem náhodné chyby je provázán z pojmem přesnosti, kterou lze definovat tak, že vyšší přesnosti je dosaženo snížením variability (variační odchylky) měření.

Systematické chyby
Jsou predikovatelné a téměř konstantní relativně vůči skutečné hodnotě. Bývají způsobeny metodickou chybou při měření nebo špatnou kalibrací měřicího přístroje. Pomocí statistických metod lze někdy identifikovat jejich zdroj a ten zcela eliminovat. Nejsou inherentní charakteristikou měřicího procesu, naopak vznikají dodatečně nevhodnou metodikou nebo původně nezamýšlenou interakcí s okolím. Lze je zcela eliminovat.

3.7.2 Statistické řízení procesů

V angličtině Statistical Process Control a odtud zkratka SPC. Statistické metody lze využít pro účelové řízení procesů. V principu je řízení navrženo jako zpětnovazební regulační smyčka, kdy se vyhodnocují zvolené statistické ukazatele a na jejich základě se regulují parametry výroby. Příkladem může být řízení ohřevu na základě průměrování teplot z několika různých tepelných čidel.

Aplikují se na procesy s definovaným a měřitelným výstupem. Jejich účelem je snížení pravděpodobnosti, že se vyrobí neshodný díl, který bude muset být opraven, nebo vyhozen. Pomocí SPC zvládnutých procesů lze dosáhnout snížení neshodovitosti, zmetkovitosti a zkrácení výrobních procesů, což všechno vede na efektivnější a ziskovější výrobu.

Aplikování metod statistického řízení se provádí ve třech fázích.

1. Analýza procesu: systémová analýza, která identifikuje podstatné znaky procesu a určuje sledované a řídicí parametry.
2. Stabilizace procesu: eliminace speciálních zdrojů chyb, zvýšení opakovatelnosti procesu na úroveň, kdy je predikovatelný a tedy řiditelný.
3. Monitorování procesu: produkční fáze, kdy je proces za pomoci statistických metod korigován pro dosažení shody se specifikací.

Metody SPC

Rozlišují se dva zdroje dat pro SPC, jsou jimi data získaná měřením a data získaná srovnáváním. Měření je proces k určení hodnoty měřené veličiny. Například měřením díry můžeme pomocí posuvného měřidla zjistit hodnotu průměru (s nějakou chybou). Srovnávání je proces, kdy se srovnávaná veličina porovnává se zvoleným etalonem a zjišťuje se shoda. V případě díry se shoda zjišťuje válcovým kalibrem. Měření tedy poskytuje více informací a jako zdroj dat pro SPC je proto preferované jako zdroj dat.

Důležitou zásadou při sběru dat je pravdivost jednotlivých záznamů. Pro potřeby SPC mohou být záznamy s hrubou chybou, jinak vyřazované, zdrojem informací o speciálních příčinách variability.

V praxi se používá těchto 7 základních nástrojů SPC:

1. záznamy, tabulky, grafy;
2. mapy vad;
3. Ishikawův diagram;
4. histogram četností;
5. Paretova analýza;
6. regulační diagramy;
7. regresní a korelační analýza.

Záznamy, tabulky, grafy

Slouží pro přesný záznam hodnot sledovaných parametrů. Tabulky poskytují strukturovaný a velmi přesný přehled o hodnotách, mohou však být méně přehledné. Grafy jsou o něco méně přesné, ale jsou velice názorné a slouží skvěle jako rychlý vizuální přehled. Vývojové grafy zase poskytují výborné znázornění vývoje procesu v čase. Všechny formy záznamu by měly splňovat alespoň tato tři kritéria: důsledná identifikace každého záznamu, pravdivost dat a přehlednost a čitelnost (pozn. srovnej s větami zenu Pythonu: v1: Explicit is better then implicit; v2: Readability counts; v3: Errors should never pass silently).

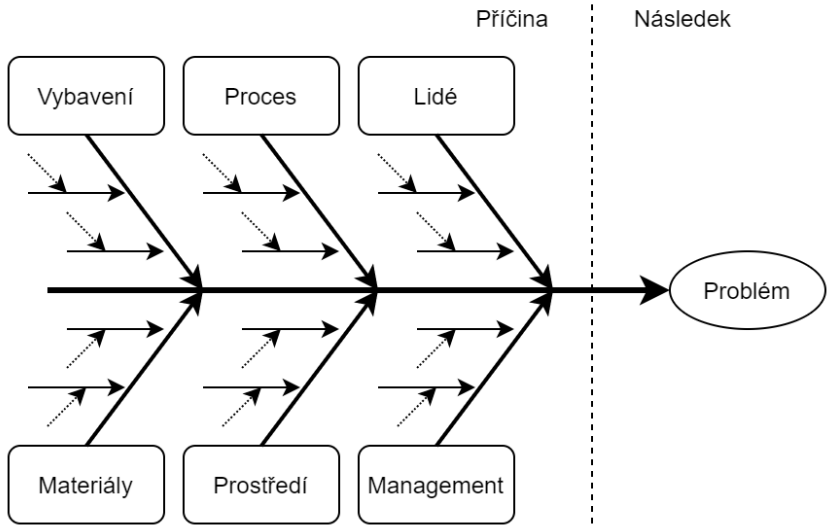
Mapy vad

Jsou vizuálním nástrojem podobně jako grafy. Slouží pro zakreslování zjištěných vad do mapy, kterou může být výkres, náčrt, fotografie, a to i v digitální podobě. Tyto mapy se používají pro identifikaci koncentrátorů vad, což může vést k odhalení nebo objasnění příčin těchto vad.

Ve sve podstatě se jedná o primitivní nástroje, avšak s velkým přínosem a navíc jsou snadno implementovatelné.

Ishikawův diagram
Je nástroj pro zjišťování příčin řešeného problému (následku). Hodí se pro týmovou práci. Je to grafický nástroj, který používá přehlednou strukturu, svým tvarem připomínající rybí kostru (proto se někdy nazývá rybí diagram), za účelem vyjmenování a roztřídění vlivů působících na problém. Je vhodný jako strukturovaná reprezentace dat.

Obrázek 3.9: Ishikawův diagram



Histogram četností
Slouží pro znázornění velkého množství záznamů do grafu, kdy by bylo nepraktické vykreslovat všechny jednotlivé záznamy. Ty se proto roztřídí do tříd o šířce h a do grafu se následně vykreslují pouze celé třídy. Volba počtu tříd a jejich šířka je prakticky libovolná a závisí na povaze statistického souboru. V základu však platí, že volba tříd by měla být takové, aby v každé třídě bylo více než 7 a méně než 20 záznamů a především aby každý záznam pasoval do některé třídy. Pro hrubé určení počtu tříd m platí vztah, kde n je počet záznamů (hodnot):

$$m = 1 + 3,3 \log n \tag{3.1}$$

Je-li n_i četnost i-té třídy, potom pro relativní četnost platí:

$$f_i^R = \frac{n_i}{n} \qquad \text{a} \qquad \sum_1^m f_i^R = 1 \tag{3.2}$$

Zvolíme třídní znak-reprezentant třídy jako její střed: $x_i^* = \min(n_i), \max(n_i)$, potom pro aritmetický průměr třídy:

$$\overline{x} = \frac{1}{n}\sum_{1}^{m}(n_i \cdot x_i^*) \qquad \text{nebo} \qquad \overline{x} = \sum_{1}^{m}\left(f_i^R \cdot x_i^*\right) \tag{3.3}$$

Pro popis variability-proměnlivosti souboru se používají dvě charakteristiky. Rozptyl (variance, disperze) s^2:

$$s^2 = \frac{1}{n-1}\sum_{n=1}^{n}(x_i - \overline{x})^2 \quad = \quad \left(\frac{1}{n-1}\sum_{n=1}^{n}x_i^2\right) - \overline{x}^2 \quad \text{pro neroztříděný soubor} \tag{3.4}$$

$$s^2 = \frac{1}{n-1}\sum_{j=1}^{m}(x_i^* - \overline{x})^2 \quad = \quad \left(\frac{1}{n-1}\sum_{j=1}^{m}f_j x_i^{*2}\right) - \overline{x}^2 \quad \text{pro roztříděný soubor} \tag{3.5}$$

Druhou důležitou charakteristikou variability je směrodatná odchylka s:

$$s = \sqrt{s_2} \tag{3.6}$$

Paretova analýza
Používá se pro identifikaci příčin, které mají největší vliv. Teze Paretova pravidla říká, že 20 % příčin má za následek 80 % všech důsledků. A tedy řešením těchto 20 % lze vyřešit 80 % nákladů. Toto pravidlo se dá uplatňovat na mnohé statistické soubory. Autor uvádí jako příklad z vlastní zkušenosti od firmy vyrábějící složité stroje, že takřka přesně 20,1 % nejdražších dílů představovalo 79,9 % celkových nákladů. Jednalo se převážně o drobný materiál (spojovací m., pneumatické prvky, aj.) v řadu miliard korun ročně, přičemž počet dílů přesahoval 1200. Jde tedy o obrovský statistický soubor a Paretovo pravidlo na něj sedělo zcela přesně.

3.7.3 Strojové učení

Pouze naťuknout — základní rešerše oblasti. Co to je strojové učení a jak by se dalo využít pro monitorování. Strojové učení v roli vylepšeného SPC.

Zajímavou oblastí, která se také zabývá zpracováním dat, je strojové učení, v angličtině Machine Learning a odtud zkratka ML. ML je soubor matematických algoritmů, které se sami zlepšují učením na dodaných datech a v čase zvyšují svou predikční schopnost. ML je podmnožinou oboru umělé inteligence. Často se používá v případech, kdy je množství dat příliš velké na to, aby byla analyzována jinými metodami, nebo v případech, kdy analytické postupy nejsou známé. V těchto případech, pokud jsou k dispozici již evaluovaná data, lze je použít pro trénování modelu ML a ten na základě naučené znalosti s jistou míru přesnosti aproximuje výsledky na neznámých datech. ML dosahuje dobrých výsledků u statisticky stabilních modelů a obecně u statisticky dobře popsatelných datech. Naopak obtížné je použití u úloh, které jsou náročné na klasifikaci. Příkladem těžkých úloh je rozpoznávání obrazu (z toho důvodu se rozpoznávání obrázků používá na webu jako test, zda se jedná o člověka nebo robota — Captcha, což je akronym pro ”completely automated public Turing test to tell computers and humans apart”)

ML představuje zajímavou možnost, jak zpracovávat některá data z výrobního procesu. Na jedné straně je nutné obětovat jistou přesnost, na druhé straně mohou tyto metody přinést výsledky v oblastech, kde je analytické řešení těžké. Lze si představit použití pro predikce teplotního stavu stroje na základě dat z tepelných čidel. Problematika ML je mimo rozsah této práce, je však vhodné je zde uvést jako možnou alternativu.

3.8 Příklad z praxe

Data driven frameworks. Big Data. Data science. V této kapitole by měl být argumentován přechod od izolovaného měření (změřím na jednom PC, v horším případě zůstanou data pouze v něm, v lepším se nahrají na sdílený síťový disk) k systematickému měření. I tady se dá aplikovat princip micro-architektury (microservices).

3.8.1 Práce s daty ve firmách

Dnes je ve výrobních firmách stále poměrně obvyklá situace, kdy data zůstávají na izolovaných ostrovech. Konkrétní příklad z praxe z nejmenované firmy: Firma si zakládala na kvalitních výrobcích a vedení firmy proto požadovalo měření ve všech fázích výroby. Na vstupu se pomocí 3D scannerů měřily odlitky. Data z těchto měření se uchovávala ve specifickém formátu výrobce na pc na měřicím stanovišti. Obráběcí stroje pro opracování odlitků byly nepravidelně proměřovány pomocí Ballbar a nárazově i pomocí LX-80 laser-interferometru od firmy Renishaw. Data z těchto měření se uchovávala na pc pracovníka, který měření zajišťoval. Obrobené díly byly měřeny na měřicích stolech CMM. Z těchto měření se tiskly protokoly, které s danými díly postupovaly dále do výroby, avšak data jako taková se ručně kopírovala do excelovských souborů a ponechávala se na sdíleném disku odboru metrologie. V průběhu výroby byly proměřovány některé konstrukční uzly, data opět ukládána pouze lokálně ve specifickém formátu. Po zkompletování celého stroje se prováděla technická přejímka, v rámci které se měřila geometrická přesnost za použití XM-60 stejné firmy. Výsledky ve formě protokolu se přikládaly k dokumentaci stroje a v tištěné podobě také do složky do archivu. Data jako taková se ponechávala opět jen na lokálním pc. Ve výrobním procesu této firmy tedy vznikalo v různých fázích výroby mnoho potenciálně velmi hodnotných dat, která by se dala využít například pro lepší konstrukci budoucích verzí. Ovšem v důsledku neexistující metodiky pro komplexní práci s těmito daty takový přínos neprobíhal. Tento typ sběru dat zahrnuje několik fatálních nedostatků, ze kterých plyne plýtvání potenciálem.

1. Lokální ukládání dat, případně ukládání na sdílený disk pro vyhrazenou část zaměstnanců (sdílený disk střediska). Tento přístup vytváří informační ostrovy, které je obtížné propojit pro další práci s daty.
2. Specifický formát dat. Pro další práci s daty jsou často využívány specifické nástroje. Buď komerční softwary nebo aplikace naprogramované na míru dané úloze. V obou případech specifický formát dat buď znesnadňuje, nebo zcela znemožňuje další práci s těmito daty. V mnohých případech firmy prodávající měřicí přístroje využívají ve svých softwarech specifické formáty. V lepším případě se jedná pouze o netypickou strukturu dat. V horším případě je formát šifrovaný a data mimo software výrobce nečitelná. Často ale software

výrobce naopak umožňuje export dat do obecného formátu a pak je pouze na firmě, aby metodicky zavedla využívání tohoto formátu.

3. Neexistence komplexní metodiky. Každé středisko mělo na starost jemu svěřené měření a samo (nezávisle na ostatních střediscích) si určovalo měřicí přístroje, metodiku měření, formát a umístění ukládaných dat. Data z jednotlivých středisek jsou v důsledku obtížně nebo vůbec zpracovatelná pro jiné účely. Ve zmíněném příkladu jednotlivá střediska neměla ani informaci o existenci dat na jiných střediscích, která by pro ně mohla být užitečná.
4. Nezájem firem. Ve zmíněném případu nebyly předchozí nedostatky nijak adresovány především proto, že vedení firmy nekladlo prakticky žádný důraz na využívaní dostupných informací. Lze se jen domýšlet, zda to bylo způsobeno neznalostí managerů, nebo jinými důvody. Pokud však neexistuje tlak a metodika se shora, informační potenciál zůstane nevyužit a znamená konkurenční nevýhodu.

Takto nevyužité informace se v zahraniční literatuře někdy označují jako disconnected assets, v překladu nepřipojená aktiva, případně jmění, potenciál. Výraz nepřipojená odkazuje na informační ostrovy [11]. Aktiva, jmění, potenciál vyjadřují utilizační potenciál těchto dat.

Zaveďme nejprve obecný pojem informační aktiva, který vyjadřuje objekty či procesy ve výrobním procesu, které generují nebo mohou generovat nějakou formou informací, které lze utilizovat. Například CMM měřicí stůl, potažmo data získaná procesem měření, je/jsou informačním aktivem.

Dále zaveďme pojem nepřipojená aktiva, kterým budou označována ta informační aktiva ve výrobních procesech, která nejsou plně nebo vůbec využívána (připojena do data zpracující infrastruktury), přestože mají utilizační potenciál. Jako příklad, pokud jsou data ze CMM používána pouze jako rozhodující kritérium pro vyhodnocení shodovitosti ve formě tištěného protokolu, jsou nepřipojeným aktivem. Podle analýzy světového ekonomického fóra z roku 2017 je 85 % utilizačních aktiv stále nepřipojených [32].

Nakonec zaveďme ještě pojem připojená aktiva, který je doplňkem nepřipojených aktiv. Za připojená aktiva se považují ta informační aktiva, která jsou buď plně utilizována nebo jsou alespoň technicky připojená do data zpracující infrastruktury firmy. Jako příklad, data ze CMM lze automaticky transformovat do obecného formátu (například JSON, csv) a ukládat do databáze přístupné na intranetu, kde k nim můžou přistupovat další střediska a využívat je pro další práci a tím je utilizovat. Potom jsou taková data považována za připojená.

Poznámka k pojmu utilizace, příp. utilizační potenciál. V praxi je používán termín monetizační potenciál a je užíván výhradně ve finančním významu. V tomto smyslu monetizovat znamená transformovat na zisk firmy. Pojem však lze chápat i šířeji jako potenciál pro zlepšení, kdy kritériem nejsou finance. Příkladem je výzkum na universitách, který je často neziskový, přesto má smysl uvažovat o informačních ostrovech a nepřipojených aktivech. Monetizační potenciál lze potom překládat jako utilizační potenciál, kdy utilizací se myslí jakékoli využití s kladným přínosem.

3.8.2 Data driven framework

Cílem každé firmy je maximalizovat poměr připojených a nepřipojených aktiv. V dnešní době stále zvyšujících se nároků na kvalitu a přesnost výroby je pro firmy důležité, aby využívaly veškerý informační potenciál, který jejich výroba nabízí. Tedy aby využívaly všechna dosud

nepřipojená aktiva. Způsob využití možných nepřipojených aktiv se liší podle toho, zda je uvažována zcela nová výroba, kdy je možné využití informačních aktiv naplánovat dopředu, přizpůsobit tomu výběr přístrojů a zařízení do výroby (upřednostnit ty výrobce, kteří umožňují další zpracování dat (nešifrují data)) a kompletně navrhnout data zpracující architekturu předem. Postup bude jiný u již existující výroby, kde jsou dána omezení stávajícím vybavením, dostupností technologií, atd.

Data zpracující architektura

Je systém složený ze soustavy hardwaru zajišťujícího propojení všech informačních bodů, a procesů, které transformují data mezi body. Soustava hardwaru zahrnuje koncové body, to jsou body, kde dochází ke vzniku informačních aktiv, a síťové prvky zajišťující propojení bodů v rámci celé sítě. Transformační procesy mohou být počítačové programy, které zpracovávají data (transformují vstupy na výstupy) s cílem jejich utilizace. V praxi data zpracující architektura znamená, že výstupy z měřicích přístrojů, CMM stolů, výrobních strojů, atd. jsou připojená na síť (ta může být intranetová nebo internetová), kde k nim přistupují další střediska, pracují s nimi a v důsledku je utilizují. V obou případech však lze zvolit obecnou metodiku pro výstavbu data-zpracující architektury.

4 Měření a kompenzace geometrické přesnosti

Cílem práce je navrhnout vhodný způsob monitorování a posouzení výrobní přesnosti. Z toho důvodu byl naplánován experiment, s cílem změřit chybu lineárního polohování, vyhodnotit výsledky měření a provést kompenzaci stroje na základě těchto výsledků.

4.1 Návrh experimentu

Byl navržen následující postup: Využít principů experimentálního designu, kdy jedním z nástrojů této disciplíny je intervence neboli zásah a posouzení jeho dopadu na zkoumaný objekt. Z toho vyplývá srovnávací charakteristika pro experiment. Srovnávanými stavy na objektu jsou stav na počátku, tedy stav před intervencí a stav na konci, tedy stav po intervenci. Intervencí je kompenzace chyb pomocí příslušných parametrů v řídicím systému. Pro experiment je potřeba zvolit měřenou veličinu a provést její měření takovým způsobem, aby byla data randomizovaná. Měřenou veličinou je absolutní odchylka polohy v daném bodě v mikrometrech. Randomizaci zajistit zvolením množství poloh pro měření, které nejsou voleny nijak systematicky s ohledem na mechanické vazby. Polohy stanovit z požadavku na měřený rozsah osy a celkový počet poloh.

Na stroji před měřením vypnout všechny případné kompenzace, aby byla zajištěna nezávislost měření.

Měření provést zvlášť pro každou osu, přičemž pro každou osu dokončit celý cyklus základního schématu. Tím se ušetří čas ustavování laseru, protože při měření nejdříve všech os iniciačním měřením, následným kompenzováním všech os a verifikačním měřením, by bylo potřeba ustavovat každou osu dvakrát a dohromady tedy ustavovat laser šestkrát. Případně by se dalo zaměnit pořadí, v jakém jsou osy měřeny a verifikační měření začít na naposled iniciačně měřené ose. Tím by se ušetřilo jedno ustavování a dohromady by tedy bylo potřeba ustavovat pětkrát. Avšak při provedení iniciačního měření, kompenzování a verifikačního měření pro každou osu zvlášť, se ustavuje každá osa pouze jednou a celkový počet ustavování je tedy 3. Jelikož ustavování laseru trvá dlouho, řádově nižší desítky minut, nižší počet ustavování laseru ušetří významné množství času.

Takto získaná data vyhodnotit pomocí proprietárního softwaru do firmy Renishaw, který obsahuje moduly pro vyhodnocení dle různých norem. Pro hodnocení geometrické přesnosti použít normu ISO 230–2.

Po vyhodnocení přesnosti polohování použít stejný software k vygenerování kompenzačních dat. Kompenzační data nahrát do stroje a aktivovat pomocí příslušných parametrů v řídicím systému.

Následně provést nové — verifikační měření s cílem stanovit efekt intervence, neboli ověřit účinnost provedené kompenzace.

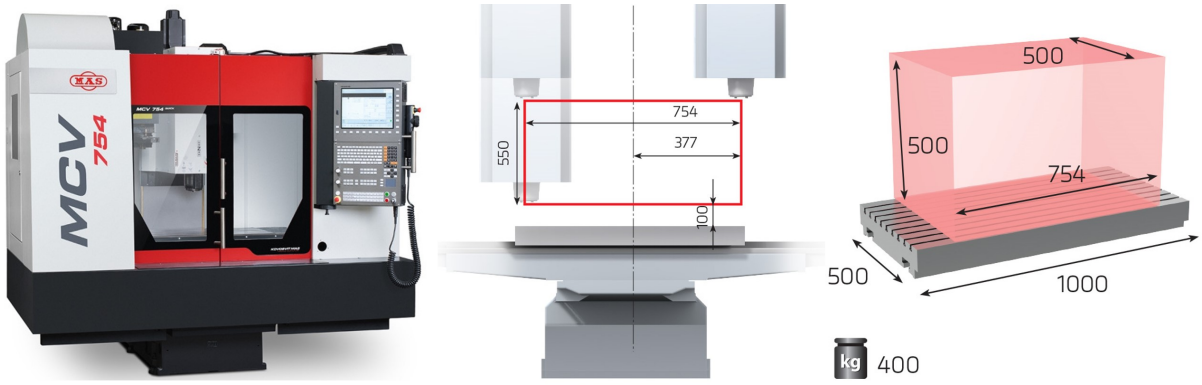
Provést zhodnocení celého experimentu a případně navrhnout jeho úpravy pro budoucí podobná měření.

Obrázek 4.1: Základní schéma experimentu

4.1.1 Zvolený výrobní CNC stroj

Zkoumaným objektem v rámci tohoto experimentu je výrobní stroj (dále jen CNC) v laboratoři výrobních strojů při ústavu výrobních strojů, systémů a robotů na fakultě strojního inženýrství na VUT v Brně. Typ stroje je MCV 754 — Quick od firmy KOVOSVIT MAS a.s. Jendá se o tří-osé vertikální obráběcí centrum.

Obrázek 4.2: KOVOSVIT MAS — MCV 754 Quick



Parametry stroje
V následující tabulce jsou vypsány podstatné parametry stroje.

Tabulka 4.1: Technické parametry stroje

Stůl	
Upínací plocha stolu	1000×500 [mm]
T-drážky	3×18×125 [mm]
Maximální zatížení stolu	400 [kg]
Pracovní rozsah	
X-osa	754 [mm]
Y-osa	500 [mm]
Z-osa	550 [mm]
Vzdálenost čela vřetene od upínací plochy stolu	100-650 [mm]
Vřeteno	
Kuželová dutina vřetena	ISO 40
Max. otáčky	10 000 [min-1]
Změna otáček	plynule měnitelné
Posuv	
Pracovní posuv X, Y, Z	1–30 000 [mm.min-1]
Rychloposuv X, Y, Z	30 [mm.min-1]
Další parametry	
Pracovní tlak vzduchu	0,55–0,6 [MPa]
Rozměry (d×š×v)	2320×2590×2560 [mm]
Hmotnost stroje	4000 [kg]

Tabulka 4.2: Technické parametry stroje, pokračování

Zásobník nástrojů	
Počet míst v zásobníku	24
Max. délka nástroje	250 [mm]
Max. průměr nástroje	75 [mm]
Max. průměr nástroje s vynecháním sousedních nástrojů	120 [mm]
Čas výměny sousedního nástroje	3 [s]
Max. hmotnost nástroje	6,5 [kg]
Motor	
Výkon motoru vřetena SIEMENS (S1/S6 — 40%)	9/13 [kW]
Jmenovitý kroutící moment (S1/S6 — 40%)	57/83 [Nm]
Max. celkový příkon stroje	20 [kVa]
Přesnost ČSN ISO 230-2	
Odměřování X, Y, Z	přímé
Přesnost nastavení polohy	0,012 [mm]
Opakovatelnost nastavení polohy	0,005 [mm]

Strojní osy

Z parametrů stroje vyplývají následující rozsahy os:

- Osa X: <0; 754> mm
- Osa Y: <0; 500> mm
- Osa Z: <0; 500> mm

4.1.2 Laser-interferometer XL-80

Pro měření lineárního polohování použít měřicí přístroj od firmy Renishaw — Laser Interferometr XL-80.

Obrázek 4.3: Laser XL-80 a kompenzační jednotka XC-80

Pro měření při experimentu na MCV Quick bylo použito následující příslušenství.

- Laser XL-80: Zdroj laseru, slouží jako měřicí jednotka. Pracuje na principu laserové interferometrie. Je to relativní měřidlo — na začátku měření je potřeba zvolit nulovou pozici — zreferovat.
- Kompenzátor XC-80: Při měření je důležité kompenzovat vliv okolního prostředí an měření. Specificky tlak, teplotu a vlhkost vzduchu. K tomu se používá kompenzační jednotka, kterou lze připojit do PC společně s laserem a software potom při výpočtu bere v potaz data z kompenzátoru.
- Koutové odražeče: Pro funkci laseru je potřeba koutový odražeč, který bude na konci dráhy odrážet, vracet paprsek zpět do laserové jednotky.
- Trojnohý stojan laseru a magnetické držáky.
- PC se softwarem Carto.

Měřitelné chyby
Měřicí soustavu lze pořídit ve dvou provedeních. Liší se vybavením, přičemž vyšší verze umožňuje měření více druhů chyb. Celkově je možné je možné měřit tyto chyby: lineární polohování, úhlové chyby, chyby přímosti, rovinnost a kolmosti.

Tabulka 4.3: Operační parametry laseru

Operační parametry		
Max. posuvová rychlost	4 m/s	
Max. vzorkovací frekvence	50 kHz	
Doba předehřevu	< 6 min	
Rozsah operační teploty	0 až 40 °C	
—	**Rozsah**	**Přesnost**
Teplota materiálu	0 až 55 °C	± 0,1 °C
Teplota vzduchu	0 až 40 °C	± 0,2 °C
Tlak vzduchu	650 až 1150 mbar	± 1 mbar
Vlhkost vzduchu	0 až 95 %	± 6 %

Parametry laseru	
Stabilita frekvence	± 0,05 ppm
Rozměry	214×120×70 mm
Zdroj napětí	90 až 264 V AC
Typ laseru	Neodymový
Vlnová délka	638,8 nm

Princip činnosti
Laser-interferometer pracuje na principu laserové interferometrie, jak ostatně napovídá název. Princip spočívá v tom, že se pomocí interference vln počítá relativní pohyb dvou paprsků. Jeden paprsek je tzv. referenční. Jeho dráha se pomocí optické soustavy vytvoří jako statická, tedy nepohyblivá. Při měření je tedy tento paprsek stabilní a slouží jako reference pro výpočet interferenčních přechodů. Druhý paprsek je měřicí. Jeho dráha se z optických dílů postaví tak, aby se koncový odražeč pohyboval společně s měřenou osou. Vzájemná interference těchto dvou paprsků slouží pro výpočet uražené vzdálenosti. Jedná se tedy o měřidlo relativní, nikoli absolutní.

Aby se dal laserový paprsek využít k interferometrii, musí být splněny tři podmínky:

- Vlnová délka paprsku musí být stabilní a přesně známá.
- Vlnová délka musí být malá, aby bylo dosaženo vysokého rozlišení a tedy přesnosti.
- Všechny paprsky vycházející z laseru musí být v jedné fázi.

Dostupnost zařízení
Tato měřicí soustava je dostupná od britské firmy Renishaw za cenu přibližně 600 tisíc korun. Finanční náročnost je relativní v kontextu použití. Pro drobného podnikatele, který provozuje 3 obráběcí centra v režimu workshop, jde o vysoké náklady, které nemusí být rentabilní. Pro podnik, který se zabývá výrobou velmi velkých strojů, kdy prodejní cena takového stroje je řádově desítky milionů, už se jedná o relativně malý náklad, který může znamenat velkou přidanou hodnotu.

Lze tedy spekulovat, že toto zařízení se používá jako opce při nákupu velkých a drahých strojů, ale v případě malých strojů je využití spíše formou placené kalibrace v pravidelných intervalech. V obou případech je však zařízení poměrně dobře dostupné a lze ho proto doporučit jako nástroj v technickém řetězci.

4.1.3 ISO 230–2

Data z měření budou vyhodnocena dle normy ISO 230–2 [15]. Norma obsahuje popis všech charakteristik, které jsou potřeba pro výpočet chyb lineárního polohování.

Funkční bod
Je střední bod řezného nástroje nebo bod spojený se součástí na obráběcím stroji, kde by se řezný nástroj dotýkal části za účelem odběru materiálu.

Zadaná poloha
Je poloha, do které je naprogramován pohyb pohybující se součásti. Značí se $P_i (i = 1 \text{ až } m)$

Skutečná poloha
Měřená poloha dosažená nastavovanou částí při j-tém nastavení do i-té polohy. Značí se $P_{ij} (i = 1 \text{ až } m; j = 1 \text{ až } n)$

Polohová úchylka
Nebo také úchylka polohy. Je skutečná poloha dosažená funkčním bodem minus zadaná poloha.

$$x_{ij} = P_{ij} - P_i \tag{4.1}$$

Jednosměrný
Výraz se vztahuje k řadě měření, při kterých se nastavování do zadané polohy v dané ose vykonává vždy ve stejném směru pohybu.

Obousměrný
Výraz se vztahuje k řadě měření, při kterých se nastavování do zadané polohy v dané ose vykonává v obou směrech.

Standardní nejistota
Nejistota výsledku měření vyjádřená jako směrodatná odchylka (3.6).

Kombinovaná standardní nejistota
Standardní nejistota výsledku měření, když je výsledek získaný z hodnot několika dalších veličin, rovnající se kladné hodnotě druhé odmocniny součtu výrazů. Výrazy jsou rozptyly nebo kovariace těchto dalších veličin vážených podle toho, jak se výsledek měření mění se změnami těchto veličin.

Rozšířená nejistota
Veličina stanovující interval kolem výsledku měření, od kterého se může očekávat, že pokryje velký podíl rozdělení hodnot, které mohou být důvodně přiřazeny k měřené veličině.

Koeficient rozšíření
Číselná hodnota činitele, užívaná jako násobek kombinované standardní nejistoty k získání rozšířené nejistoty.

Průměrná jednosměrná polohová úchylka v poloze

$$\overline{x}_i \uparrow = \frac{1}{n} \sum_{j=1}^{n} x_{ij} \uparrow \tag{4.2}$$

$$\overline{x}_i \downarrow = \frac{1}{n} \sum_{j=1}^{n} x_{ij} \downarrow \tag{4.3}$$

Průměrná obousměrná polohová úchylka v poloze
Aritmetický průměr průměrných jednosměrných polohových úchylek $\overline{x}_i \uparrow$ a $\overline{x}_i \downarrow$ (4.2) získaných při najíždění do polohy P_i (4.1.3)

$$\overline{x}_i = \frac{\overline{x}_i \uparrow + \overline{x}_i \downarrow}{2} \tag{4.4}$$

Reverzní chyba v poloze
Je rozdíl mezi průměrnými jednosměrnými polohovými úchylkami získanými při najíždění do polohy P_i (4.1.3) v obou smyslech.

$$B_i = \overline{x}_i \uparrow - \overline{x}_i \downarrow \tag{4.5}$$

Reverzní chyba v ose
Je největší z absolutních hodnot reverzních chyb v poloze B_i (4.5) ze všech zadaných poloh podél nebo okolo osy.

$$B = \max\left[|B_i|\right] \tag{4.6}$$

Průměrná reverzní chyba v ose
Aritmetický průměr necitlivostí B_i (4.5) ze všech zadaných poloh podél nebo okolo osy.

$$\overline{B} = \frac{1}{m} \sum_{i=1}^{m} B_i \tag{4.7}$$

Odhad jednosměrné opakovatelnosti osy nastavení polohy v poloze
Je odhad běžné nejistoty polohových úchylek získaných řadou n najetí do polohy P_i v jednom směru.

$$s_i \uparrow = \sqrt{\frac{1}{n-1} \sum_{j=1}^{n} (x_{ij} \uparrow - \overline{x}_i \uparrow)} \tag{4.8}$$

$$s_i \downarrow = \sqrt{\frac{1}{n-1} \sum_{j=1}^{n} (x_{ij} \downarrow - \overline{x}_i \downarrow)} \tag{4.9}$$

Jednosměrná opakovatelnost nastavení polohy v poloze
Je rozsah odvozený z odhadu jednosměrné opakovatelnosti osy natavení polohy v poloze P_i při použití koeficientu rozšíření $k = 2$.

$$R_i \uparrow = 4 s_i \uparrow \tag{4.10}$$

$$R_i \downarrow = 4s_i \downarrow \tag{4.11}$$

Obousměrná opakovatelnost nastavení polohy v poloze

$$R_i = \max\left[2s_i \uparrow + 2s_i \downarrow + |B_i|; R_i \uparrow ; R_i \downarrow\right] \tag{4.12}$$

Jednosměrná opakovatelnost nastavení polohy v ose
Je největší hodnota opakovatelnosti nastavení polohy v kterékoliv poloze P_i podél nebo okolo osy.

$$R \uparrow = \max R_i \uparrow \tag{4.13}$$

$$R \downarrow = \max R_i \downarrow \tag{4.14}$$

Obousměrná opakovatelnost nastavení polohy v ose
Je největší hodnota opakovatelnost nastavení polohy v kterékoliv poloze P_i podél nebo okolo osy.

$$R = \max R_i \tag{4.15}$$

Jednosměrná systematická chyba polohování v ose
Je rozdíl mezi největší algebraickou a nejmenší algebraickou jednosměrnou polohovou úchylkou při nastavování polohy v jednom směru $\overline{x}_i \uparrow$ nebo $\overline{x}_i \downarrow$ v jakékoliv poloze P_i podél osy nebo okolo osy.

$$E \uparrow = \max\left[\overline{x}_i \uparrow\right] - \min\left[\overline{x}_i \uparrow\right] \tag{4.16}$$

$$E \downarrow = \max\left[\overline{x}_i \downarrow\right] - \min\left[\overline{x}_i \downarrow\right] \tag{4.17}$$

Obousměrná systematická chyba polohování v ose
Je rozdíl mezi největší algebraickou a nejmenší algebraickou průměrnou jednosměrnou polohovou úchylkou při nastavování polohy v obou směrech $\overline{x}_i \uparrow$ nebo $\overline{x}_i \downarrow$ v jakékoliv poloze P_i podél nebo okolo osy.

$$E == \max\left[\overline{x}_i \uparrow ; \overline{x}_i \downarrow\right] - \min\left[\overline{x}_i \uparrow ; \overline{x}_i \downarrow\right] \tag{4.18}$$

Průměrná obousměrná chyba polohování v ose
Je rozdíl mezi největší algebraickou a nejmenší algebraickou průměrnou obousměrnou polohovou úchylkou $\overline{x}_i$ v jakékoliv poloze P_i podél nebo okolo osy.

$$M = \max\left[\overline{x}_i\right] - \min\left[\overline{x}_i\right] \tag{4.19}$$

Jednosměrná chyba polohování v ose
Nebo také jednosměrná přesnost nastavení polohy v ose je rozsah odvozený z kombinace průměrné jednosměrné systematické polohové chyby a odhadu pro opakovatelnost při jednosměrném nastavení polohy při použití koeficientu rozšíření $k = 2$.

$$A \uparrow = \max\left[\overline{x}_i \uparrow + 2s_i \uparrow\right] - \min\left[\overline{x}_i \uparrow - 2s_i \uparrow\right] \tag{4.20}$$

$$A\downarrow = \max\left[\overline{x}_i\downarrow + 2s_i\downarrow\right] - \min\left[\overline{x}_i\downarrow - 2s_i\downarrow\right] \tag{4.21}$$

Obousměrná chyba polohování v ose
Nebo taky obousměrná přesnost nastavení polohy v ose je rozsah odvozený z kombinace průměrné obousměrné systematické polohové chyby a odhadu pro opakovatelnost při obousměrném nastavení polohy při použití koeficientu rozšíření $k = 2$.

$$A = \max\left[\overline{x}_i\uparrow + 2s_i\uparrow ; \overline{x}_i\downarrow + 2s_i\downarrow\right] - \min\left[\overline{x}_i\uparrow - 2s_i\uparrow ; \overline{x}_i\downarrow - 2s_i\downarrow\right] \tag{4.22}$$

Přehled terminologie

Tabulka 4.4: Přehled terminologie normy ISO 230–2

Název charakteristiky	Značení
Obousměrná chyba polohování v ose	A
Obousměrná opakovatelnost nastavení polohy v ose	R
Obousměrná systematická chyba polohování v ose	E
Reverzní chyba v ose	B
Průměrná reverzní chyba v ose	$\overline{B}$
Průměrná obousměrná chyba polohování v ose	M

4.2 Měření

Měření proběhlo ve čtvrtek 16. dubna 2020 ve spolupráci s vedoucím diplomové práce Ing., Dipl.-Ing M. Holubem, Ph.D.. Dle navrženého postupu byly měřeny jednotlivé osy postupně. Začalo se měřením osy X.

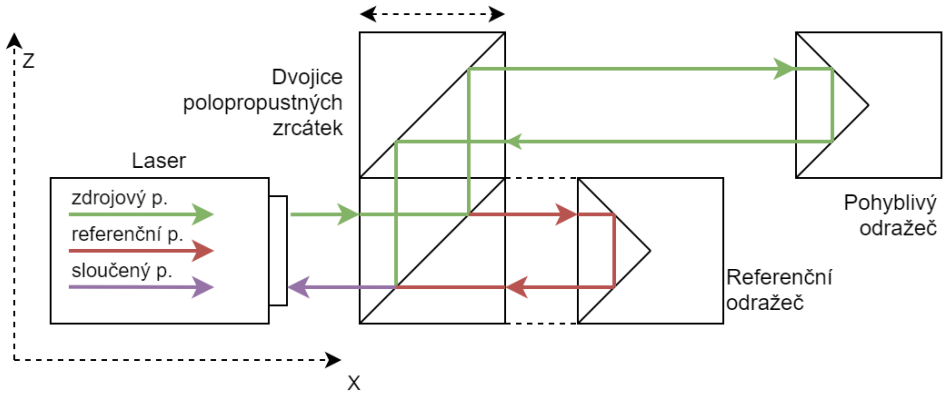
4.2.1 Osa X

Laser byl ustaven z levé strany stroje. Referenční paprsek byl ustaven do roviny XZ. Hlavní koutový odražeč byl umístěn s využitím magnetického držáku na vřeteno. Referenční koutový odražeč byl umístěn na stůl stroje. Umístění bylo zvoleno v polovině osy Y a téměř na konci stolu v rámci osy X. Toto rozložení umožňuje měření v rozsahu 2 až 752 mm. Tedy na 99,5 % délky osy. Zbylé 2 mm na každém konci jsou nutné pro přejezdy, které se používají pro vymezení vůlí při změně smyslu pohybu stolu.

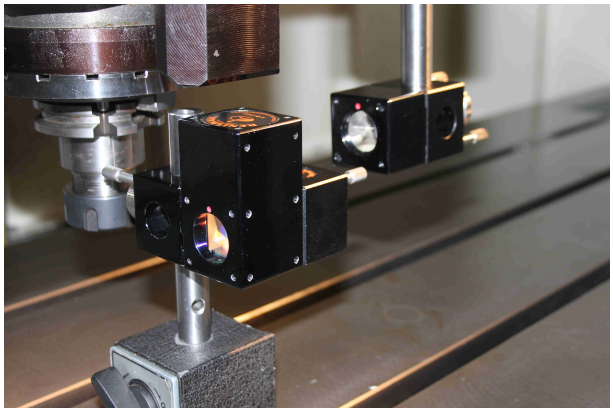
Sada s laserem a příslušenstvím, která je na ústavu k dispozici, neobsahuje díl s jedním polopropustným zrcátkem. Namísto toho obsahuje dvou-díl, který obsahuje dvě po sobě jdoucí, vzájemně rovnoběžná polopropustná zrcátka. Ustavení zrcátek bylo nutné tomu přizpůsobit. Přestože by tedy šlo sestavit schémata jednodušším způsobem, nebylo to možné kvůli konkrétním dostupným dílům. Pro sestavu s jediným polopropustným zrcátkem je charakteristické, že dráhy referenčního a měřicího paprsku jsou na sebe kolmé. V případě konfigurace se dvěma zrcátky jsou vůči sobě tyto dráhy rovnoběžné, jak je parné z následujícího schématu.

Obrázek 4.4: Programovaní stroje

Obrázek 4.5: Schéma ustavení v ose X



Obrázek 4.6: Umístění odražečů při měření osy X



Parametry měření
Délka osy je 754 mm, měřená délka 750 mm. Délka kroku měření byla zvolena na 75 mm s ohledem na celkový počet 10 intervalů. Počet poloh je tedy 11. Dále bylo zvoleno, že počet opakování měření je 5. Měřit se bude pro každý směr zvlášť.

Tabulka 4.5: Nastavení měření

Společné pro všechny osy	
Target sequence	linear
Run sequence	alternate
Run type	bidirectional
Averaging period	0,4625 s
Expansion coefficient	11,7 ppm/°C
Trigger mode	position based
Stability period	0,40 s
Stability range	0,010 mm
Tolerance	0,250 mm
Runs	5
Osa X	
First target	2 mm
Last target	752 mm
Targets	11
Osa Y	
First target	50 mm
Last target	500 mm
Targets	10
Osa Z	
First target	-500 mm
Last target	0 mm
Targets	11

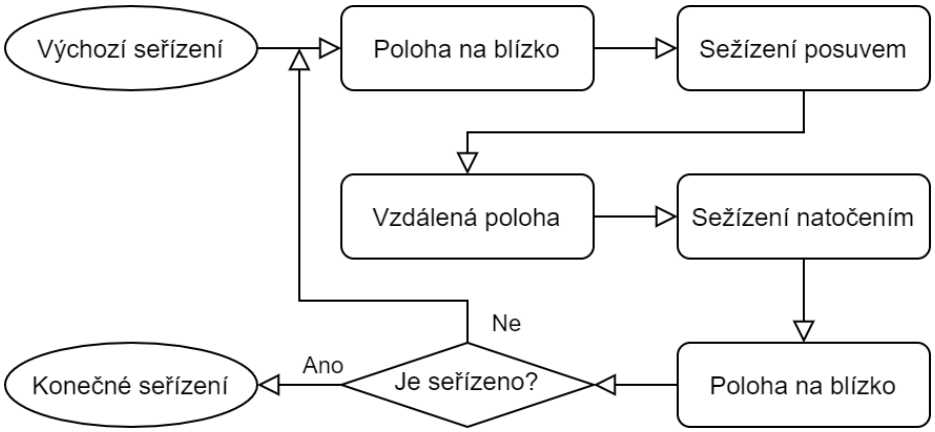
Seřízení paprsku
Po umístění zrcátek do pracovního prostoru stroje je potřeba seřídit paprsek laseru. K tomu slouží seřizovací podložka, která je umístěna na trojnohém stojanu a jednotka laseru je připevněna na tuto podložku. Podložka umožňuje seřízení úhlově ve dvou osách kolmých na směr paprsku (radiální seřízení). Dále umožňuje seřízení posuvem v vodorovné ose kolmé na paprsek. Ve druhé ose lze seřizovat pouze výsuvem stojanu, což je nepraktické a málo přesné. Jemný výsuv bohužel chybí a projevilo se to jako nevýhoda při seřizování jedné z os. V případě seřizování osy X je však možné využít výsuv vřetena.

Metodika seřizování je následující. Krytka na jednotce laseru se nejdříve pootočí do seřizovací polohy. Tím se zakryje otvor pro vstup odraženého paprsku a ten tak lze vidět a seřizovat. Krytka se umístí také na hranol s polopropustným zrcátkem. To se následně umístí na stůl stroje tak, aby paprsek dopadal na terčík krytky zrcátka.

V dalším kroku se paprsek seřizuje na koutový odražeč na druhém magnetickém držáku, který je určený pro posuv a tedy měření. Postupuje se iteračním způsobem, kdy platí pravidlo, že

když jsou od sebe polopropustné zrcátko a koutový odražeč blízko, seřizuje se posuvem a když jsou daleko, seřizuje se natočením. Přitom se pohybuje strojní osou vždy v plném rozsahu měření. Princip je shrnutý na následujícím schématu.

Obrázek 4.7: Schéma seřízení paprsku



Poté, co je paprsek tímto způsobem seřízen s krytkou, se krytka oddělá a sleduje se, zda je signál z odraženého paprsku dostatečně silný. Síla signálu se ověřuje na jednotce laseru, která má z horní strany diody. Pokud všech pět diod svítí zeleně, je signál plný a měření připraveno.

Nastavení softwaru

Pro měření se používá proprietární software od firmy Renishaw — Carto. Tento software má několik modulů: Explore pro prohlížení výsledků, Capture pro měření a Compensate pro vytváření kompenzačních NC programů. V tomto bodě se tedy používá Carto-Capture. Nastavují se především parametry měření, které jsou shrnuty v tabulce 4.5.

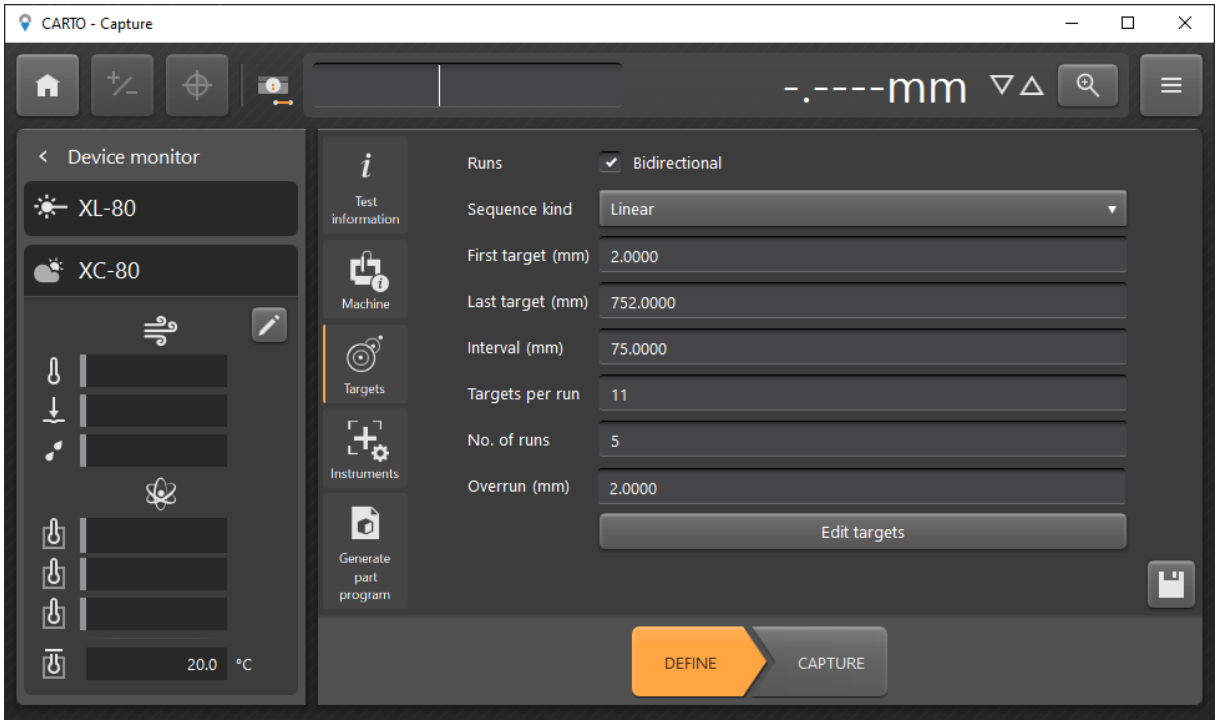
NC program

Ve stejném modulu je záložka pro generování NC programu. Na této záložce se nastaví typ řídicího systému a vygeneruje NC program. Ve většině případů by mělo být možné tento program na stroji bez dodatečných úprav spustit. U tohoto měření tomu tak bylo. Součástí vygenerovaného programu však není kód pro deaktivaci kompenzačních tabulek. To je proto potřeba pohlídat ručně. Při zjišťovacím měření je potřeba kompenzační tabulky vypnout. Po nahrání kompenzačních dat před verifikačním měřením je zase potřeba zkontroloval, že jsou zapnuté. Použité programy jsou přiloženy jako přílohy.

Zjišťovací měření

Po nahrání NC programu do stroje je možné přikročit k samotnému měření. Nejdříve se spustí NC program na stroji. V jeho začátku je příkaz M00, což je příkaz pro zastavení běhu programu až do opětovného puštění fyzickým tlačítkem NC start na řídicím panelu. Toto první programové zastavení je výzva ke spuštění programu na PC. Pustí se tedy program v Carto-Capture a následně znovu spustí NC program. V tomto okamžiku je zahájeno měření a automatické ukládání dat v programu Capture. Celková doba měření je odvislá od zvolené rychlosti posuvu, poštu měřených poloh a především fyzické délce osy.

Obrázek 4.8: Nastavení měření v Carto-Capture



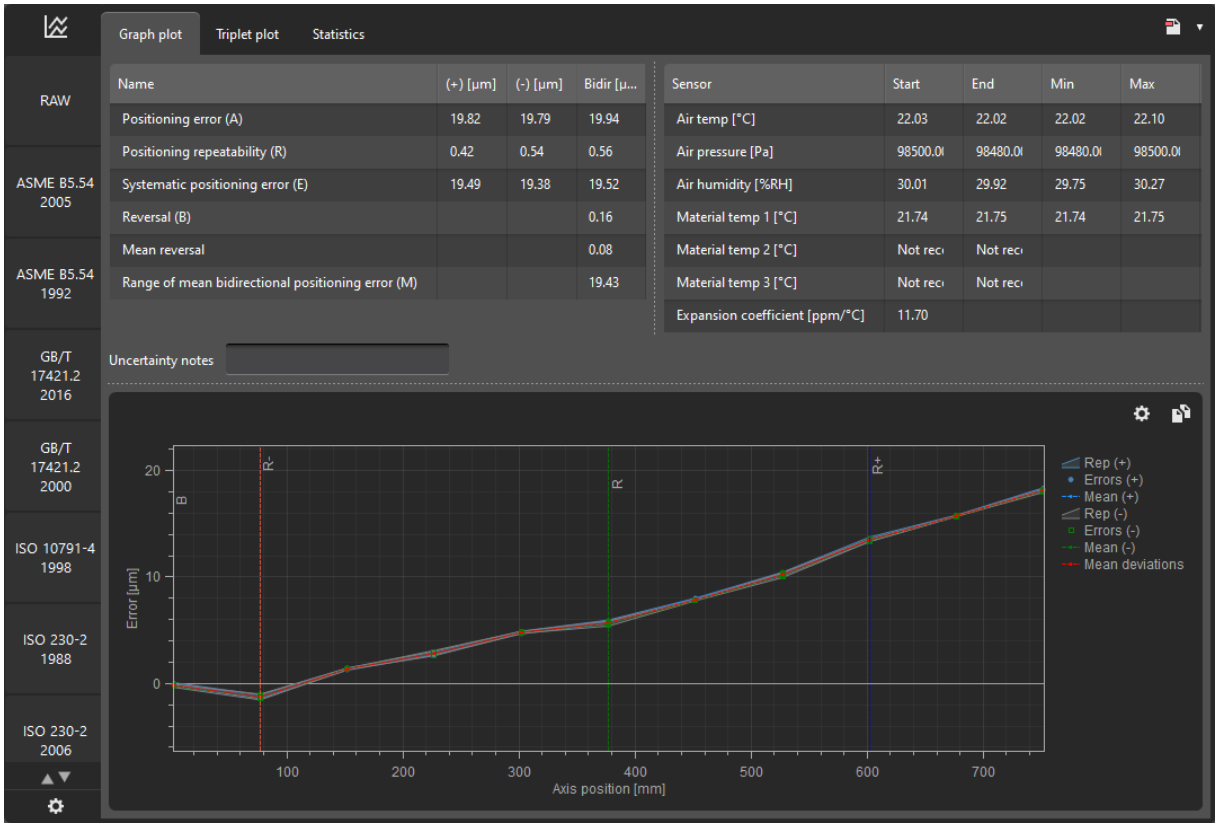
Toto měření trvalo celkem 5 min a 24 s. Laser se ponechal na pozici — po aplikování kompenzací bude spuštěno verifikační měření.

Vyhodnocení výsledků

Výsledky jsou ihned dostupné před další modul softwaru Carto — Explore. Tento modul umožňuje procházení nových výsledků, ale i historických dat, která jsou v rámci softwaru ukládána v mongo-db databázi. Data je bohužel možné exportovat mimo software pouze jako plain-text soubor, nikoli jako databázi, se kterou by bylo jednodušší dále pracovat. Exportovaný textový soubor navíc není v žádném strukturovaném formátu (např. JSON, XML, YAML, atd.). Toto lze interpretovat jako snahu výrobce (Renishaw) o tzv. vendor lock-in, což je situace, kdy se výrobce snaží zamezit či znesnadnit přechod k jinému řešení nebo jinému výrobci. Pokud by bylo možné exportovat celou databázi, bylo by možné k ní snadno přistupovat z libovolného dalšího softwaru — např. Matlab a v něm provést vlastní zpracování dat.

Vyhodnocení tedy bylo provedeno v rámci tohoto softwaru. Výsledky jsou shrnuty na konci této podkapitoly — REFERENCE.

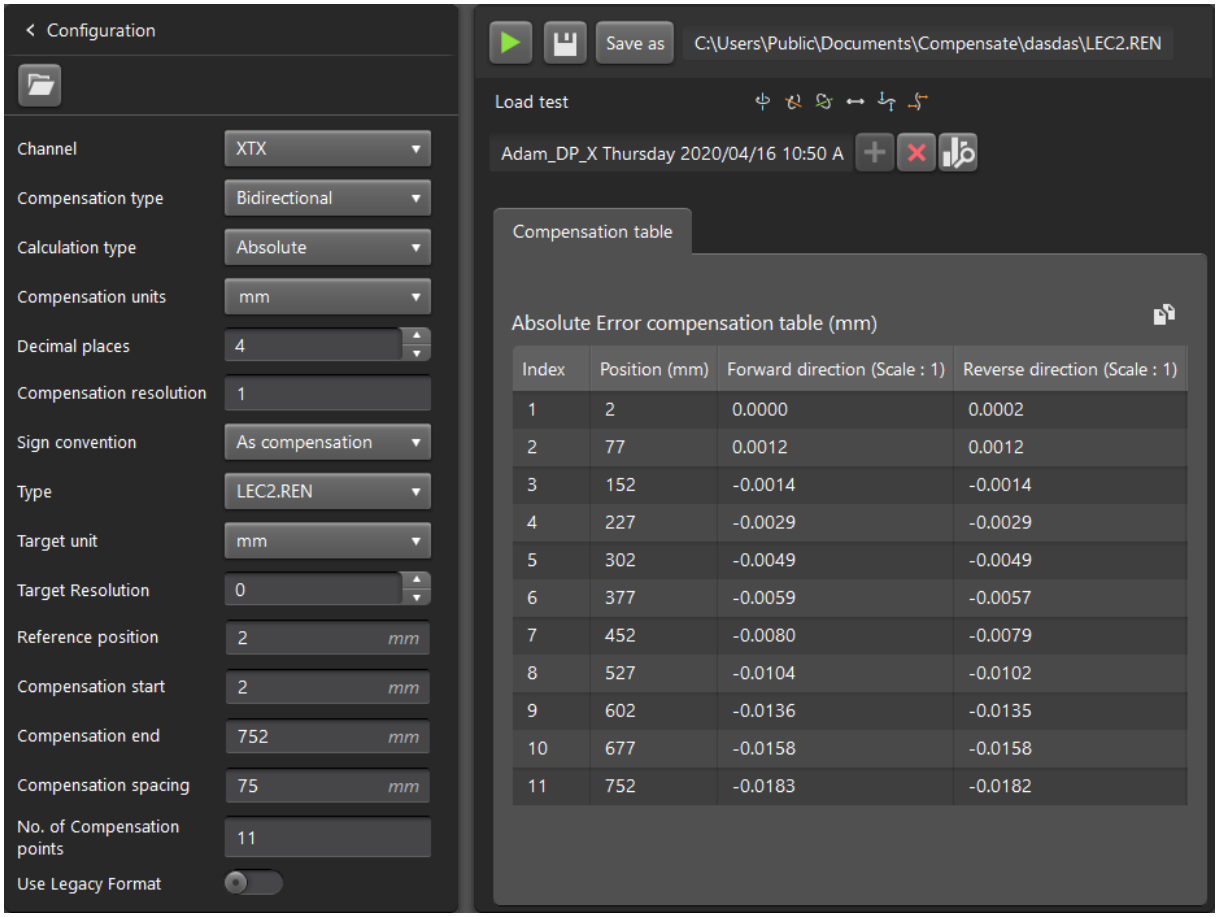
Obrázek 4.9: Výsledky zjišťovacího měření osy X



Kompenzování

Pro kompenzování so používá třetí modul aplikace Carto — Compensate. V tomto modulu se nejdříve nahraje požadované měření, které se má použít pro výpočet kompenzací. Následně se nastaví několik parametrů podle cílového řídicího systému stroje. Poté se vygenerují kompenzační data. Bohužel nelze vygenerovat NC program, který by nahrání a aktivaci příslušných kompenzačních dat ve stroji provedl programaticky. Je potřeba data vzít a ručně přepsat do stroje. Poté je potřeba, opět ručně, aktivovat příslušné tabulky a tím kompenzace aktivovat. Software tedy provádí pouze výpočet kompenzačních dat a tato umí exportovat jako plain text, vy výsledku je však potřeba data přepisovat ručně.

Obrázek 4.10: Generování kompenzačních dat v softwaru Carto-Compensate



Verifikační měření
Po aktivování kompenzací na stroji se provádí verifikační měření. Z pohledu plánování experimentu jde o srovnání s cílem zjistit dopad intervence. Verifikační měření se provádí zcela shodným způsobem jako zjišťovací měření. Díky tomu, že aparatura laseru byla ponechána na místě, není ani nutné znovu ustavovat laser. Stačí znovu spustit měření. V případě, že by zjištěná chyba byla stále příliš vysoká, přistoupilo by se k novému kompenzování s cílem chybu dále zmenšit. V tomto případě však byla chyba po kompenzování přijatelně malá a celý experiment tím pro tuto osu končí.

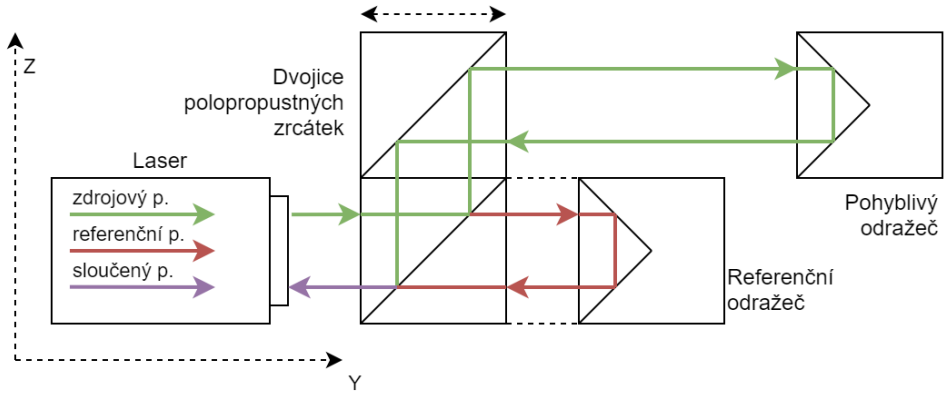
Výsledky jsou shrnuty na konci podkapitoly.

4.2.2 Osa Y

Postup při měření osy Y je identický jako u osy X, pochopitelně s rozdílem při ustavování laseru a zrcátek. Osa Y byla měřena jako poslední a to z toho důvodu, že pro měření v ose Z se použilo stejné umístění laseru jako v ose X. Aby tedy nebylo nutné laser několikrát přemisťovat, měřila se nejdříve osa X, poté Z a nakonec Y.

Ustavení zrcátek
Ustavení je schématicky stejné jako v ose X, pouze celá soustava jo otočena do osy Y.

Obrázek 4.11: Schéma ustavení v ose Y

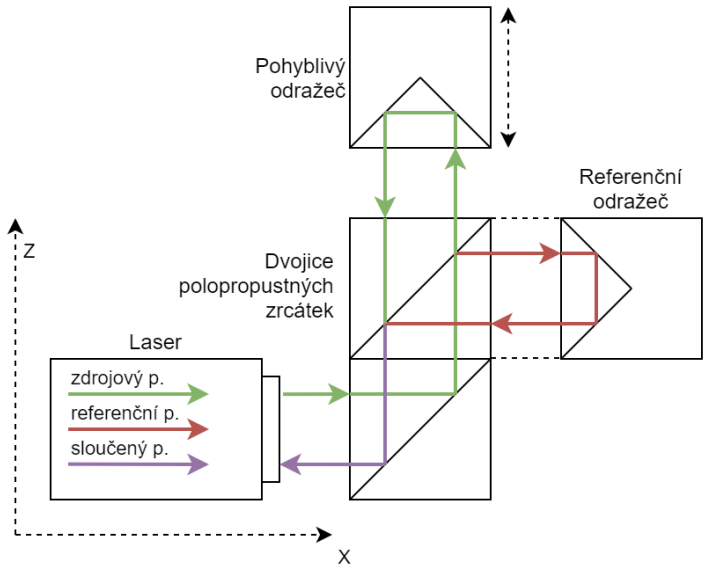


4.2.3 Osa Z

Postup při měření osy Z je opět totožný. Díky tomu, že se osa Y měřila až jako poslední, nebylo potřeba pro měření osy Z přemisťovat laser.

Ustavení zrcátek
Pro měření osy Z bylo potřeba zcela překonfigurovat soustavu zrcátek a najít novou trasu paprsku. Nové schéma je znázorněno na následujícím obrázku.

Obrázek 4.12: Schéma ustavení v ose Z



4.3 Výsledky

V této kapitole jsou ukázány výsledky z měření lineárního polohování. Pro každou osu je uvedena tabulka se statistickými ukazateli, vypočtenými podle ISO 230–2 a graf, který ukazuje pro

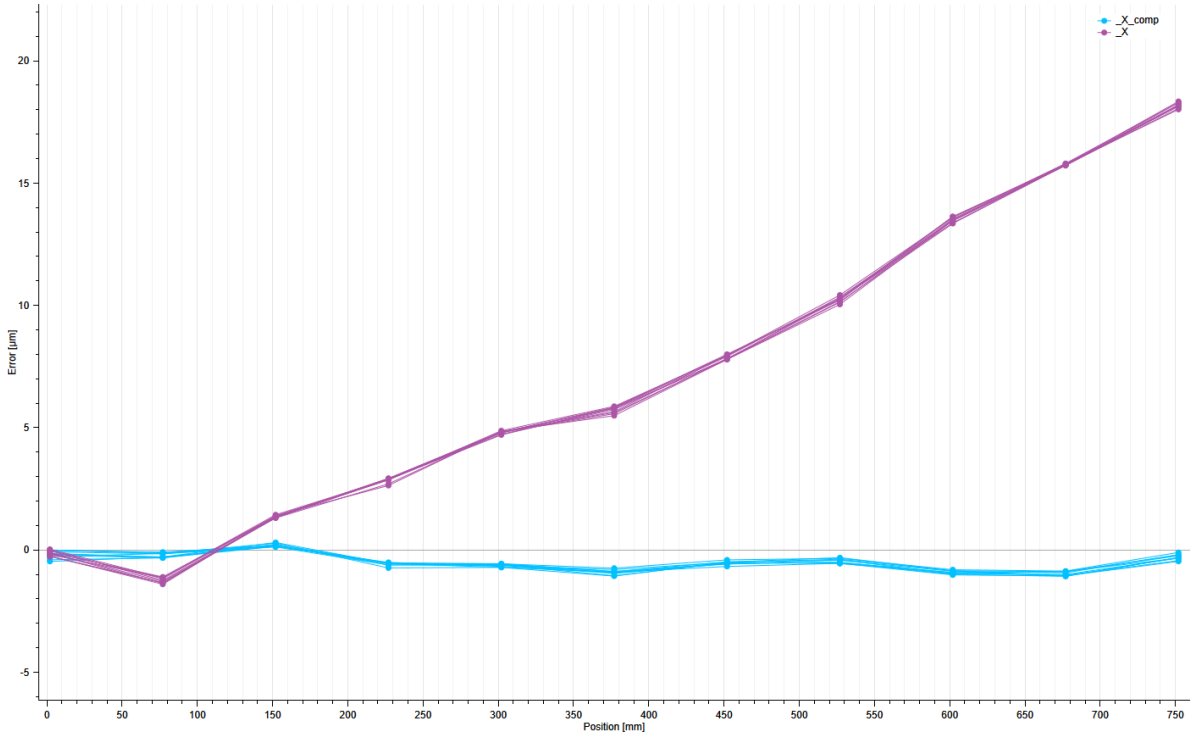
danou osu chybu polohování. Grafy jsou exportované ze softwaru Carto-Explore. Na každém grafu je fialovou barvou vyznačeno zjišťovací měření — tj. měření před kompenzováním osy a verifikační měření, což je po kompenzování.

4.3.1 Osa X

Tabulka 4.6: Přesnost polohování v ose X dle ISO 230–2

Osa X	PŘED [µm]	PO [µm]
Positioning error (A)	19.94	1.50
Positioning repeatability (R)	0.56	0.72
Systematic positioning error (E)	19.52	1.18
Reversal (B)	0.16	0.21
Mean reversal	0.08	0.07
Range of mean bidirectional positioning error (M)	19.43	1.17

Obrázek 4.13: Porovnání přesnosti před a po kompenzování osy.

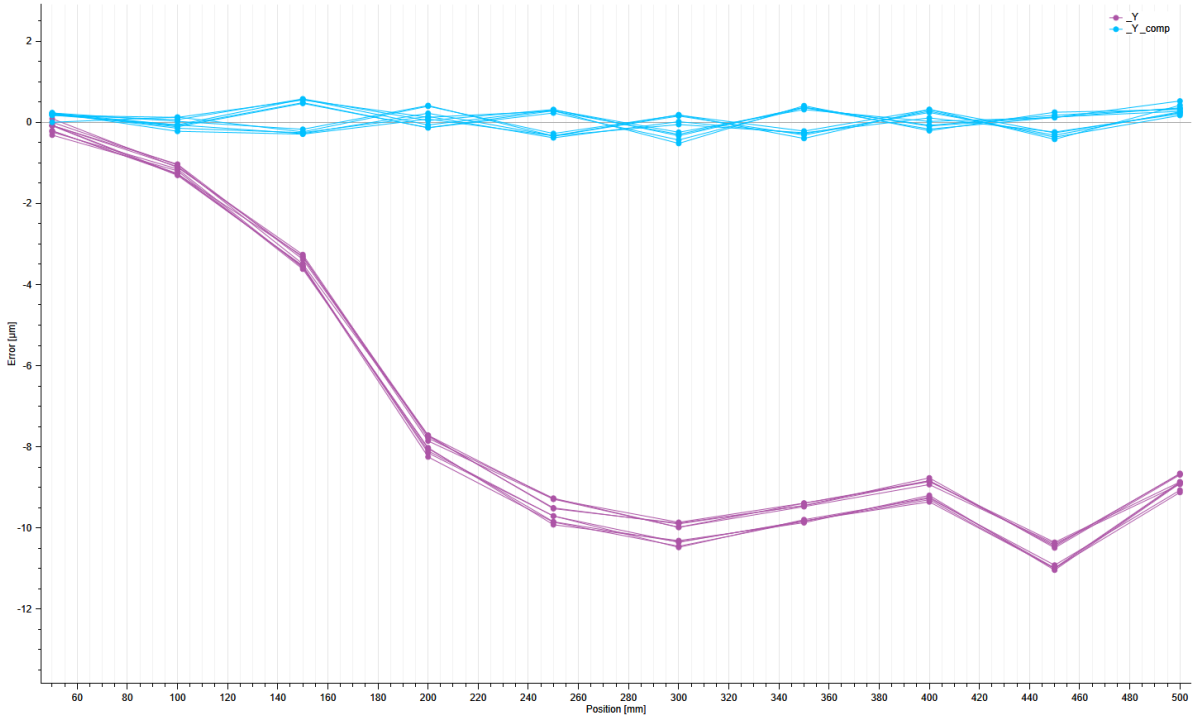


4.3.2 Osa Y

Tabulka 4.7: Přesnost polohování v ose Y dle ISO 230–2

Osa Y	PŘED [µm]	PO [µm]
Positioning error (A)	11.19	1.22
Positioning repeatability (R)	0.88	0.97
Systematic positioning error (E)	10.96	0.90
Reversal (B)	0.56	0.78
Mean reversal	0.34	-0.16
Range of mean bidirectional positioning error (M)	10.58	0.45

Obrázek 4.14: Porovnání přesnosti před a po kompenzování osy.



4.3.3 Osa Z

Tabulka 4.8: Přesnost polohování v ose Z dle ISO 230–2

Osa Z	PŘED [µm]	PO [µm]
Positioning error (A)	10.77	1.18
Positioning repeatability (R)	1.53	1.06
Systematic positioning error (E)	9.79	0.94
Reversal (B)	0.67	0.86
Mean reversal	-0.17	-0.09
Range of mean bidirectional positioning error (M)	9.46	0.60

Obrázek 4.15: Porovnání přesnosti před a po kompenzování osy.

4.4 Závěr pro měření

Byla změřena přesnost lineárního polohování všech tří os pomocí laser interferometru XL-80 of firmy Renishaw. Data byla vyhodnocena pomocí normy ISO 230–2. Zjištěné chyby polohování byly mimo tolerance stanovené výrobcem 4.1 a bylo proto přikročeno k jejich softwarové kompenzaci. Chyby polohování se po zkompenzování dostaly prakticky na hranici rozlišovací úrovně měření — na hranici 1 µm, což je skvělý výsledek. V následující tabulce je zobrazen souhrn výsledků pro všechny osy.

Tabulka 4.9: Souhrn výsledků: chyba polohování všech os

Positioning error (A)	PŘED [µm]	PO [µm]
Osa X	19.94	1.50
Osa Y	11.19	1.22
Osa Z	10.77	1.18

Dvou-díl
V kapitole 4.2.1 bylo zmíněno, že sada příslušenství k laseru, která je na ústavu dostupná, obsahuje pouze dvou-díl polopropustných zrcátek. Při použití tohoto dvou-dílu dochází vícekrát k zalomení paprsku a ztrátě signálu v důsledku jednoho polo-propuštění paprsku navíc. Bylo by vhodné provést experiment s cílem zjistit vliv použití tohoto dvou-dílu oproti použití jediného zrcátka.

5 Návrh řešení

V této kapitole je představen návrh, jaký systém zvolit pro monitorování výrobních strojů. V první části kapitoly je rozebrán zadaný problém optikou systémového přístupu. Jsou identifikovány důležité charakteristiky hledaného systému monitorování a požadované vlastnosti fyzického řešení.

V další části je z těchto výchozích úvah vyvozen vhodný systém pro monitorování a technický způsob jeho realizace. Je navržen plán realizace v jednotlivých fázích. Jsou doporučeny další kroky po rozvoj tohoto systému nad rámec této závěrečné práce.

5.1 Cíle řešení

Mezi hlavní cíle této závěrečné práce patří navrhnout vhodný způsob monitorování a vyhodnocení výrobní přesnosti zvoleného CNC stroje.

V kapitole 3.1.1 bylo definováno, že cílem monitorování je dlouhodobé sledování objektu s cílem predikce jeho budoucích stavů. Řešení by tedy mělo umožňovat sběr a analýzu dat takovým způsobem, aby bylo možné predikovat v určité míře vývoj zvolených charakteristik výrobního stroje.

V kapitole 3.2 bylo popsáno, že výrobní přesnost je souhrnný pojem, zastřešující mnoho jednotlivých charakteristik. Monitorovat je proto potřeba jednotlivé charakteristiky a získaná data zpracovávat takovým způsobem, aby bylo možné získat informaci o celkové výrobní přesnosti.

5.1.1 Zvyšování výrobní přesnosti

Dlouhodobým záměrem subjektů, které provozují výrobní stroje, však není jen výrobní přesnost monitorovat, ale zvyšovat ji za účelem tržní výhody a v důsledku za dosažení vyšších finančních příjmů. Na zvyšování výrobní přesnosti je v současné době kladen velký důraz. S celkovým technickým pokrokem ve světě a s rozvojem firem v konkurenčních ekonomikách se zvyšuje tlak na efektivitu strojů a procesů napříč různými odvětvími. Pro dosažení vyšší efektivity je ve většině případů potřeba zvyšovat přesnost dílů, které jsou součástí technických děl. Například pro dosažení úspor při výrobě elektrické energie v tepelných elektrárnách je potřeba zvýšit účinnost parních turbín. Toho lze dosáhnout jednak koncepčním zlepšením konstrukce, ale také při použití přesnějších dílů. Z toho plyne požadavek na přesnou výrobu třískového obrábění.

Platí to i v kontextu ÚVSSR při FSI na VUT, kdy v současné době je na tomto pracovišti v řešení několik projektů — českých i evropských, které mají za cíl zvyšování výrobní přesnosti. Zmínit lze například evropský projekt Level-Up, který se zaměřuje na velké výrobní stroje a jehož cílem je mimo jiné vývoj řešení pro monitorování, predikci a zvyšování výrobní přesnosti [33].

Z těchto důvodů se předkládanému řešení dává za cíl nejen monitorování výrobní přesnosti, ale i podpora takových činností, které vedou k jejímu zvyšování.

Softwarová kalibrace
Zvyšovat výrobní přesnost lze pouze zvyšováním přesnosti dílčích vlivů, které byly popsány v kapitole 3.2.3. Bylo ukázáno, že mezi významné faktory patří geometrická přesnost, a ta je zase významně ovlivněna přesností polohování. Řešení by tedy mělo podporovat činnosti, které vedou ke zvyšování těchto faktorů. Geometrickou přesnost, resp. přesnost polohování lze efektivně zvyšovat pomocí softwarové kalibrace.

5.1.2 Efektivita řešení

Řešení by mělo být co nejvíce efektivní, a to z různých hledisek, mezi která patří hledisko personální, finanční, materiálové, hledisko údržby, aj.

Hledisko lidských zdrojů zohledňuje, kolik lidí a jaké odbornosti je potřeba k pořízení a provozování daného řešení. Autor této závěrečné práce má v záměru na projektu pokračovat na doktorském studiu, přičemž potenciál a komplexita projektu budou vyžadovat spolupráci na některých částech s kolegy z ústavu. V tomto smyslu personální hledisko znamená, že řešení by mělo být navrženo takovým způsobem, aby při případné spolupráci představovalo co nejmenší časovou a mentální zátěž pro kolegy. Z pohledu firem je toto hledisko ještě důležitější. Personální náročnost totiž může znamenat rozdíl mezi použitím a nepoužitím tohoto řešení. Firmy se chovají tržně efektivně a lze proto předpokládat, že všichni odborní pracovníci jsou ve firmách plně vytíženi. To znamená, že řešení které by vyžadovalo značný časový fond pracovníků, by pro některé firmy mohlo být nedostupné a to i s ohledem na to, že v současné době není odborných pracovníků na trhu dostatek.

Finanční hledisko — řešení by mělo být finančně dostupné co do pořizovacích i režijních nákladů. Funkční řešení, pokud by bylo neúměrně drahé, by nebylo firmami přijato. Proto, pokud je snaha o vytvoření řešení, které bude prakticky použitelné pro firmy, je třeba dbát na to, aby byla jeho finanční náročnost relativně nízká oproti přínosům.

Materiálové hledisko zohledňuje jaké materiály a v jakém množství jsou pro provoz řešení potřeba. Souvisí to nepřímo s požadavky na prostor pro uložení materiálu (skladovací plocha), s finančními náklady, atd. Požadavky na materiál ale znamenají i jinou formu zátěže. Představují totiž formu závislosti na dodavateli materiálu. Například výrobci lithium iontových baterií jsou existenčně závislý na dodávkách lithia a některých těžkých kovů. Lithium, přestože je běžným prvkem co do přirozeného výskytu v přírodě v minerálech, se v ekonomicky dostatečných koncentracích vyskytuje jen v několika lokalitách na světě. Představuje tedy strategické riziko pro podnikání. V tomto smyslu by řešení mělo využívat v co největší míře běžně a snadno dostupné materiály.

Hlediskem údržby se myslí požadavky na provoz a potřebnou míru údržby. Opět to nepřímo souvisí s ostatními hledisky, především s personálním. Řešení by mělo být co nejvíce autonomní a nutná míra průběžné údržby co nejmenší.

5.1.3 Odolnost řešení

Řešení by mělo být odolné. Technické provedení by mělo být odolné vůči chybám od obsluhy a tedy co nejvíce automatizované a nezávislé na obsluze. Současně by mělo být odolné vůči technickým závadám, které se mohou vyskytnout v celém řetězci zapojených komponent. Měly by

být vyřešeny i mimořádné stavy, jako je například výpadek proudu. V takovém případě dochází zpravidla k zastavení celé výroby, která nebývá připojená na záložní zdroje energie. Požadavkem v takové situaci je, aby se po obnovení dodávek proudu řešení dostalo do stavu před výpadkem a aby nebylo nutné něco znovu nastavovat, aby nedošlo ke ztrátě dat a aby řešení bylo schopné okamžitého provozu. Takováto odolnost se označuje jako resilience. Rozdíl mezi pojmy odolnost a resilience spočívá v časové charakteristice. Odolnost vyjadřuje schopnost okamžitě odolat nastalé situaci. V případě výpadku proudu se odolnost dá zajistit například použitím záložního zdroje energie. Jde tedy o schopnost systému nebýt postižen okolnostmi. Naproti tomu resilience vyjadřuje schopnost odolat v dlouhodobém horizontu, jinak řečeno, vyjadřuje schopnost vzpamatovat se z působení okolností. V případě výpadku proudu se resilience systému projeví tak, že po obnovení dodávek proudu se systém dostane do předešlého stavu. Pojem resilience se poprvé začal uplatňovat v oboru ekologie krajiny, kdy se jím charakterizuje schopnost krajiny, například lesa, vzpamatovat se z požáru. Resilientní eko-systémy se po požárech, povodních, atd. po krátké době obnoví do původního stavu. Stejně jako v případě eko-systémů bychom měli dbát jak na odolnost, tak ale i resilienci i u technických soustav.

5.1.4 Bezpečnost řešení

Řešení by mělo být bezpečné a to z pohledu vnitřní i vnější bezpečnosti. Vnější bezpečnost znamená schopnost mít negativní dopad na okolí podniku a to jak okolí ekologické, personální, atd. Vnitřní bezpečnost potom analogicky schopnost negativně ovlivnit vnitřní okolí podniku. Řešení musí splňovat všechny požadavky na funkční bezpečnost a nad rámec toho i požadavky na bezpečnost kybernetickou a informační. Kybernetická bezpečnost vyjadřuje odolnost vůči úniku dat, vyřazení z provozu a ztrátě kontroly či řízení nad systémem.

5.1.5 Požadavky na řešení

Řešení by mělo umožňovat dlouhodobé monitorování různých parametrů stroje a mělo by umožňovat jejich analýzu různými metodami. Mělo by být snadné toto řešení zavést do výroby nových strojů, ale také dodatečné implementovat na stávajících strojích. Systém by měl být co nejvíce modulární, aby umožňoval zapojení různých stran a tedy umožňoval efektivně skrývat komplexitu jednotlivých částí.

Z výše uvedených důvodů vyplývají tyto požadavky na řešení monitorovacího systému:

- monitorování výrobní, resp. geometrické přesnosti,
- predikce vývoje na základě analýzy dat,
- kalibrace strojů na základě dat,
- flexibilita z hlediska zdrojů dat,
- modularita z hlediska použití jiných řešení.

5.2 Návrh monitorovacího systému

5.2.1 Informační systém

Z definovaných požadavků a zkušeností z praxe plyne, že vhodným řešením pro navrhovaný monitorovací systém je systém informační v užším smyslu počítačového systému. O dnešní době se mluví jako o informačním věku a je to pochopitelné. Lze se přít, zda by nebylo vhodnější užívat pojmu datový věk, ale podstata zůstává stejná. Množství generovaných dat a informací je dnes obrovské. V posledních letech bylo možné sledovat rychlý rozvoj technologií pro získávání dat, jejich zpracování a interpretaci a také jejich sdílení. Často je množství dat v oběhu terčem kritiky a mnohdy oprávněné kritiky. Je tomu tak nejčastěji v případě osobních dat, se kterými nakládají společnosti provozující sociální sítě. V mnohých jiných případech je však velké množství dat jediným klíčem k úspěchu. Mezi tyto případy lze například zahrnout celou oblast vědy a výzkumu, kde jsou všechna data užitečná. Platí to obzvláště s rozvojem metod pro strojové zpracování dat — tedy tzv. strojové učení.

Pozorovat lze i rozvoj na poli technických prostředků. Mnohé nástroje pro měření veličin se s postupem vývoje vtěsnaly do jediného mikročipu a škála veličin, které dnes lze měřit je velká. Využití těchto prostředků pro monitorovací systém je jen logické. Monitorovací systém už ze své definice naplňuje podstatu data zpracovávajícího serveru. Nejefektivnějšími prostředky pro tyto účely jsou dnes počítače.

5.2.2 Počítačové systémy

V předchozím odstavci bylo navrženo využít pro vytvoření monitorovacího systému počítače. Počítače lze využívat různými způsoby. Od izolovaného používaní jediného zařízení až po rozsáhlé soustavy počítačů — v oboru označované jako počítačové sítě. Počítače dnes mají také velké množství podob. Kromě klasických notebooků a stolních počítačů, přes servery až po miniaturní zařízení typu Arduino-Nano [3].

Je tedy otázkou, jakou počítačovou síť zvolit a jakým způsobem ji organizovat tak, aby byly v co největší míře dodrženy požadavky vyjmenované v kapitole 5.1. Podstatné z hlediska organize počítačové sítě jsou především požadavky na modularitu a flexibilitu řešení. Z povahy řešeného problému je patrné, že nelze vybrat konečné množství jasně definovaných zařízení a neměnným způsobem je organizovat do nějaké sítě. Přesně naopak je potřeba zvolit takové řešení, které umožní flexibilně přidávat a odebírat zařízení dle aktuální povahy řešených dílčích problémů. Příkladem může být rozšíření systému o novou charakteristiku výrobního stroje. V takovém případě je potřeba přidat zařízení pro sběr dat — nový senzor a nově získávaná data v systému začlenit do data zpracovávajícího řetězce.

5.2.3 Cloudová architektura

Postupná evoluce počítačových sítí napříč odvětvími ukazuje, že nejvhodnějším způsobem jejich organizace je tzv. cloudová architektura, nebo zkráceně jen cloud.

Cloud, tento pojem označuje způsob organizace počítačové soustavy, která je pro uživatele poskytována na vyžádání jako služba. Nejčastějšími dvěma funkcemi, které jsou v rámci

cloudu poskytovány jako služby jsou úložiště a výpočetní výkon. To, že jsou tyto funkce poskytovány jako služba znamená, že komplexita provozování těchto služeb jakožto počítačových sítí, je před uživatelem skryta. Příkladem z běžné praxe je e-mail. Jedná se o cloudovou službu, která je uživateli zprostředkována skrze webové rozhraní, nebo klientskou aplikaci. Pro provozování e-mailové služby jsou potřeba servery, databáze, síťová architektura, propojovací uzly, atd., avšak před uživatelem je tato komplexita zcela skryta.

Cloudová architektura umožňuje flexibilně přidávat a odebírat zařízení či celé uzly dle potřeby. Všechny nezbytné úkony s tím spojené, jako je konfigurace sítě, naprogramování nových funkcí, aj. zastanou členové týmu, který má provoz cloudu na starost. Nově vytvořenou funkci následně zpřístupní uživatelům jako službu. Příkladem může být universitní informační systém, který je spravován oddělením CVIS, které nese na svých bedrech celou komplexitu systému a výsledné funkce poskytuje zaměstnancům a studentům university jako službu. V podnicích to funguje stejným způsobem. Takto před uživateli skrytá architektura se označuje jako cloud. Od běžné počítačové sítě se odlišuje tím, že počítače pouze nepropojuje, ale zpřístupňuje uživatelům sítě služby, které běží na serverech.

Pro řešení předkládaného problému je proto zvolena právě cloudová architektura. V další části textu je popsán výběr konkrétní realizace a technologie.

5.3 Cloud

Tato kapitola se zabývá definicí cloudu, jeho základní charakteristikou, výhodami a volbou konkrétní technologie pro řešení předkládaného problému.

Služby poskytované cloudem jsou dvojího základního typu. Jsou to datové úložiště a výpočetní výkon. V případě datového úložiště je jako služba poskytováno úložiště ve formě databáze, nebo souborového systému. Příkladem může být Microsoft OneDrive nebo DropBox či Google Drive. Zákazník má k dispozici úložiště jako placenou službu a je přitom zcela oproštěn od komplexity provozování diskových polí.

Druhou základní službou je výpočetní výkon. V tomto případě je poskytován procesorový čas pro běh aplikací, avšak procesor se nachází u provozovatele cloudu a nikoli v počítači, odkud je program spouštěn. Příkladem může být webová aplikace WolframAlpha, která funguje jako chytrá kalkulačka [37]. Lze zadávat i výpočetně velmi náročné úlohy. Jejich výpočet probíhá na procesorech v cloudu poskytovatele služby a na rozhraní webové aplikace se zobrazují pouze výsledky.

5.3.1 Stručná historie cloudu

Cloud v dnešním pojetí se začal jako pojem masivně šířit až v několika posledních letech. Jeho první užití ve smyslu dnešních komerčních cloudů se datuje do roku 2006, kdy firma Amazon spustila svůj první cloud s názvem Elastic Compute Cloud. Doložené užití pojmu cloud-computing však lze dohledat již v roce 1996 v interních materiálech firmy Compaq. Úplně prvotní výskyt však lze dovodit dokonce ještě před vznikem internetu v téže organizaci, která jej vymyslela — DARPA. Ta použila symbol cloudu pro znázornění svých počítačových sítí, fakticky vzato předchůdců cloudu jako takového.

Když vezmeme do úvahy, že definice cloudu je poskytování funkcí počítačových sítí jako služeb, lze za cloudové systémy označit již tyto předchůdce, přestože dnešní služby jsou na mnohem vyšší technické úrovni. Jak lze vidět, cloud je ve své podstatě jen moderní pojem pro starý koncept. Oprávněnost jeho dnešního používání spočívá právě v technické vyspělosti poskytovaných služeb.

5.3.2 Způsob realizace cloudu

Cloud se dá realizovat třemi základními způsoby podle toho, kdo zodpovídá za jeho provoz. Provozem se v tomto smyslu myslí, kdo je vlastníkem, resp. správcem hardwaru — tedy serverů, datových úložišť, síťové infrastruktury, atd. Kdo zodpovídá za běh a správu softwaru, který je nutný k provozu celého systému a kdo je zodpovědný za kybernetickou bezpečnost celého řešení. Jsou tři režimu provozovatelů cloudu:

- Podnikový/privátní cloud (Enterprise/Private Cloud)
- Veřejný cloud (Public Cloud)
- Hybridní cloud (Hybrid Cloud)

Public cloud
Veřejný cloud je cloud provozovaný poskytovatelem pro účely pronájmu cloudových kapacit. Veškerou komplexitu výstavby, provozu a zabezpečení zajišťuje poskytovatel. Cloudové kapacity jsou poskytovatelem nabízeny jako placená služba, nejčastěji v režimu pay-as-you-go, ve volném překladu plať průběžně jen to, co skutečně využiješ. To je způsob účtování, kdy poskytovatel v průběhu celého měsíce technickými prostředky měří skutečné využití jednotlivých služeb, v případě datového úložiště v podobě souborového systému měří celkové množství přenesených bajtů dat v obou směrech, v případě databázového úložiště se měří například počet dotazů na databázi, v případě výpočetních kapacit se měří počet provedených instrukcí procesoru. Zákazník tak platí skutečně podle toho, v jaké míře funkce cloudu využívá.

Příkladem poskytovatelů veřejných cloudů pro soukromé účely jsou: Google — Google Drive, G-Suite aplikace, DropBox, Microsoft — Office-365, a další. Příkladem veřejných poskytovatelů pro komerční sektor a profesionální použití jsou: Amazon — celosvětově největší poskytoval, Microsoft — druhý největší poskytoval a Google na třetí pozici.

Hlavní výhody veřejného cloudu při srovnání s podnikovým cloudem [9]:

- Cena: Při nestálém využívání zdrojů, kdy dochází k výkonnostním špičkám jen v určitých časech, se díky modelu pay-as-you-go platí jen skutečně využité zdroje, což může být levnější, než nákup počítačů, které musí dostačovat pro tyto výkonnostní špičky, ale ve zbytku času budou využity jen na malé procento.
- Škálovatelnost: U nových projektů se může stát, že je potřeba začít jen s určitým omezeným výkonem, ale postupně je potřeba výkon zvyšovat, nebo naopak není jisté, zda bude maximální výkon potřeba i v budoucnu, což by s sebou neslo potřebu provozování serverů v maximálním rozsahu na počátku a následný prodej přebytečné kapacity. Výhoda veřejného cloudu je, že lze měnit využívanou kapacity prakticky ze dne na den přesně podle potřeby.
- Dostupnost: Pro společnosti či týmy, které nemají pracovníky s potřebnou odborností, nebo třeba v případě, že jediný pracovník potřebuje na omezenou dobo určité služby, může být veřejný cloud jediným řešením.

- Konfigurovatelnost: Poskytovatelé nabízí množství variant s předinstalovanými aplikacemi, takže pro mnohé běžné práce není potřeba pracovní prostředí složitě nastavovat.
- Široká nabídka: Přes veřejné cloudy se lze dostat k široké nabídce systémů, jak po hardwarové stránce, kdy lze provozovat minimalistické, nebo naopak velmi výkonné soustavy, tak i po stránce softwaru. Výhodou je také možnost měnit využívaný hardware podle potřeby.
- Bezpečnost: Poskytovatelé investují obrovské prostředky do zabezpečení svých cloudů a to po softwarové stránce z hlediska kyberbezpečnosti, tak i po stránce fyzické, což znamená ochranu proti vniknutí cizí osoby, odolnost proti živelným katastrofám, atd.
- Aktuálnost: Provozovatelé veřejných cloudů taktéž dbají na aktuálnost svých soustav. Aktuální se udržuje software i hardware. Dobrým příkladem jsou třeba disková pole, kde jsou neustále vyměňovány jednotlivé disky, když dojde k ukončení jejich funkčnosti. Zákazník však má datové úložiště díky redundanci stále k dispozici.
- Jednoduchost: Provoz vlastního cloudu je náročný na odbornost pracovníků zajišťujících provoz. Naproti tomu veřejné cloudy mají jednoduché webové rozhraní, pomocí kterého lze ovládat všechny základní funkce.
- Geografická dostupnost. Při poskytování služeb koncovým zákazníkům je výhodné využít služeb veřejných cloudů, které mají datová centra dostupná na všech kontinentech a snížit tím časovou prodlevu služeb. Zároveň použitím několika geografických lokalit dochází pomocí redundance ke zvýšení spolehlivosti služeb. Při lokálním výpadku proudu tak nejsou služby nedostupné.

Hlavní nevýhody veřejného cloudu při srovnání s podnikovým cloudem [9]:

- Cena: Cena za služby veřejných cloudů může být příliš vysoká třeba v případech, kdy výkonnostní požadavky jsou konstantní. V takovém případě může být výhodnější potřebný hardware provozovat ve vlastním režimu. Další nevýhodou je obtížnost odhadu nákladů. Při systému pay-as-you-go není nikdy možné přesně určit, jaké budou úhrnné roční náklady. Při špatném odhadu to může působit zásadní finanční problémy.
- Výkonnost: Veřejné cloudy zatím neposkytují velmi vysoký výkon na úrovni superpočítačů. V případě, kdy je potřeba mít velmi vysoký výkon, je výhodnější obrátit se na poskytovatele superpočítačů, nebo vybudovat vlastní infrastrukturu.
- Bezpečnost: Přestože bezpečnost veřejných cloudů je na velmi vysoké úrovni a jsou například v souladu s HIPAA normou, může být velmi obtížné z hlediska legislativy, vyřídit všechna potřebná povolení a detaily při provozování velmi citlivých údajů. Speciálně v EU jsou pro některá data veřejné cloudy zákonem vyloučeny, přestože bezpečnostní požadavky fakticky splňují.
- Závislost: Využívání služeb veřejného cloudu znamená vytvoření závislosti na jejich poskytování. Přestože se v současné době nejeví reálně možnost, že by kterýkoli z velkých poskytovatelů mohl ukončit svou činnost, může se stát třeba to, že přestane podporovat některé konkrétní funkce, které zákazník používá.

Privátní cloud

V režimu provozování privátního cloudu se výše uvedené výhody a nevýhody prakticky vzato prohazují. Při provozování soukromého cloudu je veškerý hardware ve vlastnictví a v odpovědnosti společnosti. Společnost musí zároveň řešit veškerou komplexitu s tím spojenou. Musí zajišťovat nákup, provoz a obměňování hardwaru. Musí mít odborné pracovníky pro provoz

všech úrovní softwaru od operačních systémů, hypervizorů, síťové infrastruktury až po běhová prostředí jednotlivých aplikací. Musí zároveň zajistit bezpečnost dle všech platných norem.

Výhodou soukromého cloudu je plná moc nad jeho provozem a užíváním. Lze vytvořit cloud přesně na míru potřebám společnosti. Výhodou, především z pohledu legislativy, je, že všechna data jsou uložena pouze uvnitř podniku, resp. na podnikem definovaných místech. V případě citlivých dat, například lékařských záznamů, to může být jediný možný režim provozu.

Nevýhodou je pak především to, že společnost musí sama řešit veškerou komplexitu celého řešení. V případě malých podniků to může být neřešitelný problém. Například začínající společnost o 20 lidech bude těžko platit 20 vývojářů pouze pro potřeby cloudové infrastruktury. V tomto směru se situace postupně zlepšuje s tím, jak se více věcí pro provoz cloudu automatizuje a tím se snižuje celková komplexita provozu. Je možné, že v budoucnu bude stačit jen několik málo lidí, či snad jediná osoba k provozování soukromého cloudu. Dnes to neplatí.

Hybridní cloud
Hybridní model je kombinací obou přístupů. Je využíván tedy jak veřejný, tak podnikový cloud. Služby běžící na jednotlivých cloudech mohou být, ale nemusí být provázané. Může se jednat i o zcela izolované aplikace. Příkladem může být firma, která pracuje s citlivými daty svých zákazníků a tato data spravuje výhradně na vlastním podnikovém cloudu. Kromě toho svým zákazníkům poskytují software, který pro celosvětovou dostupnost a vysokou spolehlivost (uptime) provozují na službách AWS od Amazonu.

Tento model se začíná prosazovat jako dominantní řešení, protože flexibilně kombinuje výhody obou modelů, přičemž žádné funkce se vzájemně nevylučují.

5.3.3 Režimy poskytování služeb

S postupným vývojem se dříve složité věci zjednodušují na obchodní model služby. I v případě počítačových sítí byl proces přechodu od zcela vlastních řešení k plným cloudovým službám postupný a jednotlivé kroky vývoje lze charakterizovat níže uvedeným způsobem. Podle tohoto rozdělení podle rozsahu a povahy poskytovaných služeb lze i dnes postupovat při výběru dodavatele. Jednotlivé režimy pracují s různou mírou abstrakce nad nižší vrstvou.

IaaS
Neboli Infrastructure-as-a-Service, tedy infrastruktura jako služba. V tomto režimu je zákazníkovi formou služby poskytována infrastruktura datového centra, tedy servery, na kterých může provozovat libovolný software, včetně operačních systémů. Správu samotné infrastruktury však řeší poskytovatel a zákazník je tedy na úrovni abstrakce nad hardwarem. Zákazník se však musí starat o všechny náležitosti softwaru. Pokud tedy například provozuje celé operační systémy, zodpovídá si za jejich údržbu sám.

PaaS
Platform-as-a-Service, neboli platforma jako služba. Je další úrovní abstrakce nad operačními systémy. V tomto režimu se tedy poskytovatel stará i o operační systémy, které jsou zákazníkovi poskytovány jako služba. Zákazník se v tomto režimu stará pouze o nasazování samotných aplikací. Každá platforma poskytuje vlastní softwarové řešení (něco na způsob SDK), pomocí

kterého lze aplikace nasazovat. Je potřeba definovat běhové prostředí a zdroj aplikace. Platforma se už potom sama postará o instalaci závislostí a běh aplikace. To firmám umožňuje, soustředit se výhradně na vývoj aplikací.

SaaS
Software-as-a-Service. Neboli software jako služba. Jedná se o režim, kdy je zákazníkovy poskytován formou služby již hotový software. Příkladem jsou všechny běžné služby jako e-mail, google dokumenty, WolframAlpha, aj.

Serverless
V překladu bez-serverová služba ve skutečnosti neznamená, že by v procesu nebyly použity servery. Je to odkaz na fakt, že zákazník je od serverů a veškerého provozu na serveru zcela abstrahován a využívá službu k běhu konkrétních funkcí. O kompletní běhové prostředí, včetně škálování, se stará poskytovatel serverless služby a zákazník je účtován nikoli podle instancí/-hodin, ale podle počtu iterací celé funkce.

5.3.4 Volba cloudového řešení

Výborně volbu cloudového řešení rozvádí společný článek firem Siemens a Microsoft, tedy světově významné firmy v oboru strojírenství a cloudových technologií. Článek se zabývá porovnáním mezi vlastním řešením cloudu — tedy stavbou vlastní fyzické infrastruktury a pořízením cloudových služeb od dodavatele.

Uvádí se zde, že s postupným rozvojem konceptu chytré továrny se objevují stále širší možné využití IIoT (Industrial Internet of Things). V současném zapojení technologií strojového učení, umělé inteligence a cloudu lze dosáhnout výrazných zlepšení ve výrobě a, trochu nadsazeně, revolucionizovat celá odvětví. Uvádí se analýza firmy do firmy Bain & Company, která pro rok 2021 predikuje celosvětově obrat v oboru IIoT ve výši 200 miliard amerických dolarů (Odhad před Coronavirovou krizí). Pro realizaci konceptu chytré továrny je však nezbytnou podpůrnou technologií cloud [11].

Výhody a nevýhody vlastního řešení vs. předplatného na poskytovaný cloud mají těžiště v míře komplexity, se kterou se musí firma popasovat. V případě vlastního cloudu musí téměř celou tíhu komplexity řešit sama. Vybudovat vlastní cloud znamená vybudovat k tomu určené prostory, kterých může být kvůli zajištění spolehlivosti a dostupnosti i více po světě. Výběr, nákup, instalace a provoz potřebného hardwaru stejně jako zaměstnávání k tomu určených odborníků. Provoz a údržba operačních systémů, infrastruktur pro jejich správu (typicky dnes zahrnuje několik paralelně provozovaných technologií: Terraform, Puppet, OpenStack, Docker, Kubernetes, Rancher, nebo případně jejich alternativy) a opět množství odborníků pro jejich provoz. Firma musí sama řešit kybernetickou bezpečnost, která se zabývá v kontextu podniků především ochranou a obranou proti krádeži a zneužití firemních dat, proti průniku do firemních systémů a v důsledku proti poškození, vyřazení z provozu nebo ovládnutí kybernetických systémů a v neposlední řadě proti útokům typu DDoS, nebo proti ransomwarovým útokům. K tomu je opět potřeba množství odborníků na danou problematiku.

Z provozování vlastního-tedy privátního cloudu však plynou i některé významné výhody. Mezi ty patří úplná kontrola nad fyzickou i softwarovou vrstvou, možnost úplného přizpůsobení vlastním potřebám (při alokování potřebných zdrojů) a důvěra v uložení dat. Poslední bod

je ovšem značně diskutabilní, protože na jednu stranu má firma sice jistotu, že se data nachází pouze na privátním úložišti, na stranu druhou je dnes netriviální zajistit kybernetickou bezpečnost na takové úrovni, která bezpečně zamezí krádeži dat. Zjištěných úniků firemních informací je poměrně velké množství a k únikům dochází i u renomovaných firem, které provozují privátní úložiště. Naopak poskytovatelé veřejných cloudů investují do zajištění kybernetické bezpečnosti obrovské sumy (Microsoft kolem jedné miliardy dolarů ročně). Je to dáno tím, že si nemůžou dovolit ztratit důvěru zákazníků tím, že by k úniku skutečně došlo. Naproti tomu v případě privátních cloudů není jeho provoz prvořadým cílem firmy a tudíž investice do kyberbezpečnosti často nemají dostatečnou prioritu u vedení podniku. Často potom dochází k systematickému zanedbávání kybernetické bezpečnosti a následným útokům na tyto instituce. Jedním vzorovým příkladem mohou být privátní cloudy českých nemocnic (Často se jedná spíše jen o komplexní počítačové sítě a ne cloudy, přesto princip systematického zanedbávání platí ve stejné míře).

V případě kdy není nezbytně nutné provozovat privátní cloud, se proto doporučuje využít služeb některého poskytovatele. To umožňuje firmě koncentrovat úsilí svých zaměstnanců na vývoj pouze těch aplikací, které potřebují a veškerou komplexitu provozování cloudu a řešení kybernetické bezpečnosti přenechat na poskytovateli cloudu.

5.3.5 Průmyslový cloud

Strojírenství je v tomto směru obor specifický tím, k čemu je cloud používán a konkrétně tím, jaká zařízení se do cloudu připojují. Je potřeba řešit kompatibilitu, přičemž průmyslové soustavy často používají specifické protokoly.

Z tohoto důvodu je vhodné zvolit poskytovatele, který má s průmyslovým a úžeji strojním oborem zkušenosti a umí se specifickými protokoly pracovat. Takovou společností je například Siemens nebo Fanuc, přičemž oba jsou výrobci řídicích systémů pro výrobní stroje a potřebného hardwaru. Obě firmy mají silné renomé. Firma Siemens má nepoměrně širší záběr, který se neomezuje jen na energetiku, námořní dopravu, kolejní dopravu, letecký průmysl, aj. Zároveň byl první firmou, která nabídla úplné cloudové řešení, které je možné využívat ve formě PaaS nebo SaaS.

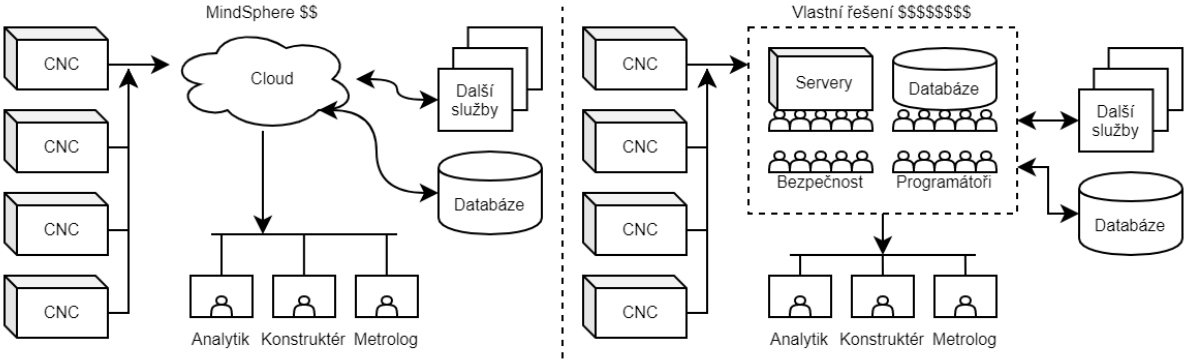
Vhodnou volbou pro řešení monitorovacího systému je tedy veřejný cloud s možností pozdějšího rozšíření na hybridní model. Režim použití se doporučuje PaaS, což umožňuje tvorbu vlastních aplikací na míru a jejich modifikaci v pozdější době. Jako poskytovatel PaaS je zvolena Siemens. PaaS od Siemensu se jmenuje MindSphere. Pro řešení této závěrečné práce a i pro navazující použití se doporučuje PaaS MindSphere.

5.4 MindSphere

Na stroji je řídicí systém Sinumerik od firmy Siemens, proto bylo jako řešení cloudové architektury zvolen systém MindSphere. MindSphere je cloudový IoT operační systém [26]. Je to PaaS řešení, což znamená, že zákazníkům je poskytována platforma jako servis (Platform as a Service). Umožňuje připojení různých počítačových systémů přes internet k serverům, na kterých běží aplikace. Výhoda PaaS řešení je, že potřebná hardwarová architektura na straně serverů a softwarová architektura sítě je poskytována jako hotové řešení od poskytovatele PaaS. Není

tedy nutné vlastnit a provozovat vlastní servery. Taky není nutné zajišťovat běhové prostředí na těchto serverech. Stačí připojit koncové body a naprogramovat aplikace pro zpracování dat.

Obrázek 5.1: Výhody MindSphere



5.4.1 Cloud Foundry

Softwarová architektura MindSphere je postavena na řešení Cloud Foundry. Cloud Foundry je open-source PaaS software. Pro pochopení funkce MindSphere je tedy nutné zabývat se nejdříve řešením Cloud Foundry.

Historie
Cloud Foundry bylo založeno v roce 2009. Počáteční návrh a vývoj vedl Derek Collison ve společnosti VMware, která se dnes věnuje především cloudovým výpočetním a virtualizačním technologiím. Projekt se původně jmenoval B29. O rok dříve, tedy v roce 2008, začala pracovat na PaaS technologii i firma SpringSource, která jej pojmenovala Cloud Foundry. SpringSource vyvíjel aplikace v programovacím jazyce Java pro nasazení na Amazon EC2. V roce 2009 VMware tuto společnost koupil a její projekt Cloud Foundry ukončil, avšak převzal jeho název a svůj vlastní projekt vedený Derekem se přejmenoval na Cloud Foundry.

Veřejné ohlášení projektu proběhlo v roce 2011. O rok později byl ohlášen nástroj BOSH. BOSH slouží pro správu a údržbu cloudových aplikací velkého rozsahu (large scale distributed services). V roce 2013 byla založena dceřiná společnost Pivotal za účelem monetizace Cloud Foundry, RabbitMQ a Spring systémů.

V roce 2014 bylo rozhodnuto o otevření kódu projektu Cloud Foundry a jeho osamostatnění pod správou nově založené organizace, která měla při založení kromě samotného Pivotalu sedm platinových a dva zlaté členy (EMC, HP, IBM, Rackspace Hosting, SAP a VMware a zlatí členové ActiveState a CenturyLink). Organizace byla následně založena v lednu 2015 jako nezisková projekt 501(c) (6) pod organizací Linux Foundation. K únoru 2019 měla organizace 65 členů, přičemž mezi platinové přibyli Dell, Google, IBM, SAP a SUSE. Mezi stříbrné například Microsoft. Siemens, přestože tento software využívá jako své výhradní řešení softwarové architektury MindSphere, není členem. Avšak stříbrným členem je společnost Mendix, která je dceřinou společností Siemens po akvizici v roce 2018. Cloud Foundry bylo první kompletní řešení PaaS na trhu.

Použití
Cloud Foundry je založeno na kontejnerové architektuře, která umožňuje nasazení aplikací v různých programovacích jazycích. Runtime je možné nasadit na hardwarovou architekturu mnoha poskytovatelů, mezi nimi například Amazon Web services, Microsoft Azure, IBM Cloud, Google Cloud Platform nebo Alibaba Cloud. Kromě nasazení runtime-u pro konkrétní programovací jazyky je možné nasadit runtime pro Docker kontejnery a do nich následně nasazovat libovolný Docker image. Mezi podporované programovací jazyky patří Java, Ruby, JavaScript, .NET (Core), Python, PHP a Go. Jelikož platforma má otevřený zdrojový kód, je možné implementovat vlastní runtime pro další jazyky. Takových komunitních implementací jsou desítky a podporují například jazyky Haskell, Lua nebo moderní Rust. V rámci jednoho nasazení je možné instalovat vícero runtime-ů a podporovat tedy vícero programovacích jazyků současně. To už ovšem není zcela v souladu s konceptem micro-services architektury, která byla rozebrána v dalších částech práce.

Nasazení
Používat Cloud Foundry je možné ve dvou základních režimech. V prvním režimu si zákazník zajistí hardwarovou architekturu sám, ať už nákupem fyzických serverů, nebo jako službu od poskytovatele (například Amazon Web Services). Nasazení Cloud Foundry je potom jeho zodpovědnost a i údržbu musí provádět sám. Ve druhém režimu přenechá tuto práci na poskytovali, který to děla za něj a Cloud Foundry nabízí jako službu. V tomto režimu funguje i Siemens, přičemž svým zákazníkům umožňuje volbu poskytovatele hardwarové architektury. Následné nasazení Cloud Foundry už ale zajišťuje namísto zákazníka. Tato kombinace přeprodávané služby poskytovatele fyzických serverů a nasazení Cloud Foundry je nabízena pod obchodním názvem Siemens MindSphere.

MindSphere jinak
MindSphere je ve své podstatě tedy jen obchodní název pro skupinu produktů, které Siemens zprostředkovává. Mezi tyto produkty patří přeprodej služeb poskytovatelů fyzických serverů (Amazon Web Services, Microsoft Azure, atd.) a nasazení svobodného softwaru Cloud Foundry. Kromě toho Siemens poskytuje ještě vlastní hardware pro připojení řídicího systému Sinumerik. Hlavní výhodou toho, že Siemens zastřešuje všechny tyto komponenty je, že za systém jako celek přebírá odpovědnost. Provozovatelé výrobních strojů tak nemusí řešit nájem serverů, instalaci softwarů, atp., ale celou službu si koupí od svého dodavatele řídicího systému.

Poznámka k obecnému pojetí
To že se jednotlivé služby přeprodávají přes další zprostředkovatele je fenomén pozorovatelný ve všech odvětvích a svědčí o vývoji v těchto odvětvích. Jde o proces, kdy původní výrobce přijde s novým výrobkem a nejprve jej prodává na přímo koncovým zákazníkům. To s sebou nese komplexitu, která zatěžuje koncového zákazníka, neboť právě ten musí umět původní výrobek použít. Jako zjednodušený příklad, když původní výrobce vynalezne spalovací motor a začne jej prodávat koncovým zákazníkům, musí si umět celý automobil postavit koncový zákazník. To vede k tomu, že vznikne nový mezičlánek v obchodním řetězci, který tuto komplexitu přebírá z koncového zákazníka na sebe a účtuje je si za to přidané náklady. Ekonomie tento mezičlánek nazývá výrobou přidané hodnoty. Ve zmíněném příkladu to znamená, že vznikne výrobce automobilů, který bude motor implementovat a prodávat celý automobil jako produkt. Přebírá tím na sebe komplexitu instalace motoru do auta.

Obecně lze tento proces charakterizovat jako abstrakci od komplexity a lze ho pozorovat nejen v ekonomii, ale i ve zdánlivě nesouvisejících situacích. Například grafické uživatelské prostředí na počítačích (GUI) lze chápat obdobně jako abstrakci od komplexity práce s programovací řádkou. Komplexitu přebírá výrobce GUI (Microsoft) od uživatele PC. Zastupitelskou demokracii lze obdobně vnímat jako abstrakci komplexity rozhodování o státních záležitostech, kdy komplexitu přebírá vláda od občanů. Tento proces má v obecném pojetí tu vlastnost, že potřeba (ekonomický či jiný potenciál pro vznik) mezičlánku je úměrná zvyšující se komplexitě. Jakmile je komplexita pro koncového zákazníka příliš vysoká, vzniká tlak na vznik mezičlánku. V ekonomice pak dochází k tzv. outsourcování práce. Jedná se opět o abstrakci od komplexity. Například malá firma, která si nejdříve řešila účetnictví sama, poté co se zvětší počet zaměstnanců, jej převede na externí firmu, která účetnictví poskytuje jako placenou službu. Osvobozuje tím firmu od komplexity účetnictví.

V případě cloudových systémů firmy konsolidující potřebné služby osvobozují konečné zákazníky, v tomto případě firmy provozující výrobní stroje, od komplexity jednotlivých komponent. Podobně se dá uvažovat o tom, že s narůstající komplexitou mezistátních problémů vzniká potřeba abstrakce v podobě nového mezičlánku, kterým je Evropská Unie. V programování je příkladem abstrakce API neboli Application Programming Interface. Plní opět v obecném pojetí stejnou úlohu. Osvobozuje uživatele kódu od komplexity používané aplikace a zprostředkovává mu pouze potřebné výstupy v podobě konkrétních příkazů.

5.4.2 Architektura aplikací

Před vysvětlením architektury Cloud Foundry systému je pro plné pochopení nutné vysvětlit nejdříve architekturu samotných aplikací.

Různým přístupům pro návrh softwarové architektury se říká architektonické vzory. Těchto vzorů je známo zhruba 20. Pro potřeby této práce jsou důležité dva. Rozdělení architektur softwaru podle vnitřní struktury kódu je na:

- monolitické a
- microservices.

Důležité charakteristiky architektur
Různé architektury mají své výhody ale i nevýhody. Aby bylo možné objektivně posoudit vhodnost architektury, je potřeba systémově definovat, jaké posuzovací charakteristiky použít pro hodnocení. Jako součást zvolené architektury se bere i programovací jazyk. Důležité charakteristiky jsou:

- Produktivita: Z pohledu lidských zdrojů. Některé jazyky jsou rychlejší a bezpečnější než jiné, ale kompromisem je, že je obtížné a pomalé v nich programovat. Příkladem takového jazyka je C nebo C++ (i když s bezpečností je to spíše naopak). Naopak třeba Python je sice jazyk pomalejší, ale produktivita práce je s ním velmi vysoká. V tomto ohledu mu dokáže konkurovat jen málo jiných jazyků. Produktivita se odráží do finančních nákladů na vývoj a je tedy potřeba na ni brát zřetel.
- Spolehlivost: V případě některých systémů je kriticky důležité, aby aplikace neměly žádné výpadky, musí být tedy spolehlivé. Spolehlivost aplikace závisí jak na kvalitě kódu (a tedy kvalitě programátorů), tak i na inherentních vlastnostech programovacího jazyka a zvolené architektury.

- Škálovatelnost: Je míra toho, jak snadno lze reagovat na narůstající výkonové požadavky, třeba v důsledku vyššího počtu uživatelů u webových aplikací. Některé architektury lze škálovat snadno a rychle, jiné jen obtížně nebo vůbec.
- Udržovatelnost a rozšiřovatelnost: Tyto dva pojmy spolu úzce souvisí. Vývoj aplikací je vždy provázen chybami, které je nutné postupně odstraňovat. Určité typy architektur to znesnadňují. Rozšiřovatelnost znamená, že se v průběhu životního cyklu aplikace mohou vyskytnout požadavky na nové funkce a potom je otázka, jak snadno je lze implementovat do již hotové aplikace. Opět, některé architektury jsou k tomu vhodnější.
- Bezpečnost: V posledních letech významně roste počet kybernetických útoků, jejich sofistikovanost a škodlivost. Různé architektury i programovací jazyky mají jiný stupeň inherentní bezpečnosti a je potřeba to zvážit. Například moderní jazyk Rust díky své konstrukci vlastnictví modelů (ownership model) vylučuje některé třídy chyb pamětí a vláken (memory-safety, thread-safety).
- Podpora: týká se především programovacích jazyků. Některé jazyky jsou plně komunitním projektem a jako takové nenabízí žádnou oficiální podporu. Pro produkční nasazení aplikací je potřeba toto zvážit. Především pak to, jak dlouhou dobu bude oficiální podpora poskytována. Z tohoto hlediska jsou lepší volbou jazyky a frameworky z produkce velkých firem jako jsou Microsoft a jejich .NET a C#.
- Vyrobitelnost: Vyrobitelností je myšleno, jak snadno lze požadovanou aplikaci se zvolenou architekturou naprogramovat z pohledu lidských zdrojů. K samotné činnosti programování jsou pochopitelně potřeba programátoři. Programátoři jsou v dnešní době značně nedostatkové zboží a firmy o ně musí často bojovat pomocí zaměstnaneckých výhod, atd. Kromě toho znalosti každého programátora jsou omezené. Lépe řečeno každý programátor je specializovaný jen na určitý typ programování a to jak z pohledu architektur, programovacích jazyků, tak i určení výsledných aplikací. Je proto potřeba volit architekturu s ohledem i na to, aby bylo na trhu dostatečné množství programátorů, kteří ji budou umět naprogramovat. Pro rozložení specializací i programátorů zhruba platí paretovo pravidlo, tedy 80 % programátorů se specializuje na 20 % programovacích jazyků, atd. Volba méně zastoupeného jazyka tedy vede k problémům s lidskými zdroji. Z tohoto pohledu je účelné volit kompromis mezi vhodností architektury pro konkrétní úlohu a její rozšířeností. Z pohledu lidských zdrojů jsou na tom nejlépe webové technologie. Back-end, front-end, případně full-stack programátorů je obecně největší zastoupení. Je to dáno tím, že největší poptávka na trhu je právě po webových aplikacích a webových stránkách. Webovým technologiím dominují jazyky HTML, CSS, TypeScript (nové projekty v dnešní době preferují TypeScript před JavaScriptem; TypeScript je syntaktický superset jazyka JavaScript) pro front-end a Go, Java, Python, Ruby pro back-end.

Monolitická architektura

Monolitická aplikace je taková, jejíž celý kód, tj. kód pro různé části programu — pro práci s databází, pro interakci s uživatelem, samotná logika, atd., je vnitřně strukturně propojena a vůči jiným aplikacím vystupuje jako kompaktní celek. Taková aplikace funguje nezávisle na jiných programech a obsahuje vše, co ke svému běhu potřebuje. Filozofie monolitické architektury je, že aplikace je zodpovědná nikoli jen za jednu úlohu, ale za všechny úlohy, které jsou potřeba. Webová aplikace se skládá ze tří hlavních komponent: uživatelského rozhraní (User Interface — UI), databáze a serveru. Monolitická aplikace obsahuje všechny tři komponenty a přestože

může být vnitřně částečně modulární, všechny komponenty jsou nasazovány jako jediná jednotka. Jednotlivé komponenty pak nelze využít mimo tuto aplikaci.

Monolitické aplikace vznikly historicky jako první vývojová etapa, kdy původně byly vázány na tzv. mainframe počítače. V dřívějších dobách byly počítače málo výkoné, měly málo paměti a tedy i aplikace měly malou komplexitu. Často se celá aplikace vlezla na jediné CD a v případě nutnosti upgradu aplikace bylo potřeba koupit nové CD a celou aplikaci přeinstalovat. Pro takové prostředí je logický vznik monolitických aplikací.

Výhodou monolitických aplikací menšího rozsahu je menší celková komplexita a s tím související vyšší produktivita. Jednodušší je také nasazení monolitické webové aplikace, protože se jedná o kompaktní celek, který lze nasadit na jediný server.

Při procházení důležitých charakteristik popořadě lze vyjmenovat tyto nevýhody. Nízká produktivita práce. U monolitické architektury lze jen obtížně práci distribuovat mezi více lidí. V případě nových členů týmu je problém nové kolegy zaškolit, protože si musí projít velkou část monolitického kódu. Spolehlivost monolitických aplikací je také nízká, protože zpravidla při chybě v jedné části kódu kolabuje celá aplikace a je potřeba provést její restart. Další nevýhodou je velmi špatná škálovatelnost. V případě, že je potřeba vyššího výkonu byť jediná části aplikace, musí se škálovat celá aplikace, nelze škálovat například pouze databázi. Udržovatelnost a rozšiřovatelnost je u monolitů také špatná, protože kód je vzájemně hodně provázaný a jeho rozšiřování často vyžaduje rozsáhlé zásahy do různých částí kódu.

Monolitické aplikace se s narůstající komplexitou (přibývající funkce) stávají více rigidní: Jakákoliv změna v software je velmi obtížná a vyžaduje řadu (nepředvídatelných) úprav na mnoha místech. Více křehké: Jakákoliv provedená úprava způsobí problémy v jiných, často nesouvisejících, částech softwaru. Méně znovupoužitelné: Vyčlenění určité části software pro znovupoužití je složitější než tuto část vytvořit znovu [21].

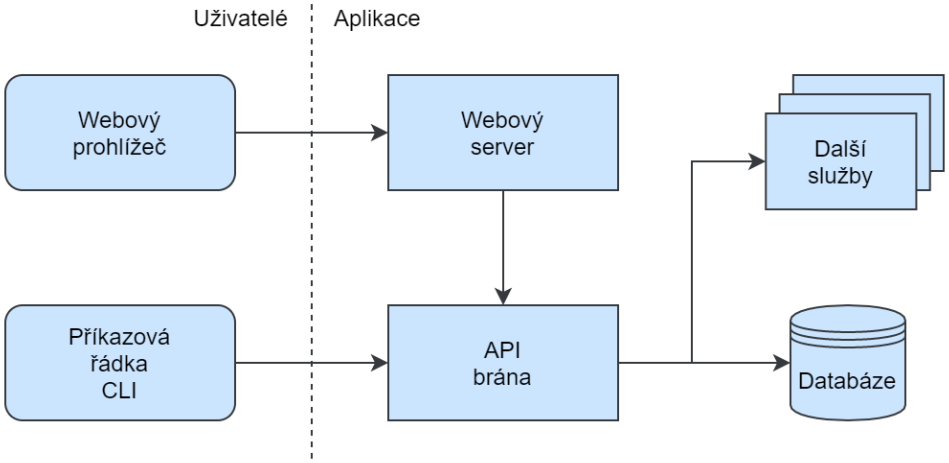
Monolitické aplikace mají tendenci nabývat na komplexitě, která se v určitý moment stane příliš velkou a aplikaci není možné dál rozvíjet, respektive jen za velmi vysoké náklady. Jak bylo popsáno v poznámce o komplexnosti, v takovém případě vzniká tlak na abstrakční mezičlánek, v programování častěji označovaný jako vrstva. Řešením problémů s monolitickými aplikacemi je tedy zavedení dalších vrstev, které poskytnou abstrakci od komplexity. Tím se dostáváme k vrstevné architektuře, která je dnes rozšířena nejvíce ve formě tzv. microservices architektuře.

5.4.3 Microservices

Monolitická aplikace se na microservices změní tak, že se rozdělí na jednotlivé komponenty, nazývající se služby (services), které spolu budou vzájemně komunikovat. Komunikace probíhá přes Application Programming Interface (API). API tím abstrakčním mezičlánkem, který umožňuje oddělení od komplexity ostatních částí. Monolitické aplikace obvykle obsahují jedinou databázi společnou pro všechny komponenty. Microservices využívají samostatnou databázi pro každou komponentu, přičemž komunikace je zajištěna přes API. Oddělené databáze zajišťují nízkou provázanost (loose coupling) kódu.

Rozčlenění aplikace na jednotlivé služby (services) dělá veškerý kód přehlednější, čitelnější (snadno pochopitelnější), a snadněji udržovatelný. Každou službu může mít na starost vyhrazený tým lidí, nebo může být klidně outsourcován jiné firmě. Každá služba se totiž chová jako black-box, na jejíž vnitřním fungování nezáleží, záleží pouze na její komunikaci přes API.

Obrázek 5.2: Základní struktura CF architektury



Lze tedy definovat API a vývoj služby delegovat na jiný tým. Lze tak velmi efektivně provádět potřebnou abstrakci od komplexity.

Separace jednotlivých služeb navíc umožňuje jednotlivým týmům využít různé technologie podle toho, jak se hodí pro konkrétní úlohu. Každá služba tak může být naprogramována v jiném programovacím jazyku. Například back-end obsluhující servery může být napsát v jazyku Go, který je právě k tomu účelu navržen, zatímco výpočetně náročné služby můžou být napsány v rychlém, nízko-úrovňovém Rustu. Každá služba může také využívat jinou databázi.

Tím je zaručena výhoda prakticky u všech důležitých charakteristik architektur. Velkou výhodou je schopnost škálovat microservices aplikace. Lze totiž škálovat jednotlivé komponenty podle potřeby samostatně. V případě mnoha požadavků na výpočty lze škálovat pouze službu výpočtů. V případě požadavků na zápis velkého množství dat lze zase škálovat pouze databázi příslušné služby. To vede i k výrazné finanční úspoře za hardware oproti monolitickému řešení.

Potenciální nevýhody microservices spočívají především v nesprávné implementaci, kdy může dojít například k rozdělení na příliš mnoho malých služeb, což celkovou komplexitu kontraproduktivně sníží. Velké množství jednotlivých služeb může být obtížnější testovat. Samostatné databáze zase můžou zbytečně duplikovat data a může tak docházet k plýtvání úložištěm, což v důsledku prodražuje hardware.

Shrnutí

Obě architektury mají své výhody i nevýhody. Pro vývoj jednoduché aplikace se spíše hodí monolitická architektura. V dnešní době se ve velké míře začínají prosazovat webové aplikace. Pro nasazení webových aplikací do cloudu je nutné zvolit cloudový přístup, což v principu znamená jenom to, že aplikace se rozdělí na jednotlivé služby, které se nasadí na vícero systémů a komunikují spolu po internetu. To znamená využití microservices.

Princip

Cloud Foundry (CF) v principu funguje tak, že zajišťuje běhové prostředí (runtime system) pro aplikace na serveru a tyto aplikace jsou přístupné přes webové rozhraní (web interface). CF je abstrakční vrstvou mezi microservice aplikacemi a potřebným hardwarem. Zajišťuje některé funkce potřebné k řízení životního cyklu aplikací nasazených na serverech a tím přesouvá kom-

Obrázek 5.3: Cloud Foundry Application Runtime

plexitu s tím spojenou od vývojářů konečné aplikace na vývojáře platformy CF. To umožňuje vývojovému týmu konečné aplikace soustředit se výhradně na vývoj samotné aplikace a neřešit implementační detaily platformy.

CF funguje na základě tzv. kontejnerů. Kontejner je program, který zpracovává vstup v podobě zdrojového kódu na běžící proces. Slouží jako abstrakční vrstva mezi hardwarem a zdrojovým kódem. Programátor se díky tomu nemusí starat o to, na jaké systému a na jakém procesoru jeho aplikace poběží. Stará se pouze o to, aby jeho aplikace běžela v kontejneru. Kontejner zajišťuje samotné běhové prostředí na konkrétním hardwaru.

Správu kontejnerů zajišťuje program CF Application Runtime (CF-AR). Aplikace se do kontejnerové architektury nahrává příkazem `cf push`, který vyvolá sérii funkcí, vykonávaných na pozadí CF-AR. Takto na pozadí je spravován kompletní životní cyklus aplikace.

5.5 Architektura cloudových aplikací

Aplikace je v základu tvořena třemi hlavními částmi. Jsou to back-end, front-end a databáze. Back-end je v určitém smyslu nejdůležitější částí, protože zajišťuje veškerou funkcionalitu. Možnosti softwaru jdou dány výhradně jeho back-endovou částí. Front-end je nějaká forma interface mezi back-endem a ovládacím subjektem či objektem (o subjekt se jedná v případě, že je softwarem ovládán člověkem; v roli objektu jsou jiné aplikace, používající funkce daného softwaru). Jsou dvě základní formy takového interface — CLI neboli Command Line Interface, což je ovládání softwaru pomocí příkazové řádky. Druhou formou je GUI neboli Graphical User Interface, což je grafické uživatelské rozhraní. Pro běžného uživatele se dnes jedná o převládající způsob interakce s počítačem. Poslední hlavní část aplikace je databáze. Ta slouží k práci s daty nebo soubory. V jistém smyslu nahrazuje počítačový souborový systém. Pro nativní aplikace není použití nutné z toho důvodu, že mají přístup právě k onomu souborovému systému. Avšak v případě webových aplikací žádný souborový systém k dispozici není, proto je nutné použití databází.

Back-end

Pro programování back-endu lze zvolit pouze takový jazyk, který je podporován běhovým prostředím cloudové platformy. Běhové prostředí zajišťuje překlad zdrojového kódu na nativní kód

pro daný hardware. MindSphere používá jako běhové prostředí Cloud Foundry Application Runtime, je proto potřeba volit ze seznamu podporovaných jazyků touto platformou. Nativní terminologie této platformy označuje balíčky pro běh jednotlivých jazyků jako buildpack-ve volném překladu stavební balíčky. Pro běh zvoleného jazyka je potřeba tento buildpack specifikovat, CF-AR si jej sám stáhne a automaticky nainstaluje. Seznam podporovaných buildpacků lze najít na stránkách platformy [30]. Mezi podporovanými jazyky najdeme Python, Go, Java a několik dalších. Jako vhodné lze doporučit Go a Python, autor je seznámen s jazykem Python a proto bude pro back-end použit ten.

Kromě oficiálně podporovaných jazyků lze využít ještě komunitní buildpacky. Jsou to buildpacky, které nemají oficiální podporu vývojářského týmu Cloud Foundry, přesto můžou fungovat bezchybně. Je také možné vytvořit si zcela vlastní buildpack, to však vyžaduje pokročilé znalosti a není-li to nutné, je lepší využít oficiálně podporované.

Front-end

Pro front-end webových aplikací existuje prakticky jediné řešení a to je JavaScript. Je vhodné zmínit jednu výjimku a totiž WASM, což je nová technologie, jejímž cílem je umožnit tvorbu webových aplikací běžících v prohlížeči rychlostí blízké nativní. Přestože dnes již existují knihovny na programování front-endu pomocí WASM, není k tomu určený — jeho uřčení je spíše back--end aplikací. Je však možné zaujmout různé postoje k tomu, jak javascriptovou aplikaci vytvořit. Nejefektivnější variantou je samozřejmě její přímé naprogramování. Další variantou je využití překladačů, které umožňují přeložit front-end do javascriptu z jiného kódu. To je zvolená varianta v této práci.

Jelikož back-endová část je programovaná v Pythonu, je vhodné zvolit k programování front-endu právě tento Jazyk. Přístupů, jak přeložit Python do webového javascriptu je víc. V této práci je využit projekt Jupyter, který je popsán v další kapitole 5.5.

Databáze

V prvotní fázi nebude databáze použita, protože MVP (Minimal Viable Product) nepotřebuje data nikam ukládat. Databáze bude dodělána v další fázi projektu jako služba, komunikující pomocí REST API.

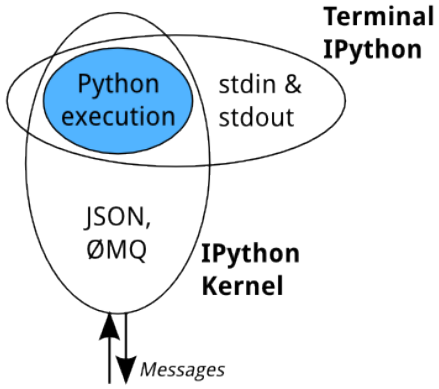
Jupyter

Jupyter je open-source webová aplikace, umožňující interaktivní manipulaci kódy. Je to webová aplikace, to znamená, že běží ve webovém prohlížeči a pomocí webových technologií zobrazuje výstup naprogramovaného kódu. Příklad. Pomocí pythonu se naprogramuje funkce vykreslující graf. V Jupyteru je možné tento graf vykresli přímo pod tímto kódem. Existuje knihovna s názvem ipywidgets, která zajišťuje interaktivitu s uživatelem uvnitř Jupyteru. Je možné používat běžné webové prvky uživatelského rozhraní, jako jsou tlačítka, posuvníky, rozevírací seznamy, atd. S využitím této knihovny je tedy možné vytvořit webovou aplikaci, která umožňuje prakticky libovolnou interakci s uživatelem.

Architektura

Jupyter je založen na architektuře server-klient. V roli klienta vystupuje webový prohlížeč a v roli serveru spuštěný Jupyter daemon. Daemon komunikuje ještě s modulem pro zvolený programovací jazyk — kernelem. Interaktivní smyčka — REPL funguje tak, že kód napsaný v No-

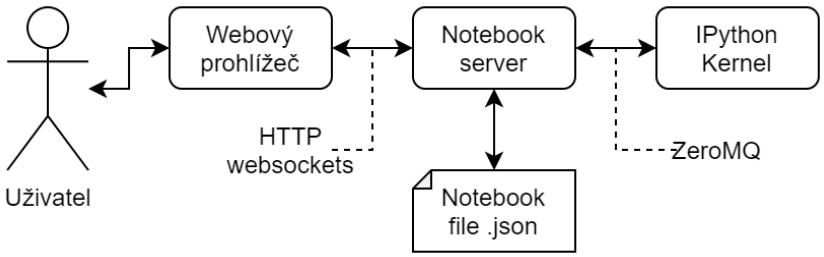
Obrázek 5.4: Architektura Jupyteru



tebooku je odeslán na server, kde je za pomoci kernelu zpracován na výstup a ten je odeslán zpět do klienta — webového prohlížeče k vykreslení.

Serverová část se skládá ze serveru notebooku a samotného kernelu. Celá smyčka potom funguje tak, že uživatel interaguje s webovým prohlížečem, ten pomocí webových socketů komunikuje s notebook serverem. Notebook server přijme požadavek, ve formátu JSON ho pomocí ZeroMQ socketů předává kernelu, který kód interpretuje a vypisuje pomocí stdout.

Obrázek 5.5: Jupyter architektura



5.6 Minimální životaschopný produkt — MVP

V této kapitole je definován minimální životaschopný produkt (z angličtiny Minimal Viable Product — MVP) pro tuto závěrečnou práci. Z výše uvedeného je zřejmé, že vývoj cloudové aplikace na produkční úrovni, tj. konkurenceschopné na volném trhu, je náročný projekt a je třeba rozličných znalostí. Konkrétně je vhodné, aby front-end byl vytvořen specialistou na JavaScript, síťová architektura opět příslušným specialistou, atd., což je mimo rozsah této práce. Proto je navrženo vytvořit MVP. Jedná se o analogii k podmínkám technické přejímky. Je to tedy seznam funkcí a charakteristik, které musí být splněny, aby byl produkt považován za kompletní v rámci první iterace vývoje. První iterací vývoje se v kontextu této závěrečné práce myslí verze pro odevzdání před obhajobou. Dalšími iteracemi se myslí následující vývoj a rozšiřování o další funkce v navazujícím doktorském studiu.

MVP

Při definování MVP se vycházelo z cíle této závěrečné práce, kterým je navrhnou vhodný způsob

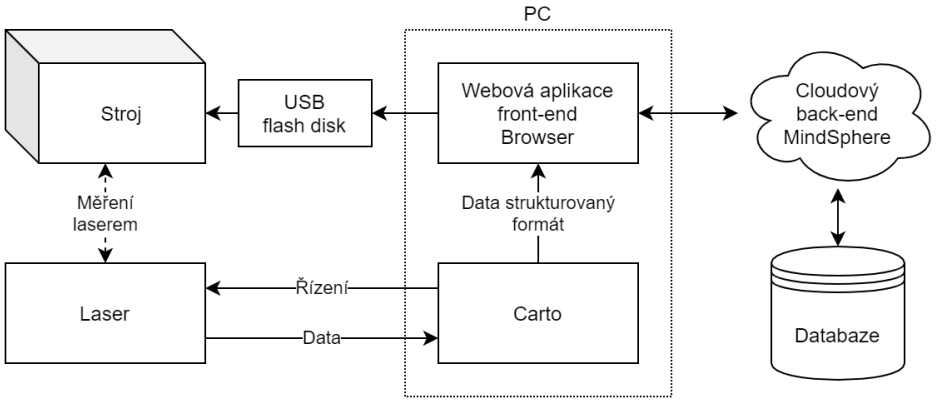
monitorování a posouzení výrobní přesnosti. Z toho vyplývá, že primárním cílem je navrhnout a ověřit monitorovací systém. Jde tedy především o ověření konceptu (proof of concept). Není proto vhodné navrhovat rozsáhlé a složité řešení, ale naopak jednoduchou aplikace, na které bude ověřeno a ukázáno, jakým způsobem lze pracovat s koncepcí cloudových aplikací a jakým způsobem je lze implementovat v rámci MindSphere.

Až následně, pokud se tato zkušenost ukáže jako dobrá, může se na základě této práce pokračovat v dalších iteracích vývoje a aplikaci posunout na produkční úroveň. Předpokládá se přitom, že bude potřeba některé části významně upravit, či zcela přepsat a to za pomoci jiných technologií.

S vedoucím práce byl definován MVP následujícím způsobem. Je potřeba vyzkoušet a ověřit koncept cloudové aplikace. Aplikace tedy musí splňovat minimálně tato kritéria:

- interaktivní aplikace běžící v cloudu na platformě MindSphere,
- funkční v prohlížečích Google Chrome a Mozzila Firefox na FHD displayích,
- funkční import dat ze softwaru Renishaw Carto pomocí uploadu,
- zobrazení importovaných dat,
- výpočet a zobrazení ukazatelů dle normy ISO 230–2,
- export ve formátu JSON pro použití v dalších softwarech,
- výpočet kompenzačních dat,
- export kompenzačních dat ve formátu NC kódu

Obrázek 5.6: Schéma měření a kompenzování pomocí MindSphere



Již při prvotním návrhu jsou však zřejmé další funkce, které budou pro tuto aplikaci užitečné. Tyto funkce budou přidávány postupně při iterativním vývoji podle časových možností. Definujme proto druhou iterační fázi, nazvěme ji MVPv2. V této druhé vlně se požadují následující funkce:

- výpočet ukazatelů dle dalších norem, minimálně dle VDI,
- rozšířené možnosti pro interaktivní prohlížení statistiky,
- systém pro přihlašování uživatelů,
- databáze pro ukládání dat,
- lepší uživatelský dojem (user experience)

5.7 Závěr pro návrh řešení

Definováním požadavků na systém pro monitorování se došlo k tomu, že nejlepší volbou pro vytvoření takového systému je informační systém realizovaný jako cloud. Cloud je dnes nad--užívaný pojem a není proto mnohdy zřejmé, co se jím vlastně myslí. V praxi to znamená organizovat architekturu informačního systému po internetové síti s využitím vysoce modulárního designu, který umožňuje vybrané části outsourcovat na dodavatele. Jinak řečeno, organizovat část infrastruktury jako placenou službu, což má mnohé výhody.

Pro strojírenské podniky je nejlepší volbou na dodavatele cloudových služeb firma Siemens, která poskytuje novou PaaS platformu MindSphere. MindSphere je produkt zastřešující vícero jednotlivých služeb od sub-dodavatelů. Využitím služeb MindSphere dochází k silné abstrakci od komplexity cloudových služeb na pouhou placenou službu. Proto byla pro řešení zadání této závěrečné práce zvolena platforma MindSphere.

V další kapitole je popsán postup při vytváření aplikace běžící v cloudu MindSphere a jsou ukázány dílčí výsledky a doporučen další rozvoj.

6 MindSphere aplikace

Cílem práce je navrhnout vhodný způsob monitorování a vyhodnocení výrobní přesnosti, přičemž zaměření práce bylo konkretizováno na monitorování chyb lineárního polohování. Polohování se měří pomocí speciálních měřicích zařízení. V této práci byl použit laser interferometr XL-80 od firmy Renishaw. Výstupem měření jako takového jsou dvourozměrná data, kde na jedné ose jsou vyneseny souřadnice měřených bodů a na kolmé ose jsou vyneseny lineární úchylky. Vyhodnocení dle normy ISO 230–2 se provádí v dodaném softwaru.

Bylo navrženo, vytvořit cloudový informační systém, který by nahradil zpracování pomocí proprietárního softwaru na PC. Jako platforma pro řešení tohoto požadavku byla zvolena nová platforma od firmy Siemens — MindSphere. Je potřeba naprogramovat aplikaci, do které bude možné importovat data z měření laserem, v této aplikaci je vyhodnocovat dle normy ISO 230–2. Aplikace by měla být navržena takovým způsobem, aby bylo možné na ní kontinuálně pracovat a postupně doplňovat dalšími funkcemi, například přidáním jiných norem pro vyhodnocování chyb.

Vývoj cloudové aplikace je komplexní záležitost a vyžaduje různorodé znalosti z oblasti programování a vývoje softwaru. Pro plnohodnotnou aplikace architektury microservices jsou nutné minimálně tyto znalosti:

- cloudová architektura,
- programování back-endu,
- programování front-endu pomocí webových technologií,
- návrh a práce s databázemi.

Pro vývoj takto komplexního systému je vhodné použít atomický iterační způsob vývoje. Tedy začít od jednoduché aplikace s minimálními funkcemi, postavené na monolitické architektuře a postupně vyvíjet další funkce. Přidávání funkcí by mělo být řešeno ve smyslu microservices a postupně tak přetvářet monolitickou aplikaci na plně cloudové řešení.

6.1 Plán implementace

Implementace byla rozdělena na několik fází tak, aby bylo možné iteračně přidávat funkce a postupem času refaktorovat na jednotlivé služby.

Fáze 1
V první fázi bylo naprogramováno výpočetní jádro aplikace. Výpočetní jádro je zodpovědné za všechny transformace s daty, které jsou potřeba k přeměně vstupů na požadované výstupy. Obsahuje tedy moduly pro načítání dat z měření, zpracování dat do jiného formátu, výpočet ukazatelů dle normy. Práce s výpočetním jádrem je možná přímo přes příkazovou řádku.

Fáze 2
Aby bylo možné s aplikací pracovat mimo příkazovou řádku, resp. webovou konzolu, bylo potřeba naprogramovat front-end, neboli GUI — grafické uživatelské rozhraní. Pro použití v cloudovém systému je potřeba GUI řešit jako webové rozhraní.

Fáze 3
Jádrem řešení MindSphere je open source platforma Cloud Foundry. Aplikační front-end i back--end je potřeba přizpůsobit běhovému prostředí CF. V této fázi byly implementovány potřebné soubory.

Fáze 4
Pro nasazení v samotném řešení MindSphere je potřeba celou aplikaci zaregistrovat přes vývojářský účet na webových stránkách Siemensu a provést další nezbytné úkony vedoucí ke zprovoznění aplikace v cloudu. Toto bylo řešeno ve čtvrté fázi.

Fáze 5
Pátou fází je přidání databáze a doprogramování potřebných funkcí pro její použití.

Další fáze
Jelikož vývoj je plánován i nad rámec této závěrečné práce a bude v něm pokračováno na doktorském stupni, jsou předpokládány další fáze, které budou přidávat funkcionality podle aktuálních potřeb a cílů. Nadto se předpokládá, že celá aplikace bude postupně přepisována pomocí jiných technologií tak, aby byl zajištěn její co nejlepší běh.

6.2 Fáze 1: výpočetní jádro

Výpočetní jádro aplikace je naprogramováno ve formě Python knihovny. V programu jsou volány další knihovny, které jsou použity pro konkrétní úlohy. V aplikaci byly použity minimálně tyto knihovny (packages) mimo standardní knihovnu:

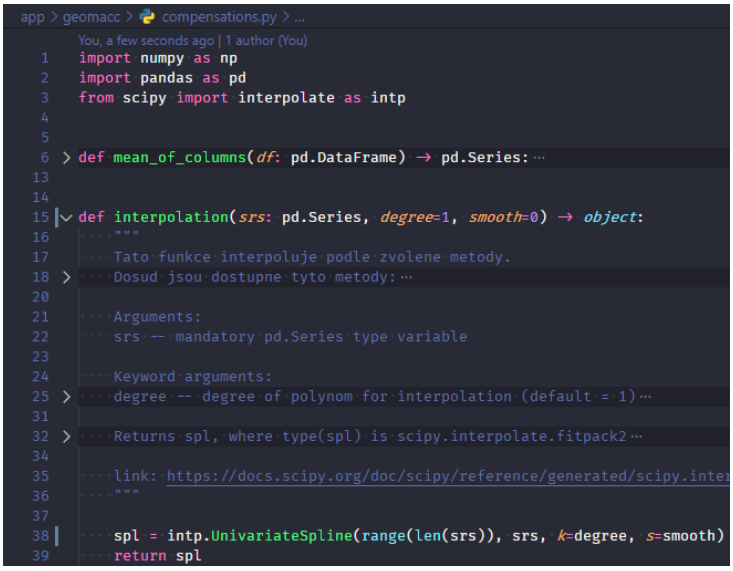
- `jupyter`
- `voila`
- `numpy`
- `matplotlib`
- `pandas`
- `ipywidgets`
- `jinja2`

Odpovědnost jádra
Při první implementaci je celé jádro napsané jako monolitický program. Jednotlivé funkce jsou rozděleny mezi několik souborů a do množství programových tříd a funkcí. Nejsou však koncipovány jako oddělené služby, komunikující přes API.

Jádro obsahuje jednotlivé programy pro dílčí funkce. Celé jádro je pak složeno ze všech programů. To zaručuje potřebnou modularitu a možnost případného pozdějšího přepisu pomocí vhodnějších technologií jednu funkci po druhé. Importování dat ze souboru je zajištěno pomocí standardní knihovny. Importovaná data jsou upravena a načtena pomocí knihovny pandas. Pomocí stejné knihovny jsou vypočteny i všechny statistické charakteristiky podle normy ISO 230–2. Pro tvorbu grafů byla použita knihovna matplotlib, která je základní knihovnou pro práci s grafy. Existují i sofistikovanější knihovny a také knihovny vhodnější pro použití ve webových aplikacích, avšak funkcionalita matplotlib je více než dostatečná pro definovaný MVP.

Pro separaci metadat byla použita knihovna re, což je knihovna pro regulární výrazy a je součástí standardní knihovny.

Obrázek 6.1: Ukázka kódu z jádra



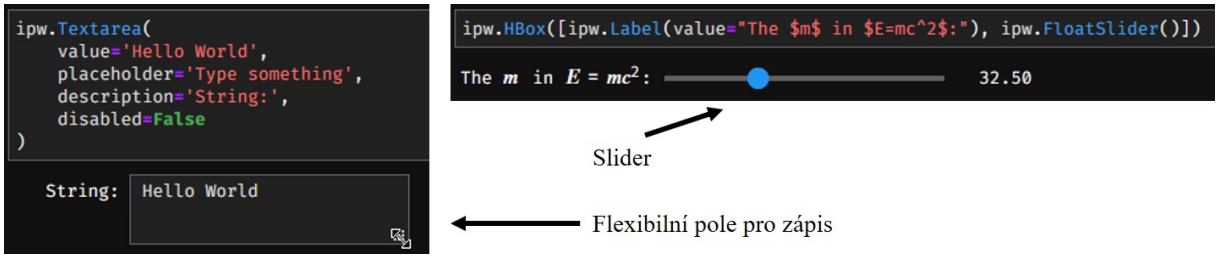
Pro generování NC kódu pro účely kompenzování geometrických chyb na stroji byl použit tzv. template engine. To jsou knihovny a metody pro vytváření šablon, které mohou být programaticky vyplněny dodanými daty. Příkladem může být šablona pro e-maily, kdy chceme rozeslat hromadný e-mail (nikoli nutně SPAM) a potřebujeme v každém e-mailu změnit pouze jméno a adresu. V Pythonu existuje více takových knihoven. Pro tuto aplikaci byla zvolena knihovna jinja2.

Modularita
Jádro byla navrženo modulárně tak, aby bylo možné snadno doplnit další moduly pro výpočet charakteristik podle jiných norem, generovat NC kód na jiné stroje, tedy použít jiné šablony. Tím je zajištěn snadný další vývoj. Pro další verze aplikace nad rámec této práce se navrhuje přepsat moduly s využitím formálních interfaců, případně rozdělit jednotlivé moduly na samostatné služby. To by byla výhoda v případě, že aplikace bude využívána ve velké míře. Lze předpokládat, že výpočet podle různých norem bude žádán s různou intenzitou a rozdělení na služby by v takovém případě umožňovalo škálování pouze těch norem, které jsou nejvíce vytížené.

6.3 Fáze 2: Grafické rozhraní

Grafické rozhraní je u cloudových aplikací nutné implementovat jako webové rozhraní. Prohlížeče umí pracovat s přesně čtyřmi (programovacími) jazyky. Jsou to HTML, CSS, JavaScript a WASM . WASM je technologie nad rámec této práce a není rozebírána. Překlad do JavaScriptu je zajištěn pomocí programu Jupyter, který byl vysvětlen v kapitole 5.5. Pro programování samotného rozhraní byla použita knihovna ipywidgets, která zajišťuje překlad mezi standardními prvky uživatelského rozhraní z JS. Umožňuje využívat všechny standardní prvky jako jsou tlačítka, posuvníky, slidery, umožňuje vkládat obrázky, mediální streamy (videa, hudbu).

Obrázek 6.2: Ukázka programování front-endu



Pomocí této knihovny bylo vytvořeno rozhraní, které komunikuje s jádrem. Knihovna umožňuje využívat několik různých přístupů pro tvorbu rozmístění prvků rozhraní. Jednou možností je tzv. grid-layout. V JS se jedná o poměrně používanou technologie. Umožňuje naprosto flexibilní rozmístění a vnořování prvků. Celkový umělecký dojem z rozhraní je tedy pouze otázkou designu a je možné jej měnit.

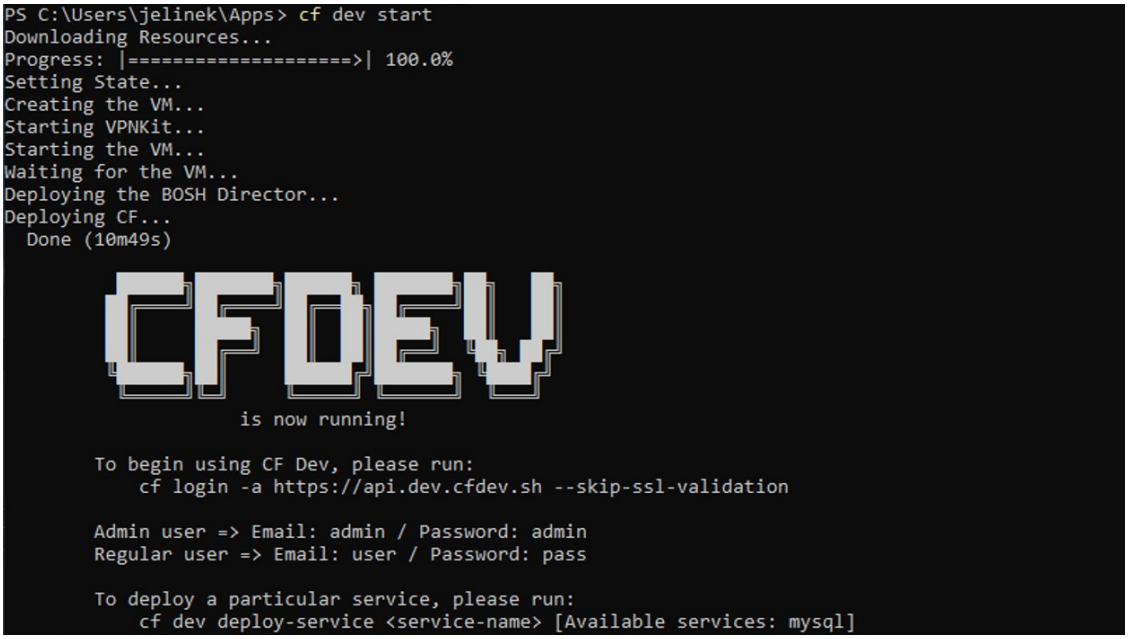
Obrázek 6.3: Ukázka programování front-endu 2

Důležité je zmínit modularitu. Jádro aplikace, neboli back-end a webové rozhraní jsou izolovaná prostředí a lze je tedy snadno nahrazovat. Je tedy možné nahradit webové rozhraní sofistikovanějším, naprogramovaným přímo v JavaScriptu, nebo ještě lépe pomocí TypeScriptu, nebo dokonce, pokud by to bylo vhodné, pomocí technologie WASM. Toto případně nahrazení lepším front-endem bude předmětem navazující práce na ÚVSSR při FSI na VUT.

6.4 Fáze 3: Cloud Foundry

Cloud Foundry je běhové prostředí, které zajišťuje vykovávání kódu na serveru. MindSphere používá jako podkladovou technologii právě Cloud Foundry. Aplikace ji možné vyvíjet i lokálně pomocí speciálního programu, který simuluje zjednodušenou architekturu cloudu na lokální stanici. Program má CLI interface, ovládá se tedy přes příkazovou řádku.

Obrázek 6.4: Spuštění lokální instance Cloud Foundry pomocí PowerShell



Pro správné fungování aplikace na Cloud Foundry bylo potřeba vytvořit několik podpůrných souborů, které definují podmínku a závislosti nasazení aplikace. Jsou to celkem tyto soubory:

- manifest.yml
- Procfile
- requirements.txt
- runtime.txt

6.4.1 Manifest file

Soubor manifest je typu YAML, což je textový soubor strukturovaných dat, který je čitelný nejen strojově, ale snadno i pro člověka. Tento soubor slouží pro předání informací do Cloud Foundry Application Runtimu o tom, jaké parametry se mají při nasazování aplikace použít. Mezi ty patří, jaké buildpacky se mají použít. Jaké má být jméno aplikace. Kolik operační paměti má mít aplikace maximálně k dispozici, kolik diskového prostoru potřebuje a další. Minimální nutné struktura manifest souboru je tato:

Obrázek 6.5: Soubor manifest

```
1  ---
2  applications:
3  - name: iso
4    memory: 256M
5    disk_quota: 756M
6
```

6.4.2 Procfile

Soubor Procfile slouží pro definování příkazů, které mají být vykonány serverem pro spuštění aplikace. Pro naprogramovanou Python aplikaci je potřeba na serveru spustit příkaz pro spuštění jupyter serveru a aplikace voila. Příkaz v Procfilu vypadá následovně ”web: voila app/app.ipynb –port=$PORT –no-browser –theme=’light’ –debug”

6.4.3 requirements

Textový soubor requirements slouží pro definování závislostí Python aplikace. Říká, jaké knihovny se mají nainstalovat. Lze definovat i konkrétní verze. Pro potřeby MVP aplikace jsou potřeba tyto knihovny:

- jupyter
- voila
- numpy
- matplotlib
- pandas
- ipywidgets
- scipy
- jinja2

6.4.4 runtime

Textový soubor runtime byl použit pro definování Python verze, která se má použít při instalaci na serveru. Aplikace vyla vyvíjena na Python verzi 3.7.6. Obsah souboru proto vypadá takto: ”python-3.7.6”.

6.4.5 Spuštění aplikace

Pro nasazení aplikace do Cloud Foundry byly postupně používány různé vývojové verze pro testování. Běh lokální instance Cloud Foundry je velmi náročný na operační paměť. Doporučuje se mít počítač s 32 GB RAM, minimálně však s 16 GB RAM. Pro vývoj byla použita stanice s 16 GB, což se ukázalo jako dostatečné pro samotný běh aplikace, bylo však nutné Cloud Foundry instanci spouštět jako první program po naběhnutí operačního systému. Především při spuštěném webovém prohlížeči již operační paměť nedostačuje a Cloud Foundry si ji nevynucuje. Instanciace potom končí chybou. Je tedy potřeba nemít spuštěné žádné další programy náročné na operační paměť, spustit nejdříve cf-dev a až poté prohlížeč, který je nutný k vykreslení uživatelského rozhraní. Při tomto postupu dojde k omezení dostupné paměti pro prohlížeč a nikoli pro instanci Cloud Foundry. Pohodlnější a pro větší aplikace, nebo pro běh více aplikací se však doporučuje použít pracovní stanici s alespoň 32 GB RAM.

Běh lokální instance Cloud Foundry je také velmi náročný na I/O operace, tedy operace typu Input a Output, kdy je využíván hard disk. Náročnost je v takové míře, že nelze použít plot-

Obrázek 6.6: Prázdný cloud po spuštění

```
PS C:\Users\jelinek\Apps>
Getting apps in org cfdev-org / space cfdev-space as admin...
OK

No apps found
```

Obrázek 6.7: Nahrání a spuštění aplikace

```
PS C:\Users\jelinek\Apps> cf push loading_data
Pushing from manifest to org cfdev-org / space cfdev-space as admin...
Using manifest file C:\Users\jelinek\Apps\manifest.yml
Getting app info...
Updating app with these attributes...
  name:               loading_data
  path:               C:\Users\jelinek\loading_data
- disk quota:         1G
+ disk quota:         756M
  health check type:  port
  instances:          1
  memory:             256M
  stack:              cflinuxfs3
  routes:
    loadingdata.dev.cfdev.sh

Updating app loading_data...
```

nový HDD, ale je nutné použít elektronický SSD o velikosti alespoň 500 GB. Vzhledem k tomu, že na použité pracovní stanici byl původně jen malý 256 GB SSD, nezbývalo již dostatek místa pro Cloud Foundry. Proběhl proto pokus o spuštění instance z HDD, ale skončil neúspěchem. Množství I/O operací v sérii je příliš velké a takový pokud končí chybou ”cf dev start failed while deploying. It got stuck at Progress x of 15 (**m**s)”. K tomu je po dotazu vyjádření jaderných vývojářů ”Running an entire PaaS on one workstation is an ambitious endeavour and the workstation has to be performant enough for the task. The most common insufficiency is the disk speed due to CF Dev being such an disk I/O intensive process. As mentioned in our recommended requirements. No less than flash storage (i.e. SSDs) is recommended for use with CF Dev. If your workstation has an HDD, this is the most likely cause of issue.

Pro vývoj jsou tedy minimální požadavky 500 GB SSD a 16 GB RAM. Doporučit lze však alespoň 1 TB SSD a 32 GB RAM.

Obrázek 6.8: Aplikace běžící v Cloud Foundry

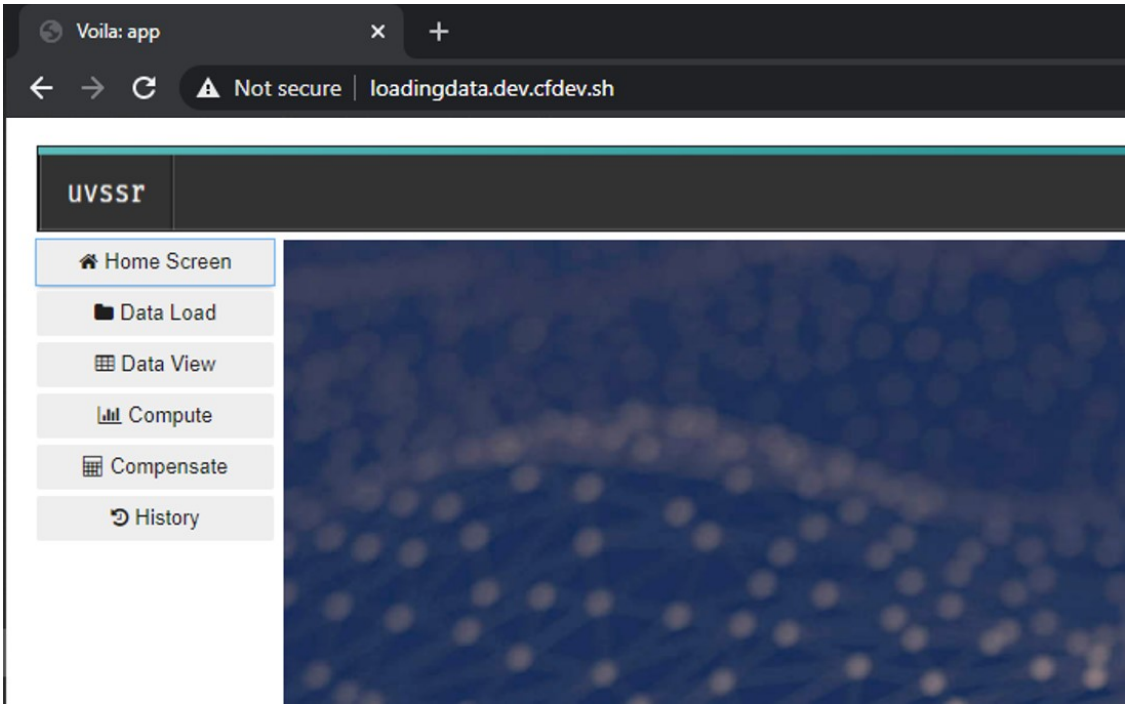
```
Waiting for app to start...

name:              loading_data
requested state:   started
routes:            loadingdata.dev.cfdev.sh
last uploaded:     Wed 24 Jun 08:41:28 CEST 2020
stack:             cflinuxfs3
buildpacks:        python

type:            web
instances:       1/1
memory usage:    256M
start command:   voila app/app.ipynb --port=$PORT --no-browser --theme='light' --debug
     state     since                  cpu    memory          disk            details
#0   running   2020-06-24T06:42:18Z   0.0%   39.1K of 256M   233.6M of 756M
```

Operační parametry aplikace
Na obrázku 6.8 lze vidět, že aplikace běží v lokální instanci Cloud Foundry na webové adrese loadingdata.dev.cfdev.sh s parametry: 233.6 MB diskového prostoru — ten je použit pro instalační soubory Pythonu, všech použitých, závislých knihoven a samotné aplikace. Bezprostředně po spuštění využívá aplikace 39.1 KB operační paměti z celkově dostupných 256 MB definovaných v manifestu. Utilizace procesoru je 0.0 % neboť aplikace není používaná.

Obrázek 6.9: Webový front-end aplikace na adrese loadingdata.dev.cfdev.sh



6.5 Fáze 4: MindSphere

Na lokální instanci Cloud Foundry vyvinutá aplikace se pro nasazení v cloudu MindSphere musí nahrát do modulu s názvem Developer Cockpit. Jedná se o modul pro správu aplikací uvnitř MindSphere. Umožňuje aplikace zveřejňovat, nastavovat přístupová práva, atp.

Dle původní dokumentace z ledna tohoto roku mělo být možné použít libovolnou technologii pro uživatelské rozhraní. Byla zvolena kombinace technologií Python, Jupyter a Voila. V průběhu roku však kontinuálně dochází k rozšiřování dokumentace MindSphere a specifikaci požadavků na aplikace, které do ní mají být umístěny.

Nové požadavky jsou především bezpečnostního charakteru a ohledně designu. Front-end aplikace musí splňovat různá bezpečnostní kritéria. Prakticky musí obsahovat technologie pro zajištění bezpečnosti, které jsou vyjmenované v požadavcích.

Design musí být konformní s předpisem dle Siemens. Aplikace musí používat pouze doporučené barvy, předepsané rozložení prvků a jednotlivé prvky rozhraní musí být použity ze Siemens knihovny. Jako příklad, tlačítka musí být použita dle Siemens specifikace. Vše je možné stáhnout z designové knihovny Siemens [19].

S těmito požadavky se na začátku vývoje nepočítalo a v důsledku toho aplikace není v aktuální podobě konformní se Siemens specifikací. Front-end tedy musí být přepracován dle těchto požadavků. Díky použití modulárního designu však není potřeba přeprogramovávat celou aplikaci, ale pouze webový front-end.

V tomto ohledu se použití Python knihovny Voila s využitím projektu Jupyter ukázalo jako nepraktické, i když samotná možnost vytvoření webové aplikace výhradně v Pythonu je nanejvýš užitečná věc. Aplikace bude pravděpodobně nutné přepracovat pomocí tradičních webových technologií. Je však nutné detailně nastudovat novou specifikaci a zvolit technologii, která bude splňovat všechny požadavky. Navrhuje se rozhraní vytvořit pomocí JavaScriptu, respektive moderního dialektu TypeScript a využít například framework Vue.js. Další alternativou je použití WASM, ale opět je nutné důkladně zvážit vhodnost technologie.

6.6 Obecné poznámky

Aplikace byla naprogramována pomocí Pythonu. Byl vytvořen tzv. Python package, neboli balíček. Balíčky slouží pro efektivní členění aplikačního kódu. Slouží jako oddělené jmenné prostory (namespace). Technicky vzato se jedná o adresáře v souborovém systému, které kromě python modulů obsahují specifický modul `__init__.py`. Balíčky zároveň slouží pro snadné sdílení mezi uživateli. Název balíčku je `geomacc`, jakožto zkratka pro geometric accuracy, tedy geometrická přesnost.

Balíček byl navržen tak, aby splňoval zásady návrhových vzorů SOLID a zásady vycházející z doporučení tvůrců jazyka, které jsou shrnuty v Zenu Pythonu.

Návrhové vzory SOLID
Jsou soubor pěti návrhových vzorů, tedy jakýchsi obecných doporučení, kterými by se měli vývojáři řídit při návrhu softwaru. Jsou to:

- princip jedné odpovědnosti (Single responsibility principle),
- princip otevřenosti a uzavřenosti (Open–closed principle),
- Liskovové princip zaměnitelnosti (Liskov substitution principle),
- princip oddělení rozhraní (Interface segregation principle) a
- princip obrácení závislostí (Dependency inversion principle).

Zen Pythonu
Je soubor 19 vět, ve většině případů po dvojicích komplementárních, které stručně charakterizují, na co by měl být při programování brán ohled. Některé důležité věty:

- Krásné je lepší než ošklivé (Beautiful is better than ugly).
- Jednoduše je lepší než složitě (Simple is better than complex).
- Čitelnost se počítá (Readability counts).

Některé věty je možné dát do souvislosti se SOLID principy. Například třetí věta zenu ”Jednoduše je lepší než složitě.”je zřejmě související s principem jedné odpovědnosti. Programovací entity (třídy, funkce, aj.) by měly být jednoduché a tedy mít jedinou odpovědnost.

Správa Pythonu
Pro instalaci a správu balíčků byl použit nástroj `miniconda`, což je vůči operačnímu systému agnostický, systémový, binární správce balíčků [20]. Jako zdroj repozitářů pro balíčky byl nastaven `conda-forge` [5]. Jako IDE byl použit `vscode`, což je open source vývojové prostředí [35]. Vscode vzniklo v roce 2015 pod křídly Microsoftu a od té doby se stalo vůbec nejpoužívanějším vývojovým prostředím [28].

6.7 Ukázka aplikace

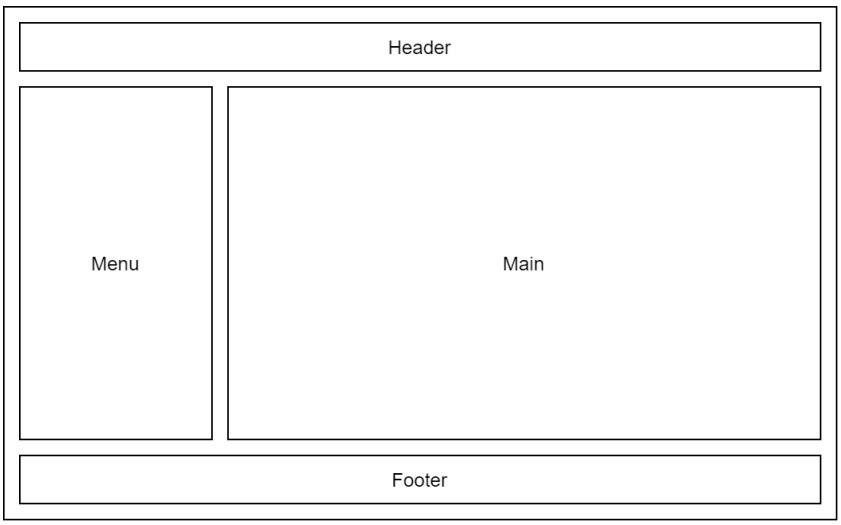
Aplikace je koncipovaná jako cloudová. To znamená, že back-end běží na serveru a front-end ve webovém prohlížeči uživatele. Aplikaci lze tímto způsobem nasadit na libovolném cloudu. Tedy jak na veřejném, ať už od firmy Siemens, nebo u jiného poskytovatele, například Amazon, Microsoft, Google. Tak ji lze nasadit i na privátním cloudu, který může být sestaven z jediné pracovní stanice. Vážou se na to všechny výhody a nevýhody popsané v kapitole 5.3.2.

Z důvodu omezeného prostoru na stránce papíru jsou používány výřezy obrazovky namísto snímků celé obrazovky.

6.7.1 Rozložení

Po spuštění aplikace v prohlížeči se zobrazí okno celé aplikace. To je rozděleno na 4 hlavní úseky. Jsou jimi Header (hlavička), Footer (zápatí), Menu a Main.

Obrázek 6.10: Schéma rozložení aplikace



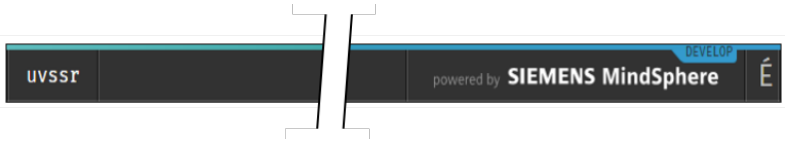
Menu
Je koncipováno jako svislý sloupec obsahující tlačítka. Tlačítka přepínají hlavní okno (Main) a tím zpřístupňují jednotlivé funkce. Tlačítek je dohromady 6. Následuje jejich stručný popis.

- Home Screen: je tlačítko pro domovskou obrazovku. Ta obsahuje základní informace pro používání aplikace a informace o autorovi, kontextu jejího naprogramování v rámci diplomové práce, domovském ústavu, aj. Domovskou obrazovku je možné libovolně změnit.

- Data Load: toto okno slouží pro nahrání souboru s daty z měření laserem, které lze exportovat ze softwaru Renishaw-Carto-Explore.
- Data View: slouží pro prohlížení importovaných dat ve formátu tabulky. Je možné toto okno přizpůsobit například tak, aby byla možná editace importovaných dat, což se může hodit pro testovací účely.
- Compute: je okno pro výpočet statistických ukazatelů dle zvolené normy a pro zobrazení odpovídajících grafů.
- Compensate: je okno pro výpočet a generování kompenzačních dat.
- History je okno pro správu historických výsledků. V další fázi vývoje zde bude zprostředkován přístup do online databáze výsledků.

Header
Okno Header je zobrazeno v horní části obrazovky přes celou její šířku. Obsahuje ovládací prvky pro MindSphere jako jsou přihlášení uživatele, aj. Pro MVP obsahuje už jen odkaz na stránky . V dalších iteracích vývoje lze doplnit další prvky.

Obrázek 6.11: Záhlaví — Header



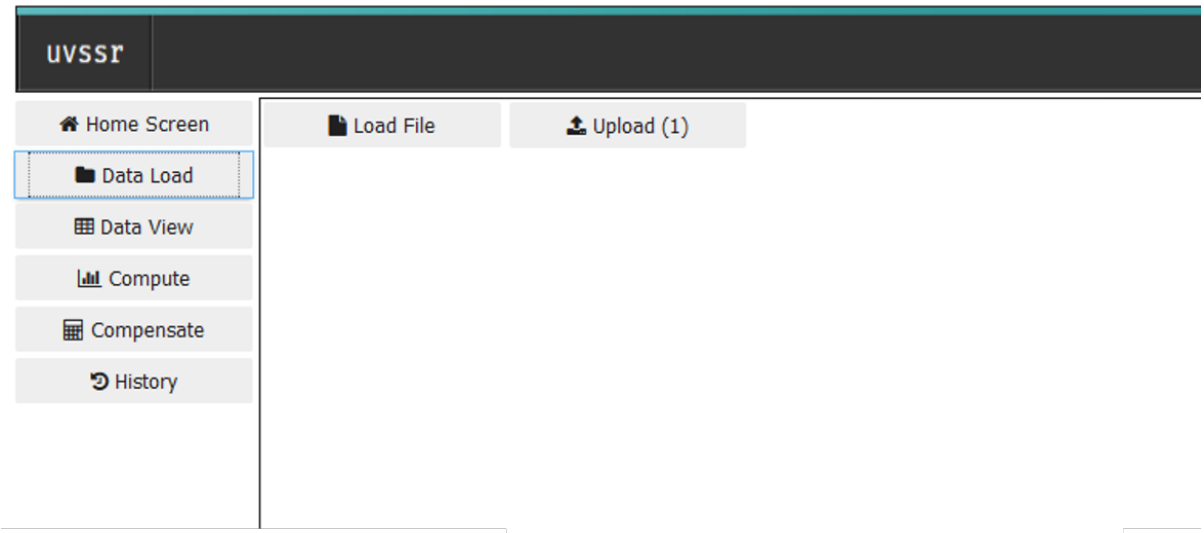
Footer
Zápatí je určeno k různým stavovým informacím typu čas, aktuální zobrazené okno, stav výpočtu — zda probíhá, nebo je již dokončen atp. Tvorba zápatí není definována do MVP a proto byla její implementace naplánována až na navazující práce. V kontextu nových požadavků na design rozhraní v MindSphere však není možné zápatí mít a bude tedy pravděpodobně zcela zrušeno pro MindSphere verzi.

Main
Hlavní okno je největší část rozhraní. Zobrazuje různý kontext podle toho, které okno je nastaveno tlačítkem z menu. Dohromady je v současné verzi 6 různých oken, každé má na starost něco jiného. Modularita aplikace teoreticky umožňuje, aby bylo každé okno naprogramováno pomocí jiné technologie.

6.7.2 Data Load — Nahrávání dat

Okno Data Load slouží pro nahrávání dat z měření. Je možné nahrát soubor z počítače — tlačítko upload otevírá dialogové okno pro výběr souboru. Tlačítko Load File otevírá testovací sadu dat, která je přímo v aplikace. Byla používaná při vývoji aplikace, aby se soubor nemusel opakovaně nahrávat. Může sloužit pro seznámení se s aplikací, pokud člověk nemá k dispozici vlastní data.

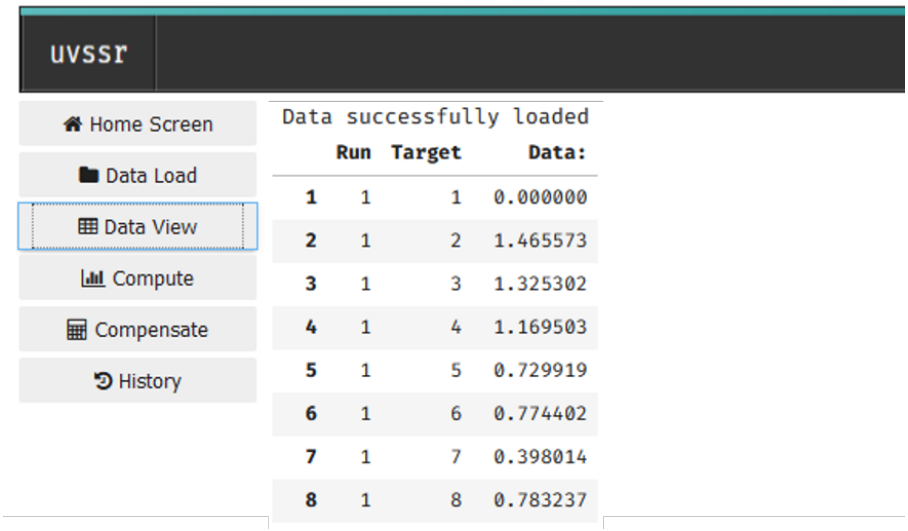
Obrázek 6.12: Okno pro Data Load — nahrávání dat



6.7.3 Data View

Okno Data View slouží pro prohlížení nahraných dat v tabulkovém režimu. V dalších iteracích vývoje by mohlo být umožněno data v tabulce editovat. To může být nebezpečné z hlediska produkčního nasazení, protože umožňuje data účelově měnit. Může to však být velká výhoda při testování.

Obrázek 6.13: Okno pro Data View — zobrazení a případná editace dat



	Run	Target	Data:
1	1	1	0.000000
2	1	2	1.465573
3	1	3	1.325302
4	1	4	1.169503
5	1	5	0.729919
6	1	6	0.774402
7	1	7	0.398014
8	1	8	0.783237

6.7.4 Compute

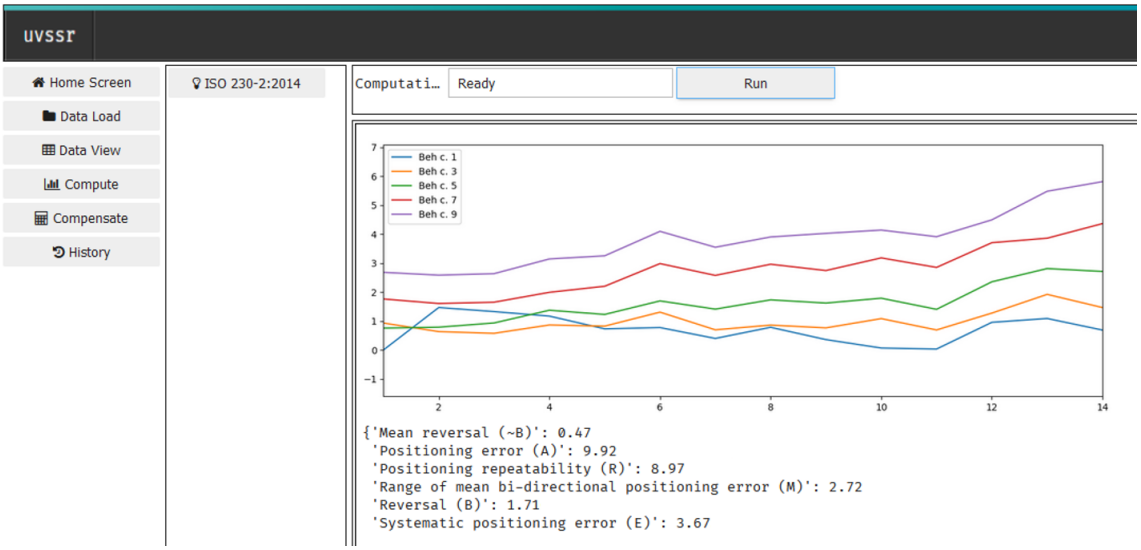
Okno Compute slouží pro výpočet statistických ukazatelů dle zvolené normy. Pro MVP verzi aplikace je k dispozici pouze norma ISO 230–2. Po výběru normy je třeba kliknout na tlačítko Run, čímž se spustí výpočet na pozadí. Výsledky jsou zobrazeny ve formě grafu a tabulky hodnot.

Pro MVP verzi byl zvolen jednoduchý graf, který znázorňuje průběh naměřených odchylek pro jednotlivé běhy. Tabulka ukazuje všechny vypočtené parametry.

Modularita

Výhodou je modularita aplikace, která umožňuje snadné dodělání dalších norem, podle kterých má výpočet proběhnout. Formát grafu, zobrazené ukazatele, formát tabulky i další designové prvky lze měnit. V důsledku požadavků MindSphere bude nutné design celkové přepracovat.

Obrázek 6.14: Okno pro Compute — výpočet dle zvolené normy



Srovnání výsledků

Zdrojová data byla použita při výpočtu v softwaru Carto i v naprogramované aplikaci. Na následujícím obrázku je srovnání vypočtených hodnot. Z obrázku jde vidět, že vytvořený program má zcela shodné výsledky jako software do Renishaw. Všechny hodnoty jsou stejné. Lze tedy konstatovat, že oba softwary počítají správně podle normy ISO 230–2.

Obrázek 6.15: Srovnání výsledků

Name	(+) [µm]	(-) [µm]	Bidir [µm]
Positioning error (A)	8.59	9.92	9.92
Positioning repeatability (R)	8.38	8.40	8.97
Systematic positioning error (E)	1.81	3.67	3.67
Reversal (B)			1.71
Mean reversal			0.47
Range of mean bidirectional positioning error (M)			2.72

```
{'Mean reversal (~B)': 0.47
 'Positioning error (A)': 9.92
 'Positioning repeatability (R)': 8.97
 'Range of mean bi-directional positioning error (M)': 2.72
 'Reversal (B)': 1.71
 'Systematic positioning error (E)': 3.67
```

7 Diskuse

V této kapitole subjektivně popisuji zkušenosti získané při psaní této práce a navrhuji změny v předložené aplikaci. Doporučuji také nové experimenty, které podle mne můžou pomoci dalšímu rozvoji v oblasti výrobní přesnosti.

7.1 Zadání a návrh řešení

Hlavním cílem práce bylo navrhnout vhodný způsob monitorování a posouzení výrobní přesnosti. Rozbor a rešerše problematiky ukázaly na možnost použití nových technologií, které se začínají ve světě uplatňovat v rozličných oborech. Jedná se o související témata IIoT — Industrial Internet of Things, digitalizace, edge-computing pod zastřešujícím názvem průmysl 4.0. Jedna z nejrychleji rozvíjejících se technologií je cloud-computing, případně pouze cloud. Cloudové systémy se začínají uplatňovat, i díky pokrokům v elektronice, ve stále více oborech. Průzkum firmy Bain & Company předpokládá pro tento rok celkové výdaje na IIoT 200 miliard dolarů [11]. Přestože toto číslo bude pravděpodobně v důsledku Koronaviru nižší, jedná se o velmi dynamicky se rozvíjející trh. Lidově řečeno vlak se již rozjel a kdo nechce zůstat pozadu, musí do něj naskočit. Pochopitelně se může časem ukázat, že tento vlak nejel zcela správným směrem, nic tomu však zatím nenasvědčuje. A v případě že směr je správný, je lepší být mezi pasažéry.

Cloud je moderní výraz, pod kterým se mnohdy skrývá ledasco. Kvalitně navržený cloudový systém má však obrovské výhody a potenciál pro mnoho podniků, výrobce výrobních strojů nevyjímaje. V praxi se používají tři formy cloudu: veřejný, privátní a hybridní. V práci byly vysvětleny výhody i nevýhody každé z nich. Zavedená firma ve strojírenském oboru — Siemens v nedávné době představila svoje vlastní řešení veřejného cloudu, který se specializuje právě na průmyslové podniky. Cloud od Siemensu s názvem MindSphere má nakročeno k tomu, stát se pro průmysl tím, co pro IT obor znamená Amazon AWS, tedy stát se prvním masově používaným poskytovatelem cloudových služeb.

MindSphere umožňuje připojení různých zařízení do počítačové sítě, sběr dat, jejich zpracovávání a zároveň poskytuje uživatelům cloudu aplikace a služby bez potřeby jejich instalace lokálně na zařízení. Cloudové aplikace jsou dostupné přes webové rozhraní v prohlížeči (Firefox, Google Chrome). S těmito možnostmi lze navrhnout a vystavět robustní systém pro monitorování nejen výrobní přesnosti. Ilustrační příklad. Výrobce CNC strojů bude každý vyrobený stroj osazovat soustavou senzorů pro měření vybraných fyzikálních veličin. Takto získaná data budou odesílána do cloudu, kde budou k dispozici výrobci pro další zpracování. Ten je může použít pro zlepšení koncepce konstrukce strojů v další vývojové verzi.

Jiný příklad. Výrobce součástek má několik výrobních hal po celém světě a v každé desítky podobných strojů. Namísto toho, aby na každém stroji měřil geometrickou přesnost jednotlivě pouze za účelem kompenzace daného stroje, bude data z měření nahrávat do cloudu. Na cloudu poběží program pro predikci vývoje geometrické přesnosti, založený klidně na strojovém učení, který bude mít díku cloudovému systému velké množství dat a bude poskytovat komplexní zpětnou vazbu.

Třetí příklad. Firma vyrábí součástky v jedné výrobní hale. Provozuje 20 strojů v nepřetržitém režimu. Pro vstupní materiál má několik dodavatelů. Vyčerpáním skladové zásoby

jednoho dodavatele začínají první stroje zpracovávat materiál od jiného dodavatele. Porovnávací měřidlo na výstupu výroby vyhodnocuje každý vyrobený kus a data odesílá do cloudu. V cloudu běžící program zaregistruje výkyv v kvalitě výrobků a okamžitě vydává pokyn ke korekci dřív, než dojde k nasazení materiálu na zbytek strojů. Použitím cloudu se v tomto případě předchází vyšším ztrátám takřka v reálném čase.

7.2 Měření geometrie

Součástí zadání bylo provést experiment na zvoleném výrobním stroji. Cílem experimentu bylo zvoleno měření a kompenzování chyb lineárního polohování os na tří-osé frézce MCV 754 Quick, která je k dispozici v laboratoři ÚVSSR při FSI na VUT. K měření byl použit laserový interferometer XL-80 od firmy Renishaw. Byl vysvětlen princip činnosti a změřeny všechny tři lineární osy stroje.

Následně byly osy kompenzovány s využitím softwaru Carto-Compensate od stejné firmy. Verifikační měření prokázalo vysokou účinnost softwarových kompenzací a tím potvrdilo užitnost této metody. Výsledky jsou shrnuty v tabulce 4.9. V případě, že je potřeba kompenzační tabulku vygenerovat pro jiné polohy, než které byly změřeny, používá software lineární interpolaci. Navrhuji provést experiment s cílem analyzovat vliv interpolační nebo obecně aproximační metody na výsledky. Tomu by mohl předcházet experiment, který by zkoumal vliv počtu měřených poloh a jejich rozmístění na účinnost kompenzací, resp. na přesnost měření. Z numerických metod je známo, že aproximační metody mají obecně vyšší přesnost při nepravidelném rozmístění měřených bodů. Měření lineárního polohování se však zpravidla dělá na rovnoměrně vzdálených bodech. To může ovlivňovat přesnost měření. Na ÚVSSR při FSI na VUT vzniká stanoviště s lineární osou a enkodéry v izolované obálce. Toto stanoviště se hodí pro navržené experimenty.

Informační ostrov
Byl vysvětlen pojem informačních ostrovů. Měření lineárního polohování s využitím softwaru Carto od firmy Renishaw je případem informačního ostrovu. Data jsou v softwaru uložena v databázi MongoDB a nebyl by tedy problém je zpřístupnit přes SQL dotazy, nebo exportovat prakticky v libovolném strukturovaném formátu. To by ovšem umožňovalo uživatelům snadný export do jiného softwaru a tedy opuštění uzavřeného ekosystému Renishaw. Carto proto umožňuje export pouze jako nestrukturovaný textový soubor.

Na druhou stranu je možné alespoň to. Existují nástroje pro práci s textovými soubory, především regulární výrazy, a lze tedy vytvořit program, který bude transformovat textové soubory na strukturované formáty, které umožňují efektivní práci s daty, jako je například json. Ve vyvinuté aplikaci je určitá forma zpracování textového souboru, aby bylo možné s daty dále pracovat, ale nebyl vytvořen modul pro export do universálních formátů. Považuji to za velmi užitečné a navrhuji proto takový transformační program vytvořit.

Chyba v softwaru Carto
Textový soubor exportovaný ze softwaru Carto je velmi nešikovně formátovaný. Konkrétně pro oddělování sloupců používá pevnou šířku mezery. Sloupce jsou tři. První sloupec značí číslo běhu, druhý sloupec číslo polohy a třetí sloupec je změřená hodnota. Mezi prvním a druhým

sloupcem je pevná mezera o šířce 3. Mezi druhým a třetím je pevná mezera o šířce 6. To je problém, protože jakmile se tato ”mantisa”zaplní číslicemi, další záznam není možný. To znamená že pro počet běhů má mantisa velikost 4 a maximální počet běhů je tedy 999 pro zachování sloupců a 9 999 při splynutí sloupců. Pro počet poloh je mantisa rovna 7 a maximální možný počet poloh je potom 999 999 pro zachování poloh a 9 999 999. Lze argumentovat, že tento počet běhů a poloh v praxi postačuje. Pro měření v provozu je to jistě pravda, ale pro experimenty to neplatí. S Ing., Dipl.-Ing M. Holubem, Ph.D. jsme v průběhu jara prováděli experiment s cílem pozorovat chování stroje v závislosti na denní době, přičemž laser se nechal běžet přes celý víkend. Řekněme, že se měření spustí v pátek odpoledne v 18 hodin a skončí v pondělí v 8 hodin. To je celkem 62 hodin = 3720 minut. Dále řekněme že jeden běh trvá přesně 1 minutu (reálná hodnota). To dává 3720 běhů, což překročilo hodnotu 999 čtyřikrát. První dva sloupce v tabulce splynou a je ověřeno, že samotný software Carto není schopen takovou tabulku přečíst. Nastává tedy situace, kdy stejný software není schopen přečíst soubor, který sám exportoval!

Obrázek 7.1: Renishaw Carto: Naprosto nevhodné formátování

```
9   1      3.9100496829
9   12     4.4922440829
9   123    5.4784992798
1   1234   5.8108478232
12  12345  6.6972634413
123 123456 6.4584261808
123412345675.6729482431
123412345673.6907478497
```

Řešení ze strany Renishaw by přitom bylo velice jednoduché. Stačilo by sloupce neoddělovat pevnou šířkou mezery, ale středníkem, nebo čárkou, jak se to v praxi běžně dělá. Řešení na straně uživatelů je naštěstí také jednoduché. Zápis hodnot totiž probíhá postupně, kdy se nejdříve zapíše hodnota prvního sloupce, poté se (při překročení hodnoty 9 999) přepíše hodnotou druhého sloupce a ta se zase přepíše hodnotou samotné odchylky. Lze tedy využít jinak zcela nevhodné konstrukce pevné šířky a data v softwaru nenačítat po sloupcích, ale vždy od 12. znaku na řádku, kde začíná hodnota odchylky. Budou tím sice zcela ztraceny informace o tom, pro kterou pozici ve kterém běhu byla hodnota odchylky změřena, to se však dá snadno dopočítat z informací o tom, kolik je měřených poloh.

Software Carto

Je jediný oficiálně podporovaný software pro měření laserem XL-80 od firmy Renishaw. Umí vyhodnocovat různé typy chyb podle několika norem. Subjektivní dojem je, že je značně nepřehledný a neintuitivní. I při opakovaném používají hledám potřebné funkce. Tuto zkušenost potvrzují i další uživatelé. Kromě vyhodnocení dle norem neumožňuje žádné další výpočty. Neumožňuje definovat vlastní funkce pro výpočet s daty. Existují tedy důvody, proč chtít data exportovat do jiného softwaru, například do Matlabu nebo do vlastního programu.

Návrh konkurenční aplikace

Navrhuji proto vytvořit vlastní aplikaci, do které bude možné importovat data z laseru XL-80, resp. libovolná data, která budou mít stejný charakter a budou převedena na požadovaný vstupní formát. Aplikace by mohla ze stejných důvodů podporovat i měřidlo Ballbal QC20-W od stejné firmy.

Je známo, že firma Renishaw reaguje na podněty uživatelů co do funkcí softwaru jen velmi pomalu. Některé žádosti o funkce jsou nevyslyšeny i několik let. Pokud by se navrhovaný software doplnil o žádané funkce, je možné, že by měl i komerční potenciál. V této práci byly představeny možnosti cloudových aplikací. Zřejmě existuje příležitost propojit tyto dvě myšlenky a vytvořenou demonstrační aplikaci rozšířit na plnohodnotnou náhradu Renishaw Carto, s přepracovaným designem, který bude konformní s požadavky Siemens, ho umístit do e-shopu MindSphere a nabízet komerčně.

V takovém případě navrhuji licenci, která by zvýhodňovala české podniky. Podle právních možností buď zcela zdarma pro podniky se sídlem a výrobou v ČR, nebo například se slevou 99 % oproti standardní ceně. Tím by se dosáhlo dvou věcí. Nezištné podpory českých podniků a finančního zisku. Tento přístup vnímám jako analogii k časté formě německého podnikání typu dceřinná společnost v ČR, která s nařízenou minimální marží prodává vše matce do Německa, kde se realizuje skutečný zisk, což je fakticky vzato moderní, právně uhlazená, forma kolonialismu.

7.3 Programování aplikace

V této části popisuji své zkušenosti z vývoje aplikace. Při zadání práce jsem měl zkušenosti s vývojem v Pythonu, vždy se ovšem jednalo pouze o jednoúčelové skripty, nebo programy které byly zakomponovány do větší aplikace. Naopak jsem neměl zkušenost s vývojem grafického uživatelského rozhraní, s webovými technologiemi a ani s cloudem. Musel jsem proto nejdříve nastudovat, co všechno vývoj aplikace pro cloud obnáší a jak vlastně cloud funguje. Například pro pochopení a naučení se s Cloud Foundry jsem prošel zdarma dostupným online kurzem, při kterém jsem si vytvořil dokumentaci k používání Cloud Foundry, která je delší než tato závěrečná práce. Přestože tedy výsledné ovládání Cloud Foundry lze abstrahovat na několik jednořádkových příkazů, musel jsem nejdříve do hloubky pochopit principy, což je nemalé investované úsilí. Drobné problémy s nedostatečným hardwarem na počátku byly promptně vyřešeny podporou ze strany ÚVSSR při FSI na VUT.

Webové aplikace se obecně skládají z front-endu, který zajišťuje interakci s uživatelem a back-endu, který dodává funkcionalitu. Protože znám jen základy HTML a nikdy jsem nevyvíjel ani rozsáhlejší webové stránky, natož aplikaci, stál jsem před rozhodnutím, zda se vydat cestou tradičních technologií a použít web-stack html, css, javascript, mysql, nebo najít úplně jinou cestu. Klasický web-stack je dnes v podstatě jedinou používanou cestou. Má mnoho různých variant v podobě různých technologií, dají se použít různé frameworky (například Angular, React, Vue), různé databáze, atd. Můj problém s těmito technologiemi je, že je neumím. Znamenalo by to tedy učit se rozsáhlou problematiku a několik různých jazyků a frameworků ”jenom”proto, abych aplikaci dal grafické uživatelské rozhraní. To se mi zdálo jako nepřiměřené úsilí vzhledem k cílům práce a proto jsem hledal nějak alternativu. Všechny alternativy jsou odsouzeny k tomu, že v principu pracují na překladu zvolené technologie do javascriptu, protože to je jediný jazyk, který prohlížeče dokážou interpretovat. Jelikož jsem měl zkušenost s Pythonem, hledal jsem možnosti programování webového front-endu právě v něm. Možností je několik, zmíním především zajímavý projekt Brython, což je implementace Pythonu, interpretovatelná v prohlížeči, která programátorovi zpřístupňuje DOM elementy [4]. Touto cestou jsem se ale nevydal a ani si nemyslím, že bude v budoucnu zajímavou alternativou ke klasickému web-stacku.

Jupyter

Alternativa, která mě naopak velmi zaujala, je projekt Jupyter. Jupyter je software, který umožňuje interaktivní programování. Prakticky vzato jde o webovou aplikaci, která se skládá z políček, která lze plnit kódem zvoleného programovacího jazyka, v mém případě Pythonem, a přímo pod tato políčka se vykreslují výsledky jako webové prvky. V pythonu přístup k prvků zprostředkovává knihovna ipywidgets. Pomocí Jupyteru lze snadno a co je důležité, výhradně pomocí Pythonu, vytvářet interaktivní webové aplikace. Doporučuji vyzkoušet demo [22].

Problém u Jupyteru je, že kód zůstává vystaven. Nejedná se tedy o plnohodnotnou aplikaci, kterou by bylo možné dodat koncovému uživateli. K tomu naštěstí vznikl další zajímavý projekt — Voilá, který přesně tuto funkcionalitu dodává [36]. Pomocí kombinace těchto dvou technologií lze vytvářet interaktivní webové aplikace výhradně pomocí Pythonu. Jasná volba pro demonstrační aplikaci v kontextu této práce.

Bylo nutné projít kompletní dokumentaci obou těchto projektů. Bez toho bych nebyl schopen začít aplikace programovat. Jednalo se o vyčerpávající činnost, ovšem stále jde o menší úsilí, než se naučit kompletní webový vývoj, alespoň dle mého soudu. Samotné programování front-endu pomocí Pythonu je poměrně příjemnou záležitostí a tyto technologie rozhodně doporučuji pro aplikace menšího rozsahu. Především umožňují každému, i méně zkušenému programátorovi jazyka Python, vytvářet webové aplikace.

Chyba lávky

Siemens intenzivně pracuje jak na vývoji MindSphere, tak na podkladové dokumentaci. Subjektivně můžu říct, že ta je dnes (červen) ve výrazně lepším stavu, než tomu bylo v lednu. Je obsáhlejší, strukturovanější a obsahuje užitečné návody. Spolu s tím ovšem Siemens zvyšuje, resp. upřesňuje i požadavky na aplikace do MindSphere. Dnes existuje již poměrně přesná specifikace, co musí aplikace splňovat a jak musí vypadat. Je k dispozici seznam povolených barev, webových prvků z dílny Siemens, které lze použít, atd. Požadavky jsou jen logické. Siemens chce, aby všechny aplikace v jeho cloudu byly uniformní a měly určitou estetickou úroveň. Aby se koncový uživatel MindSphere aplikací cítil v každé jako doma, ať už je aplikace vytvořená přímo týmem Siemens, nebo externími vývojáři.

Nicméně pro mě to znamenalo čáru přes rozpočet, protože Jupyter, potažmo Voilá jsou určeny ke tvorbě jednoduchých aplikací a jejich cílem je především nenáročnost na straně programátora, nikoli komplexita a přehršel nastavení, které vyžadují požadavky Siemens. Pravda je, že Voilá lze upravovat i na úrovni kaskádových stylů a javascriptu a teoreticky je tedy asi možné upravit vzhled aplikace tak, aby požadavky splňoval. Zaniká tím však hlavní výhoda použití této technologie, neboť se jedná o úpravy na úrovni klasického web-stacku. V takovém případě je už asi výhodnější front-end dělat od začátku pomocí vyspělých webových technologií, které jsou k tomu určeny.

Výpočetní jádro

Při programování výpočetního jádra aplikace jsem se seznámil s novými knihovnami, nebo prohloubil znalost známých. Pro samotné výpočty a manipulaci s daty jsem použil známou trojici knihoven Numpy, Scipy a Pandas. Především u posledních dvou jsem musel studovat množství dokumentace. Výpočty, především maticové, nejsou tak jednoduché a příjemné jako třeba v Matlabu, nicméně použitelnost je velká a tyto knihovny jsou výpočetně silné. Většina jaderných funkcí je implementována v jazyku C, nebo Fortranu a jsou tedy velmi rychlé. Kombinace

síly těchto knihoven a elegantnosti Pythonu způsobuje velkou oblíbenost u vědců, analytiků a vývojářů všeho druhu.

Přesto Python není pověstnou silver-bullet (jedna kulka na všechno) a místy se jeví jako vhodné doplnit jej dalšími technologiemi. V posledních letech se staly velmi populárními především dva programovací jazyky — Rust a Go (případně golang). Golang vznikl v dílnách Google jako řešení na jejich interní problémy s vývojem a především údržbou serverových soustav. Google není potřeba představovat, jen zdůrazňuji, že provozuje obří množství serverových technologií a k tomu má code-base (soustava programů) o několika milionech řádků kódu. Pro takovou firmu už má význam řešit efektivnost jazyka jak z hlediska výkonu na hardwaru, tak z hlediska lidských zdrojů (jak snadno se ho naučí nováčci, jak rychle se dokážou zorientovat v code-base, jak rychle dokážou začít dělat vlastní editace, atd.). Google pro vyřešení svých problémů vytvořil nový jazyk Go. Zle tedy konstatovat, že Go je mimořádně vhodný jazyk pro serverové aplikace, jakými jsou i cloudové aplikace. Pravděpodobně z těchto důvodů byl Golang zvolen jako primární jazyk pro vytvoření technologií Docker a Kubernetes. Obě technologie jsou kompletně napsané pomocí Golangu (s výjimkou části jádra Dockeru, kterou bylo nutné napsat v čistém C). Mimochodem Go je také tradiční stolní japonská strategická hra, která je charakteristická obrovským počtem kombinatorických možností odehrání partie (srovnej počet atomů v pozorovatelném vesmíru $= 10^{80}$ a počet možností her Go $= 2 * 10^{170}$) a z toho důvodu odolávala z tradičních stolních her nejdéle pokusům o zdolání lidských hráčů umělou inteligencí. Poprvé se to povedlo týmu DeepMind (odkoupeno Googlem) pomocí jejich programu AlphaGo v roce 2016, kdy program porazil v 5 hrát po sobě světového mistra v Go — Lee Sedola [6].

Rust je velice mladý jazyk určený primárně pro systémové programování. Jeho tvůrci chtěli vytvořit jazyk, který bude bezpečný (z obrovského průzkumu Microsoftu, který zveřejnil své interní statistiky o chybách ve vlastních softwarech za posledních 20 let, vyplynulo, že takřka přesně 70 % všech chyb je typu memory safety — memory buffer overflow, race conditions, atd.) a současně rychlý. Jde o dva protichůdné požadavky a prakticky všechny dosavadní jazyky mají jako přednost jedno, nebo druhé. C, C++ jsou jazyky extrémně rychlé, zato však extrémně nebezpečné (je snadné vytvořit v nich chybu). Naproti tomu třeba Python nebo JavaScript jsou poměrně bezpečnější, ale řádově pomalejší. Nicméně vývojářům Rustu se to povedlo a Rust nedovoluje prakticky vůbec dopustit se chyb typu memory safety, při zachování rychlosti blízké nativní (je o procenta pomalejší než C). Jak ve svém revolučním díle Pojednání o podstatě a původu bohatství národů napsal Adam Smith: ”Žádný oběd není zadarmo”a výborně to rozebírá Tomáš Sedláček ve své mezinárodně oceněné knize Ekonomie dobra a zla [34] [2]. A tedy ani v případě Rustu není dosažení rychlosti a bezpečnosti zadarmo. Cenou je komplexita jazyka, která zkrátka je poměrně vysoká a především jiná než u známých rodin jazyků. Povědomý tedy Rust není ani pro vývojáře C++, Java, Haskell, ani Python. Na stranu druhou je Rust navržen s ohledy na všechny moderní požadavky programátorů a z toho důvodu se letos stal již pátým rokem v řadě nejmilovanějším jazykem mezi vývojáři [29]. Mimochodem Python je druhý nejmilovanější jazyk hned po Rustu.

Z těchto důvodů se budu oběma jazyky zabývat a navrhuji další vývoj aplikací pomocí těchto technologií.

Nové pro mě bylo používání regulárních výrazů. Jedná se o jazykové agnostický nástroj, existuje však implementace v Pythonu. Jde o knihovnu re (Re je také egyptský bůh Slunce). Jedná se o velmi efektivní nástroj pro práci s textem.

Zcela novou technologií pro mě byl i tzv. template engine (tvorba šablon). V Pythonu existuje více implementací z nichž jsem si vybral knihovnu jinja2. Pomocí této knihovny je naprogramován program pro generování NC kódu s kompenzačními daty.

Závěr
Python je skvělý jazyk, který je však vhodné doplnit dalšími technologiemi (rozšiřovat si tool- set). Já jsem si pro další studium zvolil jazyky Rust a Go. Tato trojice je podle mne, obzvláště v kontextu technologie WASM, mocným tool-setem pro tvorbu cloudových aplikací.

Webový front-end jednoduchých aplikací lze efektivně řešit pomocí nástroje Jupyter v kombinaci s Voilá. Komplexní aplikace je však vhodnější vyvíjet pomocí plnohodnotných webových technologií, z nichž doporučuji kombinaci TypeScriptu a frameworku Vue.js.

Software od Renishaw není perfektní a existuje prostor pro konkurenční produkt, který navrhuji postavit se zmíněnými technologiemi jako cloudovou aplikaci (nemusí nutně běžet na cloudu od Siemensu).

7.4 Cloud

V posledních letech lze pozorovat dynamický rozvoj technologií označovaných jako IoT (případně IIoT), cloud-computing, edge-computing, industry 4.0, chytrá domácnost, internet věcí, atd., v podstatě jen čekáme, až bude ohlášen internet všeho a povinné připojení všech (jak to pozorujeme u čínského velkého bratra v podobě tzv. systému sociálního indexu, předpovídaný ani ne tak Orwellem, jehož svět stojí na odlišných základech, ale spíše Huxleyho Brave New World).

Nicméně přínos cloudových technologií pro různé obory od akademického výzkumu až po výrobní továrnu jsou neoddiskutovatelné a byly v práci ukázány. Cloud je možné realizovat dvěma základními způsoby: jako veřejný, nebo jako soukromí. Možný je i hybridní model, kdy jsou použity oba. Veřejný cloud znamená, že si pronajmu kýženou službu za finance od poskytovatele. Soukromý pak znamená, že si všechno musím postavit a provozovat sám. Oba přístupy mají své výhody a nevýhody. Z toho důvodu se často používá právě hybridní model, který umožňuje využít výhody obou přístupů.

Navrhuji pokračovat ve vývoji cloudové aplikace, rozšiřovat ji o další funkce, jak bylo naznačeno v předešlé podkapitole a sledovat vývoj v této oblasti.

Siemens
Není poskytovatelem cloudu v pravém slova smyslu, protože sám žádný neprovozuje. Namísto toho postavil nádstavbu nad řešením Cloud Foundry od Linux Foundation, dal tomu své logo, své barvy a svá tlačítka a umožňuje tuto nádstavbu provozovat u plnohodnotných poskytovatelů, z nichž si zákazník Siemensu může vybrat (hostování je možné u Amazonu, Microsoftu a Google). Toto není kritika postupu Siemensu. Naopak tento postup je jen logický a jde o variantu abstrahování od komplexity, což bylo v práci také popsáno. Siemens tedy poskytuje PaaS řešení, tedy platformu jako službu. Zároveň pomocí pravidel vynucuje určitou bezpečnostní a estetickou kvalitu aplikací, které přes svou platformu poskytuje. To zákazníkům konečných aplikací poskytuje záruky v oblasti kybernetické bezpečnosti a ergonomie, plynoucí z faktu, že všechny MindSphere aplikace jsou vizuálně prakticky stejné.

Zároveň však Siemens pracuje na tom, aby bylo možné na tuto platformu připojit co nejvíce (opět, podle Huxleyho ideálně všechna) zařízení z provozu. Vyvinul proto vlastní hardware pro připojení a pracuje na novém řídicím systému, který bude řekněme data-centrický. Bude tedy podporovat snadné připojení do cloudu a efektivní sdílení dat. Siemens tím na jednu stranu buduje pravděpodobně vendor lock-in ekosystém, ze kterého nebude pro zákazníky snadná cesta ven, na stranu druhou však vytváří mocnou platformu, kdy jediná firma bude schopná zajistit všechny potřeby zákazníků — od řídicího systému strojů, před PLC pro připojení, Edge-severů až po kompletní cloudové řešení.

Z veřejně dostupných informací lze dovozovat, přestože to nebylo dosud oficiálně ohlášeno, že uzavřel strategické partnerství s Microsoftem, což je firma s obrovskými možnostmi co do technologií i financí. Předpokládám, že i díky tomuto partnerství se Siemens stane firmou s monopolním postavením na trhu průmyslových cloudových systémů.

Doporučuji proto s cloudovými prvky od Siemensu na ÚVSSR při FSI na VUT dále pracovat a tyto technologie rozvíjet. Budou totiž v následujících letech hrát významnou roli.

Soukromý ústavní cloud
Podle zkušeností s některými českými firmami, i podle dostupných průzkumů (viz např. [11]), mají firmy strach posílat citlivá data do cloudu v případě, že se nachází v cizí zemi, což teprve na jiném kontinentu. V některých případech to ani není v důsledku GDPR možné. Přistupují potom ke zmíněnému hybridnímu modelu, kdy citlivá data ponechávají pouze na vlastním cloudu. Mnoho výrobních podniků v ČR však nemá personální ani odborné kapacity na to, aby realizovali vlastní cloud, není proto ostatně ani oprávněný důvod (analogická situace platí pro české nemocnice, perfektně o tom pojednává rozhovor s vedoucím informačních služeb pražské Nemocnice na Bulovce [24]).

Za předpokladu, že na ústavu bude pokračovat vývoj cloudových aplikací a dojde k vývoji komplexního systému pro řízení výrobní přesnosti (což je myslím reálná možnost), navrhuji vytvořit soukromý cloud, jehož provozovatelem by byl ústav, nebo k tomu účelu zřízená přidružená firma. Tento cloud by českým podnikům zaručoval ponechání dat na území ČR a ústav by se současně stal ručitelem za vývoj na míru. Vidím v této konfiguraci potenciál pro symbiotické partnerství. Privátní cloud by se mohl realizovat buď jako PaaS, podobně jako to dělá Siemens, ovšem v menším měřítku a u českého poskytovatele. Druhou možností je provozování vlastní infrastruktury, což je po stránce technologií velmi zajímavá oblast. Zmínit lze nástroje Docker, Kubernetes, Puppet, OpenStack, Rancher a Terraform.

7.5 Další vývoj

V této kapitole předkládám návrh na experimenty, které by podle mne bylo vhodné provést v kontextu zvyšování výrobní přesnosti.

Aproximační metody
Pro kompenzování chyb lineárního polohování se postupuje tak, že nejdříve se změří v určených polohách chyba polohování a pak se tyto chyby softwarově kompenzují. Kompenzaci je možné dělat buď ve stejných polohách, nebo v jiných, přičemž pak je ale nutné odhadnout — aproxi-

movat hodnotu chyby ve zvolené poloze. Software od Renishaw používá jednoduchou lineární interpolaci.

Je to metoda nejjednodušší, ale také ne nejpřesnější. Mám proto záměr provést experiment s cílem analyzovat vliv chyby interpolační metody na přesnost kompenzování. Druhým cílem experimentu bude analyzovat vliv počtu měřených poloh a vliv rozmístění poloh na přesnost kompenzování. Tomuto experimentu se budu věnovat přes léto a výsledky prezentovat na konferenci Mechatronika 2020, na kterou jsem již přihlášen [1].

Tuhost strukturální smyčky

Při studiu odborných článků při psaní kapitoly Přehled současného stavu poznání, mne zaujala problematiky tuhosti strukturální smyčky. To je soustava obrobku, nástroje a všech strojních uzlů mezi nimi (vřeteno, rám, stojan, atd.). Rád bych analyzoval závislost přesnosti kompenzování geometrie na velikosti výsledných sil při dokončovacích operacích. Jinak řečeno jaký vliv mají dokončovací operace, resp. tuhost strukturální smyčky na geometrickou přesnost.

Navrhuji první fázi experimentu provést pomocí přístroje Loaded Double Ballbar, což je ballbar, který je schopen vyvíjet sílu na strukturální smyčku, při měření. Jedná se v principu o simulaci dokončovacích operací.

8 Závěr

V této kapitole jsou shrnuty výsledky, kterých bylo dosaženo. Jsou připomenuty cíle práce. A jsou uvedeny návrhy autora pro další činnost.

Práce nese název Monitorování výrobní přesnosti. Název přesně vystihuje dvě ústřední témata, kterými se práce zabývá. Jsou to monitorování a výrobní přesnost. Dle zadání mělo být vypracována analýza současného stavu poznání. Bylo tak učiněno v kapitole 3, přičemž pro analýzu byl použit systémový přístup.

Bylo také zadáno, provést experiment na zvoleném CNC obráběcím stroji. Protože se práce zabývá výrobní přesností, pro experiment bylo zvoleno měření přesnosti lineárního polohování pomocí soustavy laser-interferometru XL-80 od firmy Renishaw. Kromě měření byla provedena i softwarová kompenzace všech trí lineárních os. A verifikačním měřením byla ověřena účinnost provedené kompenzace. Lze konstatovat, že experiment proběhl úspěšně, včetně skvělých kompenzačních výsledků, uvedených v kapitole 4.

Výsledky měly být také statisticky zpracovány a vyhodnoceny. Ke zpracování byl zvolen software Carto do stejné firmy a vyhodnocení proběhlo dle mezinárodní normy ISO 230–2. Takto se došlo k výsledkům uvedených v závěru kapitoly 4.9.

Dalším dílčím cílem bylo provést systémový rozbor problematiky a navrhnout vhodný systém monitorování. Na základě informací získaných analýzou současného stavu poznání a také ze zkušeností nabitých při experimentu, bylo navrženo použít pro monitorování cloud. Systémovým rozborem se došlo k tomu, že vhodným způsobem realizace cloudového systému je veřejný cloud od poskytovatele Siemens s názvem MindSphere, popsaným v kapitole 5. Pro ověření tohoto systému bylo navrženo vytvořit demonstrační aplikaci pro tento cloud. Aplikace byla naprogramována pomocí programovacího jazyka Python a umožňuje provést vyhodnocení statistických charakteristik dle zmíněné normy, vizualizovat výsledky měření na grafech a generovat kompenzační data v podobě NC kódu pro zvolený řídicí systém, takže není nutné je do stroje přepisovat ručně, ale stačí na něm spustit vygenerovaný program. Ten zajistí přepsání kompenzačních tabulek a jejich aktivaci. Popis v kapitole 6. Aplikace byla vytvořena jako vysoce modulární a lze díky tomu snadno přidávat další funkce, například vyhodnocení dle jiných norem, nebo přidávat moduly pro sledování historického vývoje.

Posledním dílčím cílem bylo vyvodit vlastní závěry z oblasti navrženého postupu. Na základě všech provedených úkonů byl zhodnocen celkový postup, konstatovány výhody, nevýhody a chyby při implementaci cloudové aplikace. Bylo také doporučeno provést další experimenty v oblasti výrobní přesnosti. V diskusi v kapitole 7 byl nastíněn další možný výzkum a vývoj, kterým se lze na ÚVSSR při FSI na VUT ubírat.

Na úplný závěr lze říci, že autor hodnotí přínosy práce jako pozitivní a plánuje věnovat se navrhovaným tématům na doktorském studiu.

Bibliografie

[1] *19th Mechatronika 2020, Prague, Czech Republic*. [Online; accessed 26. Jun. 2020]. Červ. 2020. URL: https://mechatronika.fel.cvut.cz.

[2] Smith Adam. *Pojednání o podstatě a původu bohatství národů*. cze. 1. vyd. Praha: Praha: Liberální institut, 2016. ISBN: 978-80-86389-60-8.

[3] *Arduino Nano | Arduino Official Store*. [Online; accessed 11. Jun. 2020]. Červ. 2020. URL: https://store.arduino.cc/arduino-nano.

[4] brython-dev. *brython*. [Online; accessed 26. Jun. 2020]. Červ. 2020. URL: https://github.com/brython-dev/brython.

[5] *conda-forge | community driven packaging for conda*. [Online; accessed 8. Jun. 2020]. Červ. 2020. URL: https://conda-forge.org/#page-top.

[6] Contributors to Wikimedia projects. *AlphaGo versus Lee Sedol - Wikipedia*. [Online; accessed 26. Jun. 2020]. Květ. 2020. URL: https://en.wikipedia.org/w/index.php?title=AlphaGo_versus_Lee_Sedol&oldid=959887935.

[7] Contributors to Wikimedia projects. *Information Age - Wikipedia*. [Online; accessed 9. Jun. 2020]. Květ. 2020. URL: https://en.wikipedia.org/w/index.php?title=Information_Age&oldid=959363082.

[8] *Etalon Laser-tracer NG*. [Online; accessed 19. Jun. 2020]. Červ. 2020. URL: https://www.etalon-gmbh.com/en/products/lasertracer.

[9] Ian Foster et al. *Cloud Computing for Science and Engineering (Scientific and Engineering Computation)*. The MIT Press, zář. 2017. ISBN: 978-026203724-2. URL: https://www.amazon.com/Computing-Science-Engineering-Scientific-Computation/dp/0262037246.

[10] Michal Holub. "Vliv geometrické přesnosti vybraných obráběcích center na požadované vlastnosti výrobků = Effect of geometrical precision machining centers on the desired charakteristicts of the goods". cze. In: (2011).

[11] *Industrial IoT platform: Buy it pre-built or make your own?* [Online; accessed 9. Jun. 2020]. Ún. 2020. URL: https://www.plm.automation.siemens.com/global/en/topic/industrial-iot-platform/70070.

[12] *Internetová jazyková příručka*. [Online; accessed 15. Jun. 2020]. 2004. URL: https://prirucka.ujc.cas.cz/?slovo=monitorov%C3%A1n%C3%AD.

[13] *Internetová jazyková příručka*. [Online; accessed 15. Jun. 2020]. 2004. URL: https://prirucka.ujc.cas.cz/?slovo=syst%C3%A9m.

[14] *Internetová jazyková příručka*. [Online; accessed 9. Jun. 2020]. 2004. URL: https://prirucka.ujc.cas.cz/?slovo=normalizace.

[15] *ISO 230-2:2014/Amd 1:2016*. [Online; accessed 23. Jun. 2020]. Červ. 2020. URL: https://www.iso.org/standard/63976.html.

[16] *ISO 9000:2015*. [Online; accessed 15. Jun. 2020]. Červ. 2020. URL: https://www.iso.org/standard/45481.html.

[17] Přemysl Janíček. *Expertní inženýrství v systémovém pojetí*. cze. 1. vyd. Expert (Grada). Praha: Grada, 2013. ISBN: 978-80-247-4127-7.

[18] Jiří Marek. *Konstrukce CNC obráběcích strojů IV.0*. cze. MM speciál. Praha: MM publishing, s.r.o., 2018. ISBN: 978-80-906310-8-3.

[19] *MindSphere Design System*. [Online; accessed 25. Jun. 2020]. Květ. 2020. URL: https://design.mindsphere.io.

[20] *Miniconda — Conda documentation*. [Online; accessed 8. Jun. 2020]. Květ. 2020. URL: https://docs.conda.io/en/latest/miniconda.html.

[21] *Návrhové principy: SOLID - Zdroják*. [Online; accessed 10. Jun. 2020]. Květ. 2012. URL: https://www.zdrojak.cz/clanky/navrhove-principy-solid.

[22] *Project Jupyter*. [Online; accessed 26. Jun. 2020]. Červ. 2020. URL: https://jupyter.org/try.

[23] R. Ramesh, M. A. Mannan a A. N. Poo. "Error compensation in machine tools — a review: Part I: geometric, cutting-force induced and fixture-dependent errors". In: *Int. J. Mach. Tools Manuf.* 40.9 (čvc 2000), s. 1235–1256. ISSN: 0890-6955. DOI: 10.1016/S0890-6955(00)00009-2.

[24] J. Sedlk. "Martin Koníř (Nemocnice Na Bulovce): Chtěli jsme naši eNeschopenku uvolnit zdarma. Prý bychom ale okradli stát". In: *Lupa.cz* (červ. 2020). URL: https://www.lupa.cz/clanky/martin-konir-nemocnice-na-bulovce-chteli-jsme-nasi-eneschopenku-uvolnit-zdarma-pry-bychom-ale-okradli-stat.

[25] Die Schittigs. *ctrlX CORE - The ultra-compact control platform*. [Online; accessed 4. Jun. 2020]. Květ. 2020. URL: https://apps.boschrexroth.com/microsites/ctrlx-automation/en/portfolio/ctrlx-core.

[26] *Siemens | MindSphere*. [Online; accessed 22. Jun. 2020]. Červ. 2020. URL: https://siemens.mindsphere.io/en.

[27] H. A. M. Spaan. "Software error compensation of machine tools". In: *Technische Universiteit Eindhoven onderzoeksportaal* (1995). DOI: 10.6100/IR451256.

[28] *Stack Overflow Developer Survey 2019*. [Online; accessed 8. Jun. 2020]. Červ. 2020. URL: https://insights.stackoverflow.com/survey/2019#technology-_-most-popular-development-environments.

[29] *Stack Overflow Developer Survey 2019*. [Online; accessed 26. Jun. 2020]. Červ. 2020. URL: https://insights.stackoverflow.com/survey/2019#technology-_-most-loved-dreaded-and-wanted-languages.

[30] *System Buildpacks | Cloud Foundry Docs*. [Online; accessed 22. Jun. 2020]. Červ. 2020. URL: https://docs.cloudfoundry.org/buildpacks/system-buildpacks.html.

[31] Kroly Szipka, Theodoros Laspas a Andreas Archenti. "Measurement and analysis of machine tool errors under quasi-static and loaded conditions". In: *Precis. Eng.* 51 (led. 2018), s. 59–67. ISSN: 0141-6359. DOI: 10.1016/j.precisioneng.2017.07.011.

[32] *Technology Innovation - Future of Production*. [Online; accessed 15. Jun. 2020]. 2017. URL: http://www3.weforum.org/docs/WEF_White_Paper_Technology_Innovation_Future_of_Production_2017.pdf.

[33] *The LEVEL-UP Project*. [Online; accessed 11. Jun. 2020]. Červ. 2020. URL: https://www.levelup-project.eu.

[34] Sedláček Tomáš. *Ekonomie dobra a zla: po stopách lidského tázání od Gilgameše po finanční krizi*. cze. 2. vyd. Praha: 65. pole, 2012. ISBN: 978-80-87506-10-3.

[35] *Visual Studio Code*. [Online; accessed 8. Jun. 2020]. Červ. 2020. URL: https://code.visualstudio.com.

[36] voila-dashboards. *voila*. [Online; accessed 26. Jun. 2020]. Červ. 2020. URL: https://github.com/voila-dashboards/voila.

[37] *Wolfram|Alpha: Making the world's knowledge computable*. [Online; accessed 11. Jun. 2020]. Červ. 2020. URL: https://www.wolframalpha.com.

Seznam obrázků

Seznam tabulek

SEZNAM PŘÍLOH

K práci je přiložena jediná příloha, archiv ”cfdev_app.7z”s aplikací.