

Automatické generování drah robota ABB pro tangenciální řezání

Bc. Štěpán Danzmajer

Diplomová práce
2025



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav automatizace a řídicí techniky

Akademický rok: 2024/2025

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: Bc. Štěpán Danzmajer
Osobní číslo: A23343
Studijní program: N0714A150006 Automatické řízení a informatika v průmyslu 4.0
Forma studia: Prezenční
Téma práce: Automatické generování drah robota ABB pro tangenciální řezání
Téma práce anglicky: Automatic Path Generation for ABB Robots for Tangential Cutting

Zásady pro vypracování

- Vypracujte literární rešerši na dané téma.
- Navrhněte držák řezacího nástroje i s ohledem na bezpečnost.
- Připravte simulační prostředí pro danou aplikaci a s potřebnými komponenty.
- Vytvořte postup pro zpracování křivek z CAD programu do programovacího jazyka RAPID.
- Připravte jednoduchou aplikaci na ovládacím panelu robota pro řízení parametrů řezání.
- Aplikaci otestujte alespoň virtuálně a výsledky zhodnotte.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. KOLÍBAL, Zdeněk. Roboty a robotizované výrobní technologie. Brno: Vysoké učení technické v Brně – nakladatelství VUTUM, 2016. ISBN 978-80-214-4828-5.
2. DJUKANOVIC, Milena; GRUJICIC, Rade; RADUNOVIC, Luka a BOSKOVIC, Vuk. Conceptual Solution of the Robotic Arm/Plotter. Online. In: KARABEGOVIĆ, Isak (ed.). New Technologies, Development and Application. Lecture Notes in Networks and Systems. Cham: Springer International Publishing, 2019, s. 170-179. ISBN 978-3-319-90892-2. Dostupné z: https://doi.org/10.1007/978-3-319-90893-9_21.
3. ROSSI, Cesare a SAVINO, Sergio. Robot trajectory planning by assigning positions and tangential velocities. Online. Robotics and Computer-Integrated Manufacturing. 2013, roč. 29, č. 1, s. 139-156. ISSN 07365845. Dostupné z: <https://doi.org/10.1016/j.rcim.2012.04.003>.
4. AYARI, Oussama; BOUALI, Anis a MÉAUSOONE, Pierre-Jean. Cutting forces and accuracy characterization during wood machining with serial robots. Online. European Journal of Wood and Wood Products. 2020, roč. 78, č. 4, s. 767-775. ISSN 0018-3768. Dostupné z: <https://doi.org/10.1007/s00107-020-01539-4>.
5. VILLANUEVA, Emanuel Martinez; MAMLEDESAI, Harshavardhan; MARTINEZ, Pablo; POOSTCHI, Peyman a AHMAD, Rafiq. Design and simulation of an automated robotic machining cell for cross-laminated timber panels. Online. Procedia CIRP. 2021, roč. 100, s. 175-180. ISSN 22128271. Dostupné z: <https://doi.org/10.1016/j.procir.2021.05.026>.

Vedoucí diplomové práce: **Ing. Ľuboš Spaček, Ph.D.**
Ústav řízení procesů

Datum zadání diplomové práce: **13. prosince 2024**
Termín odevzdání diplomové práce: **27. května 2025**

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan



prof. Ing. Vladimír Vašek, CSc. v.r.
ředitel ústavu

Ve Zlíně dne 13. prosince 2024

PROHLÁŠENÍ AUTORA DIPLOMOVÉ PRÁCE

Beru na vědomí, že

- odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., v platném znění bez ohledu na výsledek obhajoby;
- diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- jedno vyhotovení diplomové práce v listinné podobě bude ponecháno Univerzitě Tomáše Bati ve Zlíně k uložení;
- na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 odst. 2 a 3 mohu užít své dílo – diplomovou práci – nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá; neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji, že

- jsem na diplomové práci pracoval(a) samostatně a použitou literaturu jsem řádně citoval(a); v případě publikace výsledků budu uveden(a) jako spoluautor;
- odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné.
- při tvorbě této práce jsem použil nástroj generativního modelu AI ChatGPT (<https://chat.openai.com>) za účelem jazykové korektury, stylistických úprav a revize formulací textu. Po použití tohoto nástroje jsem provedl kontrolu obsahu a přebírám za něj plnou zodpovědnost.

Ve Zlíně, dne

.....
podpis autora

ABSTRAKT

Tato diplomová práce komplexně propojuje historický vývoj průmyslové robotiky s návrhem a implementací plně automatizovaného systému pro robotické tangenciální řezání měkkých materiálů. Praktická část demonstruje převod DXF výkresů na dráhy nástroje: knihovny ezdxf, NumPy/SciPy, OpenCV a Matplotlib automaticky segmentují polylinie, interpolují křivky metodou PCHIP a generují datové pakety, které jsou přes ABB Robot Web Services přenášeny do řadiče robota. V jazyce RAPID následně vzniká adaptivní algoritmus pohybu s ochranou proti přetočení osy 6 a s inteligentním přerušením řezu při ostrých lomových úhlech, vizualizace v RobotStudios je řízena digitálními signály. K mechanické integraci je navržen lehký, servisně přívětivý držák oscilační čepele a webový ovládací panel v AppStudio, který zjednodušuje práci obsluhy. Experimenty prokázaly škálovatelné řešení vhodné pro akademické i průmyslové nasazení.

Klíčová slova: Robot, ABB RobotStudio, Tangenciální řezání, Oscilační nůž

ABSTRACT

This thesis comprehensively links the historical development of industrial robotics with the design and implementation of a fully automated system for robotic tangential cutting of soft materials. The practical part demonstrates the conversion of DXF drawings into toolpaths: the ezdxf, NumPy/SciPy, OpenCV and Matplotlib libraries automatically segment polylines, interpolate curves using the PCHIP method and generate data packets that are transmitted to the robot controller via ABB Robot Web Services. The RAPID language then generates an adaptive motion algorithm with axis 6 roll protection and intelligent interruption of the cut at sharp break angles; the visualization in RobotStudio is driven by digital signals. For mechanical integration, a lightweight, service-friendly blade holder and a web-based control panel in AppStudio are designed to simplify the operator's work. Experiments have demonstrated a scalable solution suitable for academic and industrial deployments.

Keywords: Robot, ABB RobotStudio, Tangential cutting, Oscilating knife

OBSAH

ÚVOD.....	11
I TEORETICKÁ ČÁST	12
1 HISTORIE PRŮMYSLVÝCH ROBOTŮ	13
1.1 VÝROBA PŘED ROBOTY	13
1.2 30. LÉTA 20. STOLETÍ: ZROD PRŮMYSLVÝCH ROBOTŮ	14
1.3 80. A 90. LÉTA 20. STOLETÍ: POČÍTAČEM ŘÍZENÝ ROBOT SE STAL STANDARDEM	15
1.4 10. LÉTA 21. STOLETÍ.....	16
1.5 20. LÉTA 21. STOLETÍ.....	16
2 IDENTIFIKACE NEBEZPEČÍ.....	18
2.1 VZÁJEMNÉ PŮSOBNÍ ČLOVĚKA A STROJE	18
2.2 STAVY STROJE	18
2.3 NEČEKANÉ AKCE OBSLUHY NEBO PORUCHY STROJE.....	19
2.4 ODHAD RIZIKA	19
2.5 OBSAH RIZIK	19
2.6 UVAŽOVANÁ HLEDISKA PŘI ODHADU RIZIKA.....	20
3 ABB RAPID	23
3.1 PROGRAMOVACÍ JAZYKY	23
3.2 FUNKCE.....	24
3.3 PARAMETRY	24
3.4 PŘERUŠENÍ	24
3.5 PŘÍKLADY ZÁKLADNÍCH POHYBOVÝCH INSTRUKCÍ:	25
4 OSCILUJÍCÍ NŮŽ	26
4.1 ULTRAZVUKOVÝ NŮŽ.....	27
4.1.1 Princip UVA.....	28
5 PIECEWISE-CUBIC HERMITE INTERPOLATING POLYNOMIAL.....	31
6 TANGENCIÁLNÍ ŘEZÁNÍ ROBOTEM.....	33
6.1 KONVENČNÍ PLÁNOVÁNÍ POHYBU OD BODU K BODU	34
6.2 PLÁNOVÁNÍ PŘÍŘAZENÍM POZICE A TEČNÝCH TRAJEKTORIÍ DO TRAJEKTORIE	37
6.3 PLÁNOVÁNÍ KLOUBOVÉHO POHYBOVÉHO ZÁKONA PRO „OBÁLKU TEČEN“	39
II PRAKTICKÁ ČÁST	41
7 POUŽITÉ KNIHOVNY.....	42
7.1 EZDXF.....	42
7.2 OPENCV	42

7.3	NUMPY.....	43
7.4	MATPLOTLIB	44
7.5	SCIPIY	44
7.6	REQUESTS.....	45
7.7	URLLIB3.....	45
8	NAČTENÍ DXF SOUBORU	46
8.1	OVĚŘENÍ NAČTENÝCH DAT	47
9	HLEDÁNÍ ÚSEČEK	50
10	VYTVÁŘENÍ SKUPIN Z BODŮ, KTERÉ NEJSOU PŘÍMKY	53
11	TVOŘENÍ KŘIVEK Z BODŮ	55
12	PŘÍPRAVA DAT PRO PŘENOS	58
12.1	ÚSEČKY	58
12.2	PŘEVEDENÍ DAT DO FORMÁTU PRO PŘENOS (KŘIVKY)	58
13	PŘENOS DAT DO ROBOTA	60
13.1	PYTHON.....	60
13.2	RAPID	61
14	VYKRESLENÍ.....	62
14.1	KRESLENÍ KŘIVEK	62
14.2	VYKRESLENÍ ÚSEČKY	64
15	NÁVRH UCHYCENÍ.....	66
16	NÁVRH OVLÁDACÍHO PANELU	69
	ZÁVĚR.....	70
	SEZNAM POUŽITÉ LITERATURY	71
	SEZNAM OBRÁZKŮ	74
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	75
	SEZNAM PŘÍLOH.....	76

ÚVOD

Průmyslový roboti se za šest desetiletí posunuly od prvních počítačem řízených manipulátorů řady Unimate k dnešním kolaborativním systémům schopným vnímat okolí a bezpečně sdílet pracovní prostor s člověkem. Exponenciální nárůst jejich výkonu, přesnosti a programovatelnosti způsobil, že robotizace již není výsadou automobilového průmyslu, ale stává se hybnou silou digitalizace napříč výrobními odvětvími. Tradiční metody ručního zpracování proto ustupují plně automatizovaným pracovištím, která jsou schopna přizpůsobit se zakázce v řádu minut – typicky prostřednictvím softwarové úpravy trajektorie.

Tato diplomová práce se věnuje komplexnímu řešení robotického vyřezávání plošných materiálů oscilačním nožem. První část představuje algoritmus, jenž z DXF výkresu automaticky extrahuje úsečky a křivky, segmentuje je podle změny směru a pomocí PCHIP interpolace zhušťuje řídké polylinie tak, aby výsledná trajektorie zachovala definovanou přesnost při minimálním objemu dat. Navazující skript v jazyce Python převádí normalizované souřadnice do dvou bloků, které prostřednictvím protokolu Robot Web Services zapisuje do perzistentních struktur ovladače.

Druhá část popisuje tvorbu RAPID programu, jenž z přijatých bodů sestavuje lineární a tečné pohyby, sleduje omezení rychlosti, tlumí trhání dráhy a aktivně řídí digitální signály pro vizuální kontrolu řezu. Zvláštní důraz je kladen na bezpečné překonávání singularit a limitních úhlů šesti osy. Součástí návrhu je i webové rozhraní v ABB AppStudio, které umožňuje obsluhu jednoduše spustit či zastavit proces a v reálném čase sledovat aktuální fázi operace.

Třetí část se zaměřuje na konstrukci dvoudílného kuželového držáku oscilační čepele se standardizovanými spojovacími prvky. Geometrie nástroje byla optimalizována pro nízké rezní síly a rychlou výměnu čepele, přičemž zohledňuje prostorové limity robotické příruby i požadavky na vizualizaci dráhy v simulátoru RobotStudio.

Integrované řešení ilustruje, jak lze propojením moderních knihoven Pythonu, průmyslového standardu DXF a otevřeného rozhraní RWS vytvořit plně digitální řetězec „data-to-motion“. Přínosem práce je metodika, která výrazně zkracuje čas mezi změnou výkresu a rozběhem robotické výroby, minimalizuje rizika vyplývající z lidské chyby a posouvá hranice flexibilní automatizace směrem k inteligentní, daty řízené továrně budoucnosti.

I TEORETICKÁ ČÁST

1 HISTORIE PRŮMYSLOVÝCH ROBOTŮ

15. a 16. století nejsou běžně známé jako období robotiky, ale umělci, vědci a vynálezci z období renesance, jako například Leonardo da Vinci, poprvé vymýšleli automatizované mechanismy, které nakonec vedly ke vzniku robotů. První skutečný průmyslový robot se objevil o půl tisíciletí později, když byl v roce 1930 vytvořen ze stavebnice Meccano. Průmysl udělal obrovský skok kupředu, když v 60. letech 20. století nastoupila éra počítačů, a od té doby každé desetiletí přináší exponenciální pokrok spolu s dalšími technologiemi. Počet průmyslových robotů používaných v Americe vzrostl z přibližně 200 v roce 1970 na téměř 4 000 o pouhých 10 let později a do roku 2015 se celosvětově zvýšil na 1,6 milionu. Pro rok 2022 se odhaduje, že na celém světě bylo v aktivním provozu téměř 3 miliony průmyslových robotů. Po skromných začátcích se jedná o jeden z nejvlivnějších pokroků v historii výroby věcí, který se stal nepostradatelnou součástí společnosti a globální ekonomiky. [1; 2]

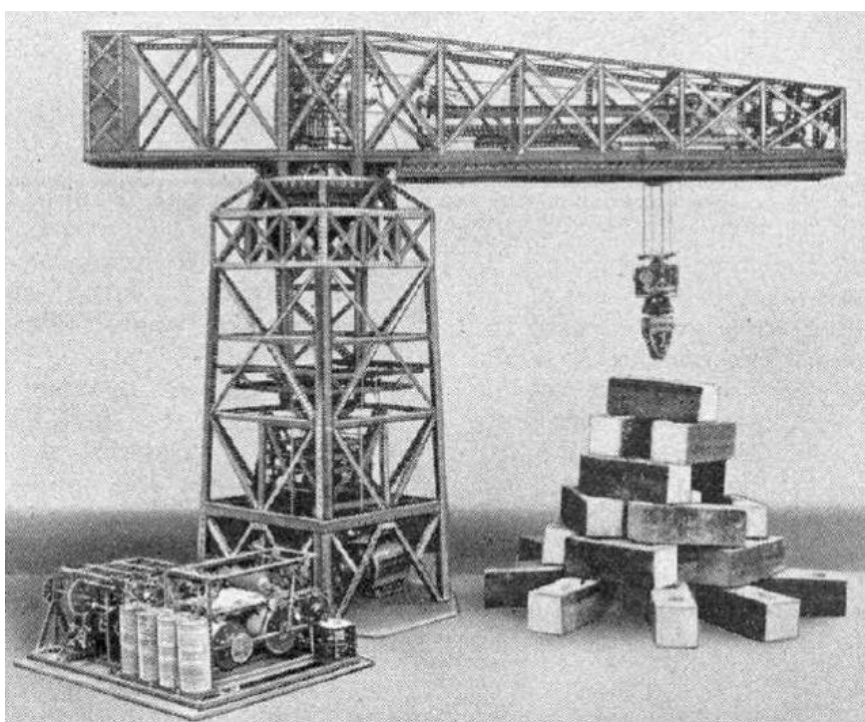
1.1 Výroba před roboty

Když se v polovině 20. století rozjel automobilový průmysl, přinesl s sebou i průmyslovou robotiku. Raná průmyslová revoluce zavedla výrobu více kusů výrobku najednou pro masovou spotřebu. Tehdejší továrny používaly extrémní technologie ohřívání a chlazení, velmi těžké stroje a obrovská zařízení, která vypouštěla olej a kouř, a vystavovala tak lidské pracovníky přímému nebezpečí. Lidé stojící na výrobních linkách připevňovali, svařovali a zatloukali což byl pomalý proces, který navíc ztěžuje přirozená omezení lidské síly a výdrže. Tento opakující se pracovní postup mohl otupit soustředění pracovníků a způsobit smrtelné nehody, stejně jako samotné stroje při katastrofických poruchách. Znečištění ničilo plíce, dřina ničila ducha a práce, která by v moderním světě byla přenechána robotům, se dodnes označuje přívlastkem „špinavá, nebezpečná nebo nudná“. [1; 2]

Po druhé světové válce se rozjel automobilový průmysl, stalo se tak ve spojení s rozvojem výpočetní techniky, takže průmysloví roboti se stali přirozenými partnery v průmyslu. Najednou mohl počítač předepisovat kroky robota – doslovné pohyby, které robot při práci prováděl, takže každý úkon byl identický a každý objekt jednotný a přeprogramovatelný tak, aby se přizpůsobil sebemenší změně. [1; 2]

1.2 30. léta 20. století: Zrod průmyslových robotů

V 30. letech 20. století sestrojil kanadsko-australský stavební inženýr (a robotik ještě předtím, než tento titul vznikl) Bill Taylor první robotický jeřáb, který nazval Gargantua. Gargantua byl vysoký jako průměrný strop v domácnosti, byl naprogramován pomocí papírové pásky s vyraženými otvory, aby mohl skládat bloky v předem naprogramovaných konfiguracích, poháněn jediným elektromotorem a postaven výhradně z dětské stavebnice Meccano. Ačkoli se Gargantua nikdy nedostal na trh, Taylorovy konstrukční nápady přispěly k nadcházející revoluci v průmyslové robotické výrobě. [1; 2]



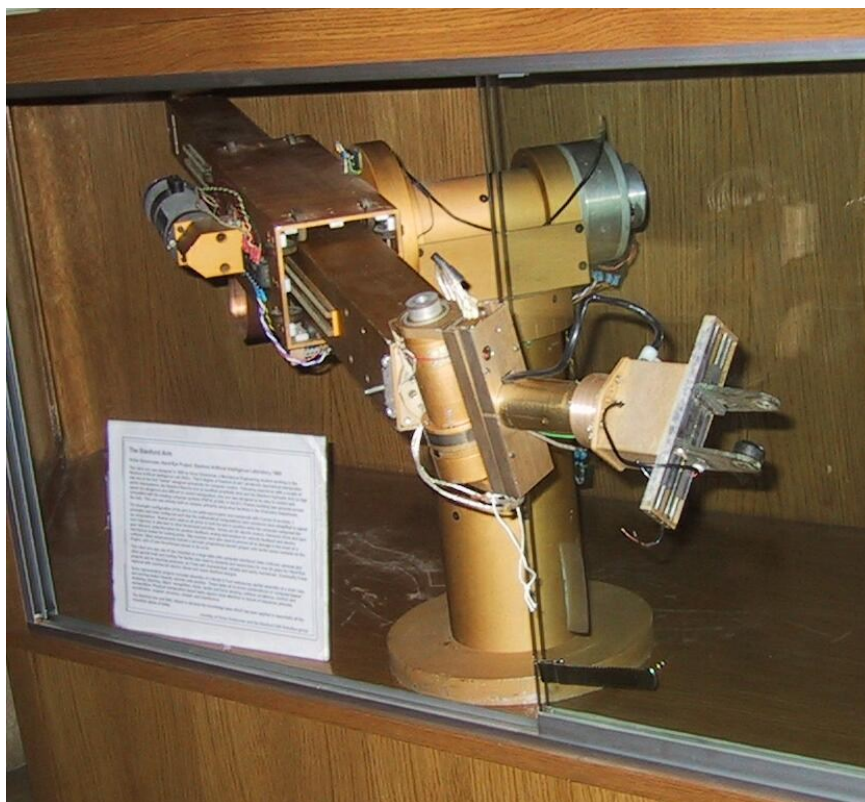
Obrázek 1 Robotický jeřáb Gargantua [30]

Trvalo ještě několik desetiletí, než se rozšířilo používání počítačů, které daly výrobě a robotice nový impuls. Connecticutská společnost Unimation uvedla na trh počítačem řízeného robota řady Unimate 1900. Jedním z jeho prvních významných zákazníků byla společnost General Motors. Přes noc se automobilová výroba obrátila naruby. Unimate 1900 měl takový dopad, že se dostal do povědomí veřejnosti a objevil se na chicagském veletrhu. [1; 2]

V roce 1966, kdy se ve Spojených státech používalo 450 robotů řady 1900, se společnost Unimation poohlédla po zámoří a jmenovala finskou společnost Nokia svým evropským výrobcem. Další iterace technologie společnosti Unimation, svařovací robot,

přišla v roce 1969 a dokázala smontovat bezprecedentních 100 automobilů za hodinu, čímž zdvojnásobila předchozí rychlost. [1; 2]

Přibližně ve stejné době se robotika rozrostla o nový rozměr díky akademikovi Victoru Scheinmanovi. Scheinman, který pracoval v laboratoři umělé inteligence na Stanfordově univerzitě, vytvořil první životaschopnou víceosou robotickou ruku na světě, která dala robotice mnohem širší možnosti pohybu. Stanfordská paže byla lehká, elektrická, antropomorfní a multiprogramovatelná, takže dokázala vykonávat širší škálu nebezpečných nebo nepříjemných prací na montážních linkách namísto lidské obsluhy. Rameno Stanford Arm, které bylo komerčně dostupné v roce 1971, znamenalo revoluci v oblasti průmyslových robotů. [1; 2]



Obrázek 2 Stanford Arm [31]

1.3 80. a 90. léta 20. století: Počítačem řízený robot se stal standardem

Gargantua, Unimate a Stanford Arm byly jako koncepty tak úspěšné, že až do poloviny 90. let byly změny spíše postupné než rozsáhlé. V 80. letech se rychle zdokonaloval pokrok, například průmyslové lasery, což umožňovalo použití senzorové technologie a základních systémů strojového vidění. Všeobecně se uznávalo, že průmysloví roboti představují budoucnost výroby. Počítačové ovládání se stalo standardem i v éře stolních počítačů a dalším pokrokem byla možnost ovládat robota z počítače díky systému MRC (multirobot

control) z roku 1994. Éra PC přinesla také prudké snížení cen mikroprocesorů, čímž se počítačem řízená robotika dostala do rukou ještě většího počtu průmyslových odvětví a hráčů. [1; 2]

Lepší senzory, levnější výpočetní výkon a vyšší stupeň programovatelnosti společně posunuly roboty z fáze jednoduchých opakovaných úkolů a daly jim to, co mnozí považovali za základní inteligenci. [1; 2]

1.4 10. léta 21. století

V roce 2010 došlo v robotickém průmyslu k další revoluci, díky níž se roboti dostali tam, kde si je už dlouho nikdo nepředstavoval: k práci po boku člověka. Kolaborativní roboti neboli coboti, jsou stroje, které pracují v těsné blízkosti lidské obsluhy nebo přímo s ní. V minulých letech museli být roboti přísně izolováni od lidí, aby se minimalizovala možnost, že velký a těžký průmyslový stroj způsobí zranění nebo se porouchá. Lepší, bezpečnější a lehčí materiály, stejně jako mnohem lepší sensorová technologie, která dokáže robota informovat o tom, kde se nachází on sám a kde operátoři, přinesly éru robotů v lidském měřítku určených pro spolupráci s lidskými spolupracovníky. [1; 2]

Tento typ úzké spolupráce člověka a robota umožnila i další důležitá inovace v oboru automatizace. Pomocí softwaru provádí automatizace stále jemnější detaily pracovního postupu. Algoritmy, které zohledňují různé okolnosti procesu, mohou v reálném čase korigovat činnosti tak, aby byly splněny stanovené výsledky. Zatímco například počítačem řízený robot provede přesnou sérii pohybů a úkonů za sebou s pravidelností zlomku sekundy, cobot čekající na člověka, který mu předá předmět ke zpracování, může pomocí svých senzorů a programování pochopit, že se jeho lidský partner zastavil, aby se napil nebo si odskočil na toaletu, a sáhne po předmětu, až když je mu nabídnut. [1; 2]

Po deseti letech vývoje si coboti vydobýly své místo ve výrobě; podle jednoho z odhadů je jich na světě v provozu přibližně 40 000, což představuje trh v hodnotě přibližně 1 miliardy dolarů v roce 2021 a podle prognóz vzroste na 11 miliard dolarů v roce 2030. [1; 2]

1.5 20. léta 21. století

Průmyslový roboti dnes nevypadají o moc jinak než v posledních 50 letech. Při bližším pohledu jsou však senzory lepší, efektorů přesnější a materiály často uživatelsky přívětivější. Stále se však jedná v podstatě o kloubové úseky tuhého materiálu s nástrojem na konci a klouby po celé délce umožňujícími pohyb v několika osách. To, čím se liší, jsou věci, které není

vidět. Díky datům, cloud computingu a technologiím internetu věcí (IoT) může každý vstup, akce a výstup (z celé továrny, jednotlivého zařízení nebo jeho části) zaznamenávat informace o svém výkonu a podávat o nich zprávy. Všechna tato data dávají výrobním podnikům mnohem větší kontrolu nad kvalitou, údržbou a produktivitou. Tato data lze také přenášet do jiných oddělení, partnerům, dodavatelům, zákazníkům a zainteresovaným stranám na druhé straně ulice nebo na celém světě. Tvoří infrastrukturu chytré továrny, globální výrobní mozek, který dokáže zlepšit způsob výroby na makroekonomické bázi a přispívá k úspoře peněz, času i planety. [1; 2]

Je to budoucnost, která se každým dnem přibližuje. Za posledních deset let se počet průmyslových robotů v tomto odvětví ztrojnásobil a v roce 2018 dosáhl téměř 422 000 jednotek, než se situace zpomalila, což ještě zhoršila pandemie v roce 2020. Největšími uživateli jsou stále automobilový a elektronický průmysl, na které v roce 2020 připadne něco málo přes polovinu nových instalací. Ohnisky tohoto odvětví jsou Japonsko, Čína, Německo a Spojené státy, ale nulovým bodem je Jižní Korea, kde na 10 000 zaměstnanců ve výrobním průmyslu připadá 930 robotů což je sedmkrát více, než je celosvětový průměr. [1; 2]

2 IDENTIFIKACE NEBEZPEČÍ

Po určení mezí strojního zařízení je prvním krokem při posouzení rizika identifikace všech rozumně předvídatelných nebezpečí v různých fázích životního cyklu stroje. Jedná se o systematickou analýzu potenciálních hrozeb a rizik, které mohou nastat během dopravy, montáže a instalace, také v průběhu uvedení do provozu a používání a následném vyřazení z provozu, demontáže a likvidace. [3]

Identifikace nebezpečí je prvním krokem k jejich eliminaci nebo snížení rizik. Pro tuto identifikaci jsou nezbytné následující kroky: identifikace činností prováděných strojním zařízením, úkoly obsluhujících a vzájemného působení stroje s uživatelem, přičemž jsou brány v úvahu různé části, mechanismy nebo funkce stroje, zpracovávané materiály a prostředí použití. [3]

2.1 Vzájemné působení člověka a stroje

Identifikace úkolů musí zohledňovat všechny fáze životního cyklu stroje, včetně (ale ne pouze) seřizování, zkoušení, učení nebo programování, změny procesu nebo nástroje, spouštění a provozu v různých režimech, jakož i přívod výrobků do stroje, jejich odstraňování po zpracování, zastavení stroje, nouzové zastavení, opětné obnovení provozu po zablokování nebo neplánovaném zastavení. Kategorie úkolů musí rovněž zahrnovat vyhledávání závad a jejich odstranění (zásah obsluhy), čištění a udržování stroje, preventivní údržbu a provádění oprav. [3]

Identifikace nebezpečí a rizik zahrnuje identifikaci všech rozumně předvídatelných nebezpečí, nebezpečných situací a nebezpečných událostí spojených s úkoly stroje. Kromě toho je nutné identifikovat také rozumně předvídatelná nebezpečí, které se přímo netýká úkolu, například seizmické události, blesky, nadměrné zatížení sněhem, hluk, rozbití strojního zařízení nebo prasknutí hydraulických hadic. [3]

2.2 Stavby stroje

Stanovení stavů provozu stroje zahrnuje dva hlavní stavy: 1) normální provoz stroje, při kterém stroj vykonává předpokládanou funkci, a 2) selhání stroje, což je stav, kdy stroj nevykonává předpokládanou funkci vlivem různých důvodů, včetně změny vlastnosti nebo rozměru zpracovávaného materiálu nebo obrobku, poruchy jeho součástí nebo funkcí, vnějších poruch (např. nárazů nebo vibrací), chyby nebo nedostatku konstrukce (např. chyb

v softwaru), přerušení dodávky energie a okolních podmínek (např. poškozených povrchů podlahy). [3]

2.3 Nečekané akce obsluhy nebo poruchy stroje

Příklady lidských faktorů zahrnují chování obsluhy stroje, které může přispět ke vzniku nehody, jako je ztráta kontroly nad strojem, reflexní chování při selhání nebo nehodě, chování vyplývající z nedostatečné koncentrace nebo neopatrnosti, chování „cesty nejmenšího odporu“ při provádění úkolu, tlaku udržet stroj v chodu za všech okolností a chování určitých osob (například dětí nebo invalidních osob). Identifikace těchto faktorů je důležitá pro zjištění potenciálních nebezpečí. Dále se doporučuje kontrola dostupné konstrukční dokumentace stroje, aby se identifikovaly možné nebezpečí spojené s pohybujícími se prvky, jako jsou motory nebo hydraulické válce. [3]

2.4 Odhad rizika

Po identifikaci nebezpečí musí být proveden odhad rizika pro každou nebezpečnou situaci pomocí prvků rizika. Pokud existují normalizované nebo jiné vhodné metody měření emisí, měly by být použity k určení hodnot emisí a porovnání hodnot emisí s jinými stroji nebo prototypy. Tato analýza umožní konstruktérovi odhadnout riziko spojené s emisemi, zhodnotit účinnost ochranných opatření, poskytnout potenciálním kupujícím a uživatelům dostatečné informace o emisích v technické dokumentaci a informacích pro používání. Stejný postup lze použít i pro jiné nebezpečí, která nelze měřitelnými parametry popsat. [3]

2.5 Obsah rizik

Riziko spojené s jednotlivou nebezpečnou situací je závislé na následujících prvcích: závažnosti úrazu, pravděpodobnosti výskytu tohoto úrazu, která je funkcí vystavení osoby (osob) nebezpečím, výskytu nebezpečné události a technických a lidských možností vyvarovat se úrazu nebo omezit jeho závažnost. [3]

Závažnost úrazu nebo poškození zdraví může být odhadnuta s přihlédnutím k její závažnosti (například lehké, těžké, smrt) a rozsahu úrazu (například u jedné osoby, u několika osob). Při posuzování rizika musí být vzata v úvahu nejpravděpodobnější závažnost úrazu spojená s každým identifikovaným nebezpečím, ale také nejvyšší předvídatelná závažnost úrazu, i když je pravděpodobnost jeho výskytu nižší. [3]

Závažnost úrazu nebo poškození zdraví lze odhadnout s ohledem na její závažnost (například lehké, těžké, smrt) a rozsah úrazu (například u jedné osoby, u několika osob). Při posuzování rizika musí být vzaty v úvahu nejpravděpodobnější závažnosti úrazů spojených s každým identifikovaným nebezpečím, ale také nejvyšší předvídatelné závažnosti úrazu, i když je pravděpodobnost jejich výskytu nižší. [3]

Vystavení osoby nebezpečím ovlivňuje pravděpodobnost výskytu úrazu. Při odhadu vystavení jsou důležité faktory jako nutnost přístupu do nebezpečného prostoru (například při normálním provozu, odstranění selhání, údržbě nebo opravě), povaha přístupu (například ruční posuv materiálů), doba strávená v nebezpečném prostoru, počet osob vyžadovaných k přístupu a četnost přístupu. [3]

Výskyt nebezpečné události je důležitým faktorem ovlivňujícím pravděpodobnost výskytu úrazu. Při odhadu výskytu nebezpečné události jsou důležité faktory jako spolehlivost a jiné statistické údaje, historie úrazů, historie poškození zdraví a porovnání rizik. Dále je třeba poznamenat, že výskyt nebezpečné události může mít původu technického nebo lidského. [3]

Možnosti vyvarování se nebo omezení úrazu jsou důležitým faktorem ovlivňujícím pravděpodobnost výskytu úrazu. Při odhadu těchto možností je třeba brát v úvahu následující faktory: různé osoby, které mohou být vystaveny nebezpečí (nebezpečím), jako jsou kvalifikované nebo nekvalifikované osobnosti; rychlost vyvinutí nebezpečné situace k úrazu, například náhlé, rychlé nebo pomalé; uvědomění si rizika prostřednictvím všeobecných informací, přímého pozorování, výstražných značek a sdělovacích zařízení; lidské schopnosti vyvarování se nebo omezení úrazu, jako je reflexivnost, hbitost a možnost úniku; a praktické zkušenosti a znalosti, například znalost strojního zařízení, podobného strojního zařízení nebo nezkušenost. [3]

2.6 Uvažovaná hlediska při odhadu rizika

Odhad rizika musí zohlednit všechny osoby, včetně obsluhujících a dalších osob, které mohou být vystaveny nebezpečím a jejichž vystavení může být rozumně předvídáno. [3]

Odhad vystavení uvažovanému nebezpečí (včetně dlouhodobého poškození zdraví) vyžaduje detailní analýzu a zohlednění všech režimů provozu strojního zařízení a pracovní postupy, včetně nutnosti přístupu při různých fázích provozu, jako je například vkládání nebo vyjímání, seřizování, záběhu, změně nebo úpravě procesu, čištění, vyhledávání závad

a údržbě. Dále musí být odhad rizika brát v úvahu i úkoly, u kterých není možné nebo je obtížné uplatnit ochranná opatření. [3]

U každé uvažované nebezpečné situace je nutné vzít v úvahu vztah mezi vystavením nebezpečí a jeho účinky, včetně účinků vícenásobného vystavení a kombinace nebezpečí. Při odhadu rizika by měly být použity vhodně ověřené údaje a pokud možno jsou tyto údaje založeny na reálných zkušenostech s úrazy. Nulový výskyt úrazů ještě nezaručuje nízkou pravděpodobnost a závažnost zranění, proto musí být odhad rizika proveden pečlivě a obezřetně. [3]

Lidské faktory mají významný vliv na odhad rizika a musí být proto zohledněny při jeho provedení. Patří sem například vzájemné působení mezi osobami a strojním zařízením, včetně opravy poruchy, vzájemné působení mezi osobami, psychická hlediska (stres), ergonomická hlediska, schopnost osob uvědomit si rizika v dané situaci, která závisí na jejich zácviku, zkušenosti a schopnosti, hlediska únavy a hlediska omezených schopností (způsobená invaliditou, věkem atd.). Zácvik, zkušenosti a schopnosti mohou ovlivnit riziko, ale žádný z těchto faktorů nesmí být používán jako náhrada za vyloučení nebezpečí, snížení rizika konstrukcí nebo bezpečnostních ochranami, pokud je to možné. [3]

Při odhadování rizika je nutné zohlednit vhodnost ochranných opatření a vzít v úvahu následující skutečnosti: identifikovat okolnosti, které mohou vést k úrazu, použít kvantitativní metody pro srovnání alternativních ochranných opatření (podle ISO/TR 14121-2) vždy když je to vhodné, uvést informace přispívající ke výběru vhodných ochranných opatření a věnovat zvláštní pozornost součástem a systémům, u nichž porucha vystupuje jako příčina zvýšeného rizika. Při odhadování rizika musí být také vzato v úvahu, že ochranná opatření zahrnující organizaci práce, správné chování, soustředěnost, používání osobních ochranných prostředků (PPE), dovednost nebo zácvik mají relativně nízkou spolehlivost ve srovnání s ověřenými technickými opatřeními. [3]

Pro nepřetržitý bezpečný provoz stroje je důležité, aby ochranná opatření byla snadno použitelná a nezabraňovala v jeho používání. Jinak je zde možnost, že ochranná opatření budou obcházena nebo vyřazena, aby bylo dosaženo maximálního využití stroje. [3]

Odhad rizika musí brát v úvahu možnost vyřazení nebo obejítí ochranných opatření a vzít v úvahu následující skutečnosti: zda ochranné opatření zpomaluje výrobu nebo překáží při jakýchkoliv jiných činnostech, zda je obtížné používat ochranná opatření, zda jsou kromě

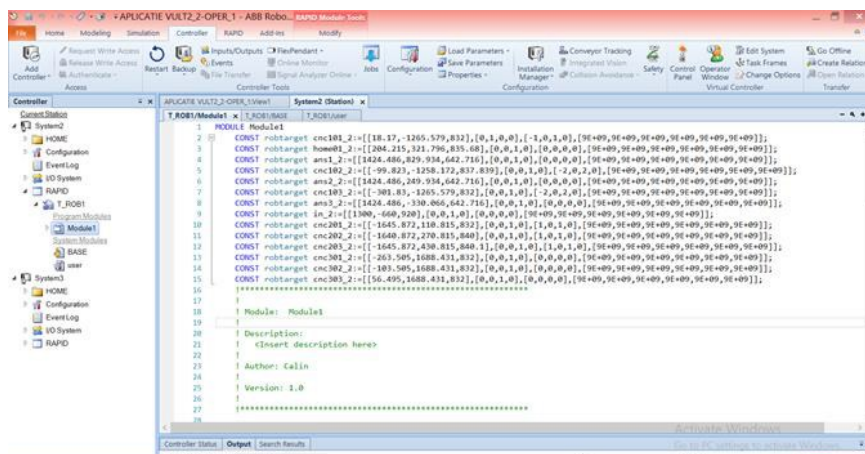
obsluhy zapojeny i jiné osoby, zda je ochranné opatření uživateli neznámé nebo nepřijatelné vzhledem ke své funkci. [3]

Možnost vyřazení ochranného opatření závisí jak na druhu ochranného opatření (například nastavitelný ochranný kryt nebo programovatelné bezpečnostní vypínací zařízení), tak i na jeho konstrukčních detailech. Používání programovatelných elektronických systémů představuje další možnost vyřazení nebo obejití, pokud přístup k bezpečnostnímu softwaru není správně navržen a monitorován. Odhad rizika musí zjistit případy, ve kterých nejsou bezpečnostní funkce odděleny od ostatních funkcí stroje a určit rozsah bezpečnostních funkcí, ke kterým je možný přístup, zejména v případě požadavku na dálkový přístup za účelem diagnostiky nebo korekce procesu. [3]

Odhad rizika musí vzít v úvahu možnost, zda mohou být ochranná opatření udržena ve správném stavu a zda jsou podmínky pro jejich údržbu nezbytné k zajištění požadované úrovně ochrany. Pokud nemohou být ochranná opatření snadno udržována, může to vést ke zvýšené možnosti vyřazení nebo obejití těchto opatření, aby bylo umožněno nepřetržité používání strojního zařízení a tím se zvýší riziko úrazu nebo poškození stroje. [3]

3 ABB RAPID

Programování robotů je nejčastěji realizováno nějakou formou strukturovaného textu, který je podobný programovacímu jazyku C. S výjimkou kolaborativních robotů, u kterých bývá programování více grafické a více uživatelsky přívětivé. [4]



Obrázek 3 Ukázka programovacího jazyka RAPID v prostředí ABB RobotStudio [32]

3.1 Programovací jazyky

Každý výrobce robotů má trochu jiný programovací jazyk pro své roboty, ale většina průmyslových robotů používá nějakou formu strukturovaného textu. Strukturovaný text (ST) je textový programovací jazyk nejvíce podobných pythonu nebo ořezané verzi jazyka C. Všechny hlavní cykly jsou dostupné v ST, jako *FOR*, *WHILE*, *CASE* a podmínky *IF*. Jinak většina standardní datových typů jsou dostupné jako i datové pole, ale ne datové listy. Nejsou v nich žádné třídy, ale může v nich být více úloh, které obsahují různé funkce nebo metody a velká část softwarových balíčků obsahuje knihovny, které usnadňují implementaci některých funkcí. [4]

Industriální sektor typicky používá ST jakožto sekundární k ladder logice, protože je součástí IEC 61131-3, což znamená, že jazyk musí dodržovat stejné zásady pro čtení instrukcí a obecnou strukturu jako ladder. [4]

Ve světě ABB robotů je používán jazyk RAPID. Ten vypadá a chová se většinou jako ST a je si velice blízký s jazykem C a jeho variacemi. [4]

3.2 Funkce

Programovací jazyk RAPID má vestavěné funkce, které může robot využívat. Tyto vestavěné funkce umožňují robotovi pohyb. Některé funkce vypočítávají matematické instrukce nebo řeší citlivost robota na vnější působení sil. [4]

Tyto funkce jsou volány stejně jako jiné funkce v ST nebo RAPID, kde pouze hlavní funkce nemohou být odstraněny a jsou používány k volání jiných funkcí. Můžeme i vytvořit naše vlastní funkce, které vracejí hodnotu specifického datového typu stejně jako v C++. [4]

3.3 Parametry

Parametr v souvislosti s voláním funkce nebo metody je proměnná, která se posílá spolu s voláním. Například „*CalculateNum(Num1, Num2)*“. *Num1* a *Num2* jsou parametry, jejich hodnoty a datové typy jsou použity ve funkci *CalculateNum*. [4]

V cyklickém ST nejsou parametry běžně k dispozici. Náhradou by byla globální proměnná, ke které by měla přístup jakákoli metoda nebo rutina v kódu. Parametry jsou k dispozici v programování RAPID a mohou zmenšit velikost programu tím, že jedna proměnná může sloužit více účelům. [4]

Příkladem může být aplikace, která vyžaduje mnoho vyučovaných pozic, ale v jednom okamžiku je přístupná pouze jedna pozice. Funkce vypočítá, ke které poloze je třeba přistupovat, a pak zavolá metodu pohybu s polohou jako parametrem. Po zavolání této metody se robot přesune do polohy uložené v parametru. Při výpočtu nové polohy se parametr pro volání nové metody recykluje. Díky použití jedné metody a odeslání polohy prostřednictvím parametru je kód pro pohyb robota velmi kompaktní. [4]

3.4 Přerušení

Přerušení v programování je signál nebo událost, která dává procesoru pokyn k provedení jiné úlohy, jakmile je aktuální úloha dokončena, bez ohledu na aktuální umístění v programu. [4]

Běžným příkladem může být tlačítko. Když uživatel stiskne tlačítko, spustí se specifický kód, který nastaví proměnné nebo zobrazí text. Přerušení se v cyklickém programování ST nevyskytují, protože přerušení cyklu snímání by bylo v rozporu s normou IEC. [4]

V programu RAPID se přerušení programování využívají, když robot vstoupí do oblasti nebo má signál chapadla. Tyto signály mohou spustit krátký specifický kód mimo standardní cyklickou úlohu. [4]

3.5 Příklady základních pohybových instrukcí:

MoveL přesouvá středový bod nástroje lineárně do cílového bodu. Udávají se mu parametry cílového bodu, rychlosti, s jakou se má do bodu dostat, jakou přesnost musí dodržet a který středový bod. Příkladem může být *MoveL p1, v1000, z30, tool1;*, robotovi tedy říká: přesuň se lineárně do bodu definovaného jako *p1* rychlostí 1000 mm/s s přesností 30 mm s nástrojem *tool1*. [5; 6]

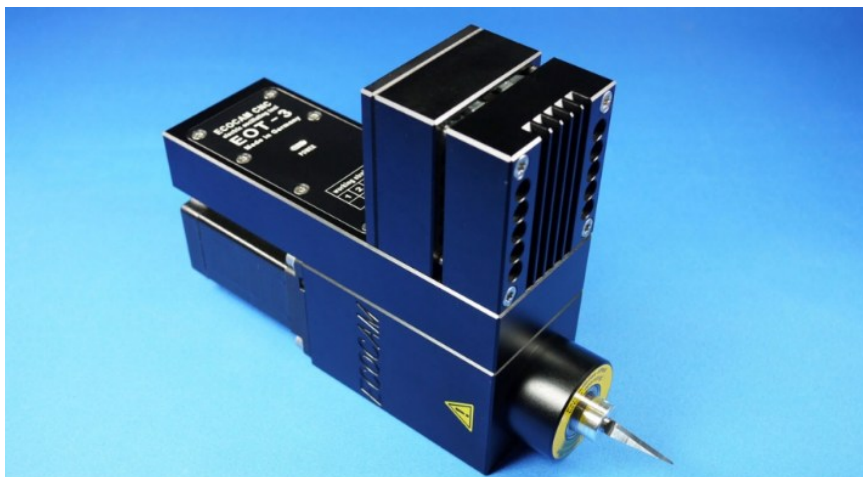
MoveJ se používá pro přesun mezi dvěma body, kdy pohyb nemusí být lineární. Robot a vnější osy se přesouvají do cílové pozice nelineární dráhou. Všechny osy dosáhnou cílové pozice ve stejný čas. Instrukce *MoveJ p1, v100, z30, tool1;*, robotovi tedy říká: přesuň se co nejrychleji do bodu definovaného jako *p1* rychlostí 100 mm/s s přesností 30 mm s nástrojem *tool1*. [5; 6]

4 OSCILUJÍCÍ NŮŽ

Přímý nůž, který během řezání vykonává vertikální pohyb nahoru a dolů (osa Z) v pravidelném rytmu, se nazývá oscilační nebo vratný nůž. Kmitání je pohyb, který se opakuje s konstantní periodou. Reciproční pohyb zahrnuje omezení, že pohyb je lineární. Takto by se nože vykonávající pohyb nahoru a dolů v jednovrstvých řezačkách měly nazývat kmitavé, ale nože ve vysoko vrstevných řezačkách, kde musí být vertikální pohyb nože přesnější/omezený, by se měly nazývat vratné. Ačkoli existuje rozdíl ve významu těchto dvou termínů (bohužel není zcela jasné vyjádření), nože pohybující se nahoru a dolů v pravidelném rytmu a provádějící jednovrstvý i vícevrstvý řez se nejčastěji nazývají oscilační nože. Tyto druhy nožů se používají k řezání houževnatých materiálů (které se obtížně řezou tažným nožem) a také silných materiálů a vícevrstvných nátěrů. Oscilační nože mohou zpracovávat jednoduché i složité tvary. [7]

Oscilační nože se liší tvarem a délkou zdvihu. Nože s plochou čepelí se doporučují k řezání rovných linií, zakřivených linií s velkými poloměry a velkých součástí (široká špička nože vytváří přesahy, které mohou narušit kontury řezu). Tenké nože s ostrou špičkou se používají k řezání malých složitých tvarových součástí se zakřivenými liniemi a malými poloměry, aby byla zajištěna vysoká přesnost řezu. [7]

Elektricky poháněné oscilační nože s krátkým zdvihem (zdvih 5 mm) se používají k řezání různých tenkých, měkkých a středně tvrdých textilií. Elektricky poháněné nože s delším zdvihem (25 mm) se používají ke zpracování silných a tvrdých textilií a také tvrdší kůže. Pneumaticky poháněné oscilační nože s dlouhým zdvihem (až 200 mm) řezou houževnaté a husté materiály, které vyžadují velké řezné síly, a také měkké, ale silné materiály a vícevrstvé textilie. [7]



Obrázek 4 Elektrický oscilační nůž [33]

Vzhledem k oscilačnímu působení nože musí být povrch řezacího stolu dostatečně volný, aby nůž mohl plně proniknout do řezaného materiálu. Při provádění vícevrstvého řezání kmitajícím nožem je řezná plocha pokryta štětinami, které umožňují noži snadno proniknout materiálem. [7]

4.1 Ultrazvukový nůž

Ultrazvukové vibračně asistované obrábění (UVA) je proces, který do pohybu hrotu řezného nástroje vnáší vysokofrekvenční vibrace v mikroměřítku, aby se interakce nástroje s obrobkem stala nespojitým procesem. Široká škála aplikací ukázala, že obrábění pomocí UVA je ve srovnání s konvenčním postupem efektivnější, což se projevuje např. nižšími obráběcími silami, vyšší stabilitou obrábění, menším opotřebením nástroje a lepší kvalitou povrchu. Obrábění UVA je výhodné zejména při povrchovém opracování obtížně obrobitelných materiálů. [8]

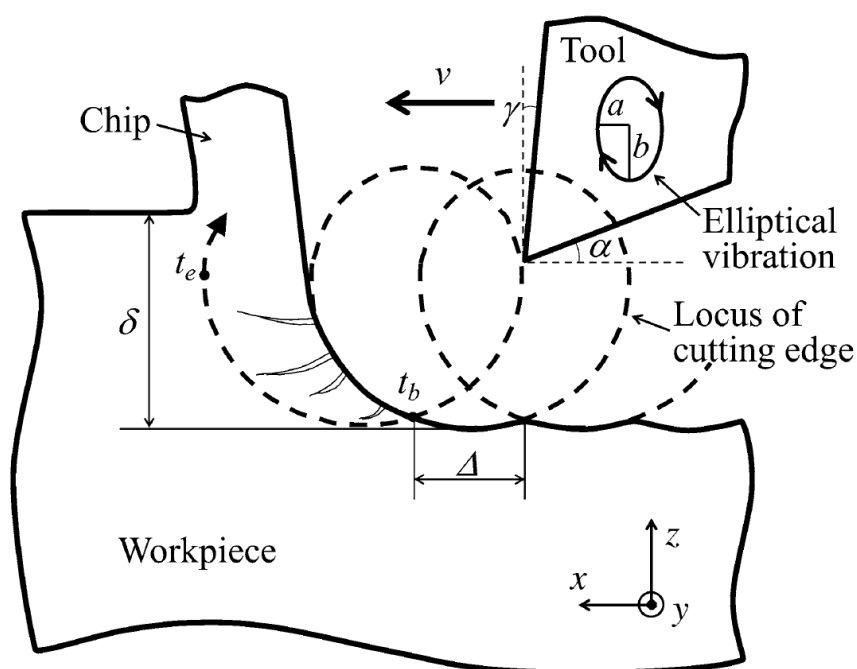
Při UVA se vysokofrekvenční elektrická energie převádí na mechanické kmity pomocí piezoelektrického a magnetostrikčního snímače. Tyto kmity jsou dále soustředěny zařízením pro fokusování energie. Nástroj pak kmitá ve směru řezu nebo ve směru tahu na vysoké frekvenci (obvykle v rezonanci nad 15 kHz) s amplitudou od 1 do 15 μm . Podle dráhy hrotu se ultrazvukové obrábění rozděluje do dvou základních typů. Jednorozměrné obrábění probíhá tak, že hrot kmitá buď ve směru řezu, nebo kolmo na něj. U dvourozměrného obrábění kmitá hrot současně ve směru řezu a ve směru tahu, čímž se vytváří eliptická trajektorie. [8]

Řezání 1D ultrazvukovou asistencí se poprvé objevilo v padesátých letech 20. století s cílem snížit řezné síly, prodloužit životnost nástroje a zlepšit přesnost obrábění kovových dílců, jako jsou hliník, mosaz, měkká ocel nebo litina. Následné práce prokázaly účinnost této metody i při obrábění skla či keramiky. V devadesátých letech byl představen dvourozměrný ultrazvukový pohyb špičky nástroje, u kterého se očekávalo další zlepšení výkonnosti procesu. Výzkumy ukázaly, že oproti jednorozměrnému řezu může 2D ultrazvukové obrábění podpořit kvalitu povrchu, snížit řezné síly a prodloužit životnost nástroje. Dnes se obě metody uplatňují při obrábění širokého spektra materiálů i dílů se složitými povrchovými vlastnostmi. Úspěch ultrazvukového asistovaného řezání zároveň vedl k výzkumu těžkoobrobitelných látek, například niklových slitin, wolframových

karbidů, křemičitanů nitridu či kompozitů s kovovou maticí a kompozitů vyztužených vlákny. [8]

4.1.1 Princip UVA

Pohyb hrotu nástroje při ultrazvukově asistovaném řezání je na obr. 5 znázorněn schematicky. Pro zjednodušení označme primární směr řezu jako kladnou osu x . Osa z je kolmá k neřezanému povrchu a určuje směr hloubky řezu. Směr y na obrázku vyjadřuje příčný posuv podobně jako při soustružení. Primární rychlost posuvu nástroje označíme v a hloubku řezu δ . [8]



Obrázek 5 Ilustrační schéma EVA [8]

Procesy ultrazvukové asistence při řezání lze podle obr. 6 rozdělit do tří skupin podle směru vibrací hrotu nástroje. První skupinu tvoří CDVA, kdy nástroj kmitá výhradně ve směru řezu. Druhá skupina NDVA popisuje vibrace ve směru kolmém k neřezanému povrchu. Třetí případ označovaný EVA zahrnuje současné kmity ve směru řezu i ve směru normály, čímž vzniká eliptická trajektorie. Proces EVA tak představuje dvourozměrné ultrazvukové obrábění, zatímco CDVA a NDVA jsou jednorozměrné speciální případy eliptické asistence. [8]

Cutting types	Vibration of tooltip	Tooltip trajectory	Tooltip displacement
Conventional cutting			$\begin{cases} x(t) = vt \\ z(t) = 0 \end{cases}$
CDVA cutting			$\begin{cases} x(t) = vt + a \cos(2\pi ft) \\ z(t) = 0 \end{cases}$
NDVA cutting			$\begin{cases} x(t) = vt \\ z(t) = b \sin(2\pi ft) \end{cases}$
EVA cutting			$\begin{cases} x(t) = vt + a \cos(2\pi ft) \\ z(t) = b \sin(2\pi ft) \end{cases}$

Obrázek 6 Metody ultrazvukového řezání [8]

Předpokládejme, že trajektorie vibrací nástroje je dána jako $x_{Tool}(t) = a \cos(2\pi ft)$, $z_{Tool}(t) = b \sin(2\pi ft)$. Platí, že a a b jsou amplitudy kmitů ve směrech x a z , a f je frekvence kmitání. Relativní pohyb hrotu nástroje vůči obrobku při ultrazvukově asistovaném řezání lze pak vyjádřit jako $x(t) = vt + a \cos(2\pi ft)$, $z(t) = b \sin(2\pi ft)$. [8]

Relativní rychlost pohybu hrotu nástroje vůči obrobku je $v_x(t) = v - 2\pi f a \sin(2\pi ft)$, $v_z(t) = 2\pi f b \cos(2\pi ft)$. [8]

Jak vyplývá z předchozího textu, ultrazvukově asistované vibrace při řezání významně prodlužují životnost nástroje, zlepšují kvalitu povrchu, snižují řezné síly a zvyšují integritu povrchu oproti tradičnímu řeznému procesu bez vibrací na špičce nástroje. Tato zjištění vycházejí ze srovnání obou přístupů, a proto je pro maximalizaci výhod klíčový správný výběr parametrů vibrací. Jediným zřejmým kritériem je, aby maximální rychlost vibrací hrotu ve směru řezu $v_{x_{max}} = 2\pi f a$ a byla vyšší než rychlost posuvu v , neboť jinak by vibrace nemohly účinně přispět k řezání. Dále proto vycházíme z předpokladu $v_{x_{max}} > v$. [8]

Řezání EVA znázorněné na obr. 5 představuje dvourozměrné ultrazvukové obrábění. Během vibračního cyklu dochází k vlastnímu řezu pouze tehdy, když se v okamžiku t_b špička nástroje zaboří do materiálu, a končí v okamžiku t_e , kdy dosáhne maximální amplitudy vibrací ve směru řezu. Eliptická trajektorie hrotu naznačuje, že vertikální složka jeho rychlosti v okolí t_b je nižší než rychlost toku třísky směrem vzhůru a tok třísky je omezen třením mezi nástrojem a třískou. Jakmile vertikální složka rychlosti vibrací překročí rychlost toku třísky, tření mezi nástrojem a třískou se obrátí ve směru toku třísky a začne usnadňovat odstraňování třísky. Každý vibrační cyklus nástroje při řezání EVA proto obsahuje fázi záporné přitlačné síly, což snižuje tloušťku třísky a průměrnou řeznou sílu a

přináší tak výhodu oproti konvenčnímu řezání i procesům CDVA. V průběhu řezání EVA se okamžitý směr pohybu nástroje mění v čase a lze jej popsat vztahem $x(t) = \arctan \frac{2\pi f b \cos(2\pi f t)}{v - 2\pi f a \sin(2\pi f t)}$.

Kde γ a α jsou jmenovité úhly sklonu a vůle. Aby bok nástroje nezasahoval do obrobku, musí být okamžitý úhel vůle $\alpha(t)$ v okamžiku zahájení řezání t_b $\alpha(t_b) > 0$. [8; 9]

Síla $F_z(t)$ vyvolaná vibracemi ovlivňuje stabilitu řezného procesu i kvalitu a přesnost povrchu obrobku. Pro pochopení tohoto vlivu uvažme svislé posuny hrotu nástroje, tedy ve směru osy z podle obr. 5, pro konvenční řezání $z_{CC}(t)$, pro jednorozměrné ultrazvukové řezání $z_{1D}(t)$ a pro dvourozměrné řezání $z_{2D}(t)$. Tyto funkce lze vyjádřit následovně $z_{CC}(t) \approx \frac{k_{CC}h\delta}{K_z}$, $z_{1D}(t) \approx \frac{t_c}{T} \frac{k_{1D}h\delta}{K_z}$, $z_{2D}(t) \approx \frac{|t_{c1}-t_{c2}|}{T} \frac{k_{2D}h\delta}{K_z}$, kde h je šířka řezu, K_z označuje ekvivalentní koeficient tuhosti obrobku ve směru z a T je perioda kmitání. Doba záběru nástroje a obrobku při jednorozměrném ultrazvukovém asistovaném řezání je $t_c = t_e - t_b$, doba, po kterou je tření nástroj-trísky opačné vůči toku třísky, je $t_{c1} = t_i - t_b$ a doba, kdy tření směřuje ve směru toku třísky, je $t_{c2} = t_e - t_i$, přičemž t_i označuje okamžik změny směru tření. Měrné odpory podél osy z pro konvenční řezání, jednorozměrné UVA a dvourozměrné UVA jsou k_{CC} , k_{1D} a k_{2D} . Studie ukazují, že k_{CC} a k_{1D} mají vyšší hodnoty než k_{2D} . Protože absolutní hodnota rozdílu t_{c1} a t_{c2} dělená T je menší než podíl t_c a T a ten je menší než jedna, z rovnic (7) až (9) plyne, že $z_{2D}(t) < z_{1D}(t) < z_{CC}(t)$. To znamená, že posunutí obrobku ve směru z je při dvourozměrném UVA nejmenší, takže dynamickou tuhost i přesnost obrábění lze nejvíce zlepšit právě eliptickým UVA. [8; 9; 10; 11]

Tahové řezné síly pro konvenční řezání, 1D i 2D ultrazvukově asistované řezání lze teoreticky popsat následujícími vztahy $F_{zCC}(t) = k_{CC}h\delta$, $F_{z1D}(t) = \frac{t_c}{T} k_{1D}h\delta + \frac{2k_{1D}h\delta}{\pi} \sum_{n=1}^{\infty} \frac{1}{n} \sin\left(\frac{t_c n}{T} \pi\right) \cos(2nf\pi^2)$, $F_{z2D}(t) = \frac{t_{c1}}{T} k_{2D}h\delta + \frac{2k_{2D}h\delta}{\pi} \sum_{n=1}^{\infty} \frac{1}{n} \sin\left(\frac{t_{c1} n}{T} \pi\right) \cos(2nf\pi^2) - \frac{t_{c2}}{T} k_{2D}h\delta - \frac{2k_{2D}h\delta}{\pi} \sum_{n=1}^{\infty} \frac{1}{n} \sin\left(\frac{t_{c2} n}{T} \pi\right) \cos(2nf\pi^2)$. [8; 12]

Jelikož je frekvence kmitání nástroje velmi vysoká, všechny členy v součtech od $n = 1$ do nekonečna v rovnicích (11) a (12) lze zanedbat. Síly $F_{z1D}(t)$ a $F_{z2D}(t)$ jsou pak úměrné poměrům $\frac{t_c}{T}$, respektive $\frac{|t_{c1}-t_{c2}|}{T}$. Z toho vyplývá, že $|F_{z2D}(t)| < |F_{z1D}(t)| < |F_{zCC}(t)|$, což dokládá, že UVA snižuje řezné síly a nejnížší řezné síly vykazuje 2D režim UVA. [8]

5 PIECEWISE-CUBIC HERMITE INTERPOLATING POLYNOMIAL

Hermitova interpolace spočívá v sestavení polynomu $H(x)$ tak, aby aproximačně zachycoval funkci $f(x)$. Základem je zadání bodů t_0 až t_n a požadavek, že v těchto bodech se hodnota polynomu i hodnoty jeho prvních derivací shodují s hodnotami původní funkce. [13]

Nejjednodušší situace, která už není pouhou polynomicou interpolací ani Taylorovou aproximací, nastává, když definujeme dva body a současně požadujeme shodu jejich prvních derivací. Vznikají tak čtyři nezávislé podmínky $H(t_0) = f(t_0) = y_0$, $H'(t_0) = f'(t_0) = y'_0$, $H(t_1) = f(t_1) = y_1$, $H'(t_1) = f'(t_1) = y'_1$, jejichž vyřešení potvrzuje existenci právě jednoho kubického polynomu splňujícího tyto požadavky. Od této chvíle budu termín „kubický“ používat i pro jeho degenerované podoby, například kvadratické. [13]

Pro určení kubického polynomu se zapisuje ve tvaru $H(x) = a + b(x - t_0) + (x - t_0)^2[c + d(x - t_{i+1})]$ a zavede se označení $h = t_1 - t_0$. [13]

Při uplatnění čtyř interpolačních podmínek pak postupně dostaneme $a = y_0$, $b = y'_0$, $c = \frac{y_1 - y_0}{h^2}$, $d = \frac{y'_1 - y'_0}{3h^2} - \frac{2(y_1 - y_0)}{3h^3}$. [13]

Při větším počtu uzlů by bylo samozřejmě možné použít polynomy vyššího řádu. V mnoha případech se ovšem uplatní konstrukce kubické aproximace po částech, kdy mezi dvěma sousedními uzly vytvoříme kubiku jen na základě hodnoty funkce a její první derivace v těchto bodech. Kusová Hermitova kubická aproximace f na intervalu $[a, b]$, kde $a = t_0 < t_1 < \dots < t_n$, se tedy skládá ze souboru n takto definovaných kubických rovnic $H(x) = H_i(x) = a_i + b_i(x - t_i) + (x - t_i)^2[c_i + d_i(x - t_{i+1})]$, $t_i \leq x < t_{i+1}$ společně s $a_i = y_i$, $b_i = y'_i$, $c_i = \frac{y_{i+1} - y_i}{h_i^2} - \frac{y'_i}{h_i}$, $d_i = \frac{y'_{i+1} - y'_i}{3h_i^2} - \frac{2(y_{i+1} - y_i)}{3h_i^3}$ kde $y_i = f(t_i)$, $y'_i = f'(t_i)$ a $h_i = t_{i+1} - t_i$. Nejčastěji jsou body rozmístěny rovnoměrně, tak že $h_i = h = \frac{b-a}{n}$. [13]

Pro tuto metodu lze odvodit vzorec pro odhad aproximační chyby, který v případě $n = 1$ zároveň slouží jako chybový vzorec pro upnutý spline. Pro $x \in [t_i, t_{i+1}]$, $f(x) - H(x) = \frac{f^{(4)}(\xi)}{4!} [(x - t_i)(x - t_{i+1})]^2$ kde $\xi \in [t_i, t_{i+1}]$ pokud tedy $|f^{(4)}(x)| \leq M_i$ pro $x \in [t_i, t_{i+1}]$, $|f(x) - H(x)| \leq \frac{M_i}{384} h_i^4$. [13; 14]

Přesnost Hermitovy aproximace se blíží přesnosti sevřených splajnů, přičemž jediným omezením je nižší hladkost – v uzlových bodech je zajištěna jen spojitost první derivace. Hlavní výhodou je ale „lokalizovaná“ chyba: pokud ve čtvrté derivaci funkce na jednom dílčím intervalu dojde k výrazné odchylce nebo není definována, projeví se to jen v tomto úseku, zatímco zbytek aproximace zůstane přesný.[13]

Toto srovnání však nebere v úvahu, že Hermitova metoda vyžaduje znalost hodnot derivací funkce, což v praxi často není možné – buď derivace nejsou dostupné, nebo není znám předpis funkce, ale jen výčet hodnot y_i . Řešením je odhadnout potřebné derivace stejným způsobem jako u modifikovaných sevřených splajnů. Abychom takový přístup mohli správně aplikovat, je třeba detailně prozkoumat techniky aproximace derivací a stanovit hranice jejich přesnosti.[13]

6 TANGENCIÁLNÍ ŘEZÁNÍ ROBOTEM

Plánování trajektorie spočívá v určení pohybu manipulátoru od výchozí pozice k cílové, přičemž se specifikuje jak geometrická dráha, tak časový průběh polohy, rychlosti i zrychlení. Pečlivé navržení trajektorie je zásadní, neboť umožňuje řídicímu systému v uzavřené smyčce provést pohyb bez rizika saturace akčních členů či nadměrného mechanického namáhání konstrukce. [15]

Plánování trajektorie se provádí v pracovním prostoru, přičemž požadovaná dráha je popsána pomocí klíčových bodů a musí respektovat jak vnější překážky, tak vnitřní limity robotické konstrukce. Při návrhu je třeba zajistit, aby byl výpočet trajektorií co nejjednodušší a nejrychlejší. Polohy i rychlosti kloubů pak musejí být spojité funkce času – nespojitá poloha by znamenala nutnost přemístit se mezi body okamžitě, a tedy nekonečnou rychlostí, což není reálné. Podobně nespojité rychlosti mohou vést k problémům, ačkoli kontinuita zrychlení není vždy nevyhnutelná. Zároveň je nutné omezit nežádoucí jevy, jako jsou nepravidelné křivosti interpolovaných úseků. [15; 16]

Roboti jsou navrženi k provádění složitých úloh vyžadujících výrazně nelineární pohyby, a definice vhodné trajektorie pro dosažení cíle představuje často náročný úkol. V robotice se při plánování pohybu běžně uplatňují dva přístupy: řízení „bod–bod“ a řízení po spojitě dráze. Při pohybu z bodu do bodu robot prochází předem určenými klíčovými body a mezi nimi trajektorii extrapoluje – jedná se o nejrozšířenější metodu. Naopak při spojitěm řízení je v každém okamžiku trajektorie vypočítávána aktuální rychlost a poloha koncového efektoru. [15; 17]

Navrhování pohybu se opírá o různá optimalizační kritéria, přičemž volba závisí na konkrétním nasazení robota. V některých případech je rozhodující rychlost vykonání, jinde se klade důraz na minimalizaci energetické spotřeby a jindy je nejdůležitější plynulý průběh trajektorie. [15; 18]

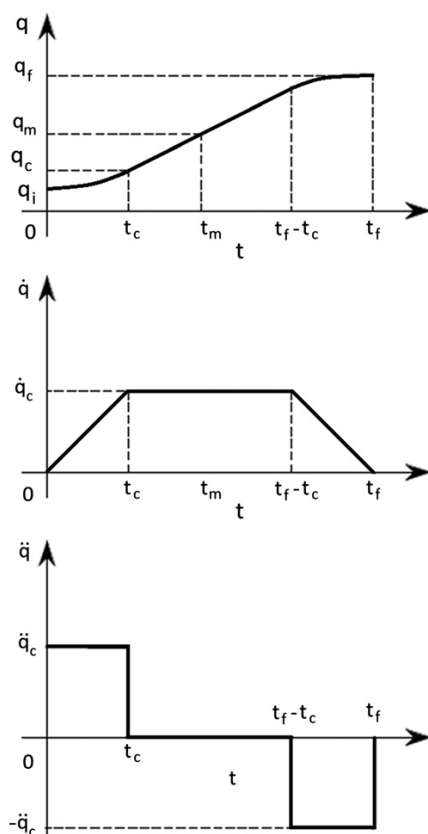
Při plánování trajektorie od bodu k bodu nebo při optimalizaci času hrozí, že výsledná dráha bude obsahovat ostré vrcholy, které mohou způsobovat nežádoucí vibrace. Pro dosažení plynulého pohybu a přesného sledování polohy v pracovním prostoru a prostoru kloubů se používají rychlostní profily lichoběžníkového nebo zvonového tvaru. Tyto profily jsou konstruovány s využitím kubických, kvartických či kvintických polynomů, jak uvádí Broquère et al. Tito autoři také shrnují techniky pro dodatečné zjemnění trajektorie, jež

omezují náhlé změny zrychlení a snižují rychlost. Ve všech případech se úprava rychlostního průběhu provádí v kloubním prostoru. [15; 19; 20]

Technika navržená Broquère et al. vychází z dat o trajektorii přiřazené v pracovním prostoru. Zatímco Broquère et al. optimalizují dráhu z hlediska rychlosti a trhání, naše metoda využívá geometrickou tečnu. Díky tomu se koncový efektor v každém přiřazeném bodě pohybuje podél tečny k požadované dráze. [15; 19; 20]

6.1 Konvenční plánování pohybu od bodu k bodu

Plánování trajektorie koncového efektoru robota v jeho pracovním prostoru spočívá v aplikaci relativních rovnic pohybu na jednotlivé klouby robota. Při přesunu z bodu do bodu se manipulátor musí během stanoveného časového úseku přenést z počáteční konfigurace kloubních proměnných do konfigurace cílové. Je nezbytné, aby jak poloha, tak rychlost každého článku byly jako funkce času plynulé. Toto vyplývá z fyzikálních zákonů, které neumožňují nespojité změny v pohybu ani skokové změny rychlosti žádné části robota. [15]

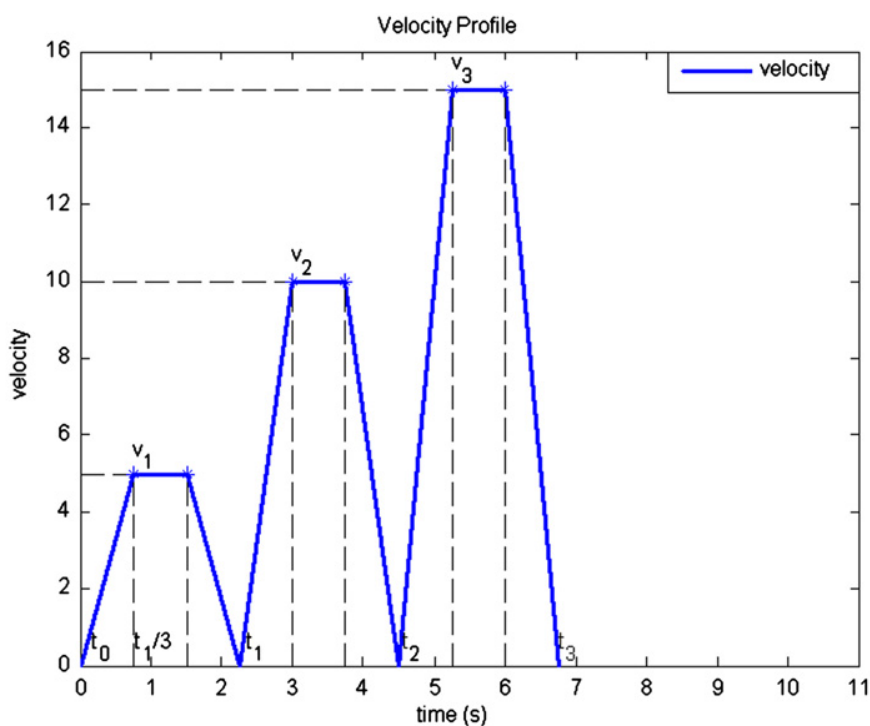


Obrázek 7 Pohybové zákony pro přesun z bodu do bodu [15]

V průmyslové praxi se často uplatňují časování založená na kombinaci polynomických zákonů. V takovém případě se použije lichoběžníkový profil rychlosti, který zajišťuje

konstantní zrychlení na počátku následované stálou cestovní rychlostí, a nakonec konstantním zpomalením. Výsledná trajektorie sestává z lineárního úseku doplněného dvěma parabolickými úsečkami vedoucími do výchozí a cílové polohy, jak je zobrazeno v obrázku 7. Řídicí systém zodpovědný za tvorbu referenční dráhy tak obdrží informace o požadovaných polohách rychlostech a zrychleních nezbytných k provedení pohybu. Díky tomu je celková doba trvání trajektorie pevně stanovena. [15]

Jednotlivé časové intervaly t_c , $(t_f - t_c) - t_c$ a $t_f - (t_f - t_c)$ lze volit nezávisle na sobě za předpokladu, že jejich součet je roven celkové době trvání pohybu a že vyhovují stávajícím vztahům mezi rychlostmi a zrychleními. Pokud tomu tak není, lze zvolit všechny tři se stejnou dobou trvání, tj. s jednou třetinou celkové doby trvání. [15; 21; 22]



Obrázek 8 Profil rychlosti v pohybu sekvencí bodů [15]

Je-li zrychlení $\overline{(q_c)}$ zvoleno tak, aby splňovalo rovnici $|\overline{(q_c)}| = 4 \cdot \frac{|q_f - q_i|}{t_f^2}$ trajektorie neobsahuje segment konstantní rychlosti a má pouze segmenty zrychlení a zpomalení. Jedná se o trojúhelníkový rychlostní profil. [15]

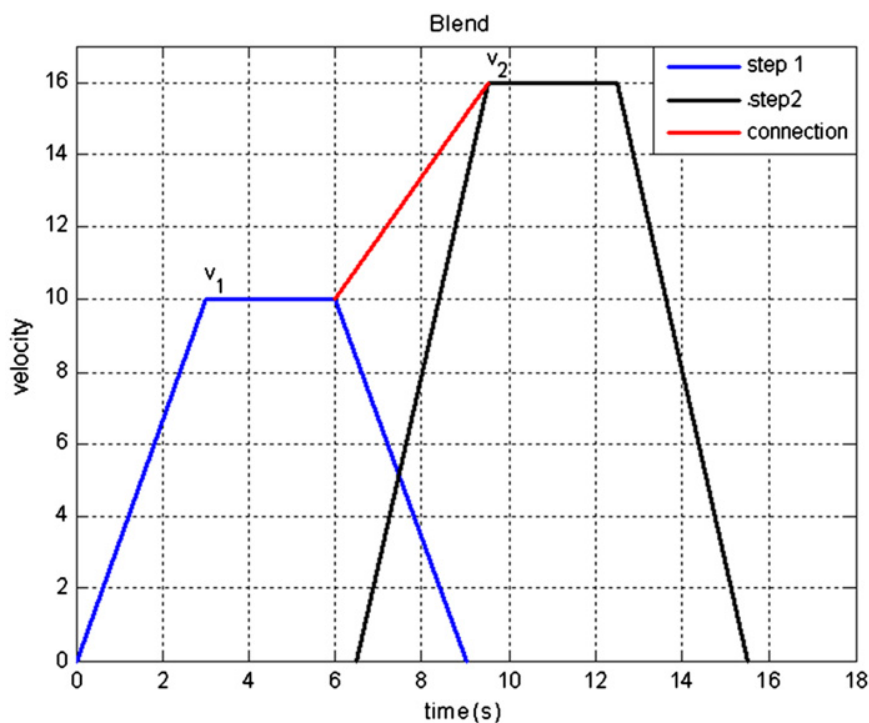
V náročnějších aplikacích se obvykle přiřazuje posloupnost bodů po geometrické dráze, aby šla trajektorie lépe řídit. V úsecích pracovního prostoru, kde je nutné obejít překážky nebo kde dráha značně zakřivuje, se body volí hustěji. Odpovídající konfigurace v kloubním

prostoru se potom vypočítávají podle těchto pracovních bodů. Úkolem je tedy vygenerovat trajektorii, mají-li být dosaženy N bodů dráhy v přesně určených časech. Pro každou kloubní proměnnou tak vzniká N podmínek, určujících požadované polohy. Jednou z možností je pro každý kloub použít polynom stupně $N-1$, což ale s sebou nese řadu nevýhod. [15]

Nelze přímo přiřadit počáteční ani konečnou rychlost, a jak roste stupeň polynomu, zvyšuje se jeho tendence k oscilacím, což může vést k nepřírodným trajektoriím pro manipulátor. Současně se s vyšším stupněm snižuje numerická přesnost výpočtu. Jakmile máme definovanou soustavu omezujících rovnic, stává se řešení komplikovanějším. Koeficienty polynomu jsou totiž závislé na všech přiřazených bodech, což znamená, že při jakékoli změně jednoho bodu je nutné znovu přepočítat všechny koeficienty. [15]

K odstranění těchto nevýhod se místo jednoho interpolačního polynomu vysokého řádu použije více polynomů nižšího řádu, které se navzájem spojí v jednotlivých bodech cesty. Řídicímu systému se pak předají body cesty, požadované rychlosti mezi sousedními body a zrychlení nebo doby jízdy. Je třeba zdůraznit, že uvedené profily platí pro jeden kloub, a proto je nutné celý postup generování trajektorie zopakovat pro každý zbylý kloub. [15]

Pro řešení tohoto problému se často používá algoritmus, který prolíná sousední rychlostní profily tak, že se rychlost mezi jednotlivými body nesnižuje na nulu a zároveň se zajistí průchod trajektorie přes každý bod. Toho se dosahuje predikcí rychlostního profilu, který se aktivuje po dosažení daného bodu dráhy. [15]

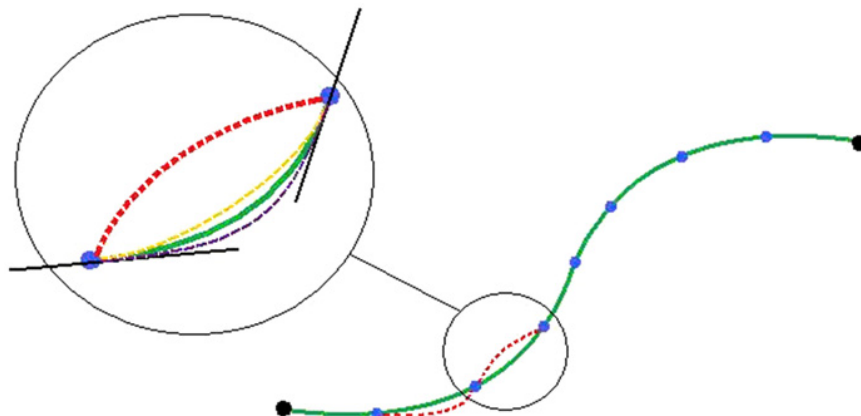


Obrázek 9 Prolnuté rychlosti [15]

Nový rychlostní profil se získá tak, že se n -tý profil spustí o Δt_i před ukončením předchozího ($i-1$)-tého profilu. Výsledkem je propojení dvou trapézových rychlostních profilů. Tím se zajistí plynulý průběh rychlosti a zabrání se tomu, aby rychlost mezi jednotlivými body klesla na nulu. Při průchodu každým bodem cesty však mohou nastat problémy. Aby manipulátor skutečně dosáhl přesně daných bodů v plánovaných časech, musí plocha pod křivkou rychlosti odpovídat ujeté vzdálenosti. Proto je nezbytné profily rychlosti upravovat změnou hodnot cestovních rychlostí nebo zrychlení. [15]

6.2 Plánování přiřazením pozice a tečných trajektorií do trajektorie

Z předešlého odstavce vyplývá, že prosté přiřazení bodů k dráze způsobí, že profil pohybu koncového efektoru závisí na pohybových zákonech každého z n kloubů. Tuto skutečnost znázorňuje obr. 10. Na něm tlustá plná čára označuje trajektorii přiřazenou body, zatímco tlustá tečkovaná čára ukazuje dráhu, kterou manipulátor skutečně projde mezi dvěma sousedními body. Představíme-li si, že má-li koncový efektor nejen procházet přiřazenými body, ale i udržet rychlost ve směru geometrické tečny k požadované dráze, dostaneme efektivní trajektorii znázorněnou tenkými tečkovanými čarami na obr. 10. Ta se pak téměř shoduje s požadovanou dráhou. [15]



Obrázek 10 Křivka s body a tangentami [15]

Na neapolské univerzitě Di.M.E. „Federico II“ se zkoumala metoda plánování trajektorie, která zlepšuje kvalitu dráhy tím, že kromě samotného průchozího bodu každé pozici přiřazuje také tečnu k požadované dráze. V kloubním prostoru to znamená, že ke každému bodu dráhy náleží i požadované rychlosti jednotlivých kloubů. [15]

Pro rozšíření popsané plánovací techniky se uvažuje, že řídicímu systému se k hodnotám kloubních proměnných q_i v každém z pevných bodů přiřadí i vektorové hodnoty rychlostí $\{q_i\}$, které musí klouby v těchto bodech dosáhnout. Po zadání požadované trajektorie koncového efektoru se na ní nejprve identifikují jednotlivé body. Je přitom nutné vzít v úvahu, že maximální počet zaznamenaných bodů je limitován rychlostí koncového efektoru (a tím i kloubů) a rovněž maximální vzorkovací frekvencí řídicího systému. [15]

Pro každý bod trajektorie se nejprve spočítají okamžité polohy všech kloubů a vyřeší se úloha inverzní kinematiky. Každému bodu je přiřazen vektor rychlosti v odpovídající velikosti a směru, tedy tečna k dráze. Díky tomu lze kloubní rychlosti pro všechny body trajektorie jednoduše určit použitím Jacobiánovy matice manipulátoru. [15]

Pro každý kloub manipulátoru se tak sestaví soubor hodnot polohy a rychlosti určených pro řídicí systém. Servomotory pak v každém okamžiku sledují příslušné pohybové zákony a vykonávají zadané hodnoty. Výsledkem je, že koncový efektor projde všemi body trajektorie s přiřazenou velikostí a směrem rychlosti. [15]

Tuto metodu lze označit jako plánování obálky tečen. V každém bodě musí interpolační polynom spojující sousední body i jeho první derivace odpovídat předem definovaným hodnotám. Díky tomu koncový efektor opustí daný bod a k dalšímu se přiblíží podél tečny k dráze. [15]

6.3 Plánování kloubového pohybového zákona pro „obálku tečen“

Vstupy pro generování referenčních bodů pro jednotlivé klouby jsou v tomto případě polohy $\{q_i\}$, kterých musí jednotlivé klouby manipulátoru dosáhnout, aby byla popsána požadovaná trajektorie v pracovním prostoru. Tyto polohy se získají po kinematické inverzi, která určuje polohy, jež je třeba uložit jednotlivým kloubům manipulátoru, aby se získaly vybrané body na geometrické trajektorii, která se vykreslí v pracovním prostoru. [15]

Dalším vstupem je rychlost průchodu pro výše uvedené body $\dot{q}_i$. Tyto rychlosti kloubů, které se odvozují od rychlosti vyjádřené velikostí a směrem (tečna ke geometrické trajektorii) v pracovním prostoru, jsou určeny podle geometrického jakobiánu manipulátoru, lze použít k určení vazby mezi rychlostí koncového efektoru a odpovídající rychlostí každého kloubu manipulátoru. [15]

Časové intervaly, v nichž se výše uvedené pohyby uskutečňují. Doba trvání těchto časových intervalů závisí na konkrétním časovém zákonu rychlosti koncového efektoru v pracovním prostoru. To je další uvažovaný vstup. [15]

Pro vytvoření rychlostního profilu mezi jedním a dalším bodem v kloubním prostoru se můžeme rozhodnout rozdělit příslušný časový interval na tři stejně dlouhé dílčí intervaly. Proto při Δt_i jako časovém intervalu použitým k přechodu z bodu p_{i-1} , odpovídajícímu okamžiku t_{i-1} , do bodu p_i , odpovídajícímu okamžiku t_i , bude mít každý dílčí interval délku $\Delta t_i/3$. Přesněji řečeno, první a třetí dílčí interval jsou fáze, ve kterých má manipulátor tendenci zrychlení nebo zpomalení podle rychlosti, která má být nastavena v obou směrových bodech p_{i-1} a p_i , tj. podle rozdílu požadovaných rychlostí. Střední podinterval, rovněž s dobou trvání $\Delta t_i/3$, je potřebný k tomu, aby bylo zaručeno, že plocha pod křivkou rychlosti se rovná vzdálenosti, kterou je třeba urazit. [15]

Aby bylo možné přejít z jednoho bodu do druhého s jinou hodnotou rychlosti, rychlost ve středním dílčím intervalu se zvýší (nebo sníží); tím se zaručí, že ujetá vzdálenost bude stejná jako požadovaná vzdálenost a že budou samozřejmě dodrženy časy stanovené při plánování. [15]

Rychlost ve středním úseku se vypočítá po stanovení vzdálenosti, která má být uražena, a stanovení počáteční rychlosti, konečné rychlosti a časů jízdy. Rovnice, která určuje plochu pod křivkou rychlosti, tj. vzdálenost, kterou je třeba ujet, je $s = \frac{t_f - t_i}{3} \cdot \frac{v_i + v^*}{2} + \frac{t_f - t_i}{3} \cdot v^* + \frac{t_f - t_i}{3} \cdot \frac{v_f + v^*}{2} = \frac{t_f - t_i}{3} \cdot \frac{v_i + v_f + 4 \cdot v^*}{2}$ kde s je ujetá vzdálenost, t_i počáteční okamžik trajektorie, t_f

konečný okamžik trajektorie, v^* cestovní rychlost středního úseku, v_i rychlost dosažená v počátečním bodě v okamžiku t_i , v_f rychlost, které má být dosaženo v konečném bodě v okamžiku t_f . Dosažená rychlost ve středním úseku bude dána $v^* = \frac{3 \cdot s}{2 \cdot (t_f - t_i)} - \frac{v_f + v_i}{4}$. [15]

II PRAKTICKÁ ČÁST

7 POUŽITÉ KNIHOVNY

V této kapitole jsou popsány použité knihovny a jejich význam pro implementované řešení.

7.1 ezdxf

Knihovna ezdxf je open-source nástroj pro programovací jazyk Python, který slouží k práci s formátem DXF (Drawing Exchange Format). Tento formát je standardně využíván pro výměnu dat mezi CAD aplikacemi, zejména programem AutoCAD. Knihovna ezdxf poskytuje robustní a přehledné rozhraní pro načítání, vytváření, modifikaci a ukládání DXF souborů. [23]

Jednou z hlavních výhod knihovny ezdxf je její schopnost efektivně zpracovávat různé typy geometrických entit. Umožňuje přímý přístup k objektům jako jsou čáry (LINE), kružnice (CIRCLE), oblouky (ARC), polyliny (POLYLINE a LWPOLYLINE) nebo spline křivky (SPLINE). [23]

V rámci této diplomové práce je knihovna ezdxf využita zejména pro automatizované načítání geometrických dat z CAD výkresů ve formátu DXF. Tyto data jsou dále zpracována a analyzována pro účely generování trajektorií pro řízení robotického ramene. [23]

7.2 OpenCV

OpenCV (Open Source Computer Vision Library) představuje softwarovou knihovnu s otevřeným zdrojovým kódem, která je určena pro oblast počítačového vidění a strojového učení. Tato knihovna byla vyvinuta s cílem vytvořit jednotnou infrastrukturní platformu pro aplikace v oblasti počítačového vidění a zároveň podpořit širší implementaci metod strojového vnímání v rámci komerčních produktů. Díky licencování pod Apache 2 je možné knihovnu snadno využívat i v průmyslovém sektoru, včetně možnosti její úpravy a přizpůsobení konkrétním potřebám. [24]

OpenCV disponuje více než 2500 vysoce optimalizovanými algoritmy, které pokrývají široké spektrum úloh od tradičních metod počítačového vidění až po pokročilé techniky strojového učení. Mezi typické aplikace těchto algoritmů patří detekce a rozpoznávání obličejů, identifikace objektů, klasifikace lidských činností ve videozáznamech, sledování pohybu kamery nebo objektů v obraze, konstrukce 3D modelů objektů, generování mračen bodů ze stereo záznamů, spojování obrazových dat do panoramat ve vysokém rozlišení, vyhledávání vizuálně podobných obrazů v databázích, odstraňování efektu červených očí na

fotografiích, sledování pohybu očí, rozpoznávání scén a jejich označování pro potřeby rozšířené reality a mnoho dalších. [24]

OpenCV se těší široké uživatelské základně čítající více než 47 tisíc vývojářů a uživatelů po celém světě, přičemž počet stažení knihovny přesahuje 18 milionů. Díky své otevřenosti, flexibilitě a rozsáhlým možnostem využití nachází OpenCV uplatnění jak v komerční sféře, tak i ve výzkumných institucích a státních organizacích. [24]

7.3 NumPy

Knihovna NumPy (Numerical Python) je jedním ze základních nástrojů pro vědecké výpočty v jazyce Python. Poskytuje podporu pro práci s vícerozměrnými poli (tzv. n-dimenzionální arrays) a nabízí rozsáhlou sadu matematických funkcí pro efektivní manipulaci s numerickými daty. NumPy je optimalizovaná pro rychlé výpočty a umožňuje vektorové operace, které jsou výrazně efektivnější než klasické iterace. [25]

V rámci této diplomové práce je knihovna NumPy využívána především pro zpracování a analýzu geometrických dat načtených z DXF souborů. Konkrétně se jedná o práci se souřadnicemi bodů a výpočty geometrických vlastností jako jsou vzdálenosti mezi body, směrové vektory, úhly mezi úsečkami nebo úhlové změny mezi jednotlivými segmenty. Díky NumPy bylo možné realizovat tyto výpočty ve vektorové podobě, což významně zjednodušuje kód a zároveň zvyšuje jeho výkonnost. [25]

NumPy hraje také důležitou roli při segmentaci geometrických entit. Například při hledání bodů, kde dochází ke změně směru trajektorie. Pomocí dalších operací s poli jsou dále vyhledávány skokové změny těchto úhlů, na jejichž základě je následně trajektorie rozdělena do jednotlivých segmentů. [25]

Dalším příkladem využití NumPy je normalizace vektorů nebo výpočet středních hodnot, které jsou potřeba pro konstrukci oblouků nebo interpolovaných křivek. V kombinaci s knihovnou SciPy, která rozšiřuje možnosti numerických výpočtů, umožňuje NumPy i interpolaci dat a plynulé vykreslení spline křivek. [25]

Díky svému výkonu, jednoduchosti a rozsáhlé dokumentaci je NumPy nepostradatelným nástrojem při práci s numerickými a geometrickými daty. V této práci se tak NumPy stal klíčovým prvkem při vývoji algoritmu pro analýzu a přípravu dráhy robotického nástroje. [25]

7.4 Matplotlib

Knihovna Matplotlib je rozsáhlý vizualizační nástroj pro programovací jazyk Python, který umožňuje tvorbu dvourozměrných (a částečně i trojrozměrných) grafických výstupů. Byla navržena tak, aby se co nejvíce podobala knihovnám pro kreslení grafů z prostředí MATLABu, a díky tomu nabízí intuitivní a flexibilní API pro generování grafů, vykreslování bodů, úseček, křivek a dalších prvků. [26]

Pomocí knihovny Matplotlib jsou zobrazovány mezikroky výpočtu, jako je například segmentace podle úhlových změn, spojování bodů do skupin nebo vykreslování oblouků mezi dvěma body s daným úhlem. Tyto vizualizace nejsou jen pomocné při vývoji algoritmu, ale slouží také jako důkaz funkčnosti zvoleného řešení. Při konstrukci oblouků je využita i možnost vykreslování objektů z modulu patches, konkrétně objektu Arc, který umožňuje vykreslit přesný oblouk mezi dvěma body s daným středem a úhlem. [26]

Díky své jednoduchosti, přehlednosti a širokým možnostem přizpůsobení je Matplotlib ideálním nástrojem pro vizualizaci v technických a vědeckých aplikacích. V kontextu této práce přispěla knihovna k lepší srozumitelnosti výstupů, ladění algoritmů a prezentaci výsledků analýzy geometrických dat. [26]

7.5 SciPy

Knihovna SciPy (Scientific Python) je rozšířením knihovny NumPy a slouží jako pokročilý nástroj pro vědecké výpočty, technické simulace a numerickou analýzu. Obsahuje množství modulů pokrývajících široké spektrum úloh – od integrace, optimalizace, statistiky, až po interpolaci nebo práci s diferenčními rovnicemi. Využívá výkonné implementace algoritmů založené na jazycích C a Fortran, což zajišťuje vysokou rychlost výpočtů i při zpracování rozsáhlejších datových sad. [27]

V rámci této diplomové práce je knihovna SciPy využita zejména pro účely interpolace. Tato funkce je klíčová při práci s entitami typu POLYLINE, které jsou v CAD datech reprezentovány pomocí řídicích bodů. Pomocí modulu `scipy.interpolate` je z těchto bodů vytvořena hladká křivka, která reprezentuje trajektorii, již má následně vykonávat nástroj robotického ramene. Konkrétně je využita funkce `splprep` pro vytvoření parametrické křivky a následně `splev` pro její vyhodnocení v jemných krocích. [27]

Díky svému propojení s NumPy a rozsáhlé dokumentaci je SciPy velmi vhodnou volbou pro technicky orientované aplikace. V této práci se osvědčila jako stabilní a přesná knihovna pro generování hladkých drah podle vstupních dat z konstrukčního výkresu. [27]

7.6 Requests

Knihovna Requests je v Pythonu používána jako de-facto standardní rozhraní pro práci s protokolem HTTP(S). Je navržena tak, aby abstrahovala nízkoúrovňové detaily připojení (adresování socketů, zpracování TLS certifikátů, kódování dat) a uživateli poskytla intuitivní API založené na relačních metodách *GET*, *POST*, *PUT*, *DELETE* apod. V této práci je Requests využívána k odesílání příkazů a přijímání odpovědí z rozhraní Robot Web Services (RWS) řadiče ABB. Díky automatické správě relací usnadňuje knihovna opakované dotazy a snižuje režii šifrované komunikace. [28]

7.7 urllib3

Součástí balíku urllib3, na němž Requests interně staví, je třída výjimky *InsecureRequestWarning*. Toto varování je generováno vždy, když se navazuje HTTPS spojení k serveru, jehož certifikát nelze ověřit proti důvěryhodné autoritě (typicky self-signed certifikát vestavěný v řadiči ABB). V rámci diplomové práce je varování programově potlačeno. [29]

Tím je výstup konzole očištěn od opakovaných hlášek a uživatel se může soustředit na relevantní logy spouštění a zastavování robotického programu. Samotné spojení se přesto realizuje přes TLS, pouze se vynechává kontrola řetězce důvěryhodnosti certifikátu; tento postup je běžný při vývoji a offline simulacích, kde by instalace vlastního CA certifikátu byla neúměrně složitá. [29]

8 NAČTENÍ DXF SOUBORU

V této části se implementuje skript pro načítání geometrických entit ze souboru ve formátu DXF pomocí knihovny *ezdxf*. Na začátku je definována proměnná *file*, která obsahuje název souboru, a proměnná *imgSize*, jejíž hodnota se v tomto úseku kódu dále nevyužívá. Při načítání souboru pomocí funkce *ezdxf.readfile()* se ošetřují dvě možné výjimky. Pokud soubor neexistuje, je vyhozena výjimka *IOError*, uživatel je informován zprávou „*File not found*“ a skript je ukončen s návratovým kódem 1. V případě, že má soubor neplatnou nebo poškozenou strukturu, je vyhozena výjimka *DXFStructureError*, vypsána odpovídající chybová hláška a program je ukončen s návratovým kódem 2.

Jako testovací byl využit následující obrázek převedený pomocí online nástroje dostupného z <https://convertio.co/jpg-dxf/>.



Obrázek 11 Původní obrázek [34]

Po úspěšném načtení souboru se získává přístup k modelovému prostoru pomocí metody *modelspace()*, odkud se postupně iterují všechny obsažené entity. Pro jejich uložení se inicializují prázdné seznamy pro jednotlivé typy geometrických objektů. *lines* pro úsečky, *circles* pro kružnice, *arcs* pro oblouky, *lwpolylines* pro lehké polylinie a *polylines* pro obecné polylinie. Zároveň se inicializuje čítač *countOfEntities*, do kterého se zaznamenává celkový počet zpracovaných entit, který se používal v rámci vývoje kódu.

Každá entita se vyhodnocuje podle svého typu pomocí metody *dxftype()*. Pokud se jedná o úsečku (LINE), získávají se souřadnice počátečního a koncového bodu, které se ukládají jako dvojice bodů do seznamu *lines*. V případě kružnice (CIRCLE) se ukládají souřadnice středu a hodnota poloměru do seznamu *circles*. U oblouku (ARC) se získávají souřadnice středu, poloměr a úhly počátečního a koncového bodu a následně se zapisují do seznamu *arcs*.

Lehké polylinie (LWPOLYLINE) se zpracovávají pomocí metody *get_points()*, která vrací seznam jejich vrcholů. Každý vrchol se ukládá jako dvojice souřadnic do seznamu *lwpolylines*. U běžných polylinií (POLYLINE) se vrcholy iterují metodou *entity.points()* a převádějí se do seznamu dvojic (X, Y), který se následně ukládá do seznamu *polylines*.

V případě spline křivek (SPLINE) se získávají kontrolní body podle stupně křivky (degree). Každý bod se ukládá jako dvojice celočíselných souřadnic, přičemž všechny tyto body se po zpracování převádějí do objektu typu *numpy.array*.

Výsledkem výše uvedených operací, je obdržení parametrů entit. Nicméně práce je zaměřena na zpracování entit typu polylinie, jelikož ostatní entity mohou být na tuto jednoduše převedeny a jsou schopné obsáhnout takto všechny možné entity.

8.1 Ověření načtených dat

V této fázi se přechází k vykreslení načtených entit. Tato část slouží hlavně pro kontrolu správnosti načtených entit. Jako výstupní médium je použita proměnná *img*, která představuje obrázkovou matici, do níž se jednotlivé geometrické útvary vykreslují pomocí knihovny OpenCV. Kromě OpenCV je pro vykreslení některých entit (např. oblouků nebo spline křivek) použita i knihovna Matplotlib.

Nejprve se vykreslují úsečky uložené v seznamu *lines*. Každá úsečka se kreslí pomocí funkce *cv2.line()* na základě souřadnic počátečního a koncového bodu. Barva čáry je nastavena na bílou a tloušťka je nastavena na 1 pixel. Následně se vykreslují kružnice ze seznamu *circles*, kde se pro každý záznam využívá střed (X, Y) a poloměr. Kreslení probíhá funkcí *cv2.circle()* se stejným barevným a tloušťkovým nastavením.

Pro oblouky, uložené v seznamu *arcs*, se vytváří úhlové rozmezí mezi počátečním a koncovým úhlem pomocí *numpy.linspace()*. Poté se vypočítávají souřadnice bodů ležících na oblouku podle parametrického tvaru rovnice kružnice, kde se úhly převádějí na radiány.

Výsledná množina bodů se vykresluje pomocí funkce *plt.plot()*, čímž se zajišťuje hladké zobrazení oblouku.

Lehké polylinie ze seznamu *lwpolylines* se vykreslují pomocí spojování sousedních bodů funkcí *cv2.line()*. Pro každou dvojici sousedních bodů se vykreslí úsečka, přičemž poslední bod je spojen s prvním, aby byl uzavřen tvar, pokud je to potřeba. I zde se používá bílá barva a tloušťka 1 pixel.

Následně se vykreslují běžné polylinie ze seznamu *polylines*. Každá polylinie se iteruje a její souřadnice se oddělí na složky x a y pomocí funkce *zip()*. Tyto body se následně vykreslí pomocí funkce *plt.scatter()* a jsou navíc opatřeny popiskem pomocí proměnné *nPolyline*, která slouží jako identifikátor segmentu.

V případě, že je aktuální entita typu SPLINE, se získají souřadnice všech kontrolních bodů, které byly dříve uloženy do pole *splines*. Tyto body se interpolují pomocí funkce *interpolate.splprep()* a následně se vyhlazená křivka získá funkcí *interpolate.splev()*. Původní kontrolní body i interpolovaná spline křivka se vykreslí pomocí *plt.plot()*.

Na závěr se celý obrázek zobrazí pomocí funkce *plt.imshow()*, přičemž se vypne zobrazení os a vykreslí se legenda, která zajišťuje označení jednotlivých polylinií. Funkce *plt.show()* slouží k finálnímu zobrazení vytvořeného výstupu.



Obrázek 12 Vykreslení načtených dat

V této části jsou tedy načtené geometrické entity jsou po zpracování vykreslovány do výstupního obrazu. Každý typ entity se vizualizuje specifickým způsobem s využitím

knihoven OpenCV a Matplotlib. Úsečky a kružnice se kreslí přímo do obrázku. Oblouky jsou konstruovány jako množiny bodů podle parametrických rovnic a vykreslovány pomocí Matplotlibu. Lehké a běžné polylinie se tvoří spojením jednotlivých vrcholů, přičemž polylinie jsou navíc opatřeny popisky. Spline křivky jsou zpracovány interpolací kontrolních bodů a vykresleny jako hladké křivky. Celkový výstup se zobrazí jako obrázek s vypnutým zobrazením os a doplněnou legendou, která usnadňuje orientaci mezi jednotlivými segmenty. Tím je umožněna přehledná vizualizace obsahu DXF souboru.

9 HLEDÁNÍ ÚSEČEK

V tomto úseku skriptu se provádí podrobná analýza tvaru jedné konkrétní polylinie za účelem identifikace jednotlivých přímkových segmentů. Nejprve jsou vytvořeny dva vektory rozdílů souřadnic *np.diff(x)* a *np.diff(y)*, z nichž je funkcí *np.arctan2()* vypočítán směrový úhel každého elementárního úseku, tedy úhel čáry spojující bod *i* a *i + 1* vzhledem k ose x. Úhly jsou vráceny v radiánech v rozsahu $(-\pi, \pi]$, což usnadňuje následný výpočet jejich změn.

Ze sekvence úhlů je pomocí *np.diff(angles)* odvozen vektor *angle_changes*, jenž reprezentuje změny směru mezi sousedními úseky. Absolutní hodnota těchto změn zajišťuje, že se neposuzuje orientace (vpravo nebo vlevo), nýbrž pouze velikost odchylky. Poslední vypočtená změna je zreplikována na konec pole, aby byla zachována délková konzistence s počtem bodů polylinie. Tato kopie je nezbytná, protože funkce *np.diff()* vrací pole o jeden prvek kratší, než je vstupní velikost, a indexace v dalším kroku vyžaduje shodu délek.

Změny úhlů jsou porovnávány s prahovou hodnotou *angle_threshold*, jež činí 2 stupně. Pokud změna směru překročí tento práh, aktuální segment se ukončí a je zahájen nový segment. Jinak je bod považován za součást stávajícího segmentu. Segmentace je řízena dvěma pomocnými strukturami seznamem *segments*, který uchovává indexy bodů patřících do jednotlivých částí, a aktuálně otevřeným seznamem *current_segment*, kam se postupně přidávají indexy, dokud nedojde k překročení prahu.

Po dokončení hlavní segmentace je aplikována korekce pro případ, že analyzovaná polylinie tvoří uzavřený tvar. Podmínka ověřuje, zda poslední bod posledního segmentu splývá s posledním bodem původní polylinie a zda první bod prvního segmentu splývá s prvním bodem polylinie. Pokud ano, vzniká dodatečný segment *extraSegment*, jenž propojí oba koncové body, aby byl zachován topologický uzávěr.

V závěrečném průchodu se prochází každým segmentem. Jestliže segment *i* a segment *i + 1* obsahují alespoň dva body, přidá se do segmentu *i* index prvního bodu následujícího segmentu. Tím se dosahuje sloučení hranic sousedních segmentů, které společně leží na téže přímce, což umožní plynulé navazování při dalším zpracování, například při konverzi na samostatné entitní úsečky nebo při výpočtu délek.

Přesnost detekce lomených bodů je přímo ovlivněna volbou prahu. Příliš nízká hodnota vede k nadměrné segmentaci, zatímco vysoká hodnota může způsobit, že skutečné zlomy

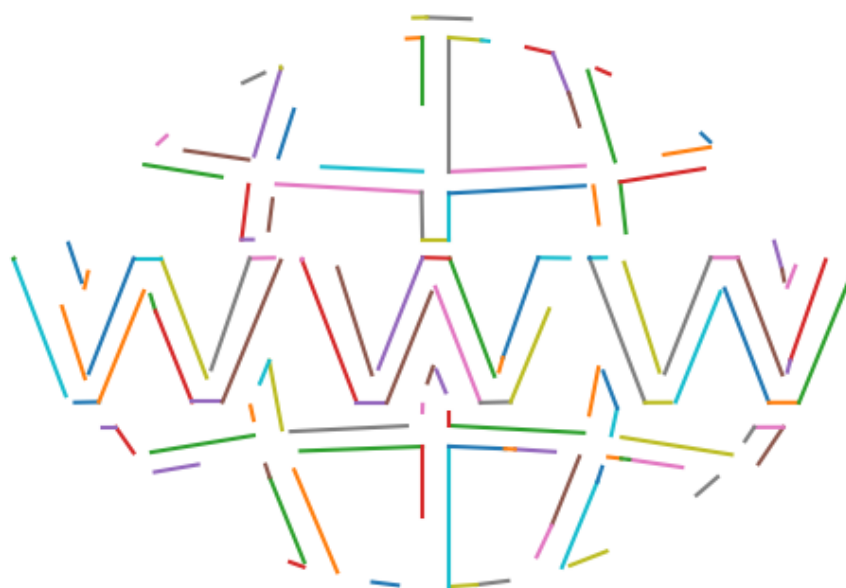
zůstanou nerozpoznány. Hodnota 2 je zde zvolena jako kompromis mezi citlivostí a robustností.

V následující fázi algoritmu se provádí transformace surových segmentů na dvě cílové datová pole, které se dále využívají. První pole *linePoints* je určeno k uchování čistých přímkových úseků reprezentovaných pouze jejich koncovými body, čímž se výrazně snižuje datový objem a zároveň se zachovává veškerá geometrická informace potřebná pro rekonstrukci čáry. Druhé pole *otherPoints* slouží jako zásobník bodů, které nelze okamžitě přiřadit k přímkám, ale které mohou představovat uzlové vrcholy lomených nebo zakřivených prvků.

Během iterace přes pole *segments* se nejprve vyhodnocuje délka každého segmentu. Pokud segment obsahuje více než jeden bod, je považován za kandidáta na přímku. Získává se souřadnice prvního a posledního bodu v příslušných polích *x* a *y*, přičemž tyto hodnoty se ukládají do *linePoints* ve tvaru $[x_start, y_start, x_end, y_end]$. Touto operací se celý lineární úsek redukuje na dvojici koncových vrcholů, protože mezilehlé body nepřinášejí žádnou dodatečnou informaci o směru ani o poloze čáry, a jejich vypuštěním se dosahuje kompaktnější reprezentace dat.

Pokud segment obsahuje jediný bod, nastupuje cyklus, jehož cílem je rozhodnout, zda bod představuje skutečný izolovaný vrchol, nebo zda má fungovat jako spojovací uzel mezi okolními přímkami. Nejprve se testuje, jestli segmenty bezprostředně před a za aktuálním bodem vykazují délku alespoň dvou bodů, čímž se považují za přímky. Pokud ano předpokládá, že bod leží na průsečíku dvou přímkových úseků, algoritmus proto ukládá trojici bodů do *otherPoints* tak, aby byl zachován správný sled vrcholů v budoucím spojování geometrie. Jestliže přímka existuje pouze na jedné straně, ukládají se do *otherPoints* dva body, a to koncový bod přímky a aktuální vrchol, čímž se zajišťuje, že bod bude mít v topologii odpovídající návaznost. V situaci, kdy segmenty na obou stranách rovněž obsahují jediný bod, je aktuální vrchol považován za izolovaný a bez dalších souvislostí se vkládá do *otherPoints* samostatně.

Tato vícestupňová rozhodovací logika zabraňuje ztrátě kritických uzlových bodů, které by později mohly být potřeba při rekonstrukci křivek, a zároveň brání vytváření přebytných topologických vazeb mezi nesourodými úseky. Datová struktura seznamů navíc umožňuje rychlé indexové přístupy, čímž se minimalizuje režie spojená s častými dotazy na sousední segmenty.



Obrázek 13 Nalezené úsečky

10 VYTVÁŘENÍ SKUPIN Z BODŮ, KTERÉ NEJSOU PŘÍMKY

V této části skriptu se připravuje logika pro seskupování bodů, které nebyly přiřazeny k přímkovým úsekům. Nejprve jsou inicializovány dva prázdné seznamy *groupOfNonLine* a *groupsOfNonLine*, jež v dalším kroku poslouží k uchování dočasné a finální podoby těchto skupin. Pro každý pár po sobě jdoucích bodů ze seznamu *otherPoints* se vypočítává eukleidovská vzdálenost a výsledky se vkládají do pole *distances*. Tím se vytváří přehled o vzájemných rozestupech mezi sousedními body, který je nezbytný pro odhalení přirozených shluků.

Pokud pole *distances* není prázdné, je nad ním nejprve sestaven histogram, jehož četnosti se ukládají do proměnné *distDistri*, zatímco *histDist* obsahuje hodnoty hranic jednotlivých binů. Tato statistická analýza umožňuje identifikovat typickou vzdálenost mezi body a současně odfiltrovat odlehlosti. Jestliže histogram vykazuje obsazenost v méně než třech binech, považuje se datová sada za příliš malou. V takovém případě se za reprezentativní prahovou vzdálenost přijímá maximum *np.max(distances)*. V opačné situaci se nejprve detekují biny s četností 1, které signalizují ojedinělé, tedy pravděpodobně nepříslušné hodnoty. Pro každý takto označený bin se nalezne jeho nejvyšší hranice *maxDist*, vyhledá se jí nejblížejší konkrétní hodnota v poli *distances* funkcí *closest_value()* a tato vzdálenost se z pole odstraní, jelikož se očekává, že jde o vzdálenost mezi body, které spolu přímo nesousedí. Po eliminaci odlehlostí se z aktualizovaného pole *distances* opět odečte maximum, jež se ukládá do proměnné *justDistance*. Tato proměnná tak reprezentuje robustní odhad horní hranice rozestupu mezi body a poslouží jako dynamický práh pro následné seskupování. Body, jejichž vzájemná vzdálenost nepřekračuje *justDistance*, budou přiřazovány do stejné skupiny, zatímco větší rozestup bude interpretován jako hranice mezi dvěma shluky.

Ve zbývající části algoritmu se z pole *otherPoints* vytvářejí shluky bodů, které spolu souvisejí buď geometrickou blízkostí, anebo tím, že představují uzlové vrcholy sdílené s přímkovými úseky. Nejprve se inicializují pomocné proměnné *foundOne*, *needSkip* a *oi*, jejichž úkolem je řídit stav, v němž se právě nachází proces seskupování. Proměnná *foundOne* signalizuje, zda byl v aktuálním průchodu detekován bod, který je současně koncovým bodem některé přímky. Proměnná *needSkip* slouží k přeskočení části kódu, pokud se po detekci takového bodu má okamžitě pokračovat další iterací smyčky. Proměnná *oi* si uchovává polohu bodu, jenž byl posledním detekovaným sdíleným uzlem, aby bylo možné korektně uzavřít logiku, která řetězí sousední body do společné skupiny. Tohoto rozvětvení

je zapotřebí, aby se rozlišily skutečné počátky a konce úseček. Tedy, aby se nemohlo stát, že celá úsečka bude identifikována jako úsečka a zároveň jako bod, který netvoří úsečku.

Hlavní smyčka prochází všechny body. V každém kroku se nejprve provádí test, zda aktuální bod *otherPoints[i]* odpovídá některému z koncových bodů přímkových úseků v seznamu *linePoints*. Pokud je shoda nalezena (porovnávají se vždy obě souřadnicové složky), nastaví se *needSkip* na *True*, aby se po zpracování přeskočily další operace vzdálenostní analýzy. Dále se podle stavu příznaku *foundOne* a relace indexů *oi* rozhoduje, zda má být aktuální uzlový bod pouze připojen k rozpracované skupině *groupOfNonLine*, nebo zda tato skupina již má být uzavřena a vložená do *groupsOfNonLine*. Tím se zajišťuje, že sdílené uzly (koncové body linek) vždy tvoří přirozenou hranici shluku, ale současně nezůstanou opomenuty.

Jestliže *needSkip* zůstane *False*, pokračuje algoritmus k vzdálenostnímu testu mezi aktuálním bodem a následujícím bodem. Eukleidovská vzdálenost se porovnává s prahovou hodnotou *justDistance*, která byla dříve odvozena z histogramu rozestupů. Je-li vzdálenost menší nebo rovna tomuto prahu, považují se body za součást stejného geometrického útvaru a jsou ukládány do otevřené skupiny *groupOfNonLine*. Pokud vzdálenost mezi body práh překročí, vkládá se aktuální bod do rozpracované skupiny, skupina se uzavírá přesunem do *groupsOfNonLine* a následně se inicializuje nová prázdná skupina. Výjimku představuje poslední iterace, kde se po jejím dokončení explicitně připojuje i poslední bod pole *otherPoints[i + 1]*, aby žádný prvek nezůstal nezařazen.

Takto sestavené pole *groupsOfNonLine* tedy obsahuje uspořádané seznamy bodů, mezi nimiž nejsou vzdálenosti větší než statisticky odvozený práh a které zároveň respektují hranice definované sdílenými uzly. Díky kombinaci topologické kontroly (koncové body linek) a adaptivního prahu se docílí robustního seskupení i v případech, kdy jsou body rozmístěny nepravidelně nebo kdy jejich hustota v různých částech kresby výrazně kolísá.

11 TVOŘENÍ KŘIVEK Z BODŮ

V předchozím textu bylo popsáno, jak se tvoří proměnná *groupsOfNonLine*, která obsahuje skupiny bodů reprezentující různé křivky. Některé tyto křivky jsou reprezentovány nedostatečným počtem bodů, tudíž by při vykreslení došlo ke zkreslení, kdy by místo křivky vznikl soubor úseček. Proto je zapotřebí tyto křivky interpolovat tak, aby se toto zkreslení minimalizovalo.

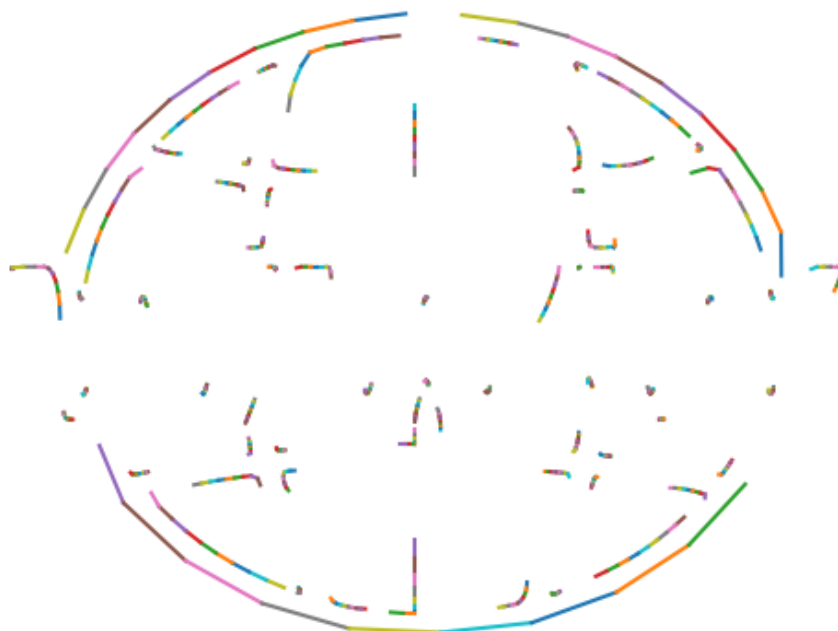
Funkce *densify_pchip()* slouží k přesně k tomuto účelu, tedy k zpřesnění vložením rovnoměrně rozmístěných bodů mezi původní uzly. Na vstupu je očekáván dvourozměrný seznam či pole *pts*, jehož každý řádek obsahuje souřadnici x a y jednoho bodu, a volitelně parametr *n_samples*, jenž určuje, do kolika vzorků má být křivka *převzorkována*.

Nejprve jsou vstupní body převedeny na objekt *numpy.ndarray* typu *float*, aby mohlo být dále pracováno s vektorovými operacemi. Následně je provedena parametrizace bodů podle kumulativní délky oblouku. Ta se získává rozdílem sousedních bodů *np.diff(pts, axis=0)*, z něhož je pomocí *np.hypot()* vypočtena eukleidovská délka každého elementárního segmentu. Akumulací těchto délek vzniká vektor *t*, který představuje „*arc-length parametr*“ celé křivky: hodnota *t[i]* vyjadřuje vzdálenost bodu *pts[i]* od počátku křivky měřenou podél její trajektorie. Tento způsob parametrizace se volí proto, že délka oblouku odstraňuje problémy spojené s nerovnoměrnými projekcemi bodů do os x nebo y. Křivka se tak vyhodnocuje rovnoměrně podle skutečné geometrické dráhy.

Na základě celkové délky *t[-1]* je sestavena hustější mřížka *t_dense* o *n_samples* bodech, která je rovnoměrně rozprostřena po celé délce křivky. Následně jsou vytvořeny dva samostatné PCHIP interpolátory – *PchipInterpolator(t, pts[:, 0])* pro x souřadnice a *PchipInterpolator(t, pts[:, 1])* pro y souřadnice. PCHIP (Piecewise-Cubic Hermite Interpolating Polynomial) se volí záměrně. Na rozdíl od klasické kubické spline zachovává monotonicitu (pořadí prvků) dat a zabraňuje vzniku artefaktů, které by mohly vést k nerealistickým průhybům křivky v oblastech s řídko vzorkovanými body nebo s prudkými změnami směru. [27]

Interpolátory jsou následně evaluovány na husté mřížce *t_dense*, čímž vzniknou vektory *x_dense* a *y_dense*. Tyto dva vektory jsou sloučeny do pole pomocí *np.vstack()* a transpozice, aby výsledná struktura kopírovala formát původního pole bodů. Funkce tak vrací soubor bodů, které jsou rovnoměrně rozloženy podél křivky a zároveň přesně leží na PCHIP aproximaci původních dat.

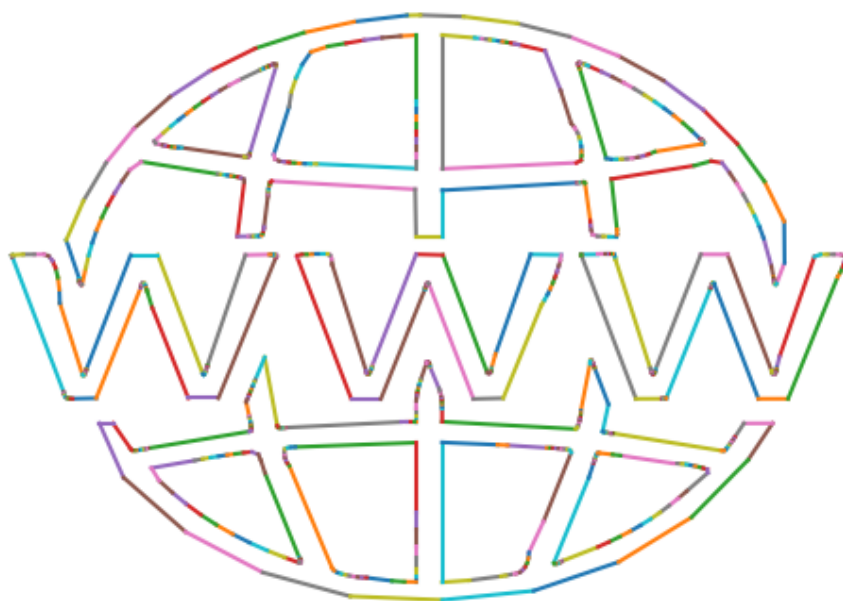
Z hlediska numerické stability zajišťuje PCHIP, že odvozená křivka prochází všemi původními uzly a v nich má spojité první derivace, avšak nikoli nutně derivace vyššího řádu – to je kompromis umožňující vyhnout se lokálním extrémům, které by mohly vzniknout u klasických cubic-spline metod.



Obrázek 14 Vykreslené křivky

Zhuštěné body se používají v aplikacích, kde je vyžadována hladká a detailní reprezentace křivky – například při generování CNC drah, při simulaci pohybu robotů, při FEM analýzách, kde se jemnější diskretizace promítá do přesnějších výsledků, nebo při vizualizaci, již by příliš řídké vzorkování zubatělo. Parametr *n_samples* lze přizpůsobit nárokům na přesnost a výkon; vyšší hodnota poskytuje detailnější křivku, avšak zvyšuje paměťové nároky a dobu následného zpracování.

Je-li třeba zachovat uzavřenou smyčku, je vhodné před voláním funkce doplnit na konec pole *pts* kopii prvního bodu, aby PCHIP mohl správně vyhladit hranu spoje. Pro extrémně dlouhé křivky s neúměrně variabilní hustotou bodů se může zvážet adaptivní vzorkování (např. na základě lokální zakřivenosti), avšak základní lineární zhuštění s PCHIP zde představuje robustní a obecně použitelný kompromis mezi přesností a jednoduchostí implementace.



Obrázek 15 Celkové vykreslení

12 PŘÍPRAVA DAT PRO PŘENOS

Před samotným přenosem dat je zapotřebí, aby odeslaná data měla jasnou a co nejjednodušeji čitelnou strukturu. To umožňuje lepší zpracování v systému, který tyto data přijímá.

12.1 Úsečky

V této části kódu se připravuje druhý datový blok, který obsahuje přímkové úseky. Na začátku se alokuje statické pole *full_array_points* o 1000 prvcích, kde každý prvek je předvyplněn hodnotou $[-1, 0, 0, 0, 0]$. Hodnota -1 slouží k rozpoznání konce platných dat. Následně se seznam *sendLinePoints*, v kterém dosud jsou hodnoty typu *numpy.float64*, převádí na typ *float*. Tato konverze je nezbytná, aby správně proběhl přenos dat.

Po převodu se detekuje nejmenší x souřadnice v poli *float_data*. Ta se ukládá do proměnné *min_in_array_x* a slouží k normalizaci souřadnicového systému. Všechny body se totiž posunují tak, aby nejmenší x začínalo přibližně hodnotou 10. Od hodnot *x_start* a *x_end* každého záznamu se odečítá výraz $min_in_array_x - 10$. Tím se zaručí, že celý výkres leží v kladné části osy x a že zůstává malá rezerva před nulou, ta může být využita pro bezpečnostní odsazení uvnitř pracovního prostoru.

Jakmile jsou souřadnice posunuty, kopírují se převedené záznamy do počáteční části *full_array_points*. Zbytek pole zůstává vyplněn předvyplněnou hodnotou. Výsledkem je homogenní pole pevné délky, které zaručuje, že robot obdrží vždy stejně velký paket dat nezávisle na tom, kolik segmentů daný výkres skutečně obsahuje. Díky tomu se eliminuje nutnost dynamické alokace.

12.2 Převedení dat do formátu pro přenos (křivky)

V této závěrečné fázi se připravuje datová struktura, která se následně odesílá kontroleru robota. Výstup má jednotný formát $[x, y, is_last, angle]$, kde *x* a *y* představují souřadnice bodu, *is_last* signalizuje, zda je bod posledním prvkem příslušné křivky (1 = ano, 0 = ne) a *angle* uchovává směr (v radiánech) od aktuálního bodu k bodu následujícímu.

Nejprve se deklarují dočasné seznamy *non_in_middle* a *nonLines*. Algoritmus prochází všechny skupiny nepřímkových bodů *finGroupsOfNonLine*. Jednotlivé skupiny (křivky), se interpolují voláním dříve definované funkce *densify_pchip()* s parametrem *n_samples=10*.

To zajistí, že každá křivka je reprezentována deseti rovnoměrně rozprostřenými body – počet bodů je zde kompromisem mezi přesností a velikostí výsledného datového balíku.

Pro každý bod se do pole *non_in_middle* ukládají souřadnice *x* a *y*. Následně se podle indexu cyklu nastavuje příznak *is_last*, což je příznak posledního bodu křivky, který, pokud se jedná o poslední bod křivky přiřazuje 1, všem ostatním 0. Takto naplněný seznam *non_in_middle* se vloží do *nonLines* a proměnná se resetuje pro další iteraci. Po ukončení dvojité smyčky je ještě poslední uložený prvek rozšířen o úhel 0, čímž se sjednotí délka všech záznamů.

V dalším kroku se provádí globální posun všech křivek po ose *x*. Od každé hodnoty *nonLines[i][0]* se odečítá *min_in_array_x - 10*, kde *min_in_array_x* je definována jako nejmenší z *x* souřadnic přímkových bodů. Tím se geometrie posune doleva aby *x* souřadnice bodů začínali co nejbližší nule.

Následně se mezi všemi vypočítá směrový úhel pomocí *np.arctan2()*. Tento úhel vyjadřuje orientaci segmentu spojujícího aktuální bod s bodem následujícím a ukládá se jako čtvrtý prvek každého záznamu. Zároveň se pro čistě ilustrační účely vykresluje každé nenulté spojení do grafu Matplotlib (propojují se segmenty, jejichž *is_last* = 0, aby se předešlo spojení křivek, které spojeny být nemají).

Pro samotné odeslání se alokuje pevná délka datového balíku – pole *full_array_curves* má kapacitu 1000 záznamů, přičemž nevyužitá místa se předvyplňují hodnotou [0, 0, -1, 0]. Hodnota -1 reprezentuje konec dat, který přijímající systém snadno rozpozná. Do počáteční části pole se kopírují *nonLines* a zbytek zůstává ve výchozím stavu. Výsledkem je homogenní pole vhodné pro přenos, které se díky fixní velikosti snadno zpracovává na straně robota.

13 PŘENOS DAT DO ROBOTA

V závěrečné fázi skriptu se připravená data odesílají do řídicího systému robota prostřednictvím REST rozhraní ABB Robot Web Services (RWS).

13.1 Python

Nejprve jsou deklarovány dvě možné IP adresy cílového zařízení. *IP_RC* reprezentuje adresu skutečného řídicího kontroleru (port 443 pro HTTPS), zatímco *IP_VC* odkazuje na lokální virtuální řadič (port 80). Proměnná *ip* je nastavena na *IP_VC*, a tak skript implicitně komunikuje s offline simulátorem. Přepnutím hodnoty lze snadno směřovat provoz přímo na fyzický stroj. Dále jsou definována jména RAPID proměnné *arrLine* a *arrCurve* v modulu *MainModule*, do nichž se budou zapisovat úsečkové a křivkové body.

Následně se vytváří sdílená instance *requests.Session()*, která zajišťuje persistentní TCP spojení a ušetří režii TLS handshaku při opakovaných dotazech. K session se přiřazuje základní autorizace (uživatel *Default User*, heslo *robotics*) a standardní hlavičky protokolu RWS ve verzi 2.0. Konkrétní URL adresa pro zápis dat se skládá z prefixu */rw/rapid/symbol/RAPID/T_ROBI/*, jména modulu a proměnné, přičemž parametr *mastership=implicit* zajišťuje, že webová služba dočasně převezme exkluzivní oprávnění k úpravě úlohy RAPID bez nutnosti explicitního přihlášení přes FlexPendant.

První *POST* dotaz odesílá pole *full_array_points* do proměnné *arrLine*. Datová část požadavku (*data={'value': f'{full_array_points}'}*) předává textovou reprezentaci Python listu. Po odeslání se vyhodnocuje návratový kód. Stav *204* znamená úspěšné přepsání hodnoty, zatímco *404* signalizuje neexistenci cílové proměnné (např. nesoulad názvu modulů nebo proměnných). Jakýkoli jiný kód je vypsán jako obecná výstraha.

Analogický postup se opakuje pro druhý datový blok *full_array_curves*, jenž obsahuje body křivek. K prvnímu URL segmentu se pouze připojí odlišné jméno proměnné *arrCurve*. Po odeslání se stavový kód opět vyhodnocuje a v případě chybného zápisu se skript pokusí vytisknout tělo odpovědi *resp.json()*. Pokud JSON selže, vypisuje se syrový text, což usnadňuje ladění. Odchytávání výjimky *ConnectionError* chrání program před chybou, pokud je server nedostupný (například při vypnutí simulátoru nebo ztrátě sítě).

Celý komunikační blok pracuje v režimu HTTPS s vypnutým ověřováním certifikátu (*verify=False*), protože ABB ovladače zpravidla používají self-signed certifikáty. V produkčním nasazení by měl být tento parametr vypnut nebo nahrazen důvěryhodnou

certifikační autoritou, aby se zamezilo útokům. Odeslané body trajektorie pak může RAPID procedura převzít a převést na konkrétní robotické pohyby.

13.2 RAPID

V RAPID kódu se definuje modul *MainModule*, v němž jsou deklarovány datové struktury sloužící jako cílové kontejnery pro body odeslané z Python skriptu. Nejprve se zavádí datová definice *line_data*. Ta obsahuje pět skalárních položek typu *num*. souřadnice počátečního bodu $x1$, $y1$, souřadnice koncového bodu $x2$, $y2$ a atribut *angle*, který uchovává orientaci segmentu v radiánech. Touto strukturou se v RAPIDu reprezentuje každý přímkový úsek přesně v tom formátu, jaký Python skript generuje do pole *full_array_points*.

Následuje záznam *curve_data*, jenž slouží pro body křivek. Obsahuje souřadnice x , y , příznak *end*, který indikuje poslední bod konkrétního segmentu (0 = mezilehlý, 1 = konečný), a *angle*, jenž vyjadřuje směr k následujícímu bodu. Formát přesně odpovídá podobě, v níž jsou body ukládány do pole *full_array_curves*.

Pro trvalé uchování dat v řídicím systému jsou obě struktury persistentní pole *PERS*. Řádek *PERS line_data arrLine{1000};* rezervuje statické pole *arrLine* o tisíci prvcích typu *line_data*. Analogicky *PERS curve_data arrCurve{1000};* alokuje pole *arrCurve* pro křivkové body. Pole *PERS* zajišťuje, že hodnoty v poli přetrvávají i po stopnutí programu nebo restartu řadiče, takže trajektorie zůstanou k dispozici, dokud nejsou přepsány novým *POST* dotazem.

Tato datová rozhraní tvoří most mezi předzpracováním a řídicím cyklem robota. RAPID program může nyní iterovat skrze *arrLine* a *arrCurve*, transformovat každý záznam na odpovídající instrukci, a tím vykreslit importovanou geometrii v simulovaném prostoru.

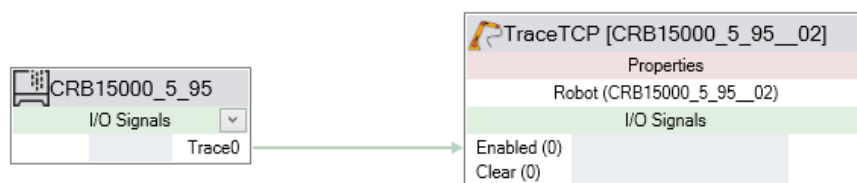
14 VYKRESLENÍ

V proceduře *main()* se inicializuje stav robotického systému a následně se spouštějí dvě dílčí podprogramy, jež vykonávají vlastní řezné pohyby podle dříve načtených dat. Nejprve se příkazem *SetDO doCuttingCurve, 0;* vypíná digitální výstup *doCuttingCurve* – tím je zajištěno, že příznak pro zobrazování právě vykonávané operace na displeji ovladače, odpovídá skutečnému stavu. Stejným způsobem se okamžitě poté nuluje výstup *doCuttingLine* prostřednictvím příkazu *SetDO doCuttingLine, 0;*, čímž se zabezpečuje identická výchozí logika.

Po inicializaci I/O signálů se volá procedura *pohyb_Curve*, v níž jsou iterovány záznamy z persistentního pole *arrCurve* a generovány příslušné trajektorie. Jakmile je celý křivkový úsek zpracován, přechází se příkazem *pohyb_Line* k obdobné smyčce nad polem *arrLine*, která převádí každou dvojici koncových bodů na instrukci *MoveL*.

Tím je zajištěno, že robot nejprve vyřeže všechny křivky a poté úsečky. Pořadí lze v případě potřeby jednoduše změnit prohozením volání obou procedur.

Samotná vizuální simulace dráhy, kterou špička nože projede zajišťuje funkce ABB RobotStudio „*TraceTCP*“. Sama o sobě tato funkce vykresluje projetou dráhu po celou dobu trvání programu, proto je za potřebí toto ošetřit, aby vykreslování probíhalo pouze když uvažujeme, že nůž vyřezává do materiálu. Toto umožňuje simulační logika, kde pomocí ovládacího signálu, kterého hodnotu měníme v programu robota ovládáme, v kterých okamžicích je vykreslení aktivní a v kterých není.



Obrázek 16 Bloková logika vykreslování drah

14.1 Kreslení křivek

V proceduře *pohyb_Curve()* je nejprve deklarována lokální proměnná *angle*, indexová proměnná *i* a proměnná *curJ* typu *jointtarget*, která slouží k uložení okamžité polohy kloubů. Při nastavení *CornerPathWarning FALSE* se zaznamenávání této události vypíná, takže případný *corner path failure* se v řídicím protokolu nevyskytuje a nepřerušuje chod

programu. Instrukce *ConfL\Off* vypíná ověření, že vykonávané instrukce dodržují přednastavenou konfiguraci os.

Pro první bod křivky se ze struktury *arrCurve{1}* načítá směr v radiánech, který je převáděn na stupně, negován a uložen do *angle*. Souřadnice x a y se přiřazují do globálního cíle *mytarget*. Následuje lineární pohyb *MoveL reltool*, který využívá relativní přepočtení k nástrojovému souřadnému systému. Translace v osy z je nulová, zatímco rotace *Rz* je nastavena na vypočtený *angle*. Takto je proveden výjezd z počáteční polohy robota do prvního bodu. Po dokončení se digitální výstup *Trace0* nastaví na 1, čímž začne vykreslování.

Hlavní smyčka *FOR i FROM 2 TO Dim(arrCurve,1)-1* postupně zpracovává všechny zbývající body v poli. Pokud proměnná *end* v aktuální položce *arrCurve{i}* nese zápornou hodnotu, je detekován neplatný záznam, smyčka je okamžitě přerušena, jelikož tato hodnota indikuje konec platných dat. Pro každý platný bod jsou souřadnice uloženy do *mytarget*. Jestliže příznak *end* nabývá hodnoty 1 signalizuje konec křivky, provádí se nejprve odsazení nástroje o 20 mm v negativním směru osy z.

U mezi bodů (tedy u těch kde *end* má hodnotu 0) se sleduje změna orientace. Pokud rozdíl mezi aktuálním a předchozím úhlem přesáhne 15° , provádí se přerušování dráhy. Vykreslení je vypnuto, robot vyjede 20 mm nad rovinu řezu, a teprve poté pokračuje k dalšímu bodu. Tím se předchází poškození materiálu při ostrých lomových úhlech. V běžném případě se bod projede jedním nepřerušným pohybem *MoveL* po rovině řezu.

Po každém kroku je do *curJ* načten aktuální stav kloubů, kdy řídicí logika testuje, zda se šestá osa nedostává mimo interval $(-180^\circ, 180^\circ)$. Pokud dojde k přetočení, mohlo by se stát, že 6. osa robota by se dostala na limitní úhel svého natočení, což by vedlo k přerušování programu. Proto je vykreslování vypnuto, robot přesune nástroj 20 mm nad rovinu, provede se přetočení osy 6 na hraniční hodnotu ($\pm 179^\circ$) a nástroj je vrácen na bezpečnou neutrální orientaci *Rz* = 0° . Teprve poté se dráha resetuje na správný *angle* pro pokračování řezu.

Před přesunem na následující bod *arrCurve{i+1}* se *angle* přepočítá a do *mytarget* se nahrají cílové souřadnice příštího bodu. Pokud aktuální bod není koncovým. Vykreslování se z novu aktivuje. Pohyb k dalšímu bodu se opět provádí příkazem *MoveL* po řezné rovině.

Narazí-li řídicí smyčka na bod s *end* = 1, vykreslování je vypnuto, robot vyjede 20 mm nad rovinu a opět je zkontrolován úhel šesté osy. V případě jejího překročení se aplikuje stejná kompenzační sekvence rolování jako výše. Po návratu na neutrální orientaci je pohyb

k poslednímu odskočnému bodu dokončen a procedura pokračuje dalším záznamem, dokud nejsou všechny body z *arrCurve* vyčerpány nebo není splněna podmínka *end < 0*.

Celá dráhová logika je navržena tak, aby se udržela plynulost řezu v mezích technologických limitů nástroje a zároveň aby se předešlo singularitám či přetočení osy 6.

14.2 Vykreslení úsečky

Proceduře *pohyb_Line()* se nejprve deklarují lokální proměnné *angle* a *i*. Digitální výstup *Trace0* se nastavuje na 0, aby bylo vykreslování při startu deaktivováno, a příkazem *ConfL\Off* se ruší omezení na konkrétní konfiguraci kloubů, takže plánovač dráhy může volit libovolný. Následně se zapíná výstup *doCuttingLine* tím se indikuje aktivita procedury pro výstup na ovládacím panelu.

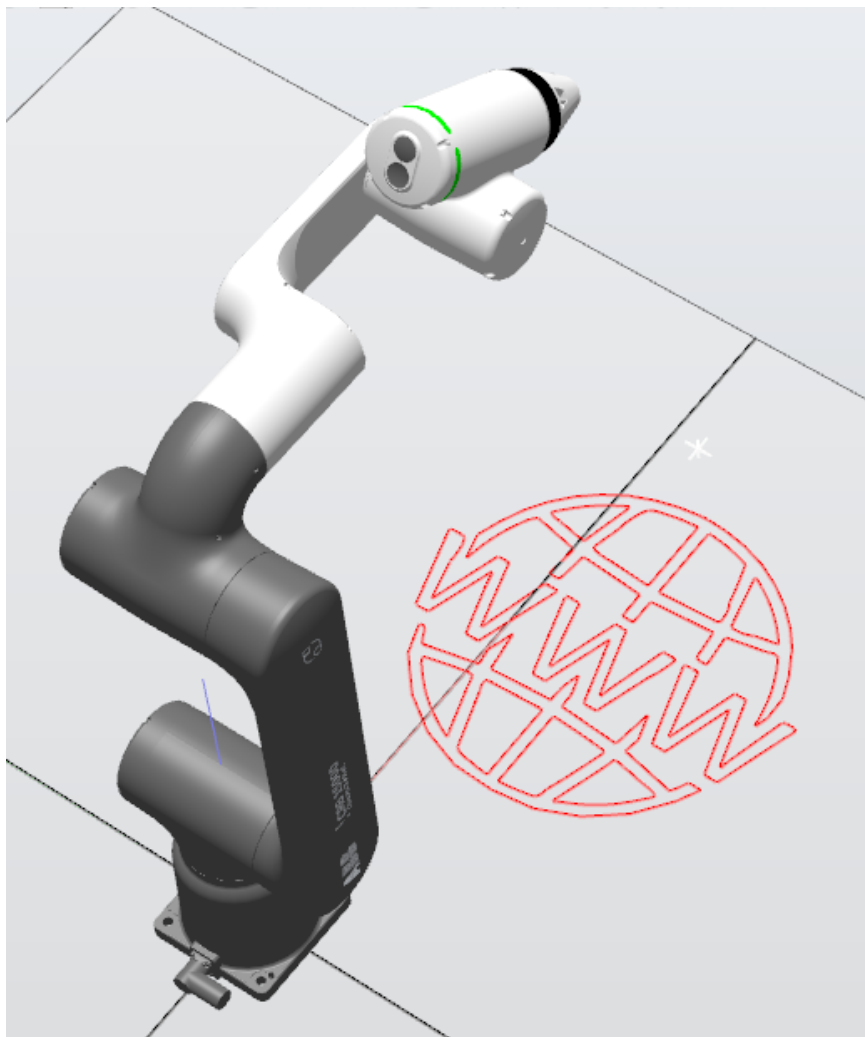
Hlavní smyčka *FOR i FROM 1 TO Dim(arrLine, 1)* postupně prochází celé pole *arrLine*. Každý záznam je typu *line_data* a nese souřadnice počátku a konce úseku i orientaci segmentu *angle* uloženou v radiánech. Pokud se v položce *x1* objeví hodnota menší než 0, je to uvažováno jako signalizace konec platných dat a smyčka se přeruší příkazem *BREAK*.

Pro každý platný záznam je úhel převeden z radiánů na stupně a negován. Negace je nutná, protože nástroj *tKnife* má otočenou osu z dolů směrem k podložce neboli směr osy z nástroje je obrácený oproti souřadnému systému „světa“.

Dále jsou do globální proměnné *mytarget* zapsány souřadnice počátečního bodu. Robot nejprve sjede 20 mm nad rovinu řezu, čímž vytvoří nájezdový odstup. Druhým příkazem *MoveL* se nástroj přiblíží k rovině řezu. Po potvrzení, že je robot ve v pozici pomocí příkazu *WaitRobInPos*, se zapíná *Trace0*, takže vykreslení je aktivní pouze mezi počátečním a koncovým bodem úseku.

Po nastavení koncových souřadnic se provádí lineární pohyb po rovině řezu na konec segmentu. Jakmile robot dorazí, vykreslení se vypíná a následuje dvojice výjezdových příkazů, kdy první vyjede 20 mm vzhůru s původní orientací nástroje, druhý pak ještě v stejné výšce stáčí nástroj na neutrální $R_z = 0$, aby byla osa 6 připravena na libovolnou orientaci při následujícím startu.

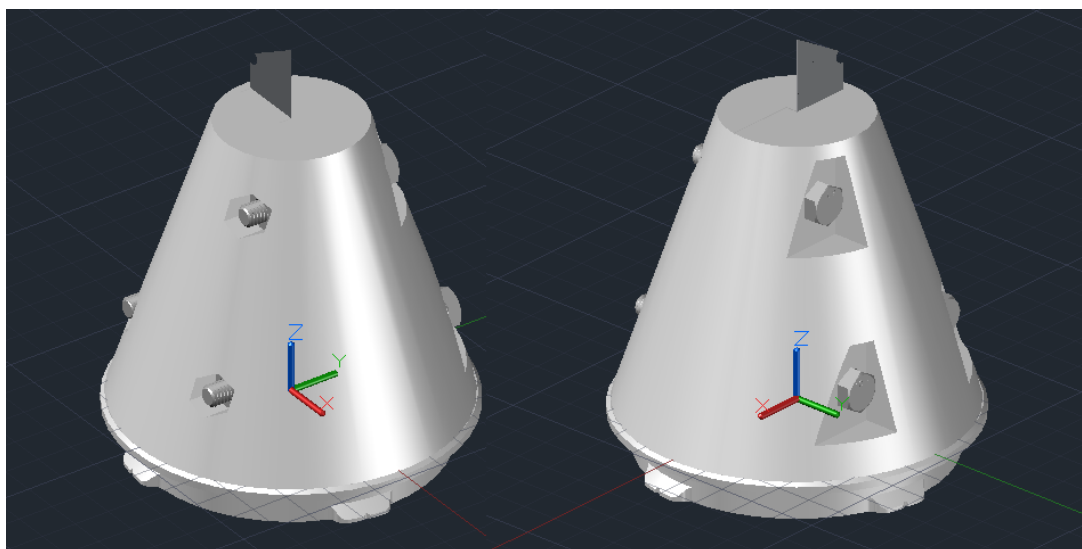
Celá procedura tak zajišťuje, že každý lineární segment je vyřezán jako samostatná trajektorie s bezpečným nájezdem a přesným odjezdem.



Obrázek 17 Obrázek vykreslený robotem

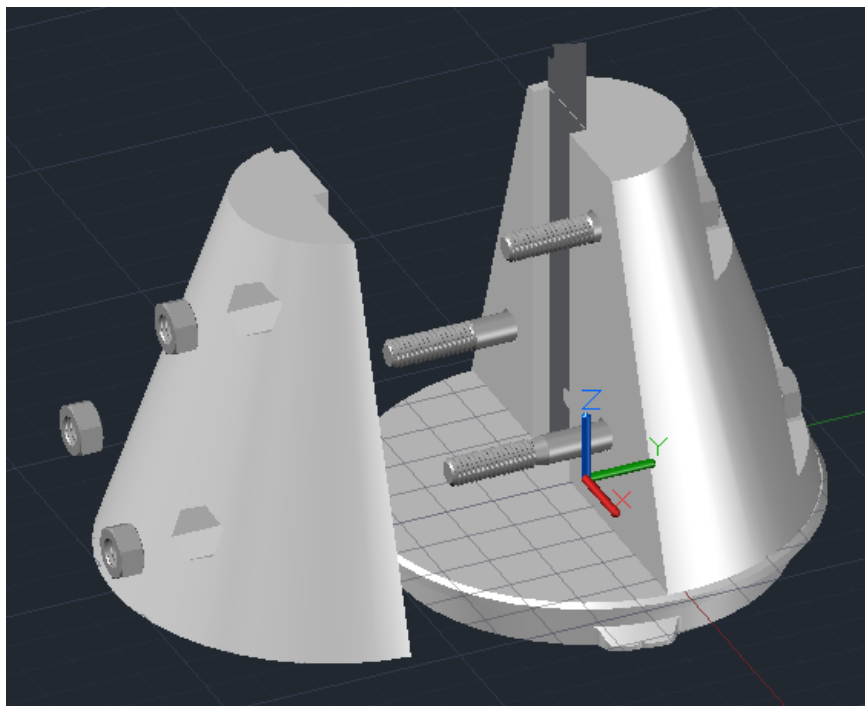
15 NÁVRH UCHYCENÍ

Držák čepele je koncipován jako kužel o poloměru základny 46,5 mm a výšce 80 mm. Plášť svírá s podstavou úhel 59,8 °. Těleso je rozděleno na dvě poloviny tak, aby se uprostřed vytvořil pravoúhlý zářez, do něhož se zasouvá čepel univerzálního nože (model čepele převzat z https://grabcad.com/library/utility-knife/details?folder_id=128911). Čepel je uložena v délce 80 mm uvnitř kuželu, zatímco 20 mm vyčnívá, což zajišťuje jednak dostatečnou tuhost upnutí, jednak potřebnou délku řezu bez nadměrného vyložení nástroje.



Obrázek 18 Držák nože

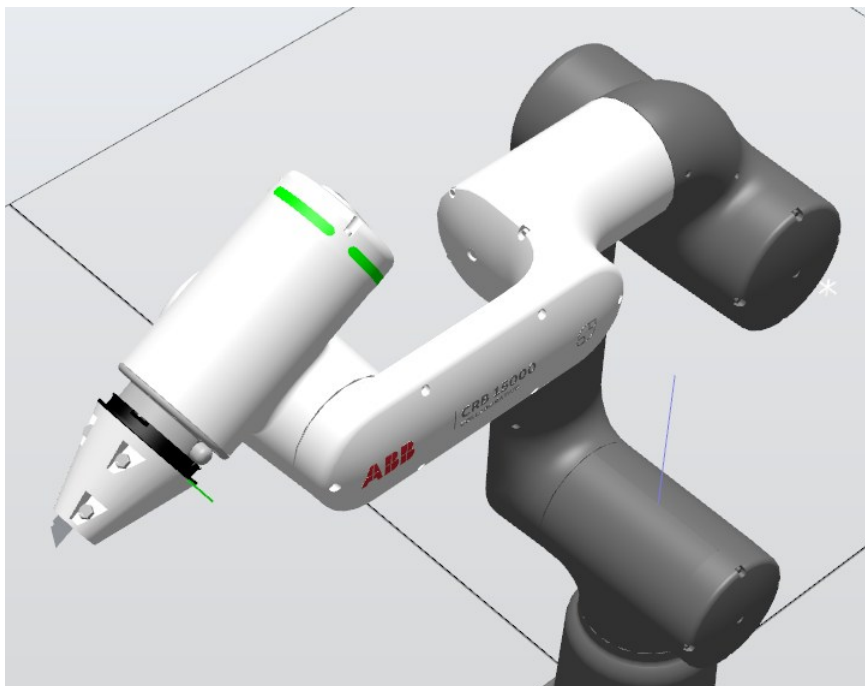
Obě poloviny kuželu jsou k sobě přitahovány trojicí šroubů ISO 4014 různých délek – $M6 \times 80$, $M6 \times 50$ a $M6 \times 40$ mm – aby se utahovací síla přenesla do míst s odlišnou tloušťkou stěny kužele. Na protilehlé straně jsou zhotoveny zápusťné otvory pro šestihranné matice, které se po montáži zapustí do tělesa a nevystupují nad plášť. Tím se zachovává kompaktní vnější tvar a snižuje se riziko kolize s okolními objekty. Rozteč šroubů odpovídá nepravidelnému tvaru kuželu. Každý spoj je dimenzován tak, aby po dotažení vytvořil rovnoměrný úchyt na celé ploše čepele.



Obrázek 19 Ukázka systému uchycení

Kužel se k robotu připojuje přes samostatný adaptér, který zajišťuje kompatibilitu s montážním vzorem příruby. Vzdálenost špičky čepele od montážní plochy činí 113 mm. Předpokládané řezné síly jsou nízké, protože se zpracovávají měkké materiály (např. karton nebo pěnový polyetylen), pevnostní rezervu tak zajišťuje už samotná přídržná délka 80 mm a rovnoměrné sevření šrouby M6.

Výměna čepele se provádí rozepnutím trojice šroubů a odklopením jedné poloviny kuželu. Celý úkon vyžaduje přerušení provozu a dá se provést během 2–3 minut mimo robota, čímž odpadá riziko poranění obsluhy. Převzaté 3D modely šroubů a matic ze standardizovaného internetového katalogu garantují, že držák je konstruován pro běžné spojovací prvky a nevyžaduje speciální díly (modely šroubů a matic převzaty z <https://www.mcmaster.com/>). Celkově je tak navrženo tuhé, snadno servisovatelné a kompatibilní upnutí lámacího nože pro robotické řezání.



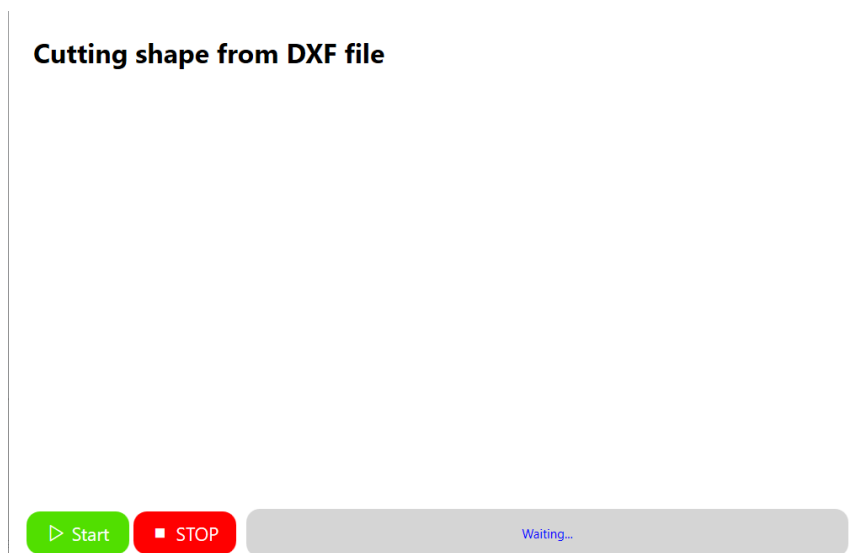
Obrázek 20 Držák přichycený na virtuálního robota

16 NÁVRH OVLÁDACÍHO PANELU

Ovládací panel byl vytvořen v prostředí ABB AppStudio s využitím výhradně předinstalovaných grafických objektů. Rozhraní bylo navrženo jako jednoduchá webová aplikace složená ze dvou akčních tlačítek a stavového pruhu. Tlačítko Start bylo instancováno z komponenty *ActionButton*, jeho text byl změněn na „Start“, barva byla přenastavena na zelený odstín. Do obsluhy události *On Click* byl vložen příkaz na start programu.

Tlačítko „Stop“ bylo rovněž převzato z knihovny Standard Controls a bylo přebarveno na odstín červené. V jeho obsluze *On Click* byl zaveden jediný příkaz, a to stop programu.

Ve spodní části obrazovky byl umístěn šedý stavový pruh realizovaný komponentou text. Výchozí text „Waiting...“ byl zobrazen modrou barvou. AppStudio následně provádělo automatický dotaz na digitální výstupy */rw/iosystem/signals/doCuttingCurve* a */rw/iosystem/signals/doCuttingLine*. Pokud oba signály zůstávaly nulové, zůstával i text „Waiting...“. Při aktivaci *doCuttingCurve* se text přepisoval na „Cutting curves...“, při aktivaci *doCuttingLine* na „Cutting lines...“. Tímto způsobem bylo zajištěno, že panel poskytoval jednoznačnou obsluhu spuštění a zastavení programu i okamžitou vizuální zpětnou vazbu o právě probíhající operaci robota.



Obrázek 21 Ovládací obrazovka

ZÁVĚR

Tato práce si kladla za cíl prozkoumat a prakticky demonstrovat kompletní workflow, který převádí dvourozměrná CAD data do bezpečného, spolehlivého a uživatelsky přívětivého procesu robotického řezání. Úvodní přehled vývoje průmyslové robotiky od renesančních automatů až po dnešní kolaborativní systémy ukázal, jak rychle se roboty staly nepostradatelnou součástí moderní výroby. Historický exkurz zároveň zdůraznil, že klíčem k úspěchu není jen mechanická síla stroje, ale především kvalita řídicích algoritmů, bezpečnostní filozofie a ergonomická integrace do pracovního prostředí.

Praktická část propojila tuto teoretickou základnu s konkrétním softwarovým řešením. Zvolená kombinace knihoven ezdxf, NumPy/SciPy, OpenCV a Matplotlib umožnila automaticky načítat DXF výkresy, segmentovat je na úsečky a křivky, interpolovat řídké body pomocí PCHIP a vygenerovat homogenní datové pakety pro ABB Robot Web Services. RAPID skripty pak tato data proměnily v trajektorie s adaptivním přerušováním dráhy, ochranou proti přetočení šesté osy a jednoznačnou signalizací pro vizuální trace. Celý řetězec je navíc řízen přes jednoduchý webový panel vytvořený v AppStudio, takže obsluha zvládne spustit i zastavit úlohu bez znalosti vnitřní struktury kódu.

Velký důraz byl kladen také na mechanickou stránku: držák nože s přesným sevřením čepele a adaptérem na přírubu robota zaručuje tuhost řezu při zachování minimální hmotnosti. Díky fixní vzdálenosti hrotu od příruby lze navíc snadno simulovat a validovat kolizní stavy v RobotStudios. Spojením robustního hardware a flexibilního softwaru tak vznikl ucelený systém, který prokazatelně převádí digitální návrh do fyzického řezu.

Přesto zůstává prostor pro další rozvoj. V budoucnu lze rozšířit algoritmus o adaptivní hustotu vzorkování podle lokálního zakřivení, doplnit online korekce pomocí kamerového systému či vibračních senzorů a implementovat plnohodnotný dashboard pro sledování OEE a prediktivní údržby. Mechanický modul by bylo možné dovybavit aktivními tlumiči rázů a rychlovýměnným rozhraním čepele. Tyto kroky by nejen zvýšily produktivitu, ale posunuly systém směrem k principům Průmyslu 4.0.

SEZNAM POUŽITÉ LITERATURY

- [1] TURNEY, Drew. *The history of industrial robots, from single taskmaster to self-teacher*. Online. AUTODESK. 2018. Dostupné z: <https://www.autodesk.com/design-make/articles/history-of-industrial-robots>. [cit. 2024-11-10].
- [2] WALLEN, Johanna. The History of the Industrial Robot. Online. In: . Linköping, 2008, s. 18. ISSN 1400-3902. Dostupné z: <https://www.semanticscholar.org/paper/The-History-of-the-Industrial-Robot-Wallen/b658782dd9c075d768bd772a271eaf639a0ba5b8#related-papers>. [cit. 2024-11-10].
- [3] ÚŘAD PRO TECHNICKOU NORMALIZACI, METROLOGII A STÁTNÍ ZKUŠEBNICTVÍ. ČSN EN ISO 12100, *Bezpečnost strojních zařízení - Všeobecné zásady pro konstrukci - Posouzení rizika a snižování rizika*. 2010.
- [4] DIETRICH, Shawn. *Introduction to ABB Robot Programming Language*. Online. CONTROL.COM. 2022. Dostupné z: <https://control.com/technical-articles/introduction-to-abb-robot-programming-language/>. [cit. 2024-11-13].
- [5] *Technical reference manual RAPID Instructions, Functions and Data types*. Online. Revize: Z. ABB, 2024. Dostupné z: <https://search.abb.com/library/Download.aspx?DocumentID=3HAC050917-001&LanguageCode=en&DocumentPartId=&Action=Launch>. [cit. 2024-11-14].
- [6] KOLÍBAL, Zdeněk. *Roboty a robotizované výrobní technologie*. Brno: Vysoké učení technické v Brně - nakladatelství VUTUIM, 2016. ISBN 978-80-214-4828-5.
- [7] INETA; VILUMSONE-NEMES a INETA. Automated cutting of textile material. Online. In: *Industrial Cutting of Textile Materials*. 2nd Edition. 2018, s. 36. ISBN 978-0-0810-2123-1. Dostupné z: <https://app.knovel.com/hotlink/pdf/id:kt011Q3XV5/industrial-cutting-textile/oscillating-knife>. [cit. 2025-05-09].
- [8] XU, Wei-Xing a ZHANG, Liang-Chi. Ultrasonic vibration-assisted machining: principle, design and application. Online. *Advances in Manufacturing*. 2015, roč. 3, č. 3, s. 173-192. ISSN 2095-3127. Dostupné z: <https://doi.org/10.1007/s40436-015-0115-4>. [cit. 2025-05-20].
- [9] MA, Chunxiang; SHAMOTO, E.; MORIWAKI, T. a WANG, Lijiang. Study of machining accuracy in ultrasonic elliptical vibration cutting. Online. *International Journal of Machine Tools and Manufacture*. 2004, roč. 44, č. 12-13, s. 1305-1310. ISSN 08906955. Dostupné z: <https://doi.org/10.1016/j.ijmachtools.2004.04.014>. [cit. 2025-05-20].
- [10] SHAMOTO, Eiji a MORIWAKI, Toshimichi. Study on Elliptical Vibration Cutting. Online. *CIRP Annals*. 1994, roč. 43, č. 1, s. 35-38. ISSN 00078506. Dostupné z: [https://doi.org/10.1016/S0007-8506\(07\)62158-1](https://doi.org/10.1016/S0007-8506(07)62158-1). [cit. 2025-05-20].
- [11] SHAMOTO, Eiji; MORIMOTO, Yoshiyuki a MORIWAKI, Toshimichi. Elliptical Vibration Cutting (1st Report). Online. *Journal of the Japan Society for Precision Engineering*. 1996, roč. 62, č. 8, s. 1127-1131. ISSN 0912-0289. Dostupné z: <https://doi.org/10.2493/jjspe.62.1127>. [cit. 2025-05-20].
- [12] MORIWAKI, Toshimichi; SHAMOTO, Eiji a INOUE, Kenji. Ultraprecision Ductile Cutting of Glass by Applying Ultrasonic Vibration. Online. *CIRP Annals*. 1992, roč.

- 41, č. 1, s. 141-144. ISSN 00078506. Dostupné z: [https://doi.org/10.1016/S0007-8506\(07\)61171-8](https://doi.org/10.1016/S0007-8506(07)61171-8). [cit. 2025-05-20].
- [13] 4.4. *Piecewise Polynomial Approximating Functions: Splines and Hermite Cubics*. Online. COLLEGE of CHARLESTON. Dostupné z: <https://lemesurierb.people.charleston.edu/introduction-to-numerical-methods-and-analysis-julia/docs/piecewise-polynomial-approximation-and-splines.html>. [cit. 2025-05-21].
- [14] KINCAID, David a CHENEY, Ward. *Numerical Analysis Mathematics of Scientific Computing*. Pacific Grove: Brooks/Cole Publishing Company, 1991. ISBN 0-534-13014-3.
- [15] ROSSI, Cesare a SAVINO, Sergio. Robot trajectory planning by assigning positions and tangential velocities. Online. *Robotics and Computer-Integrated Manufacturing*. 2013, roč. 29, č. 1, s. 139-156. ISSN 07365845. Dostupné z: <https://doi.org/10.1016/j.rcim.2012.04.003>. [cit. 2024-11-14].
- [16] DJUKANOVIC, Milena; GRUJICIC, Rade; RADUNOVIC, Luka a BOSKOVIC, Vuk. Conceptual Solution of the Robotic Arm/Plotter. Online. In: KARABEGOVIĆ, Isak (ed.). *New Technologies, Development and Application*. Lecture Notes in Networks and Systems. Cham: Springer International Publishing, 2019, s. 170-179. ISBN 978-3-319-90892-2. Dostupné z: https://doi.org/10.1007/978-3-319-90893-9_21. [cit. 2025-05-21].
- [17] VILLANUEVA, Emanuel Martinez; MAMLEDESAI, Harshavardhan; MARTINEZ, Pablo; POOSTCHI, Peyman a AHMAD, Rafiq. Design and simulation of an automated robotic machining cell for cross-laminated timber panels. Online. *Procedia CIRP*. 2021, roč. 100, s. 175-180. ISSN 22128271. Dostupné z: <https://doi.org/10.1016/j.procir.2021.05.026>. [cit. 2025-05-21].
- [18] AYARI, Oussama; BOUALI, Anis a MÉAUSOONE, Pierre-Jean. Cutting forces and accuracy characterization during wood machining with serial robots. Online. *European Journal of Wood and Wood Products*. 2020, roč. 78, č. 4, s. 767-775. ISSN 0018-3768. Dostupné z: <https://doi.org/10.1007/s00107-020-01539-4>. [cit. 2025-05-21].
- [19] BROQUERE, X.; SIDOBRE, D. a HERRERA-AGUILAR, I. Soft motion trajectory planner for service manipulator robot. Online. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2008, s. 2808-2813. ISBN 978-1-4244-2057-5. Dostupné z: <https://doi.org/10.1109/IROS.2008.4650608>. [cit. 2024-11-14].
- [20] BROQUÈRE, Xavier; SIDOBRE, Daniel a KHOI NGUYEN. From motion planning to trajectory control with bounded jerk for service manipulator robots. Online. *2010 IEEE International Conference on Robotics and Automation*. 2010, s. 4505-4510. ISBN 978-1-4244-5038-1. Dostupné z: <https://doi.org/10.1109/ROBOT.2010.5509152>. [cit. 2024-11-14].
- [21] FU, K.S.; GONZALEZ, R.C. a LEE, C.S.G. *ROBOTICS: CONTROL, SENSING, VISION, AND INTELLIGENCE*. McGraw-Hill Book Co., 1987. ISBN 0-07-100421-1.
- [22] RATHMILL, K. (ed.). *Robotic assembly*. International Trends in Manufacturing Technology. Kempston: IFS, 1985. ISBN 0-903608-71-5.
- [23] *Ezdx*. Online. Dostupné z: <https://ezdx.readthedocs.io/en/stable/index.html>. [cit. 2025-01-18].
- [24] *OpenCV*. Online. Dostupné z: <https://opencv.org/about/>. [cit. 2025-01-19].

- [25] *Python NumPy*. Online. GeeksfoGeeks. Last Updated : 26 Mar, 2025. Dostupné z: <https://www.geeksforgeeks.org/python-numpy/>. [cit. 2025-03-10].
- [26] *Introduction to Matplotlib*. Online. GeeksforGeeks. Last Updated : 18 Feb, 2025. Dostupné z: <https://www.geeksforgeeks.org/python-introduction-matplotlib/>. [cit. 2025-02-19].
- [27] *SciPy*. Online. May 08, 2025. Dostupné z: <https://docs.scipy.org/doc/scipy/>. [cit. 2025-05-10].
- [28] *Requests: HTTP for Humans™*. Online. Dostupné z: <https://docs.python-requests.org/en/latest/index.html>. [cit. 2025-04-20].
- [29] *Urllib3*. Online. Dostupné z: <https://urllib3.readthedocs.io/en/stable/>. [cit. 2025-04-20].
- [30] *1937 – The Robot Gargantua – “Bill” Griffith P. Taylor – (Australian/Canadian)*. Online. In: Cyberneticzoo.com. 2013. Dostupné z: <https://cyberneticzoo.com/robots/1937-the-robot-gargantua-bill-griffith-p-taylor-australiancanadian/>. [cit. 2025-05-09].
- [31] *Mobie Robot, developed at Stanford*. Online. In: Robots and their Arms. Dostupné z: http://infolab.stanford.edu/pub/voy/museum/pictures/display/robots/IMG_2404ArmFrontPeekingOut.JPG. [cit. 2025-05-09].
- [32] INDRE, C I; BLAGA, F S a MICULAŞ, C. Research on designing a manufacturing cell for the Disk Limiter. Online. *ANNALS OF THE ORADEA UNIVERSITY. Fascicle of Management and Technological Engineering*. 2019, roč. XXIX (XIX), 2019/3, č. 3, s. 7. ISSN 15830691. Dostupné z: <https://doi.org/10.15660/AUOFMTE.2019-3.3477>. [cit. 2024-12-10].
- [33] *Oscillating Tangential Motorised Cutting Knife Unit*. Online. In: Prototools. Dostupné z: <https://www.prototools.co.uk/oscillating-tangential-cutting-knife.html>. [cit. 2024-12-10].
- [34] Online. In: ClipArtBest.com. Dostupné z: <https://www.clipartbest.com/clipart-xTg66eRBc>. [cit. 2024-11-26].

SEZNAM OBRÁZKŮ

Obrázek 1 Robotický jeřáb Gargantua [30]	14
Obrázek 2 Stanford Arm [31]	15
Obrázek 3 Ukázka programovacího jazyka RAPID v prostředí ABB RobotStudio [32]	23
Obrázek 4 Elektrický oscilační nůž [33]	26
Obrázek 5 Ilustrační schéma EVA [8]	28
Obrázek 6 Metody ultrazvukového řezání [8]	29
Obrázek 7 Pohybové zákony pro přesun z bodu do bodu [15]	34
Obrázek 8 Profil rychlosti v pohybu sekvencí bodů [15]	35
Obrázek 9 Prolnuté rychlosti [15]	37
Obrázek 10 Křivka s body a tangentami [15]	38
Obrázek 11 Původní obrázek [34]	46
Obrázek 12 Vykreslení načtených dat	48
Obrázek 13 Nalezené úsečky	52
Obrázek 14 Vykreslené křivky	56
Obrázek 15 Celkové vykreslení	57
Obrázek 16 Bloková logika vykreslování drah	62
Obrázek 17 Obrázek vykreslený robotem	65
Obrázek 18 Držák nože	66
Obrázek 19 Ukázka systému uchycení	67
Obrázek 20 Držák přichycený na virtuálního robota	68
Obrázek 21 Ovládací obrazovka	69

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ST	Structured text
MRC	Multirobot Control
IoT	Internet of Things
PC	Personal Computer
ČSN	Česká technická norma
EN	European Norm
PPE	Personal Protective Equipment
CAD	Computer Aided Design
3D	Trojrozměrný
DXF	Drawing Exchange Format
2D	Dvojrzměrný
API	Application Programming Interface
TLS	Transport Layer Security
CA	Certificate Authority
PCHIP	Piecewise Cubic Hermite Interpolating Polynomial
CNC	Computer Numerical Control
FEM	Finite Element Method
TCP	Transmission Control Protocol
UVA	Ultrasonic Vibration-Assisted Machining
CDVA	Cutting-Direction Vibration-Assisted
NDVA	Normal-Direction Vibration-Assisted
EVA	Elliptical Vibration-Assisted

SEZNAM PŘÍLOH

Příloha P I: Python kód

Příloha P II: Rapid kód

PŘÍLOHA P I: PYTHON KÓD

```
import ezdxf
import sys
import cv2
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Arc
from scipy import interpolate
from scipy.interpolate import PchipInterpolator

import requests
from urllib3.exceptions import InsecureRequestWarning

# FILE OPEN
#####
file = "www.dxf"
imgSize = 750

try:
    doc = ezdxf.readfile(file)
except IOError:
    print(f"File not found")
    sys.exit(1)
except ezdxf.DXFStructureError:
    print(f"Invalid or corrupt file")
    sys.exit(2)

msp = doc.modelspace()

lines = []
circles = []
arcs = []
lwpolylines = []
polylines = []

# LOADING PARAMETERS OF ENTITIES
#####
countOfEntities = 0
for entity in msp:
    countOfEntities += 1
    # If the found entity is LINE, then put all start and end points
    (both x and y) into their respective brackets in 'lines'.
    if entity.dxf.type == "LINE":
        line = []
        start_line = []
        end_line = []

        start_line.append(entity.dxf.start[0])
        start_line.append(entity.dxf.start[1])
        end_line.append(entity.dxf.end[0])
        end_line.append(entity.dxf.end[1])

        line.append(start_line)
        line.append(end_line)
        lines.append(line)

    # If the found entity is CIRCLE, then put all centre points
    (both x and y) and radius into their respective brackets in
    'circles'.
    elif entity.dxf.type == "CIRCLE":
```

```

        circle = []
        circle.append(entity.dxf.center[0])
        circle.append(entity.dxf.center[1])
        circle.append(entity.dxf.radius)
        circles.append(circle)

    # If the found entity is ARC, then put all centre points (both x
    and y) and start an angle with an end angle into their respective
    brackets in 'arcs'.
    elif entity.dxf.type() == "ARC":
        arc = []
        arc.append(entity.dxf.center[0])
        arc.append(entity.dxf.center[1])
        arc.append(entity.dxf.radius)
        arc.append(entity.dxf.start_angle)
        arc.append(entity.dxf.end_angle)
        arcs.append(arc)

    # If the found entity is LWPOLYLINE, then put all centre points
    (both x and y) into their respective brackets in 'lwpolylines'.
    elif entity.dxf.type() == "LWPOLYLINE":
        lwpolyline = []
        lwpolyline.append(entity.get_points())
        for i in range(len(lwpolyline[0])):
            center = []
            center.append(lwpolyline[0][i][0])
            center.append(lwpolyline[0][i][1])
            lwpolylines.append(center)

    # If the found entity is POLYLINE, then put all centre points
    (both x and y) into their respective brackets in 'polylines'.
    elif entity.dxf.type() == "POLYLINE":
        polyline = []
        points = [(p.x, p.y) for p in entity.points()]
        polylines.append(points)

    # If the found entity is SPLINE, then put all centre points
    (both x and y) into their respective brackets in 'lwpolylines'.
    elif entity.dxf.type() == "SPLINE":
        splines = []
        spline = []
        for i in range(0, entity.dxf.degree + 1):
            spline.append(int(entity.control_points[i][0]))
            spline.append(int(entity.control_points[i][1]))
            splines.append(spline)
        spline = []

    splines = np.array(splines)

img = np.zeros((imgSize, imgSize))

# DRAWING PICTURE LOADED
#####
# Draw line with use of two points.
for i in range(len(lines)):
    cv2.line(img, (int(lines[i][0][0]), int(lines[i][0][1])),
              (int(lines[i][1][0]), int(lines[i][1][1])),
              (255, 255, 255), 1)

# Draw a circle with centre coordinates and radius.
for i in range(len(circles)):

```

```

        cv2.circle(img, (int(circles[i][0]), int(circles[i][1])),
int(circles[i][2]), (255, 255, 255), 1)

# Draw arc with use of linspace containing start and end angles in
deg, then calculate, then calculate coordinates of points via the
equation below.
for i in range(len(arcs)):
    theta = np.linspace(arcs[i][3], arcs[i][4], 100)
    x = arcs[i][0] + arcs[i][2] * np.cos(np.deg2rad(theta)) # x =
center_x + radius * cos(theta)
    y = arcs[i][1] + arcs[i][2] * np.sin(np.deg2rad(theta)) # y =
center_y + radius * sin(theta)
    plt.plot(x, y)

# Draw neighbours close to each other in lwpolylines
for i in range(len(lwpolylines)):
    if i < len(lwpolylines) - 1:
        cv2.line(img, (int(lwpolylines[i][0]),
int(lwpolylines[i][1])),
(int(lwpolylines[i + 1][0]), int(lwpolylines[i +
1][1])), (255, 255, 255), 1)
    else:
        cv2.line(img, (int(lwpolylines[i][0]),
int(lwpolylines[i][1])),
(int(lwpolylines[0][0]), int(lwpolylines[0][1])),
(255, 255, 255), 1)

nPolyline = 0
# Draw polylines together
for polyline in polylines:
    x, y = zip(*polyline)
    plt.scatter(x, y, 2, label=f"Segment {nPolyline}")
    nPolyline += 1

# Draw splines
if entity.dxftype() == "SPLINE":
    x = splines[:, 0]
    y = splines[:, 1]

    tck, u = interpolate.splprep([x, y], s=0)
    xnew, ynew = interpolate.splev(np.linspace(0, 1, 100), tck,
der=0)
    plt.plot(x, y, 'o', xnew, ynew)

# plt.imshow(img)
# plt.legend()
plt.axis('off')
plt.show()

# -----
def plot_arc(ax, start, end, angle):
    """Plots an arc between two points with the given angle on the
provided axes."""
    start = np.array(start)
    end = np.array(end)

    # Compute the midpoint
    mid = (start + end) / 2

    # Compute the distance between start and end points

```

```

chord_length = np.linalg.norm(end - start)

# Compute the radius using the given angle
theta_rad = np.radians(abs(angle)) / 2 # Half-angle in radians
radius = (chord_length / 2) / np.sin(theta_rad)

# Find the perpendicular bisector direction
dir_vector = end - start
perp_vector = np.array([-dir_vector[1], dir_vector[0]]) #
Perpendicular vector
perp_vector = perp_vector / np.linalg.norm(perp_vector) #
Normalize

# If the angle is negative, flip the centre to the other side
flip_direction = -1 if angle < 0 else 1
center = mid + flip_direction * perp_vector * np.sqrt(radius **
2 - (chord_length / 2) ** 2)

# Compute start and end angles
start_angle = np.degrees(np.arctan2(start[1] - center[1],
start[0] - center[0]))
end_angle = np.degrees(np.arctan2(end[1] - center[1], end[0] -
center[0]))

# Ensure correct arc drawing based on the angle direction
if angle < 0:
    start_angle, end_angle = end_angle, start_angle # Swap for
a clockwise direction

# Plot the arc on the given axes
arc = Arc(center, 2 * radius, 2 * radius, angle=0,
theta1=start_angle, theta2=end_angle, color="blue", linewidth=2)
ax.add_patch(arc)

# Plot the start, end, and centre points
ax.plot(*start, 'go', markersize=3) # Start point
ax.plot(*end, 'ro', markersize=3) # End point
# ax.plot(*center, 'bo', markersize=3, alpha=0.5) # Arc center
# ax.set_ylim(480,500)

def closest_value(lst, target):
    return min(lst, key=lambda x: abs(x - target))

def processing(groupNumber):
    choose_poly = groupNumber
    x, y = zip(*polylines[choose_poly])

    # FINDING LINES
    #####
    # Calculate angles between consecutive points
    angles = np.arctan2(np.diff(y), np.diff(x)) # Angles of lines
between consecutive points

    angle_changes = abs(np.diff(angles)) # Change in an angle
between consecutive lines
    angle_changes = np.append(angle_changes, angle_changes[-1])
    angle_threshold = 2 * np.pi / 180

    # Segment points based on angle changes
    segments = []
    current_segment = [0] # Start with the first point

```

```

for i_enum_angle, angle_change in enumerate(angle_changes):

    if angle_change > angle_threshold:
        segments.append(current_segment)
        current_segment = [i_enum_angle + 1] # Start a new
segment
    else:
        current_segment.append(i_enum_angle + 1)
        segments.append(current_segment) # Add the last segment

# If
extraSegment = []

    if (x[segments[-1][-1]] == x[-1] and y[segments[-1][-1]] == y[-
1]) and (
        x[segments[0][0]] == x[0] and y[segments[0][0]] ==
y[0]):
        extraSegment.append(segments[-1][-1])
        extraSegment.append(segments[0][0])
        segments.append(extraSegment)

    for i_segmenting, segment in enumerate(segments):
        if i_segmenting != len(segments) - 1:
            if len(segment) > 1 and len(segments[i_segmenting + 1])
> 1:
                segment.append(segments[i_segmenting + 1][0])
    x = np.array(x)
    y = np.array(y)
    # Plot the segmented lines

    '''plt.figure(figsize=(8, 6))
    for segment in segments:
        #print("segment",segment)
        plt.plot(x[segment], y[segment], marker='o', label=f"Segment
{segments.index(segment)}")
    plt.title("Segmenting Lines by Angle Changes")
    plt.scatter(x[0], y[0], s=500, color="red")
    plt.scatter(x[-1], y[-1], s=500, color="red")
    plt.legend()
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.show()'''

    # Segmented lines only as start and end point
    linePoints = []
    otherPoints = []

    # MAKING LINES ONLY AS TWO POINTS AND ADDING LINE END POINT TO
GROUP OF NON-LINE AFTER THE LINE AND BEGINNING TO GROUP OF NON-LINE
BEFORE
#####

    for i_segment_line, segment in enumerate(segments):

        # If there are more inliers than 2.
        # ONLY SAVING the FIRST AND LAST POINT OF LINE
        if len(x[segment]) > 1:
            startPointX = x[segment][0]
            endPointX = x[segment][-1]
            startPointY = y[segment][0]
            endPointY = y[segment][-1]

```

```

        linePoints.append([startPointX, startPointY, endPointX,
endPointY])

    else:
        # Check if there is a line before and after the point
        and get the start point of line after and end point of line before.
        if len(segments[i_segment_line - 1]) >= 2 and
len(segments[i_segment_line + 1]) >= 2:
            xPoint = x[segment][0]
            yPoint = y[segment][-1]
            startPointX = x[segments[i_segment_line + 1]][0]
            endPointX = x[segments[i_segment_line - 1]][-1]
            startPointY = y[segments[i_segment_line + 1]][0]
            endPointY = y[segments[i_segment_line - 1]][-1]

            if [endPointX, endPointY] == [xPoint, yPoint]:
                otherPoints.append([xPoint, yPoint])
                otherPoints.append([startPointX, startPointY])
            elif [startPointX, startPointY] == [xPoint, yPoint]:
                otherPoints.append([endPointX, endPointY])
                otherPoints.append([xPoint, yPoint])
            else:
                otherPoints.append([endPointX, endPointY])
                otherPoints.append([xPoint, yPoint])
                otherPoints.append([startPointX, startPointY])

        # Check if there is a line only before get the end point
        of the previous line.
        elif len(segments[i_segment_line - 1]) >= 2 >
len(segments[i_segment_line + 1]):
            xPoint = x[segment][0]
            yPoint = y[segment][-1]
            endPointX = x[segments[i_segment_line - 1]][-1]
            endPointY = y[segments[i_segment_line - 1]][-1]

            if [endPointX, endPointY] == [xPoint, yPoint]:
                otherPoints.append([xPoint, yPoint])
            else:
                otherPoints.append([endPointX, endPointY])
                otherPoints.append([xPoint, yPoint])

        # Check if there is a line only after get also start
        point of a future line.
        elif len(segments[i_segment_line - 1]) < 2 <= len(
            segments[i_segment_line + 1]) and i_segment_line
        != len(segments) - 1:
            xPoint = x[segment][0]
            yPoint = y[segment][-1]
            startPointX = x[segments[i_segment_line + 1]][0]
            startPointY = y[segments[i_segment_line + 1]][0]
            if [xPoint, yPoint] == [startPointX, startPointY]:
                otherPoints.append([xPoint, yPoint])
            else:
                otherPoints.append([xPoint, yPoint])
                otherPoints.append([startPointX, startPointY])

        # Point with no line before or after, add it
        else:
            xPoint = x[segment][0]
            yPoint = y[segment][-1]
            otherPoints.append([xPoint, yPoint])

```

```

groupOfNonLine = []
groupsOfNonLine = []

# CONNECTING GROUPS OF POINTS THAT DON'T FORM A LINE
#####

distances = []
for i_distances in range(0, len(otherPoints) - 1): # find
distances between consecutive points
    distances.append(np.sqrt(
        (otherPoints[i_distances][0] - otherPoints[i_distances
+ 1][0]) ** 2 + (
            otherPoints[i_distances][1] -
otherPoints[i_distances + 1][1]) ** 2))

distances = np.array(distances)
np.set_printoptions(precision=20)

#####
#####

if len(distances) != 0: # if there are some distances, make a
histogram
    distDistri, histDist, _ = plt.hist(distances,
bins=len(distances))
    plt.close()

    # plt.title("Distance Histogram")
    # plt.show()

    if len(np.where(distDistri > 0)[0]) < 3: # note where there
are outlier distances
        justDistance = np.max(distances)

    else:
        indices = np.where(distDistri == 1)[0]

        for i_max_distances in range(0, len(indices)): # find
max of indicated distances
            maxDist = np.max(histDist[indices[i_max_distances]])
            closest = closest_value(distances, maxDist)
            distances = distances[distances != closest]

        justDistance = np.max(distances)

#####
#####
# MAKE GROUPS OF NEAR POINTS IN OTHERPOINTS
foundOne = False
needSkip = False
oi = 0
for i in range(0, len(otherPoints) - 1): # go through
otherPoints and find points within a certain distance

    for j in range(0, len(linePoints)):
        if otherPoints[i] == [linePoints[j][0],
linePoints[j][1]] or otherPoints[i] == [linePoints[j][2],

```

```

linePoints[j][
3]]: # Check if currently looked at otherPoint is the same as some
linePoint

        needSkip = True
        if i == 0:
            groupOfNonLine.append(otherPoints[i])
            # foundOne = True
        elif foundOne and oi == (i - 1):
            groupOfNonLine.append(otherPoints[i])
            foundOne = False
            # if there was already a point that was also a
line point, add it to the group
        else:
            groupOfNonLine.append(otherPoints[i])
            groupsOfNonLine.append(groupOfNonLine)
            groupOfNonLine = []
            foundOne = True
            oi = i
            # if there was not already a point that would
also be a line point, add this line point to the group and end the
group

        if needSkip:
            needSkip = False
            continue

        distance = np.sqrt(
            (otherPoints[i][0] - otherPoints[i + 1][0]) ** 2 +
            (otherPoints[i][1] - otherPoints[i + 1][1]) ** 2)
        if distance <= justDistance:

            groupOfNonLine.append(otherPoints[i])
        else:
            groupOfNonLine.append(otherPoints[i])
            groupsOfNonLine.append(groupOfNonLine)
            groupOfNonLine = []
        if i == len(otherPoints) - 2:
            groupOfNonLine.append(otherPoints[i + 1])
            groupsOfNonLine.append(groupOfNonLine)

        # Sample data structure: groupsOfNonLine = [(x1, y1), (x2, y2),
        ..., [(xA, yA), (xB, yB), ...]]
        # FINDING ANGLES BETWEEN POINTS AND LINES
        #####

        groupofAnglechanges = []
        for i in range(0, len(groupsOfNonLine)): # Go through groups of
points;
            x, y = zip(*groupsOfNonLine[i])
            if len(x) < 2:
                angles = np.arctan2(y, x)
            else:
                angles = np.arctan2(np.diff(y), np.diff(x)) # Angles of
lines between consecutive points

            for j in range(0, len(linePoints)):
                angleLine = np.arctan2(linePoints[j][3] -
linePoints[j][1],
                                linePoints[j][2] -

```

```

linePoints[j][0]) # Angle of line

        # If a point of groupsOfNonLine is the same as the first
or second line point, then add a line angle after
        if (groupsOfNonLine[i][-1][0] == linePoints[j][2] and
groupsOfNonLine[i][-1][1] == linePoints[j][3]) or (
            groupsOfNonLine[i][-1][0] == linePoints[j][0]
and groupsOfNonLine[i][-1][1] == linePoints[j][
            1]) and len(groupsOfNonLine[i]) > 1:
            angles = np.insert(angles, len(angles), angleLine)

        elif (groupsOfNonLine[i][0][0] == linePoints[j][2] and
groupsOfNonLine[i][0][1] == linePoints[j][3]) or (
            groupsOfNonLine[i][0][0] == linePoints[j][0] and
groupsOfNonLine[i][0][1] == linePoints[j][
            1]) and len(
            groupsOfNonLine[i]) > 1:
            angles = np.insert(angles, 0, angleLine)

        angle_changes = np.diff(angles) # Change in an angle
between consecutive lines

        # Make sure angle_changes aren't zero
indWhereZero = np.where(angle_changes == 0)

        angle_changes[indWhereZero] += 0.01 * np.pi / 180

        if len(angle_changes) == 0:
            angle_changes = angles
        groupofAnglechanges.append(angle_changes)

    return linePoints, groupsOfNonLine, groupofAnglechanges

# TRANSFORMING TO ONE ARRAY
finLinePoints = []
finGroupsOfNonLine = []
finAngleChanges = []
sendLinePoints = []
for g in range(len(polylines)):
    grouped_data = processing(g)

    # TRANSFORMING DATA FOR SENDING
    # output -> [[angle, 0, 0, 0], [x1, y1, x2, y2]... ]

    for i in range(len(grouped_data[0])):
        angle = np.arctan2(grouped_data[0][i][1] -
grouped_data[0][i][3], grouped_data[0][i][0] -
grouped_data[0][i][2])
        sendLinePoints.append([*grouped_data[0][i], angle])

    finLinePoints.append(grouped_data[0])
    finGroupsOfNonLine.append(grouped_data[1])
    finAngleChanges.append(grouped_data[2])

# CHANGING FROM NP.ARRAY64 TO FLOAT AND ADDING TO PREDEFINED ARRAY
full_array_points = [[-1, 0, 0, 0, 0]] * 1000
float_data = [[float(value) for value in sublist] for sublist in
sendLinePoints]
print("len(float_data):", len(float_data))

# MOVING X VALUES TO BEGIN CLOSER TO 0 FOR FULL ARRAY POINTS

```

```

# find the smallest x in full_array_points
min_in_array_x = min(item[0] for item in float_data)
print("min_in_array", min_in_array_x)
for i in range(len(float_data)):
    float_data[i][0] -= min_in_array_x - 10
    float_data[i][2] -= min_in_array_x - 10
full_array_points[: len(float_data)] = float_data
print("full_array_points:", full_array_points, "\n",
len(full_array_points))

fig, ax = plt.subplots()
# PLOTTING LINES
for g in range(len(full_array_points)):
    plt.plot([full_array_points[g][0], full_array_points[g][2]],
[full_array_points[g][1], full_array_points[g][3]],
            label=f"Segment {g}")

# Plotting arc
'''for g in range(len(finGroupsOfNonLine)):
    #print("i",i)
    for h in range(len(finGroupsOfNonLine[g])):
        for l in range(len(finGroupsOfNonLine[g][h])-1):

#print(f"finAngleChanges{[j]}{[k]}",finAngleChanges[j][k])
        plot_arc(ax, finGroupsOfNonLine[g][h][l],
finGroupsOfNonLine[g][h][l+1], finAngleChanges[g][h][l] * 180 /
np.pi)'''
# plt.show()

# CURVE INTERPOLATION
def densify_pchip(pts, n_samples=200):
    # Ensure numeric array
    pts = np.asarray(pts, dtype=float)

    # Parameterize by cumulative arc-length
    deltas = np.diff(pts, axis=0)
    seg_lengths = np.hypot(deltas[:, 0], deltas[:, 1])
    t = np.concatenate([[0], np.cumsum(seg_lengths)])

    # Create dense parameter grid
    t_dense = np.linspace(0, t[-1], n_samples)

    # Build PCHIP interpolators for each coordinate
    pchip_x = PchipInterpolator(t, pts[:, 0])
    pchip_y = PchipInterpolator(t, pts[:, 1])

    # Evaluate on dense grid
    x_dense = pchip_x(t_dense)
    y_dense = pchip_y(t_dense)

    # Combine into (n_samples, 2)
    return np.vstack([x_dense, y_dense]).T

# PREPARING FOR SENDING
'''
output -> [[x, y, is last?, angle]... ]
is last? -> 0 = no, 1 = yes
'''

non_in_middle = []
nonLines = []

```

```

for i in range(len(finGroupsOfNonLine)):
    for j in range(len(finGroupsOfNonLine[i])):
        pchip = densify_pchip(np.array(finGroupsOfNonLine[i][j]),
n_samples=10)
        for l in range(len(pchip)):
            for m in range(len(pchip[l])):
                non_in_middle.append(float(pchip[l][m]))
            if l == len(pchip) - 1:
                non_in_middle.append(1)

            else:
                non_in_middle.append(0)
            nonLines.append(non_in_middle)

        non_in_middle = []
nonLines[-1].append(0)

for i in range(len(nonLines)): # Move curve points to the left
    nonLines[i][0] -= min_in_array_x - 10

for i in range(len(nonLines) - 1): # Calculate angles between
interpolated curve points and plot those points
    nonLines[i].append(float(np.arctan2(nonLines[i][1] - nonLines[i
+ 1][1], nonLines[i][0] - nonLines[i + 1][0])))
    if nonLines[i][2] == 0:
        plt.plot([nonLines[i][0], nonLines[i + 1][0]],
[nonLines[i][1], nonLines[i + 1][1]], label=f"Segment {i + 1}")

# PREPARE ARRAY FOR SENDING
full_array_curves = [[0, 0, -1, 0]] * 1000
full_array_curves[: len(nonLines)] = nonLines
print("\nfull_array_curves:", full_array_curves, "\n",
len(full_array_curves))

'''plt.xlim(350,400)
plt.ylim(300,350)'''
plt.axis("off")
plt.show()

def shape(lst): # Check dimensions of a list
    if isinstance(lst, list):
        # compute this dimension...
        current = len(lst)
        # ...and recurse on the first element for deeper dims
        return [current] + shape(lst[0])
    else:
        return []

print("shape of full_array_points", shape(full_array_points))
print("shape of full_array_curves", shape(full_array_curves))

#####
#####
send = False

if send:
    # SEND DATA TO ROBOT
    IP_RC = '192.168.125.1:443'
    IP_VC = '127.0.0.1:80'
    ip = IP_VC
    module = 'MainModule'

```

```

var_name = 'arrLine'
var_name_2 = 'arrCurve'

# disable an insecure warning message in the output terminal
requests.packages.urllib3.disable_warnings(InsecureRequestWarning)

session = requests.Session()
session.auth = ('Default User', 'robotics')
session.headers.update(
    {'Content-Type': 'application/x-www-form-urlencoded;v=2.0',
'accept': 'application/hal+json;v=2.0'})

url =
f'https://{ip}/rw/rapid/symbol/RAPID/T_ROB1/{module}/{var_name}/data
?mastership=implicit'

try:
    resp = session.post(url, data={'value':
f"{full_array_points}"}, verify=False)
    if resp.status_code == 204:
        print("DONE (full_array_points)")
    elif resp.status_code == 404:
        print(f"Variable unknown (full_array_points).")
    else:
        print(f"Status code (full_array_points):
{resp.status_code}")
except ConnectionError:
    print("Server unreachable")

url =
f'https://{ip}/rw/rapid/symbol/RAPID/T_ROB1/{module}/{var_name_2}/da
ta?mastership=implicit'

try:
    resp = session.post(url, data={'value':
f"{full_array_curves}"}, verify=False)
    if resp.status_code == 204:
        print("DONE (full_array_curves)")
    elif resp.status_code == 404:
        print(f"Variable unknown (full_array_curves).")
    else:
        print(f"Status code (full_array_curves):
{resp.status_code}")
        try:
            print(resp.json())
        except ValueError:
            print(resp.text)
except ConnectionError:
    print("Server unreachable")

```

PŘÍLOHA P II: RAPID KÓD

```
MODULE MainModule

  RECORD line_data
    num x1;
    num y1;
    num x2;
    num y2;
    num angle;
  ENDRECORD

  RECORD curve_data
    num x;
    num y;
    num end;
    num angle;
  ENDRECORD

  PERS line_data arrLine{1000};
  PERS curve_data arrCurve{1000};

  PROC main()

    SetDO doCuttingCurve,0;
    SetDO doCuttingLine,0;
    !      vypis;
    pohyb_Curve;
    pohyb_Line;

  ENDPROC

  PROC vypis()
    VAR jointtarget curJ;

    SetDO doCuttingCurve,1;
    WaitTime 1;
    SetDO doCuttingCurve,0;
    SetDO doCuttingLine,1;
    WaitTime 1;
    SetDO doCuttingLine,0;
    WaitTime 1;
```

```

TPWrite "Angle Line " \Num:=arrLine{1}.angle;
WaitTime 1;
TPWrite "Angle Curve " \Num:=arrCurve{1}.angle;
WaitTime 1;

curJ:=CJointT();
TPWrite "Angle of 6th axis " \Num:=curJ.robax.rax_6;
ENDPROC

PROC pohyb_Line()
VAR num angle;
VAR num i;
SetDO Trace0,0;
Confl\Off;
SetDO doCuttingLine,1;

FOR i FROM 1 TO Dim(arrLine,1) DO
    IF arrLine{i}.x1<0 THEN
        ! Check for end of lines
        TPWrite "Ended on x1 < 0: " \Num:=arrLine{i}.x1;
        SetDO doCuttingLine,0;
        BREAK;
    ENDIF

    angle:=arrLine{i}.angle*180/pi;
    angle:=angle*(-1);
    !protože tool má z dolů a ne nahoru vůči světu
    TPWrite "Number of line: " \Num:=i;
    TPWrite "Angle: " \Num:=angle;

    mytarget.trans.x:=arrLine{i}.x1;
    mytarget.trans.y:=arrLine{i}.y1;
    MoveL reltool(mytarget,0,0,-
20\Rz:=angle),v100,z5,tKnife\WObj:=wobjStul;
    MoveL
reltool(mytarget,0,0,0\Rz:=angle),v100,z0,tKnife\WObj:=wobjStul;
    WaitRob\InPos;
    SetDO Trace0,1;

    mytarget.trans.x:=arrLine{i}.x2;

```

```

        mytarget.trans.y:=arrLine{i}.y2;
        MoveL
reltool(mytarget,0,0,0\Rz:=angle),v100,z0,tKnife\WObj:=wobjStul;
        WaitRob\InPos;
        SetDO Trace0,0;

        MoveL
20\Rz:=angle),v100,z5,tKnife\WObj:=wobjStul; reltool(mytarget,0,0,-
        MoveL
20\Rz:=0),v100,z5,tKnife\WObj:=wobjStul; reltool(mytarget,0,0,-

ENDFOR
ENDPROC

PROC pohyb_Curve()
    VAR num angle;
    VAR num i;
    VAR jointtarget curJ;
    CornerPathWarning FALSE;
    SetDO Trace0,0;
    Confl\Off;
    SetDO doCuttingCurve,1;

    angle:=(arrCurve{1}.angle*180/pi)*(-1);
    mytarget.trans.x:=arrCurve{1}.x;
    mytarget.trans.y:=arrCurve{1}.y;
    MoveL
reltool(mytarget,0,0,0\Rz:=angle),v400,z0,tKnife\WObj:=wobjStul;
    WaitRob\InPos;
    SetDO Trace0,1;

    FOR i FROM 2 TO Dim(arrCurve,1)-1 DO
        IF arrCurve{i}.end<0 THEN
            ! Check for last point
            TPWrite "Ended on end < 0"\Num:=arrCurve{i}.end;
            SetDO doCuttingCurve,0;
            BREAK;
        ENDIF

        TPWrite "Number of curve: "\Num:=i;

        mytarget.trans.x:=arrCurve{i}.x;

```

```

mytarget.trans.y:=arrCurve{i}.y;

IF arrCurve{i}.end=1 THEN
    ! if found end of curve go up or check if the angle is
too big
    MoveL                                reltool(mytarget,0,0,-
20\Rz:=angle),v100,z5,tKnife\WObj:=wobjStul;

ELSE
    IF                                Abs(arrCurve{i-1}.angle*180/pi-
arrCurve{i}.angle*180/pi)>=15 THEN
        ! Check if the angle change isnt too big
        TPWrite                        "Abs(arrCurve{i-1}.angle*180/pi)-
Abs(arrCurve{i}.angle*180/pi):        "\Num:=Abs(arrCurve{i-
1}.angle*180/pi-arrCurve{i}.angle*180/pi);
        WaitRob\InPos;
        SetDO Trace0,0;
        WaitDO Trace0,0;
        MoveL                                reltool(mytarget,0,0,-
20\Rz:=angle),v100,z5,tKnife\WObj:=wobjStul;

    ENDIF

    MoveL
reltool(mytarget,0,0,0\Rz:=angle),v100,z0,tKnife\WObj:=wobjStul;
    WaitRob\InPos;
    SetDO Trace0,1;
    WaitDO Trace0,1;

ENDIF

curJ:=CJointT();
TPWrite      "Current      angle      of      6th      axis:
"\Num:=curJ.robax.rax_6;
IF curJ.robax.rax_6<=-180 THEN
    ! if angle of 6th axis is too high then roll back
    WaitRob\InPos;
    SetDO Trace0,0;
    MoveL                                reltool(mytarget,0,0,-20\Rz:=-
179),v100,z5,tKnife\WObj:=wobjStul;
    MoveL                                reltool(mytarget,0,0,-
20\Rz:=0),v100,z5,tKnife\WObj:=wobjStul;
    angle:=(arrCurve{i}.angle*180/pi)*(-1);

```

```

        MoveL
reltool(mytarget,0,0,0\Rz:=angle),v100,z5,tKnife\WObj:=wobjStul;
        WaitRob\InPos;

ELSEIF curJ.robax.rax_6>=180 THEN
        WaitRob\InPos;
        SetDO Trace0,0;
        MoveL
20\Rz:=179),v100,z5,tKnife\WObj:=wobjStul;
        reltool(mytarget,0,0,-
        MoveL
20\Rz:=0),v100,z5,tKnife\WObj:=wobjStul;
        reltool(mytarget,0,0,-
        angle:=(arrCurve{i}.angle*180/pi)*(-1);
        MoveL
reltool(mytarget,0,0,0\Rz:=angle),v100,z5,tKnife\WObj:=wobjStul;
        WaitRob\InPos;

ENDIF

angle:=(arrCurve{i}.angle*180/pi)*(-1);
mytarget.trans.x:=arrCurve{i+1}.x;
mytarget.trans.y:=arrCurve{i+1}.y;
WaitRob\InPos;

IF NOT arrCurve{i}.end=1 THEN
        SetDO Trace0,1;
        WaitDO Trace0,1;

ENDIF

MoveL
reltool(mytarget,0,0,0\Rz:=angle),v100,z0,tKnife\WObj:=wobjStul;
        WaitRob\InPos;

IF arrCurve{i+1}.end=1 THEN
        ! Turn off drawing if end of curve reached
        WaitRob\InPos;
        SetDO Trace0,0;
        MoveL
20\Rz:=angle),v100,z5,tKnife\WObj:=wobjStul;
        reltool(mytarget,0,0,-

        curJ:=CJointT();
        IF curJ.robax.rax_6<=-180 THEN

```

```

        ! Angle back 6th axis to prevent error
        MoveL                                reltool(mytarget,0,0,-20\Rz:=-
179),v100,z5,tKnife\WObj:=wobjStul;

        ELSEIF curJ.robax.rax_6>=180 THEN
            MoveL                                reltool(mytarget,0,0,-
20\Rz:=179),v100,z5,tKnife\WObj:=wobjStul;
            WaitRob\InPos;

        ENDIF

        MoveL                                reltool(mytarget,0,0,-
20\Rz:=0),v100,z5,tKnife\WObj:=wobjStul;
        WaitRob\InPos;

    ENDIF
ENDFOR
ENDPROC
ENDMODULE

```