

**ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE**

Fakulta strojní – Ústav mechaniky, biomechaniky a mechatroniky

**DIPLOMOVÁ PRÁCE**

**ALGORITMY  
PRO AUTOMATICKÉ LAKOVÁNÍ**

**AUTOMATED PAINTING ALGORITHMS**

Prohlašuji, že jsem tuto práci vypracoval samostatně s použitím literárních pramenů a informací, které cituji a uvádím v seznamu použité literatury a zdrojů informací. Zároveň deklaruji, že interní zdroje od společnosti ABB nebudou veřejně sdíleny.

Datum: .....

podpis

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hulínský** Jméno: **Lukáš** Osobní číslo: **420500**  
Fakulta/ústav: **Fakulta strojní**  
Zadávající katedra/ústav: **Ústav mechaniky, biomechaniky a mechatroniky**  
Studijní program: **Průmysl 4.0**  
Studijní obor: **bez oboru**

## II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

**Algoritmy pro automatické lakování**

Název diplomové práce anglicky:

**Automated painting algorithms**

Pokyny pro vypracování:

1. Seznamte se s postupy technologické přípravy a programování lakovacích robotů.
2. Popište základní publikované přístupy k návrhu trajektorie lakovacího robota a charakteristice rozstříku trysky.
3. Implementujte vybraný plánovací algoritmus na platformě Matlab
4. Pro vybranou trysku experimentálně ověřte tvar rozstříku.
5. Získaný tvar rozstříku využijte pro optimalizace stejnoměrnosti vrstvy

Seznam doporučené literatury:

1. Marshall Burns. Automated fabrication: improving productivity in manufacturing, englewood cliffs, n.j. : Ptr prentice hall, 1993.
2. Heping Chen, T. Fuhlbrigge, and Xiongzi Li. Automated industrial robot path planning for spray painting process: A review, ieee. page 523, Publikováno 2008.
3. CHEN Wei and DZHAO Dean. Path planning for spray painting robot of workpiece surfaces, school of electrical and information engineering, jiangsu universit. Hindawi Publishing Corporation,, (Article ID:659457), 2013.

Jméno a pracoviště vedoucí(ho) diplomové práce:

**Ing. Pavel Steinbauer, Ph.D., odbor mechaniky a mechatroniky FS**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **31.10.2018**

Termín odevzdání diplomové práce: **18.01.2019**

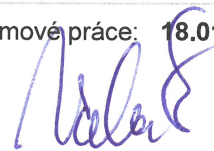
Platnost zadání diplomové práce: \_\_\_\_\_



Ing. Pavel Steinbauer, Ph.D.  
podpis vedoucí(ho) práce



prof. Ing. Milan Růžička, CSc.  
podpis vedoucí(ho) ústavu/katedry



prof. Ing. Michael Valášek, DrSc.  
podpis děkana(ky)

## III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

**12.11.2018**

Datum převzetí zadání



Podpis studenta

# Anotace

Diplomová práce se primárně zabývá vývojem algoritmu pro automatické hledání trajektorií v oblasti robotického lakování. Vytvořený postup umožňuje na základě znalosti distribuce barvy aplikátoru vyhodnotit optimální vzdálenost rozteče průjezdů robota. Dále je algoritmus schopný nalézt trajektorii na STL modelu a vygenerovat zdrojové kódy pro roboty ABB (RAPID). V diplomové práci jsou také uvedeny vybrané algoritmy z výzkumných institucí z celého světa a jsou představeny technologické postupy při hledání optimálního nastavení procesu lakování.

**Klíčová slova:** robotické lakování, plánování trajektorie, STL model, MFT, optimální trajektorie, analýza rozstříku

# Abstract

The diploma thesis deals primarily with the development of an algorithm for automatic path planning in the area of robotic painting. Based on knowledge of the paint distribution of the applicator, the created procedure enables the user to evaluate the optimal distance of the robot passes. Furthermore, the algorithm is able to find the trajectory on the STL model and generate source codes for ABB robots (RAPID). The thesis also lists selected algorithms from research institutes around the world and introduces technological procedures to find optimal settings for the painting process.

**Key words:** robotic painting, path planning, STL model, MFT, optimal path, paint gun analysis

# Poděkování

Diplomové práce se přímo, či nepřímo zúčastnilo mnoho osob, bez kterých by dosažení cíle bylo špatně dosažitelné. Především bych chtěl poděkovat panu:

**Pavlu Steinbauerovi**, který byl při diplomové práci v roli vedoucího. S panem Steinbauerem jsem na pravidelné bázi konzultoval dílčí etapy práce a snažil se nacházet řešení problémů. Panu Steinbauerovi bych také rád poděkoval za motivaci, profesionální vedení a za skvělé technické nápady.

**Jiřímu Duspivovi**, expertu přes robotické lakovací systémy, za poskytnutí objektivního pohledu na projekt. Nasměrování práce správným směrem, a také za získání vzorků lakování.

**Janu Kudláčkovi** za konzultaci ohledně měření povrchových vrstev a zapůjčení měřících zařízení.

**Petru Škrhovi** za konzultaci v problematice zpracování trajektorie v ABB Robot Studiu.

**Mayur Andulkarovi**, expertu na robotiku, za představení jeho přístupu k problému v oblasti lakování a získání cenných informací.

Velké poděkování patří také rodině, která mě během celých studií podporovala a motivovala.

# Obsah

<b>1</b>	<b>Úvod</b>	8
<b>2</b>	<b>Cíl práce</b>	9
<b>3</b>	<b>Část teoretická</b>	10
3.1	Počátky robotiky, automatizace a mechanizace	10
3.2	Počátky lakovacích robotů	12
3.3	Robotické lakovací systémy	13
3.3.1	Řízení vzduchu	14
3.3.2	Řízení barvy	14
3.3.3	Řízení vysokého napětí	17
3.3.4	Trysky (atomizéry) používané v robotice	17
3.4	Hledání optimálního nastavení procesu aplikace	20
3.4.1	Měření reálné šířky paprsku	21
3.4.2	Efektivita přenosu barvy	23
3.4.3	Vzhled vrstvy	24
3.4.4	Odhad rychlosti pohybu robota	25
3.4.5	Odhad nastavení průtoku barvy	25
3.4.6	Konvence položení trajektorie	26
3.5	Uvedení do problematiky plánovacích algoritmů	27
3.6	Vstupní a výstupní parametry	30
3.6.1	CAD Model	30
3.6.2	Orientace nástroje	31
3.6.3	Charakteristika rozstříku	31
3.6.4	Výstupní kód trajektorie pro roboty ABB	32
3.6.4.1	Movement (Způsoby pohybu)	34
3.6.4.2	ToPoint	34
3.6.4.3	Speed	34
3.6.4.4	Zonedata	34
3.7	Přístupy hledání trajektorie aplikátoru	36
3.7.1	Standardní analytická metoda	36
3.7.2	Plánování trajektorie použitím rekonstrukce Iso-parametrických ploch	37
3.7.3	Metoda sledování povrchu (MFT)	38
3.7.4	Hledání trajektorie metodou Line-sweep	40
3.8	Průzkum trhu	42
<b>4</b>	<b>Část Praktická</b>	43
4.1	Návrh procesu použití	44
4.1.1	Celková analýza	44
4.1.2	Analýza profilu vrstvy	44

4.1.3	Hledání optimální trajektorie . . . . .	45
4.2	Návrh uživatelského rozhraní a jeho ovládání . . . . .	45
4.2.1	Záložka: Model . . . . .	46
4.2.2	Záložka: Inspection of normals . . . . .	47
4.2.3	Záložka: Pitch optimization . . . . .	47
4.2.4	Záložka: Trajectory planner . . . . .	48
4.2.5	Záložka: ABB Rapid code generator . . . . .	48
4.2.6	Záložka: Help . . . . .	49
4.3	Načtení a vizualizace modelu . . . . .	50
4.4	Výpočet normálových vektorů povrchu . . . . .	52
4.4.1	Algoritmus výpočtu normál a přidružených operací . . . . .	52
4.4.2	Vizualizace normál . . . . .	56
4.5	Aproximace tloušť kového profilu vrstvy . . . . .	57
4.5.0.1	Interpolace polynomem n-tého stupně . . . . .	57
4.5.0.2	Interpolace pomocí kubického splinu . . . . .	58
4.6	Hledání optimální rozteče průjezdů . . . . .	60
4.6.1	Algoritmus výpočtu optimální rozteče překryvů . . . . .	63
4.7	Vizualizace optimálního překryvu . . . . .	67
4.8	Výpočet trajektorie a její zobrazení . . . . .	69
4.9	Generování trajektorie GeneratePath() . . . . .	70
4.9.1	FindTrianglePlaneIntersection() . . . . .	74
4.10	Generování výstupů . . . . .	77
4.11	Generování RAPID kódu pro ABB roboty . . . . .	80
4.12	Další vývoj aplikace . . . . .	82
<b>5</b>	<b>Část experimentální . . . . .</b>	<b>83</b>
5.1	Měření profilu tloušťky barvy . . . . .	84
5.2	Nalezení trajektorie na modelu a simulace . . . . .	88
<b>6</b>	<b>Závěr . . . . .</b>	<b>90</b>
	<b>Seznam použitých značek a symbolů . . . . .</b>	<b>91</b>
	<b>Seznam použitého SW . . . . .</b>	<b>99</b>
	<b>Seznam příloh . . . . .</b>	<b>100</b>
	<b>Seznam příloh na přiloženém CD . . . . .</b>	<b>100</b>

# 1 Úvod

Robotické lakování patří v dnešní době k velmi důležitým součástem výrobních procesů mnoha výrobků. Nejčastěji se jedná o výrobky z automobilového, nábytkářského či jiného průmyslu, kde se produkují velké série. Dále se můžeme setkat s robotickým lakováním při výrobě velkých produktů a částí konstrukcí, jako jsou například lopatky rotorů větrné turbíny, trupy a křídla letadel.

Rovnoměrnost tloušťky nástřiku na výrobku výrazně ovlivňuje jeho kvalitu a tudíž i hodnotu. Mezi rozhodující faktory při získávání rovnoměrné vrstvy bez vad patří trajektorie aplikačního zařízení a nastavení technologického procesu. Právě zkoumáním těchto faktorů se aktivně zabývají výzkumné instituce a technologické firmy po celém světě. I přesto, že automatické hledání trajektorie aplikačního zařízení je velice aktuální téma, je tato oblast výzkumu velice neprobádána. Důvodem je především komplexnost a náročnost hledání optimální trajektorie a upřednostňování vývoje atomizérů a aplikačních zařízení.

Jedna z největších výzev a motivace pro oblast robotického lakování dnešní doby je nalezení takového algoritmu, který bude schopen automaticky naplánovat trajektorii vedoucí k optimalizaci vybraného kritéria nebo více kritérií zároveň. Pokud by byl takový algoritmus nalezen, bylo by možné začlenit robotické lakovací systémy snadno do kusové, či malosériové výroby. Tímto směrem se ubírá i ideologie čtvrté průmyslové revoluce.

Aplikace vytvořená v prostředí App Designer - MATLAB bude obsahovat plánovací algoritmus, který bude schopný naplánovat trajektorii robota pro tělesa a plochy definované stereolitografickým popisem. Aplikace bude také umožňovat nalézt optimální rozteč mezi jednotlivými průjezdy robota, nezbytně nutnými pro nalezení trajektorie aplikačního zařízení. Výstupem z aplikace budou souřadnice bodů, definující pohyb aplikačního zařízení. Tyto souřadnice bodů bude také možné vygenerovat ve formátu programovacího jazyka RAPID, jež se používá pro programování robotů ABB. Dále bude představena teorie spojená s technologickými postupy při návrhu výrobních procesů a technologických zařízení.



## 2 Cíl práce

Hlavním cílem této práce je navrhnout a vytvořit aplikaci, pomocí které si především inženýři z oblasti automatizace lakovacích procesů usnadní práci a ušetří tak čas při návrhu trajektorie. Sekundárním cílem je vytvořit takový algoritmus, který zajistí optimální stejnoměrnost pokrytí plochy. Tyto cíle není ale možné dosáhnout přímo a práce je tak složena z více dílčích cílů.

### **Jednotlivými cíli této práce tedy jsou:**

- poskytnout teoretický úvod o robotickém lakování
- poskytnout teoretický základ nutný k pochopení, jak funguje technologie lakovacích zařízení
- provést problematikou hledání optimálního nastavení lakovacího procesu
- uvést do problematiky plánovacích algoritmů a přístupů hledání trajektorie
- navrhnout architekturu aplikace
- implementovat tuto aplikaci v rámci zvolených technologií
- připravit virtuální případovou studii
- uskutečnit experiment s vybraným aplikačním zařízením
- otestovat funkčnost a použitelnost aplikace

## 3 Část teoretická

### 3.1 Počátky robotiky, automatizace a mechanizace

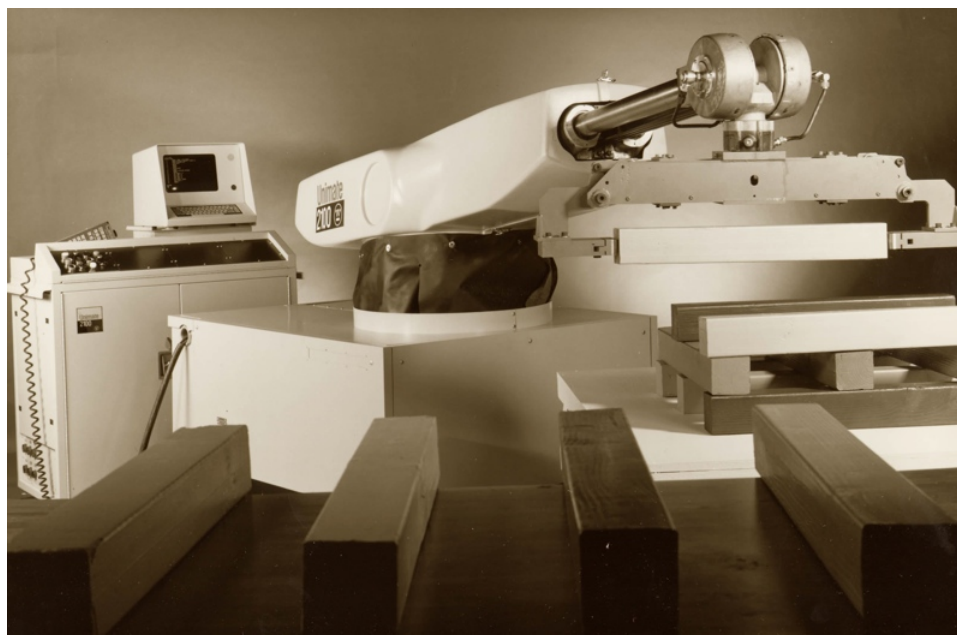
První myšlenky na automatizování některých činností se zrodily již před několika stoletími. Ve starověkém Římě se objevovaly nápady na automatické zvedáky, otevírání dveří, hydraulické systémy a systémy, které by vykonávaly jednoduché činnosti místo lidí. V roce 850 byla v Iráku dokončena jedna z prvních publikací *Book of Ingenious Devices*, která shromáždila stovky nápadů na mechanismy a jejich aplikaci. Po dokončení dílo usnadnilo konstruktérům získávání informací a pomáhalo tak k výběru optimálního řešení problému.

Velký pokrok v oblasti konstrukčních návrhů a mechanizace byl zaznamenán v období renesance. Začaly vznikat instituce, které se technickými problémy začaly aktivně zabývat. Jeden z nejznámějších konstruktérů byl geniální vynálezce, malíř a umělec Leonardo Da Vinci. Ten během svého života připravil stovky zajímavých konstrukčních řešení, které svět obdivuje dodnes. Většina z nápadů nebyla v počátku využívána, jelikož řešení měla často velké nedostatky. Postupem času docházelo k dalšímu vývoji a optimalizaci. Nápady tak začaly přinášet lidstvu radost, usnadňovaly náročnou manuální práci a občas sloužily i k pobavení.

Dramatické změny se v oblasti výroby začaly odehrávat od začátku první průmyslové revoluce (od 18. do 19. století). Ve výrobních procesech docházelo častěji k přechodům od ruční výroby k tovární strojní výrobě. Byly k dispozici nové zdroje energie a také začalo docházet k dělbě práce, specializacím a plánování výroby. S mechanizací výroby se stávaly produkty dostupnější, dramaticky rostly výrobní kapacity a začala se objevovat velká motivace pro automatizaci.

Dalším velkým skokem v oblasti vývoje výrobních zařízení bylo vynalezení integrovaných obvodů a počítačů (1950) [1]. Díky těmto prostředkům George Charles Devol, přezdívaný otec robotiky vynalezl roku 1954 první průmyslový robot Unimate. Ten byl o 7 let později vyroben v několika prototypch a nainstalován na výrobní linku americké společnosti General Motors. Na lince byly roboty využívány na bodové svařování a zakládání plechových dílců do lisu. Cena jednoho robotu bez aplikace byla 65 000 USD, což v přepočtu na dnešní hodnotu činí přibližně 12 000 000 Kč.

Po velkém zájmu o robot Unimate si konkurenční společnosti z celého světa všimly

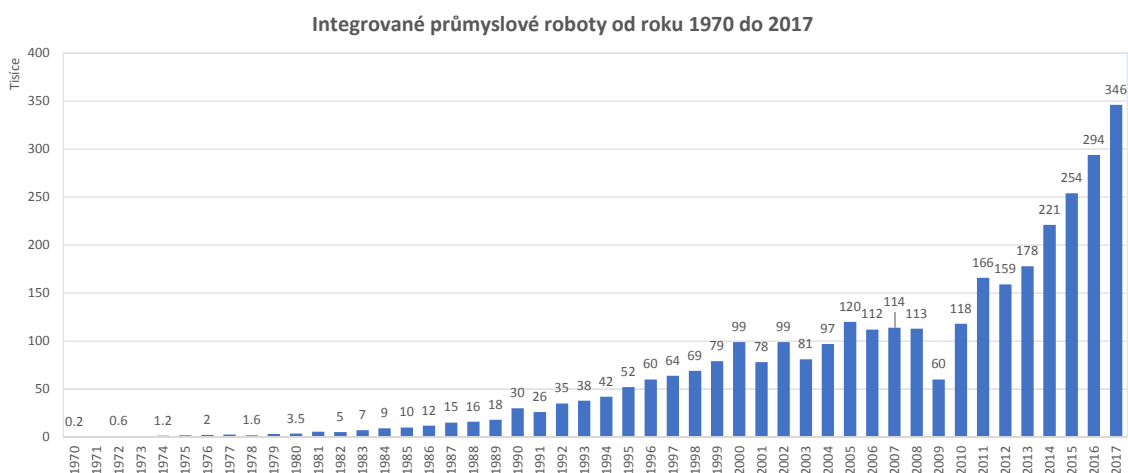


**Obr. 1:** První průmyslový robot Unimate (1961) [35]

velkého potenciálu v průmyslových robotech. To vedlo k ještě intenzivnějšímu výzkumu a technickému vývoji, jak po stránce mechanické, elektrické, tak i řízení. Na trhu se začaly objevovat mikroprocesory s výkonem takovým, který byl schopný počítat složité řídicí algoritmy. Zároveň pořizovací ceny polovodičů klesaly a tím se stávaly roboty postupně dostupnější.

Inženýři po celém světě začali hledat uplatnění průmyslových robotů v řadě velmi odlišných aplikací, napříč různými průmyslovými odvětvími. To rozšířilo možnosti využití a tak následoval enormní růst poptávky, který v polovině 70. let dosahoval meziročně až 50 %. V počátku 80. let začaly automobilky zaplavovat společnosti prodávající robotické systémy objednávkami. Bohužel se později ukázalo, že robotické systémy jsou poměrně poruchové a investice do oprav a odstávek začaly vytvářet finanční nestabilitu. Tyto skutečnosti snížily v letech 1984-1989 meziroční růst na 10%. V 90. letech šla opět cena výpočetní techniky dramaticky dolů a díky tomu se snížila i pořizovací cena robotů. Zároveň investoři získali opět důvěru a tím se odstartovala druhá investiční vlna, která až na malé výkyvy přetrvává dodnes. Dle IFR (International Federation of Robotics) je nasazeno po celém světě přes 2 000 000 průmyslových robotů a tento počet se má dle predikcí IRF do roku 2022 téměř zdvojnásobit.

Na grafu níže je znázorněný vývoj počtu integrovaných průmyslových robotů (ročně) v průběhu posledních 47 let.



**Obr. 2:** Integrované průmyslové roboty od roku 1970 do roku 2017 (ročně) [10] [42]

## 3.2 Počátky lakovacích robotů

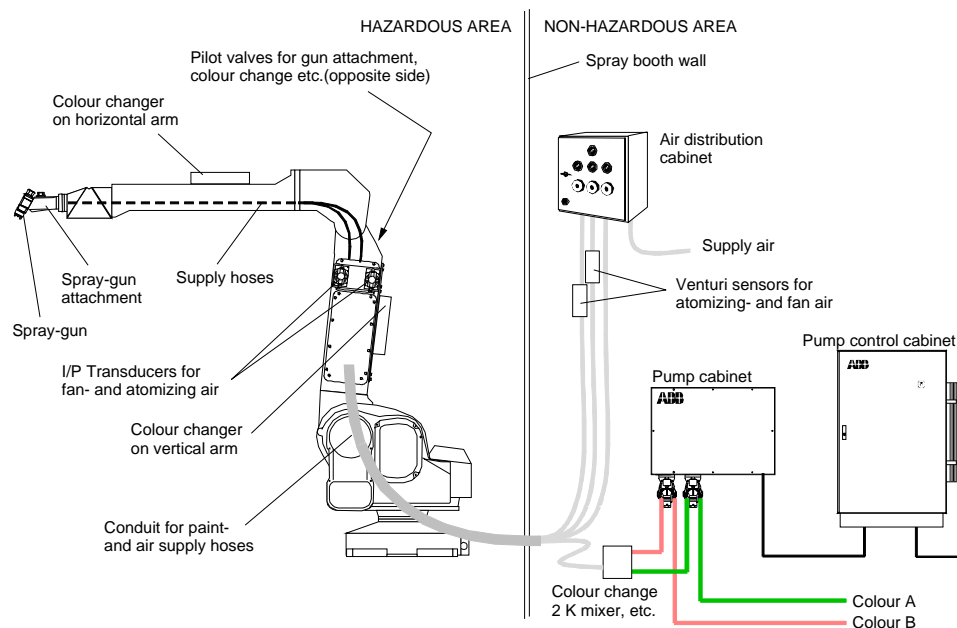
Počátek historie začíná v roce 1941, kdy Nils Underhuag založil společnost na výrobu manipulační techniky a přepravních vozíků. Nils byl velmi nadaný a již v 17 letech navrhl a sestrojil automobil. Monstrum s motorem o výkonu 1,5 koňských sil. Po zakončení svého vzdělávání jako automechanik dostal od banky úvěr 2000 USD na založení společnosti na výrobu rudlů. Ta oficiálně vznikla v únoru 1941.

Společnost měla v počátku dva zaměstnance, kteří spolu s Nilsem vyráběli vozíky ve stodole. Postupem času se začali specializovat na výrobu koleček a společností se začalo velmi dařit. Dokonce natolik, že již brzy se mohli přestěhovat do výrobní haly a násobně zvýšit produkci. Cena koleček se snižovala a tím se o ně výrazně zvýšil zájem. Nils se snažil zvýšit produkci jak jen to šlo. Po čase ale narazil na velký problém. Kolečka byly barveny ručně a navzdory skutečnosti, že pracovníci byli vybaveni tím nejmodernějším vybavením a pracovali ve směnách, natírání se stalo úzkým místem výroby.

Molaug, manažer výroby přišel z nápadem použít pro lakování automatický systém. To se Nilsovi zdálo jako velmi dobrý nápad a dal Moluagovi prostor na vývoj robotického systému. Po 3 letech, roku 1967, byl poprvé představen a spuštěn robot, který na pásu lakoval korby pro kolečka. Výsledek byl na tolik dobrý, že se společnost rozhodla roboty začít vyrábět a prodávat. V roce 1969 začaly z výrobní linky vyjíždět první sériově vyráběné lakovací roboty. Společnost Trallfa byla později akvizována společností ABB a ta vyrábí jedny z nejlepších lakovacích systémů dodnes.

### 3.3 Robotické lakovací systémy

Následující sekce stručně popisuje řídicí systém, který se pro lakování často používá ve společnosti ABB. Na obrázku 3 je zobrazený příklad lakovacího robotického systému. V systému jsou použity prvky: I/P měnič, atomizér, jednotka distribuce stlačeného vzduchu, jednotka pro míchání barvy, čerpadla a jednotka pro jejich ovládání.



**Obr. 3:** Příklad lakovací sestavy IPS s čerpadlem na robotu IRB5400 [21]

Proces lakování s atomizační jednotkou se skládá standardně ze tří řídicích systémů. Nastavení každého ze systémů ovlivňuje výsledný lakovací proces a tedy i kvalitu nanášení barvy. V praxi patří právě vyladění procesu lakování pomocí hledání správných parametrů řízení systémů k nejnáročnějším disciplínám lakování.

Obecně máme:

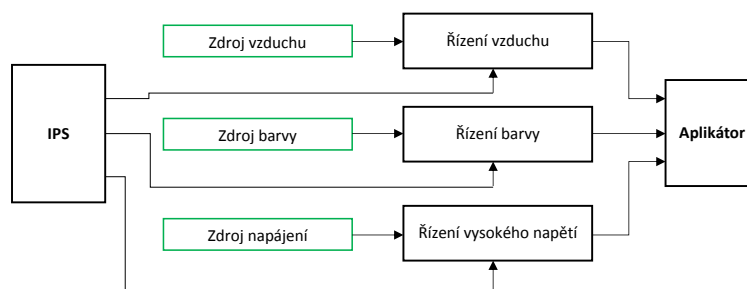
**Systém řízení vzduchu**, který reguluje požadované průtoky vzduchu do aplikátoru.

**Systém řízení barvy**, jež reguluje požadované průtoky barvy do aplikátoru.

**Systém řízení vysokého napětí**, který reguluje napětí elektrostatického aplikátoru.

Integrovaný procesní systém IPS (Integrated Process System), který řídí celý systém se

skládá z uzavřených (closed-loop) smyček regulátoru. Vysokorychlostního řídicího systému pro ovládání aplikátoru, jež je přesně synchronizován s pohybem robota. Stručné schéma celého systému je na následujícím obrázku.



**Obr. 4:** Systémový diagram

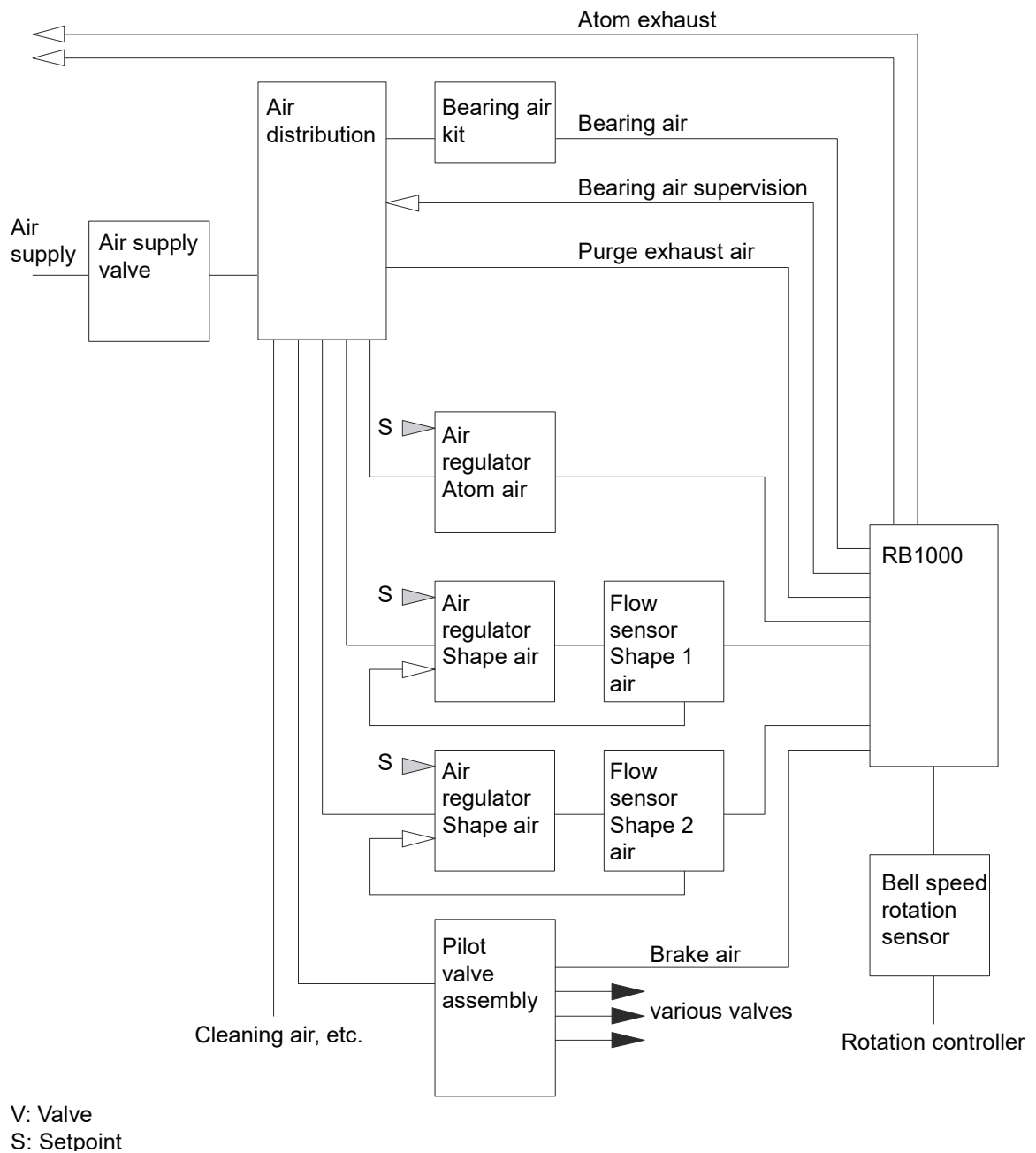
### 3.3.1 Řízení vzduchu

Princip řízení jednotky vzduchu je popsán pro aplikátor ABB RB1000. Tvar vzduchové distribuce je řízen tokem vzduchu pro atomizér (atomizing air) a tokem vzduchu pro tvarování (shaping air). Atomizační vzduchový přívod má za úkol atomizovat a vaporizovat barvu. Atomizační vzduch je řízen IPS jednotkou s IP regulátorem, který reguluje průtok a tlak vzduchu tekoucího do lakovacího zvonu. Vzduch pro tvarování je regulován taktéž IPS jednotkou. Jeho hlavní funkce je nastavení požadovaného tvaru a šířky rozprašování barviva. Zjednodušené schéma je zobrazeno na obrázku 5.

Vzduch přivedený do vzduchového ložiska (Bearing Air) zásobuje zároveň vzduchový motor, jež roztáčí lakovací zvon. Otáčky zvonu jsou snímány a zpětnovazebně řízeny. Aby bylo možné rychle zmenšovat otáčky zvonu, je do aplikátoru přiveden přívod s brzdícím vzduchem (Brake Air). Přívod je realizován přes ventilovou sestavu, která je řízena Integrovaným procesním systémem IPS.

### 3.3.2 Řízení barvy

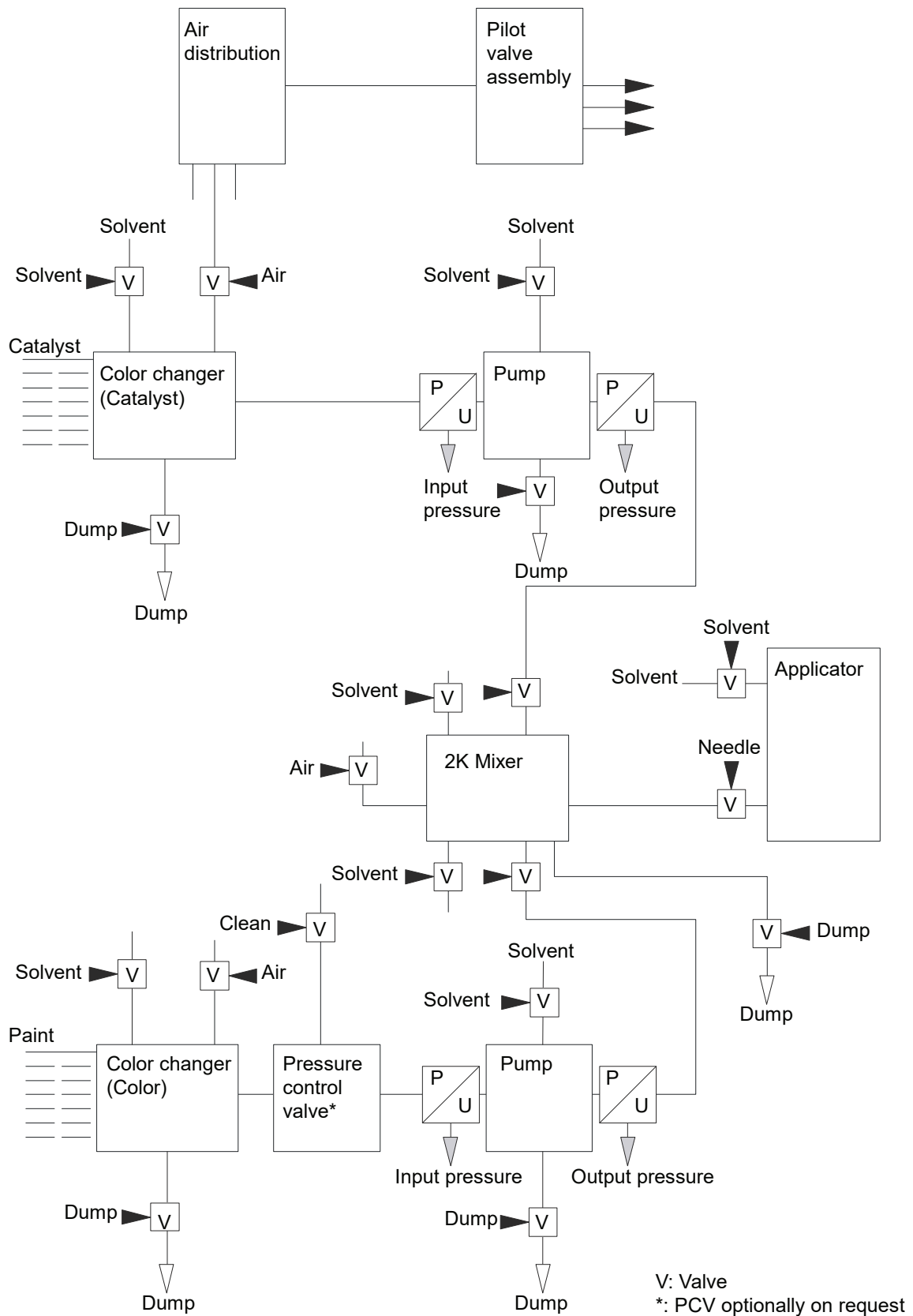
Proces pro přípravu barvy je procesem nejsložitějším. V této kapitole je pro hrubý přehled uveden pouze elementární popis procesu. Jeden měnič barvy dodává jeden z vybraných katalyzátorů za jednotku času. Druhý měnič dodává vybranou barvu za jednotku času a obě složky jsou čerpadly dopraveny do míchacího prostoru (2K-Mixer). V míchacím prostoru se dvousložková substance promíchá a dostane ideální konzistenci. Dále je



**Obr. 5:** Procesní diagram vzduchového okruhu pro RB1000 [22]

vzduchem vtlačena do aplikátoru.

Měnič barvy (Color Changer) je speciálně vyvinutý tak, aby umožňoval velmi rychle měnit barvu. Vnitřní struktura měniče je navržena bez "mrtvých konců", což minimalizuje časy a ztráty během čistících cyklů.



**Obř. 6:** Diagram toku barvy pro RB1000 s pouřitím jednotky 2K Mixer [23]



### 3.3.3 Řízení vysokého napětí

Řízení napětí na elektrostatickém aplikátoru je součástí jednotky IPS. Vysokonapěťová kaskáda je integrována do samotného aplikátoru.

### 3.3.4 Trysky (atomizéry) používané v robotice

Nejčastěji se setkáváme se třemi typy atomizérů. Ty se navzájem od sebe liší technologií dopravy barvy k cíli, tvarem stříkaného paprsku, velikostí stříkaného paprsku a také pořizovací cenou.

Nejjednodušším typem je atomizér (Obr. 7 vlevo) bez tvarovacího vzduchu a vysokého napětí. Šířka stříkaného paprsku se obvykle pohybuje od 30 mm do 350 mm. Tvar paprsku závisí, jak na tvaru trysky, tak i vzdálenosti trysky od cíle a případně i tlaku média.

Více používaný je systém (Obr. 7 střed), který využívá pro usměrnění barvy tvarovací vzduch (shaping air). Šířka stříkaného paprsku je od 50 do 500 mm. Výhodou tohoto systému je možnost měnit tvar a distribuci barvy pomocí změn tlaku tvarovacího vzduchu. Dále může být tvar stříkaného paprsku ovlivňován tlakem média a vzdáleností trysky od cíle.

V robotice jsou nejvíce používané atomizéry s rotačním zvonem (Obr. 7 vpravo). Jejich schopností rovnoměrně nanášet barvu se zpravidla využívá v automobilovém průmyslu. Šířka kruhového paprsku se pohybuje od 100 do 500 mm a lze jej měnit pomocí změn mnoha parametrů, jako například: Napětí, otáčky turbíny, tlaky tvarovacích vzduchů, apod.



**Obr. 7:** Typicky dosahované (používané) tvary stříkaných paprsků při robotickém lakování

Volba aplikátoru velmi podstatně ovlivňuje účinnost aplikace a tím návratnost investice celého systému. Pokud bychom porovnali manuální nanášení, kde se převážně používají atomizéry, u kterých dochází k přenosu laku na cíl pouze za pomoci tvarovacího vzduchu, můžeme hovořit o účinnosti přenosu mezi 15 až 30 %. Pro robotické systémy s nejmodernějšími atomizéry a správně naprogramovanou trajektorií se může účinnost pohybovat až na 90%. Při použití těchto atomizérů v sériové výrobě je na první pohled zřejmé, že úspory ve spotřebě barviv, filtrů odsávacích zařízení a času se pozitivně projeví do výrobních nákladů. Porovnání jednotlivých technologií z pohledu účinnosti je zobrazen v tabulce

<b>Technologie</b>	<b>Interval [%]</b>	<b>Typická [%]</b>
Vzduchová stříkácí pistole	15 - 30	25
Vzduchová stříkácí pistole s tvarovacím vzduchem	25 - 45	30
Vzduchová stříkácí pistole HVLP	25 - 45	35
Bezvzduchová stříkácí pistole	20 - 50	40
Vzduchová stříkácí pistole s VN	35 - 50	40
Vzduchová stříkácí pistole s tvar. vzd. a VN	45 - 55	50
Rotační zvon s VN (ABB G1 Cope)	70 - 80	75
Rotační zvon s VN a int. nabíjením (ABB Bell 925)	75 - 85	80
RZ s CBS, VN a int. nabíjením	75 - 85	80
Digitální RZ s CBS, VN, a int. nabíjením	80 - 98	90

**Tab. 1:** Tabulka účinností aplikátorů

Vysokonapět'ové elektrostatické lakování s použitím vysokootáčkového rotačního zvonu je založeno na principu vytvoření elektrostatického pole, které usměřuje částičky k cíli. Lakovaný předmět je zapojený jako anoda a elektrostatický rotační zvon jako katoda s negativním napětím od -50 kV do -120 kV. Rotační zvon je standardně poháněný vzduchovou turbínou, která dosahuje od 20 000 do 70 000 otáček za minutu. Barva, která se do dostane do vysokootáčkového zvonu je odstřed'ována po jeho vnitřní straně a vytváří velmi tenkou vrstvu. V okamžiku, kdy se vrstva dostane na okraj zvonu, dojde vlivem elektrostatického pole a odstředivé síly k rozštěpení vrstvy na mikroskopické kapénky. Kapénky jsou opačně nabitě než je cílový objekt a tak se vydávají správným směrem. Správnému směřování a ochraně procesu před vměšováním prachu pomáhá tvarovací vzduch (tzv. Shaping air).

Konstrukce, jakým jsou atomizéry s rotačním zvonem navrženy se může u různých výrobců lišit. Princip však zůstává stejný. Detailnější popis funkce je uveden například ve člunku *Automotive Rotary-Bell Spray Painting* [36], dále také v dokumentu *How a Rotary Bell Atomizer Works* od společnosti GRACO [17].

### 3.4 Hledání optimálního nastavení procesu aplikace

Aby bylo možné splnit vysoké požadavky zákazníka na kvalitu povrchové vrstvy, je nutné dbát na precizní nastavení procesních parametrů a nalezení optimální trajektorie aplikátoru. Pro to, aby bylo možné tento cíl splnit, musí se vypracovat analýza lakovacího procesu.

Analýza se může skládat z několika možných částí v závislosti na tom, jak vysoké kvality chceme dosáhnout. Pro představu je vhodné říci, že tato fáze se ve většině případů provádí již zákazníkem zakoupeným systémem, buď v aplikačních centrech, nebo přímo u zákazníka. Je tedy nutné splnit domluvené kvalitativní náležitosti s takovým vybavením, které je obsaženo v dodávce a změna technologie nepřipadá k úvahu.

Během vývoje procesu se zabýváme například analýzou:

1. hmotnosti pevné složky u aplikované vrstvy
2. efektivity přenosu barvy
3. šířkou vrstvy
4. tloušťkou vrstvy
5. světových/firemních konvencí způsobu lakování
6. vynikajících vad
7. odhadované rychlosti robota
8. barevnosti
9. vlnové délky
10. optimálních překryvů
11. atd.

### 3.4.1 Měření reálné šířky paprsku

Po té, co máme nastavené parametry procesu, můžeme vytvořit první vzorky lakování. To provedeme tak, že si připravíme rovné a hladké plechy o dostatečné délce a šířce. Ty umístíme ideálně do takové polohy, v jaké bude reálný dílec lakován (pokud lze specifikovat). Následně připravíme program robota tak, aby přešel přes plech rovnoměrným přímočarým pohybem horizontálně s kratší stranou vzorku. Provedeme další nezbytně nutné kroky technologického postupu předepsané výrobcem barvy a dokončíme tak přípravu vzorku.



**Obr. 8:** Vzorek před (vlevo) a po (vpravo) nástřiku barvou

Analýza probíhá standardně dvěma způsoby. První z nich je naprosto primitivní a používá se pouze v provozech, kde není důležité držet přísné tolerance tloušťky vrstvy. Může se například jednat o nanášení barev u hraček. Metoda je založená na subjektivním určení hranice, kde končí celistvá hranice barvy. Po odhadu obou hranic se změří vzdálenost mezi nimi a výsledek určuje šířku paprsku. U této metody se také předpokládá, že požadovaná tloušťka vrstvy je přes celý interval zvoleného paprsku.

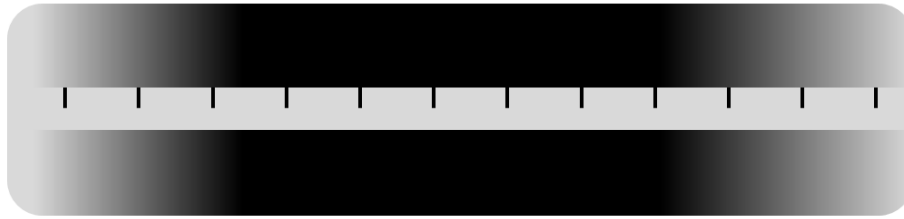


**Obr. 9:** Příklad posuzování šířky paprsku

Druhá metoda je sofistikovanější a přesnější. Je založena na principu měření tloušťky vrstvy laku po celé délce vzorku. Používá se ve většině aplikacích robotického lakování a je nedílnou částí postupu hledání optimálního nastavení procesu.

Na připravený vzorek s povrchovou úpravou se po celé délce připraví stupnice s rozestupy 15 nebo 20 mm. Tak, jako je tomu na obrázku 10.

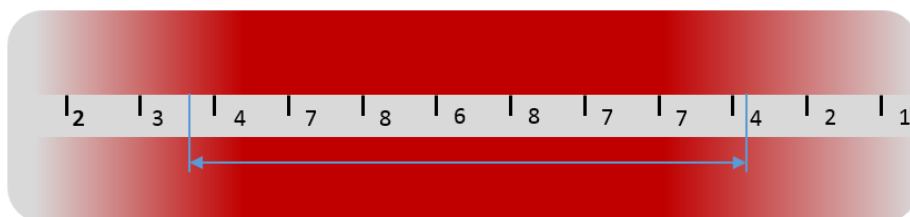
V dalším kroku se provede měření tloušťky ve vyznačených bodech. Měření se provádí měřicími přístroji, které ovšem nemusí mít velkou přesnost měření a na totožném



**Obr. 10:** Příprava stupnice

místě mohou naměřit hodnoty s rozestupem až 10 %. Z tohoto důvodu je možné provést více měření na jednom místě a hodnoty zprůměrovat. Poté co máme kompletní dataset, vyneseme hodnoty do grafu a hledáme takové místo, kde tloušťka dosahuje poloviny průměrné tloušťky určené ze střední části nástřiku. Místo, z kterého se vypočte průměrná hodnota se získá expertním odhadem.

Příkladem může být vzorek na obrázku 11. Zvýrazněná plocha představuje expertní odhad z kterého se vypočítá průměrná tloušťka vrstvy. Průměrná tloušťka se vydělí dvěma a získáme tak koeficient  $t_{50}$ . Do grafu vykreslíme horizontální úsečku ve výšce tloušťky  $t_{50}$  a určíme průsečíky s křivkou tloušťkového profilu. V posledním kroku se určí vzdálenost mezi průsečíky, jež představuje šířku paprsku.



**Obr. 11:** Analýza vzorku

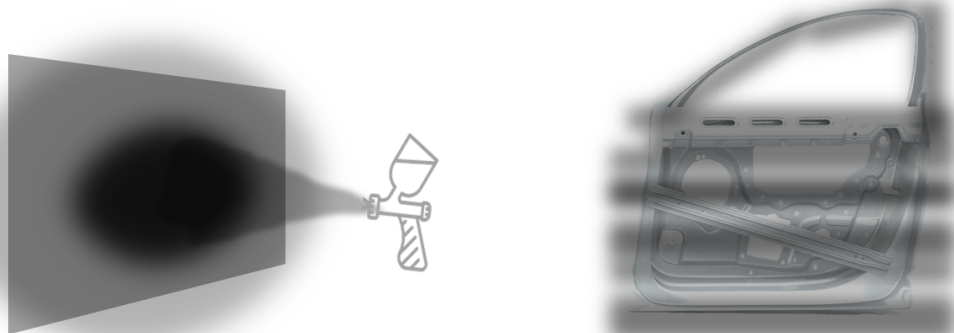
### 3.4.2 Efektivita přenosu barvy

Samozřejmostí dnešní doby je i sledování efektivity přenosu barvy. Efektivita přenosu je dána jak použitou technologií (viz. tabulka 1), tak i mnoha dalšími vlivy. Aby bylo možné získat přehled o tom, kolik barvy opravdu proces spotřebuje, je nutné provést jeho analýzu.

Celkovou analýzu můžeme rozdělit na dvě separátní úlohy. Úloha první se zabývá přenosem barvy bez ohledu na tvar součásti. Na připravenou hliníkovou folii o známé hmotnosti se nastříká vrstva barvy. Vrstva se vytvrdí v peci a vzorek se zváží. Před a po provedení experimentu je také nutné zvážit zásobník barvy, jelikož průtokoměry instalované v pumpách systému nejsou standardně dostatečně přesné pro zjištění objemového úbytku. Abychom mohli určit koeficient účinnosti přenosu  $TE$ , musíme znát podíl pevné složky v barvě a použít následující vztah:

$$TE_1 = \frac{\text{weight of applied substance}}{\text{weight of used substance} \cdot \frac{SC}{100}} \cdot 100 \quad [\%] \quad (1)$$

Pro vytvoření závěru analýzy je nutné experiment opakovat a výsledky měření mezi sebou porovnat. V případě, že efektivita přenosu nesplňuje standardní (viz.1), či požadované kritéria, musíme proces upravit. V opačném případě je proces dostatečně stabilní a může se přejít na druhou část analýzy.



**Obr. 12:** Nástřik barvy na hliníkovou folii (vlevo); efektivita přenosu barvy na součást (vpravo)  
[41] [29]

Druhá úloha analyzuje efektivitu přenosu barvy během výrobního procesu. Metoda je téměř totožná, jako je tomu u úlohy první. Rozdílem je, že se přímo lakovaná součást potáhne hliníkovou fólií a nalakuje se takovým způsobem, jakým bude lakována ve výrobním

procesu. Po nalakování se hliníková fólie odstraní, vrstva laku se vytvrdí a zaváží. Jistou výhodou toho experimentu je, že získáme efektivitu přenosu během celého procesu.

### 3.4.3 Vzhled vrstvy

Vzhled nanesené vrstvy patří k základním kritériím kvality výrobku. Například moderní automobilové povrchové nátěry často obsahují metalické vločky pro dosažení metalických efektů. Používají se matné barvy s nízkým koeficientem odrazivosti a barvy různých odstínů. Všechny parametry (a mnoho dalších) související s kvalitou je nutné během nastavování procesu analyzovat a ověřit tak, zda korespondují s požadavky [33].

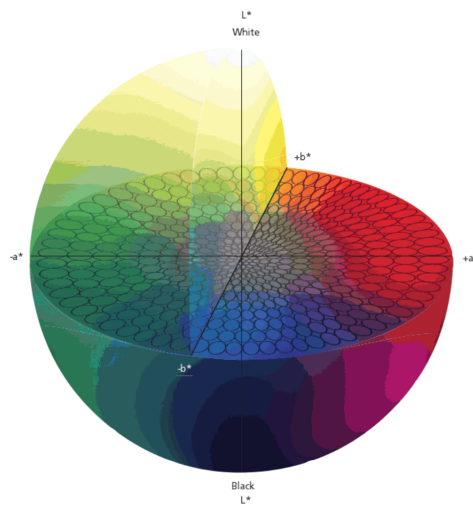
Důvod, proč se touto částí velice důkladně zabýváme, je přiblížen níže. Bohužel tato část analýzy mnohdy vyžaduje velmi nákladné vybavení a je také časově náročná. Představme si, že automobilka má dvě lakovací linky (A, B) na auta, která jsou lakovaná různými barvami. Dále výroba disponuje speciální lakovnou na kapoty a páté dveře (C). Cílem produkce je dosáhnout stejných parametrů nátěru na všech provezech. Jinými slovy můžeme říct, že po doručení automobilů zákazníkovi z produkce A i B se předpokládá, že auta budou naprosto totožná. Například, kapota nebude mít jinou odrazivost, než karoserie či páté dveře. Aby toto bylo možné zabezpečit, je nutné provádět důkladnou analýzu a nastavit proces tak, aby byl každý z předepsaných parametrů splněn v rámci každé lakovny.

#### **Analýzou se standardně ověřují následující parametry:**

1. Odrazivost
2. Barevnost (Odstín, Sytost, Jas)
3. Vlnová délka
4. Popř. distribuce metalických vloček

Pro analýzu barevnosti se používá model barevného prostoru CIE-LAB. Systém LAB vychází z barvového prostoru XYZ. Je složen z imaginárních barev, které jsou vytvořeny pouze matematicky a jsou proto nezávislé na přístrojovém barevném tělesu (oproti modelům RGB nebo CMYK). Model je zvolen tak, aby obsáhl množinu všech barev, zachytitelných lidským okem. [44]





**Obr. 13:** Model barevného prostoru CIE-LAB  
[25]

V automobilovém, vlakovém, či leteckém průmyslu hraje také podstatnou roli vlnová délka. Ta totiž ovlivňuje, jak se budou opticky projevovat nedokonalosti povrchu na kterém je barva aplikována. V rámci analýzy vlnové délky se sledují parametry: SW (Short Wave), LW (Long Wave), DOI (Distinctness of image). [19]

#### 3.4.4 Odhad rychlosti pohybu robota

Pokud chceme odhadovat rychlost pohybu robota, může použít následující vztah:

$$v_A = \frac{w_p \cdot \frac{t}{N}}{\frac{TE}{100} \cdot \frac{SC}{100} \cdot \frac{PFR}{60}} \quad (2)$$

kde,  $w_p$  [m] je šířka paprsku aplikátoru,  $t$  [ $\mu\text{m}$ ] tloušťka vrstvy,  $N$  [-] počet vrstev,  $TE$  efektivita přenosu barvy,  $SC$  podíl pevné složky a  $PFR$  [cc/min] představuje průtok barvy.

#### 3.4.5 Odhad nastavení průtoku barvy

Pro první nastavení procesu potřebujeme odhadnout, jaký průtok barvy budeme při zvolené rychlosti robota, šířce paprsku, tloušťce požadované vrstvy a dalších parametrů

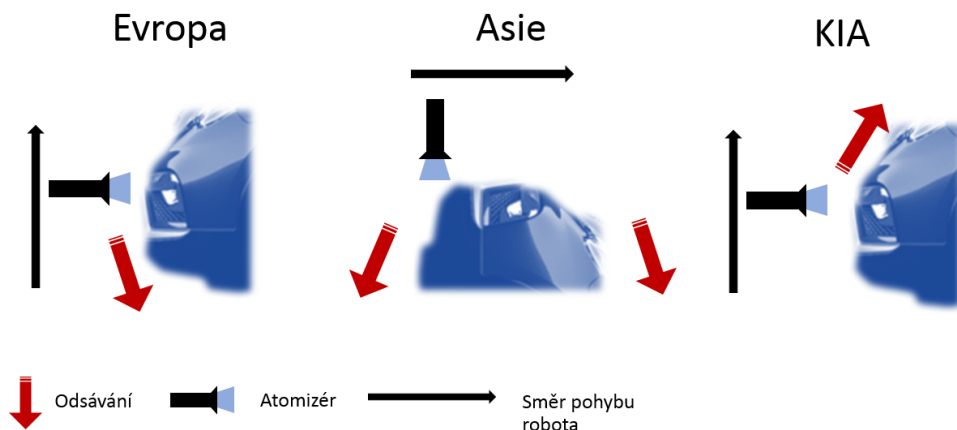
potřebovat. Nastavení průtoku můžeme odhadnout z následujícího vzorce a následně proces upravovat.

$$PFR = \frac{v_A \cdot w_p \cdot t \cdot N}{\frac{TE}{100} \cdot \frac{SC}{100}} \quad [cc/min] \quad (3)$$

Pro výpočet  $PFR$  (*Paint Flow Rate*) je nutné znát všechny proměnné kde:  $v_A$  [m/s] je rychlost pohybu atomizéru/robotu,  $w_p$  [m] šířka paprsku při předpokládané vzdálenosti nanášení,  $t$  [um] tloušťka vrstvy,  $N$  [-] počet vrstev,  $TE$  efektivita přenosu barvy,  $SC$  podíl pevné složky.

### 3.4.6 Konvence položení trajektorie

Zajímavostí jsou konvence položení trajektorie, na které je nutné dávat velký pozor při návrhu procesu. Je nutné znát důkladně způsoby a metody, které se v dané zemi, společnosti, či provozu používají. Podle konvence, která se v provozu používá musíme zohlednit především směr odsávání a směr jakým se má robot pohybovat. Dobrým příkladem může být například lakování nárazníku automobilu, kde se mohou projevit velké rozdíly ve způsobu nanášení.



Obr. 14: Příklady rozdílného způsobu položení trajektorie

### 3.5 Uvedení do problematiky plánovacích algoritmů

Již z první kapitoly je jisté, že lakovací proces patří k velmi důležitým úsekům výroby a jeho kvalita ovlivňuje celkovou kvalitu a hodnotu výrobku.

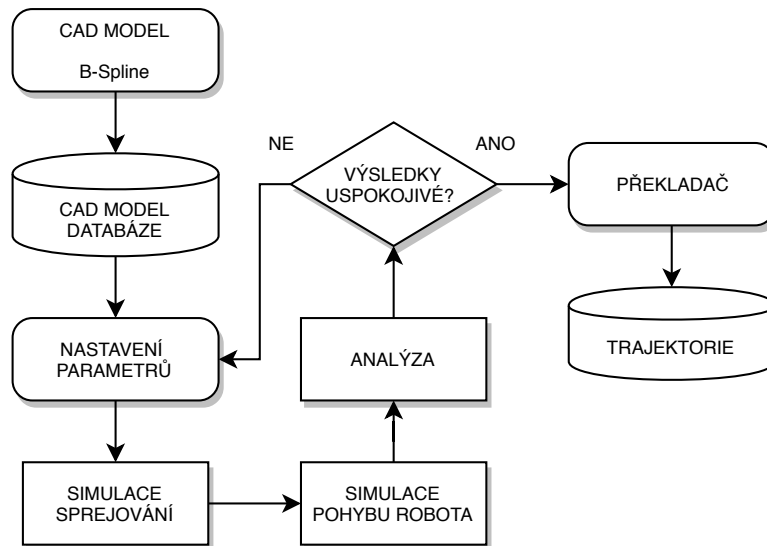
Plánování trajektorie je kritickou oblastí, která ovlivňuje celistvost, tloušťku vrstvy, ale také i časovou náročnost cyklu. V dnešní době máme dvě metody, jak se můžeme docílit požadované trajektorie. Metodou klasickou, kdy operátor robota vytváří manuálně programovou proceduru. Druhá metoda využívá algoritmů pro automatické generování trajektorie v závislosti na vstupních parametrech a typu optimalizace.

Klasická metoda je velmi časově náročná a tloušťka nanesené vrstvy záleží na zkušenostech operátora. Na rozdíl od klasické metody, je automaticky generovaná trajektorie časově méně náročná i přesto, že ve většině případech musí být manuálně dopravena. Také tloušťku vrstvy můžeme teoreticky lépe optimalizovat automatickým způsobem.[38]

První algoritmus (ATPS - Automatic Trajectory Planning System) pro generování trajektorie lakování byl vyvinut profesorem Suk-Hwan Suh a jeho týmem roku 1991. Jejich metoda je založená na aproximaci povrchu na malé části. Pro rozložení na sub-elementy byl použit bikubický B-spline popis. Algoritmus bere v potaz také možnosti mechanického systému. [45]

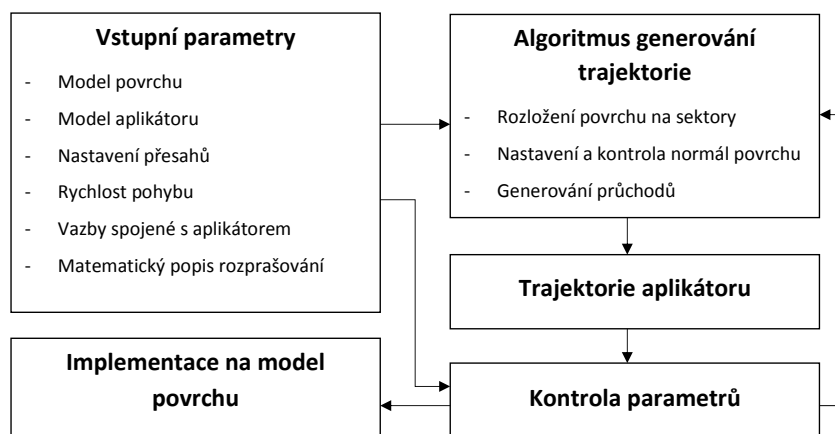
ATPS nabízel uživateli generovat trajektorii stříkací pistole a trajektorii robota. Vytvořený program také umožňoval uživateli analyzovat distribuci barvy na povrchu a její tloušťku. Simulace ale neřešila optimalizaci tloušťky vrstvy, nýbrž pouze pokrytí.

Framework pro optimalizaci tloušťky vrstvy byl představen poprvé Johanem Antoniem [31]. Algoritmus frameworku umožňoval nastavovat parametry procesu jako je: úhel sklonu pistole, úhel rozstříku a rychlostní profil. Metoda měla ale problémy zvládat dílce s komplikovanou topologií. Výpočet totiž využívá parametrický popis ploch a s tímto popisem je velmi náročné se vypořádat při složité geometrii plochy. Aplikace této metody byla tedy použitelná pouze pro jednodušší plochy [31].



**Obr. 15:** Schéma ATPS algoritmu

Později byl problém rozvíjen Antoniem a Ramabhadranem. Ti vyvinuli zobecněný framework pro optimalizaci rychlosti, přičemž při simulaci je předpokládáno, že trajektorie je již známá z jednoduššího distribučního modelu. Dále doktor Weihia s jeho týmem [34] vyvinuli algoritmus, který garantuje 100% pokrytí povrchu cílové plochy s minimalizací překryvů jednotlivých překryvů. Nicméně, ani tento algoritmus neřeší optimalizaci tloušťky vrstvy. Zajímavý přístup představil Pål Johan spolu s Gravdahl Jan Tommy [48], kteří se zabývali optimalizací natočení nástroje tak, aby pokrytí barvou dosahovalo stejné tloušťky.



**Obr. 16:** Metodika hledání trajektorie - současnost

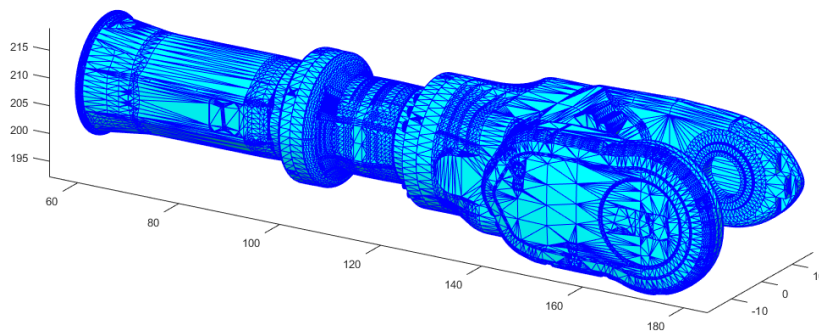
Souhrnem doposud provedených výzkumů můžeme říci, že automatické generování trajektorie je limitované na topologicky jednoduché součásti. U součástí, které mají složitou topologii, obecné metody selhávají a je zřejmé, že při vývoji optimalizačního SW se dá

přístupovat z několika různých úhlů pohledu. Počet parametrů, které ovlivňují optimální trajektorii po stránce tloušťky vrstvy je více, než je možné optimalizovat zároveň a musíme hledat kompromisy. V praxi nás mohou zajímat tyto parametry, které ovlivňují tloušťku vrstvy: Nastavení technologické hlavice vůči povrchu, kolize hlavice, možnosti robotického systému, geometrie rozstříku, vzdálenost hlavice od povrchu, rychlost pohybu robota, topografie povrchu, rychlost pohybu média, limitní přesahy, vlnová délka barevného profilu a další. Vzhledem k velkému množství parametrů je plánování optimální trajektorie lakování jednou z nejnáročnějších oblastí výzkumu až do dnes.

## 3.6 Vstupní a výstupní parametry

### 3.6.1 CAD Model

Plánování trajektorií se nejčastěji provádí na parametrickém, či síťovém modelu plochy. Představitelé parametrických CAD modelů jsou například formáty STEP a IGES. Ty jsou reprezentovány Beziérovým, B-Spline, či NUBRS popisem. Jedná se o popisy, kde plocha je reprezentována popisovými funkcemi. I přesto, že tento popis geometrie je matematicky přesný, pro plánování trajektorií je zbytečně složitý. Z tohoto důvodu se algoritmy, které řeší obecně problém hledání trajektorie na parametrických plochách používají jen zřídka. [13]



**Obr. 17:** Síťový model - trojúhelníková aproximace [7]

Představitelé síťových modelů jsou například formáty: STL, NASTRAN a VRML. Tyto modely popisují geometrii pomocí trojúhelníků. Tento systém je jednodušší v tom, že jednotlivé trojúhelníky jsou na sebe vázány pouze ve třech bodech. Problém, který může při aproximaci nastat spočívá ve špatně zvolené velikosti elementů. V případě, že by zvolený počet elementů byl příliš malý (přesnost modelu bude příliš malá), může se snadno stát, že plánování trajektorie nebude vlivem velké odchylky probíhat správně. [7]

CAD model používající trojúhelníkovou aproximaci pro popis plochy  $M$ , zobrazené na obrázku 17 může být reprezentován 76 110 trojúhelníky / 228 330 uzly. Obecně lze plocha  $M$  zapsat jako:

$$M = T_n : n \quad (4)$$

kde  $n = (1, 2, \dots, R)$  a  $R$  je celkový počet trojúhelníků. Každý trojúhelník  $T_n$  plochy  $M$

je tvořen třemi uzly  $N_i$ ,  $N_j$  a  $N_k$  [8]. Každý uzel je popsán v globálním souřadném systému souřadnicemi  $x$ ,  $y$  a  $z$ . Obecně můžeme uzel  $m$  plochy  $M$  zapsat:

$$N_{m|x,y,z} = \{N_m(x, y, z) : m = 1, 2, \dots, V\} \quad (5)$$

kde  $V$  je celkový počet uzlů.

### 3.6.2 Orientace nástroje

Orientace nástroje může být popsána v Euklidovském souřadném systému například šesti dimenzionální vektorovou funkcí  $A(t)$ , která je časově závislá.

$$A(t) = [p(t)o(t)] \quad (6)$$

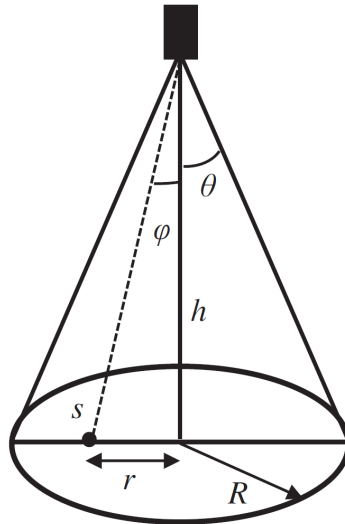
Po rozepsání vektorových funkcí  $p(t)$  a  $o(t)$  do Euklidovského souřadného systému, dostaneme zápis ve formě:

$$A(t) = \begin{bmatrix} p_x(t) & o_x(t) \\ p_y(t) & o_y(t) \\ p_z(t) & o_z(t) \end{bmatrix}$$

kde vektorová funkce  $p(t)$  reprezentuje polohu nástroje a funkce  $o(t)$  reprezentuje natočení nástroje. Důležité je, aby výsledný vektor  $o(t)$  měl přesně opačný směr k normálovému vektoru plochy. Tím se zaručí, že nástroj bude správně orientovaný.

### 3.6.3 Charakteristika rozstříku

Ve většině odborných článcích je předpokládáno, že distribuce barvy odpovídá kruhovitému průřezu s možností aplikace distribučního modelu, takového, který nejvíce odpovídá změřeným datům. Také se předpokládá, že tryska zaujímá polohu rovnoběžnou s normálou k ploše, tak jako je tomu na obrázku 18.



**Obr. 18:** Poloha trysky vůči povrchu [9]

kde  $\theta$  je úhel rozstříku,  $h$  představuje vzdálenost povrchu od trysky a  $R$  je aplikační rádius.

**Mezi nejčastěji používané distribuční modely patří:**

1. Parabolický distribuční model
2. Gaussův distribuční model
3. Beta distribuční model

### 3.6.4 Výstupní kód trajektorie pro roboty ABB

Roboty od společnosti ABB využívají vlastní programovací jazyk RAPID. Ten se řadí mezi programovací jazyky vysoké úrovně a disponuje vlastnostmi jako například: multitasking, modulární programování, automatické zpracování chyb (Automatic error handling), možnost tvořit vlastní funkce a procedury.

Pro snadné použití aplikace je vhodné vygenerovat kód, který se bude moci přímo použít pro provedení simulace. Případ, jímž se práce zabývá, využije pouze malou část funkcí tohoto jazyka. Převážně se bude jednat o definici proměnných a definici pohybových instrukcí. Veškeré potřebné vlastnosti jsou popsány v této kapitole.



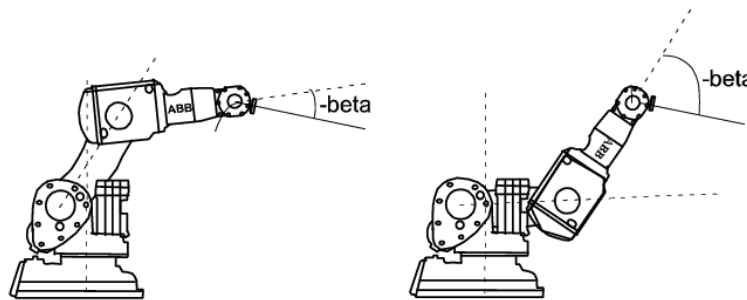
## Deklarace bodů

Proměnné, neboli body (v tomto případě) se mohou deklarovat ve tvaru:

```
CONST robtarget p10 := [ [x, y, z], [a, b, c, d],  
[p1, p2, p3, p4], [ e1, e2, e3, e4, e5, e6 ] ];
```

kde, `CONST` říká, že se jedná o konstantu; `robtarget` říká, že se jedná o bod kam má robot dojet a `p10` je identifikátor bodu, kterému přísluší přiřazené informace. `[x, y, z]` jsou souřadnice bodu v kartézském souřadném systému robota, `[a, b, c, d]` definují natočení nástroje v kvaternionech<sup>1</sup>, `[p1, p2, p3, p4]` určuje konfiguraci robota a `[ e1, e2, e3, e4, e5, e6 ]` definuje natočení externích os pro systémy využívající koordinované pohyby s externí osou/osami.

Konfigurací se rozumí poloha částí robota v jednotlivých kvadrantech. Více informací ohledně konfigurací je možné získat v technickém manuálu *confdata - Robot configuration data* [5].



Obr. 19: konfigurace robota [5]

## Deklarace pohybů

Pohybové instrukce se mohou deklarovat ve tvaru:

```
Movement ToPoint, Speed, Zonedata, Tool;
```

kde `Movement` je typ pohybu, jakým se má robot dostat do cílové destinace; `ToPoint` určuje bod, kterého má robot zvoleným pohybem dosáhnout; `Speed` určuje rychlost pohybu koncového bodu nástroje; `Zonedata` vymezuje toleranční zónu v jaké se může koncový bod nástroje pohybovat a `Tool` je aktivní pracovní nástroj.

<sup>1</sup>Bližší informace o kvaternionech jsou uvedeny v kapitole 4.10 (generování výstupů)

#### 3.6.4.1 Movement (Způsoby pohybu)

**MoveL** je pohybová instrukce, která definuje pohyb robota mezi dvěma body tak, že robot jede po přímce. Nevýhodnou tohoto pohybu je nižší rychlost a větší riziko singulárních poloh.

**MoveJ** je instrukce, která definuje pohyb tak, že robot mezi dvěma body nemusí jet po přímce a může si trasu upravovat dle zvoleného tolerančního pole. Tento typ pohybu je méně náchylný na singularitu a je rychlejší, než je tomu u pohybu způsobem MoveL. Jelikož není trajektorie pohybu přesně známá, je nutné dbát zvýšené opatrnosti kvůli vzniku kolizím.

**MoveC** definuje pohyb po kružnici.

#### 3.6.4.2 ToPoint

**ToPoint** nám pouze označuje jaký je cílový bod pohybu.

#### 3.6.4.3 Speed

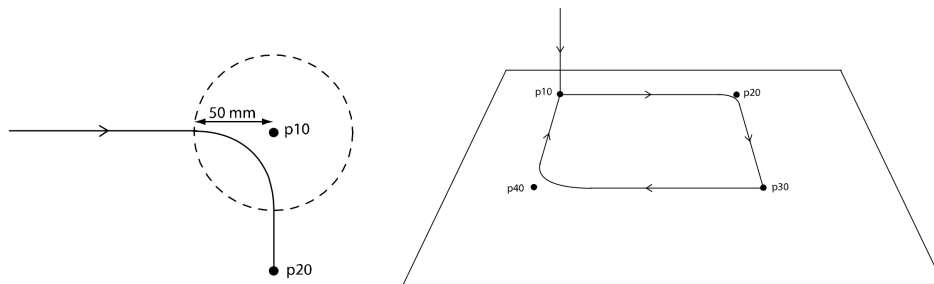
Definuje maximální rychlost pohybu mezi dvěma body. Hodnoty rychlosti jsou deklarovány v mm.s<sup>-1</sup> a jejich zápis se provádí ve tvaru: vxxx (v5, V10, V7000).

#### 3.6.4.4 Zonedata

**Zonedata** určují toleranci, jakou má robot do cílového bodu najet. Příkladem je situace znázorněná na obrázku 20. Robot má příkaz jet lineárním pohybem MoveL z bodu p00 do bodu p20, přes bod p10. V bodě p10 je nastavená zóna z50, což znamená, že robot může po najetí do zóny pohyb optimalizovat okolo definovaného bodu ve vzdálenosti 50 mm nemusí dosáhnout přesně bodu p10. Tato vlastnost velice ovlivňuje rychlost robota a následně i čas cyklu.

Pro definici zón se používají zóny **z0** až **z200**, kde robot body projíždí (metoda fly-by points). Speciální zóna, kdy robot zastaví přímo v definovaném bodě má označení **fine**. Tato instrukce by měla být primárně používána pro najíždění do bodů, v kterých následuje akce jiná, než pohybová (uchopení, bodové svaření, atd.).

V našem výzkumu se zabýváme především modelování lakovacích systémů, které využívá pro přenos kapek kombinaci obou mechanismů. Pro takový případ se ve většině technologií používá technologická hlavice, která je v ose povrchové normály. Její vzdálenost



**Obr. 20:** Znáznění zóny pohybu robota [3] [4]

od povrchu se pohybuje od 15 do 40 cm, v závislosti na technologii a parametrech procesu.

## 3.7 Přístupy hledání trajektorie aplikátoru

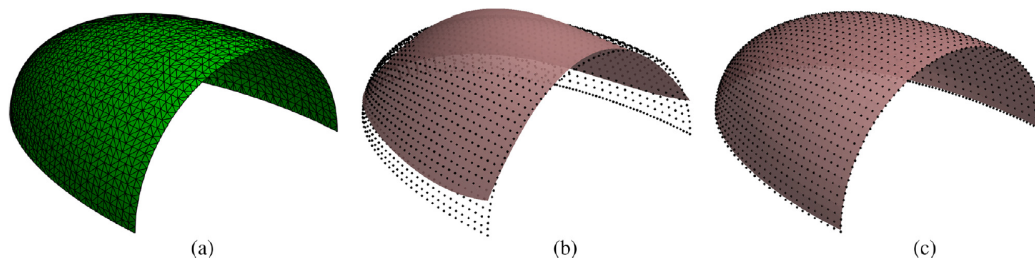
### 3.7.1 Standardní analytická metoda

V dnešní době existuje široká škála algoritmů vhodných pro generování trajektorie nástrojů pro obecné plochy a objemy. Některé z nich jsou určeny na generování trajektorie pro CNC stroje [14] s předpokladem, že počet říditelných os stroje je 4 a méně. Zároveň tyto metody používají pro generování trajektorie plochy aproximaci sítě (které je v řadě případů tvořena mnohostěny).

Standardním přístupem, který převádí nehladké plochy sít'ového modelu v plochy po částech hladké je aproximace využívající metodu Non-Uniform Rational Basis Spline (NURBS). NURBS metoda se vyznačuje možnostmi vytvořit povrchy s různou úrovní geometrické návaznosti. Tj. poziční návaznost, označována také G0, tangenciální návaznost (G1) a křivková návaznost (G2). Druhým způsobem, jenž je využíván je aproximace pomocí polynomů.

Při hledání vhodného algoritmu pro plánování trajektorie je také dobré si uvědomit, že NURBS křivky mohou být poměrně jednoduše přeneseny do STL modelu, jelikož se jedná o zjednodušení. Opačně, rekonstrukce NURBS z STL modelu je náročná a nezaručuje v každém případě správný výsledek, jelikož z jednoduchého a méně přesného modelu se vytváří model složitý a přesný. Aby bylo možné co nejlépe provést rekonstrukci, bylo by zapotřebí použít neuronových sítí, které by pro dané předměty upravovaly sít' na základě předcházejících zkušeností [20]. Toto bychom mohli přirovnat k rekonstruování předmětů z fotografie, která nezachycuje daný předmět dostatečně detailně a dokonce může být i částečně poškozená. Do této doby není na trhu software, který dokáže plně automaticky a efektivně provést rekonstrukci předmětu. Nejvíce vyvinutý je CAD software Rhinoceros®, který umožňuje převádět modely z STL do NURBS formátu. Bohužel ale využívá pouze nejjednodušší plochy, které se nazývají bilinéární (Definovány NURBS křivkou 1. stupně). Pro polynomiální aproximaci velice dobře funguje MATLAB® surface fitting toolbox. Bohužel proces také není optimální a pro uspokojivé výsledky je nutné používat polynomy vysokých stupňů. Důkazem může být například test z práce Carmela Mineoa, který provedl analýzu se zaměřením na minimalizaci odchylky. Test proběhl na jednoduchém modelu, který se skládal z 1914 vrcholů a pro jeho ideální aproximaci bylo potřeba použít polynom 30. stupně.

Zároveň výpočetní náročnost roste exponenciálně s řádem polynomu. Pro uvedený test byl použit počítač osazený čtyřjádrovým procesorem i7 a operačním systémem Windows



**Obr. 21:** Aproximace polynomiální plochou, (a) STL model (b) polynom 3. stupně, (c) polynom 30. stupně[16]

10, kdy celkový výpočetní čas pro proložení polynomiální plochou 30. řádu dosáhl téměř 4 vteřin. [26].

Další velkou nevýhodou tohoto přístupu aproximace je ten, že polynomiální plochou mohou být proloženy pouze křivky, které jsou popsateľné surjektivní funkcí [26]. Tj. funkce  $z = f(x, y)$  s doménou  $X - Y$  a kodomenénou  $Z$  je definována jako surjektivní, pokud každý element  $z$  v  $Z$  má korespondující  $z = f(x, y)$ , kde funkce  $f$  může mapovat více než jednu dvojici proměnných  $x, y$  pro stejné  $Z$ . Naopak to ale nelze.

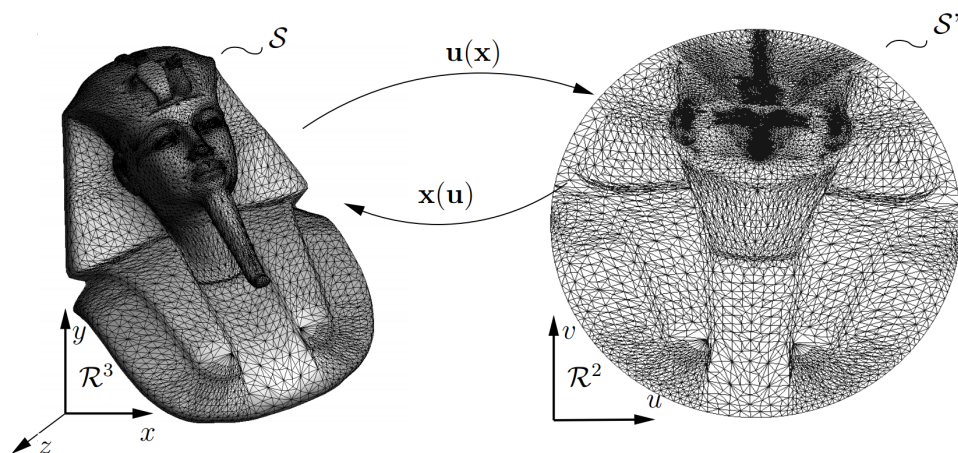
Z předcházejících poznatku je zřejmé, že použití polynomu, jako aproximační křivky/plochy je vhodné pouze pro jednoúčelové aplikace.

### 3.7.2 Plánování trajektorie použitím rekonstrukce Iso-parametrických ploch

V roce 2005 byl představen další algoritmus pro plánování trajektorie z trojúhelníkového síťového modelu pomocí rekonstrukce iso-parametrických ploch.

Algoritmus představený Sun Yuwenem využívá strategii [46] parametrizace trojúhelníkové sítě na harmonickou mapu mezi úseky promítnuté do 2D roviny a plochou ve 3D prostoru. Tímto namapováním se umožní pracovat s plochou ve 3D, jako kdyby se jednalo o plochu ve 2D. Rozlišení (CC), jakým je parametrizace provedena, závisí na přesnosti, jakou má být trajektorie hledána (dáno tolerancemi výrobního procesu).

Velkou výhodou této metody je, že trajektorie je navržena přirozeně až na samotný okraj součásti, což například u metody MFT není. Na druhou stranu metoda je limitována použitelností pro surjektivní plochy a tím pádem její praktická využitelnost v průmyslu není příliš dobrá.



Obr. 22: Příklad vytvoření harmonické mapy z STL modelu pro kruhový výřez [43]

### 3.7.3 Metoda sledování povrchu (MFT)

Metoda MFT (Mesh Following Technique) patří k nejnovějším přístupům hledání trajektorie na površích. Metoda si na rozdíl od standardních analytických metod vystačí se síťovým modelem, který může dosahovat vysoké složitosti.

Myšlenka MFT metody je založena na využití trojúhelníkové sítě modelu přímo pro vedení nástroje. Tím se odstraňuje nutnost provádět aproximaci plochy po částech hladkými plochami, které jsou pro určité modely v praxi jen těžce dosažitelné.

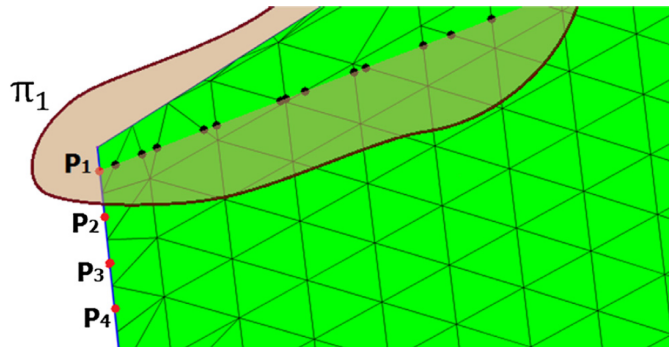
Pro vysvětlení metody je použit přístup z práce Camela Mineoa a jeho týmu [15], kde je jako vzorek použita konvexní plocha. U takové plochy můžeme předpokládat, že okraj je definován takovými body sítě, jež jsou vrcholy pouze dvou sousedních trojúhelníků. Směrový vektor mezi dvěma vrcholy (hrana) trojúhelníku má tvar:

$$\vec{v} = [u_i, v_i, w_i] \quad (7)$$

kde  $u_i$ ,  $v_i$  a  $w_i$  jsou složky podél os  $x$ ,  $y$  a  $z$ . Z tohoto vektoru můžeme snadno získat střední bod mezi vrcholy segmentu (hrany trojúhelníku). Hrana na které tento bod leží nazýváme referenční. Referenční proto, že nám právě tato přímka, definuje normálový vektor a bod roviny  $\pi$ , která je definovaná jako:

$$\pi_i : (u_i \cdot x) + (v_i \cdot y) + (w_i \cdot z) + d_i = 0 \quad (8)$$

Pro každý segment referenční hrany je možné nalézt body, které vzniknou průsečíkem mezi rovinou a hranami trojúhelníků sítě, tak jak je tomu na obrázku 23. Mezi referenčním bodem se dopočítávají vzdálenosti mezi po sobě jdoucími body, které se kumulativně sčítají až do okamžiku, kdy následující vypočtený bod přesáhne chtěnou rozteč průjezdů robota. Takový bod a bod jemu předcházející definuje přímkou, na které leží bod určující ideální vzdálenost průjezdu robota od bodu referenčního. Tento proces se opakuje pro všechny body nalezené na referenční hraně.



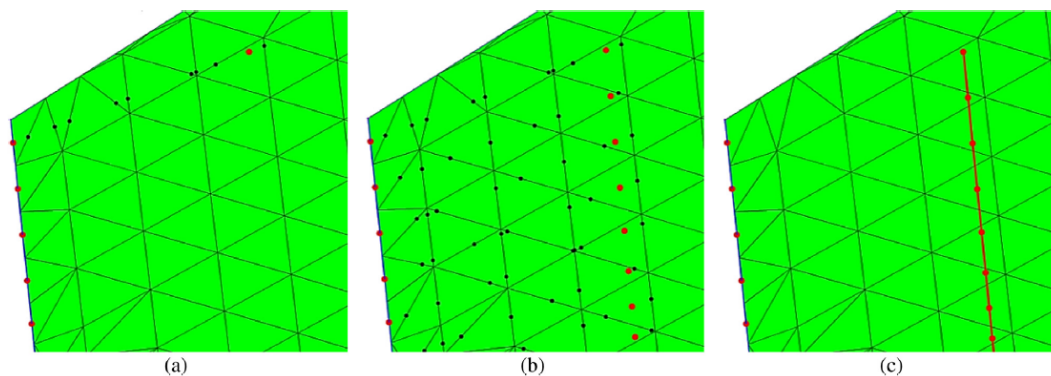
Obr. 23: Body vzniklé průsečíkem roviny se sítí modelu [16]

Nalezené body tímto postupem jsou následně použité pro definici trajektorie, tak jak je tomu na obrázku 24. Chyba s jakou je trajektorie pokládána na model odpovídá přímo chybě trojúhelníkové sítě od analytického modelu vytvořeného v CAD systému.

Z logiky algoritmu zmíněného výše je zřejmé, že výsledná trajektorie téměř nikdy nemůže konec generované křivky protnout vnější hranu sítě a algoritmus není tedy zcela ideální. Problém je také viditelný na obrázku 24(c).

Nicméně tento problém je odstranitelný metodou, která pomocí přidání dvou bodů propojí křivku s hranicemi modelu a zachová správný směr trajektorie. Metoda vychází z výpočtu minimální vzdálenosti mezi dvěma přímkami, jenž svírají ostrý úhel. Bližší informace jsou popsány v práci *Introducing a novel mesh following technique for approximation-free robotic tool path trajectories* [15].

Tato metoda může být mimo lakování použita v oblasti 3D tisku, svařování s přidaným materiálem či lepení. Bohužel i tato metoda nespĺňuje požadavky dnešní doby na 100% a nenahradí manuální vstupy člověka do struktury kódu pro roboty.



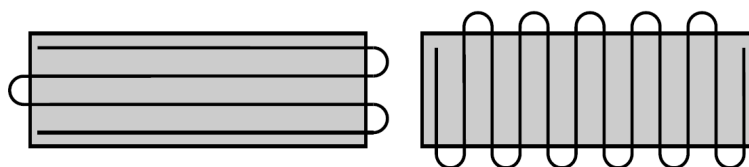
Obr. 24: Postup generování trajektorie [16]

### 3.7.4 Hledání trajektorie metodou Line-sweep

Algoritmy zmíněné v předcházejících kapitolách se zabývají primárně hledáním trajektorie na modelu v příslušném matematickém popise ploch. Tato kapitola je průvodcem jednoho z přístupů hledání trajektorie s důrazem na optimalizaci času cyklu se 100% pokrytí cílené vrstvy.

Cílem takového algoritmu je nalézt trajektorii, která zajistí, že každá část cílové plochy bude pokryta a zároveň bude optimalizovaný nějaký z parametrů (např. čas, tloušťka vrstvy). Ve většině aplikacích je to právě čas, který se stává kritickým kritériem produkce a zásadně ovlivňuje výrobní cenu produktů. Mnoho stávajících algoritmů využívá k hledání cesty rozklad plochy na segmenty a ty dále na pod-segmenty, které slouží jako vstup pro tzv. traveling-salesman algoritmus [37]. Ten vygeneruje sekvenci sub-segmentů, přes které by trajektorie měla být vedena. Tento algoritmus je standardně doplněn o další algoritmus (line-sweep) a algoritmus jež vyplňuje patřičný segment [27].

Při návrhu je nutné si uvědomit, že v okamžiku, kdy robot vyplní jednu řadu, musí se téměř zastavit, otočit směr a začít nanášet druhou řadu. Abychom minimalizovali čas, tak je logické, že musíme hledat takovou trajektorii, která odpovídá orientaci, kdy počet takových stavů je minimální (například trajektorie na obrázku 25 (vlevo) je výhodnější ve srovnání s trajektorií vpravo).



Obr. 25: Příklad způsobu položení trajektorie: horizontální (vlevo), vertikální (vpravo)[27]

Pro určení správného natočení cílové plochy byla použita Wesley H. Huangem funkce



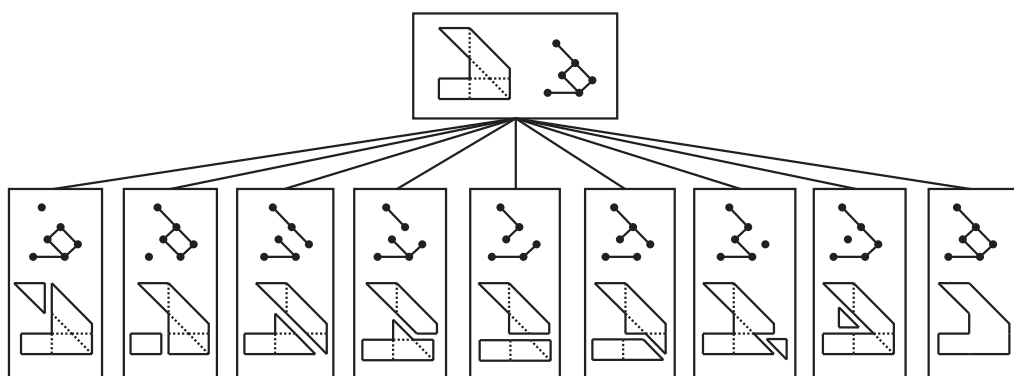
$S$ , která hledá minimální součet průměrových funkcí.

$$\min_{i \rightarrow N} \{S_i(\theta)\} = \min_{i \rightarrow N} \left\{ d_p(\theta) + \sum_{i=1}^N d_{H_i}(\theta) \right\} \quad (9)$$

kde  $d_p(\theta)$  je funkce průměrů v závislosti na natočení plochy v 2D prostoru.  $d_{H_i}(\theta)$  představuje průměrovou funkci děr. Cílem je najít takové natočení, aby  $S$  bylo minimální. Více informací je k dispozici v práci *Optimal Line-sweep-based Decompositions for Coverage Algorithms* [27].

V dalším kroku celého postupu je nutné formulovat problém formou dynamického programování. To provedeme tak, že pro sub-segmentovou dekompozici vytvoříme přidružený graf (Obrázek 26). Pak každý uzel představuje jeden sub-segment a každé dva uzly jsou spojeny hranou právě tehdy, když sub-segmenty sdílejí stejnou hranu [28]. Definujme si graf  $G$ , který představuje kompletní sub-segmentovou dekompozici. Ten rozdělíme na dvě separátní úlohy nižší náročnosti (podproblémy) a zároveň ponecháme graf v celku, tak jako je tomu na obrázku. Pokud graf  $G$  rozdělíme na dva podgrafy  $G_1$  a  $G_2$ , grafy musí dohromady obsahovat všechny uzly z  $G$ . [28] Řešení následně hledáme jako minimum funkce  $S(G)$ , kde:

$$S(G) = \min \left\{ C(G), \min_i \{S(G_1^i) + S(G_2^i)\} \right\} \quad (10)$$



**Obr. 26:** Graf a jeho možnosti rozložení[28]

**Závěrem:** Tento přístup řešení dokáže velmi efektivně řešit 2D úlohy a optimalizovat tak časovou náročnost spolu se zaručením 100% pokrytí. Bohužel není rozvinutý na 3D úlohy a tělesa s obecnými plochami. Další nevýhodou je exponenciální výpočtová náročnost ( $2^n$ , kde  $n$  je počet sub-segmentů). Z těchto důvodů není tato metoda vhodná pro implementaci v této práci.

### 3.8 Průzkum trhu

Za účelem prověření situace na trhu se softwary pro automatické hledání trajektorie v oblasti lakování byl provedený online průzkum trhu. Průzkum proběhl v rozsahu 12 hodin<sup>2</sup> s použitím vyhledávače Google. Celkem bylo nalezeno pro klíčová slova [software for robot painting; abb paint powerpack; kuka Painting software; Fanuc painting software; automatic path planner painting robotics Roboterbahn Malerei-Software; Software robotické lakování; trajektorie robotické lakování SW; robot master manual painting] nalezeno 9 uspokojivých výsledků.

#### Výsledky průzkumu odpovídají následujícímu seznamu SW:

1. ABB RobotStudio® - Painting PowerPac
2. Robotmaster
3. DELFOI PAINT
4. Inropa™ OLP CAD
5. CMA ROBOTICS SPA ITALY – AWPS
6. CMA ROBOTICS SPA ITALY – ADPS
7. Lecrob Robot Manager
8. PaintiXen
9. FANUC PaintPRO

Závěrem můžeme říci, že trh se softwary v oblasti lakování není přesycený. Z celkových devíti uspokojivých výsledků jsou pouze dva (Robotmaster a DELFOI PAINT) plně uspokojivé<sup>3</sup>. Dle marketingových podkladů oba softwary nabízí funkce automaticky nalézt trajektorii, vyhodnotit tloušťku vrstvy, vizualizace pokrytí a mnoho dalších.

Detailní informace pro každý software jsou uvedeny v příloze: *Průzkum trhu*.

---

<sup>2</sup>Započtený čas je včetně editace výsledků hledání

<sup>3</sup>Průzkum trhu se uskutečnil pouze na základě podkladů uvedených na internetových stránkách. To zda SW opravdu funguje, tak jak výrobce uvádí nebylo prověřováno

## 4 Část Praktická

V praktické části práce jsou detailně rozvedeny principy, které bylo nutné použít pro naplnění zadání. V první části je provedený návrh procesu použití aplikace. Tedy to, jak se bude v aplikaci uživatel pohybovat a jaké operace bude provádět. V druhé části je popsán návrh uživatelského rozhraní. V poslední části jsou detailně popsány hlavní postupy a funkce z kterých je aplikace složena.

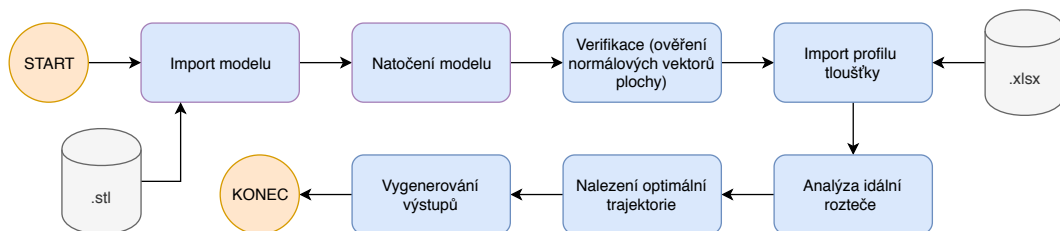
## 4.1 Návrh procesu použití

Uživatelsky příjemné a intuitivní prostředí aplikace je nezbytnou částí, kterou je nutné se při návrhu software zabývat. V tomto případě je aplikace dostatečně komplexní, skládá se z několika dílčích modulů a kroků, že je vhodné provést návrh procesu použití (use case).

Aplikaci lze použít jak k analýze modelu a nalezení trajektorie, tak i k určení optimální rozteče mezi průjezdy aplikátoru. Všechny tyto aplikace jsou proveditelné nezávisle na sobě a to umožňuje uživateli větší volnost používání. Standardně se můžeme setkat se třemi různými scénáři použití.

### 4.1.1 Celková analýza

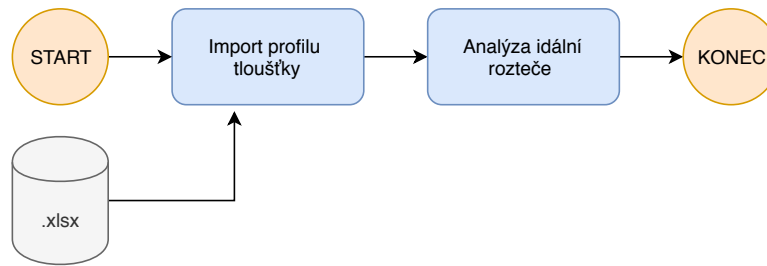
Celková analýza umožňuje uživateli projít celý proces. To znamená od importu modelu, až po samotné nalezení trajektorie a vygenerování kódů. Jak je celý proces realizován z uživatelského hlediska je znázorněno na obrázku 27.



Obr. 27: Diagram použití aplikace

### 4.1.2 Analýza profilu vrstvy

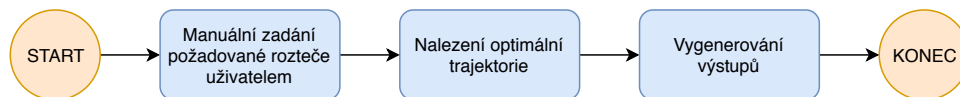
Jeden z oddělených celků aplikace je analýza profilu vrstvy, jejímž výsledkem je optimální rozteč průjezdu. Jako vstupní data je použitý Excel soubor formátu xlsx. Pro vstup ale není nutné mít k dispozici model, a tudíž aplikace může být používána pouze k tomuto účelu. Pro takový případ vypadá diagram použití uživatelem následovně: (Obrázek: 28)



**Obr. 28:** Diagram procesu pouze pro analýzu tloušťkového profilu

### 4.1.3 Hledání optimální trajektorie

Samozřejmostí aplikace je také možnost provést nalezení optimální trajektorie bez ohledu na znalost profilu nanášené vrstvy. To znamená, že je možné zadat informaci o požadovaném rozestupu průjezdů manuálně. Pro takový případ odpovídá postup uživatele následujícímu diagramu na obrázku 29.



**Obr. 29:** Diagram procesu pouze pro generování optimální trajektorie

## 4.2 Návrh uživatelského rozhraní a jeho ovládání

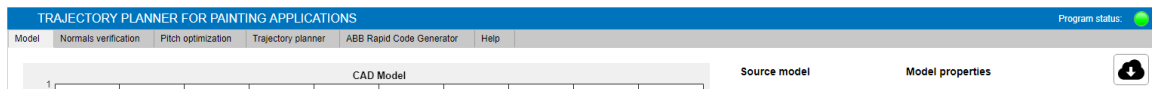
Uživatelské rozhraní bylo vytvořeno v nadstavbě programu MATLAB nazývaným *App Designer*. Tento nástroj umožňuje efektivně vytvářet desktopové a webové aplikace. App Builder kombinuje primárně dvě úlohy nezbytně nutné pro vytváření aplikací. A to, vytváření grafického uživatelského rozhraní (GUI) a programování jeho chování spolu s propojením se scripty a funkcemi.

Cílem navrženého uživatelského rozhraní je logicky a systematicky provést uživatele programem tak, aby co nejnadhěji dosáhl svého cíle.

Aplikace je rozdělena do šesti bloků (záložek), které odpovídají svým logickým rolím. Bloky jsou přepínány z menu, které umožňuje uživateli pohyb skrz všemi funkcemi aplikaci.

V pravém horním rohu je umístěna signalizační kontrolka zaneprázdněnosti aplikace. Pokud svítí červeně, program vykonává požadovanou úlohu a nelze spustit v tentýž okamžik úlohu jinou. V případě, že je kontrolka rozsvícena zeleně, program je připravený k vykonání požadované úlohy.

Dále uživateli pomáhá v orientaci postupné povolování tlačítek podle toho jaké kroky mohou teoreticky následovat. Tato vlastnost také zamezuje nepředpokládaným případům použití (např. spuštění analýzy bez vybraného modelu).



Obr. 30: Pohled na hlavní menu okna aplikace

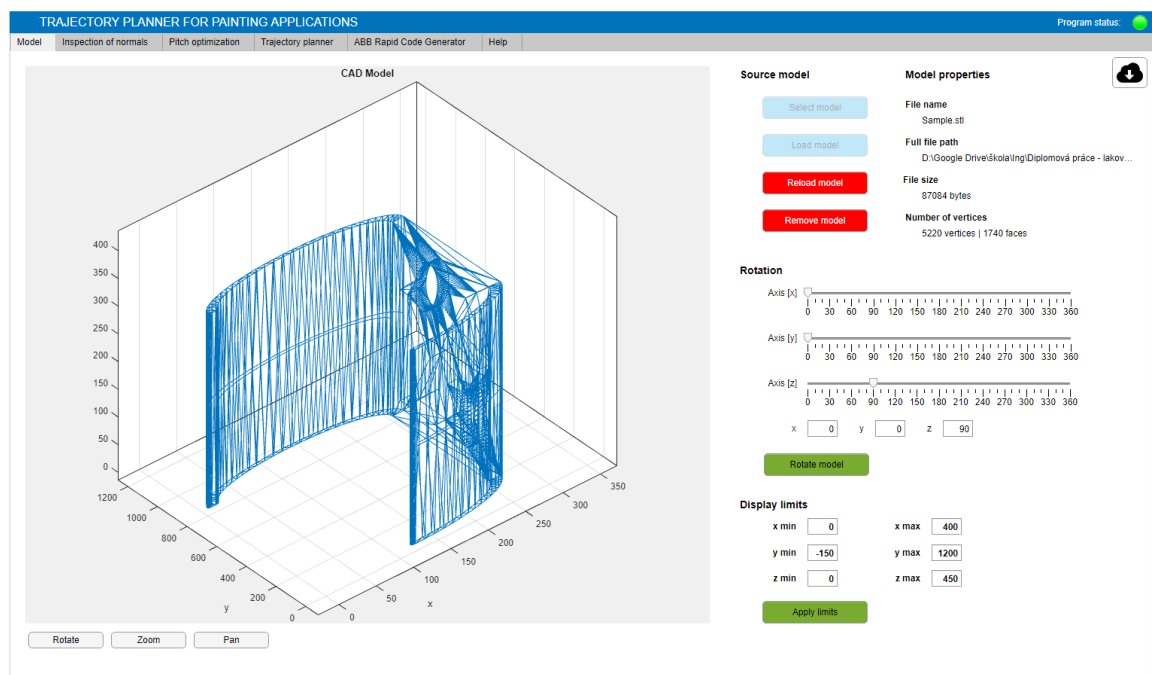
Vybrané bloky a zvolené metody spojené s UI jsou stručně popsány v následujících kapitolách.

#### 4.2.1 Záložka: Model

Záložka *Model* provádí uživatele správou zdrojového modelu, který si přeje uživatel vyšetřovat.

Uživatel má také možnost souřadnice modelu rotovat podle osy  $x$ ,  $y$  a  $z$ . Tato funkce je do aplikace přidána z toho důvodu, že hledání trajektorie je implementováno pouze metodou hledání průsečíků modelu s rovinou  $xy$ .

Dále je uživateli umožněno modelem natáčet, posouvat a přibližovat. V poslední řadě si uživatel může nastavit vlastní limitace zobrazení jednotlivých os.



Obr. 31: GUI - Správa modelu

## 4.2.2 Záložka: Inspection of normals

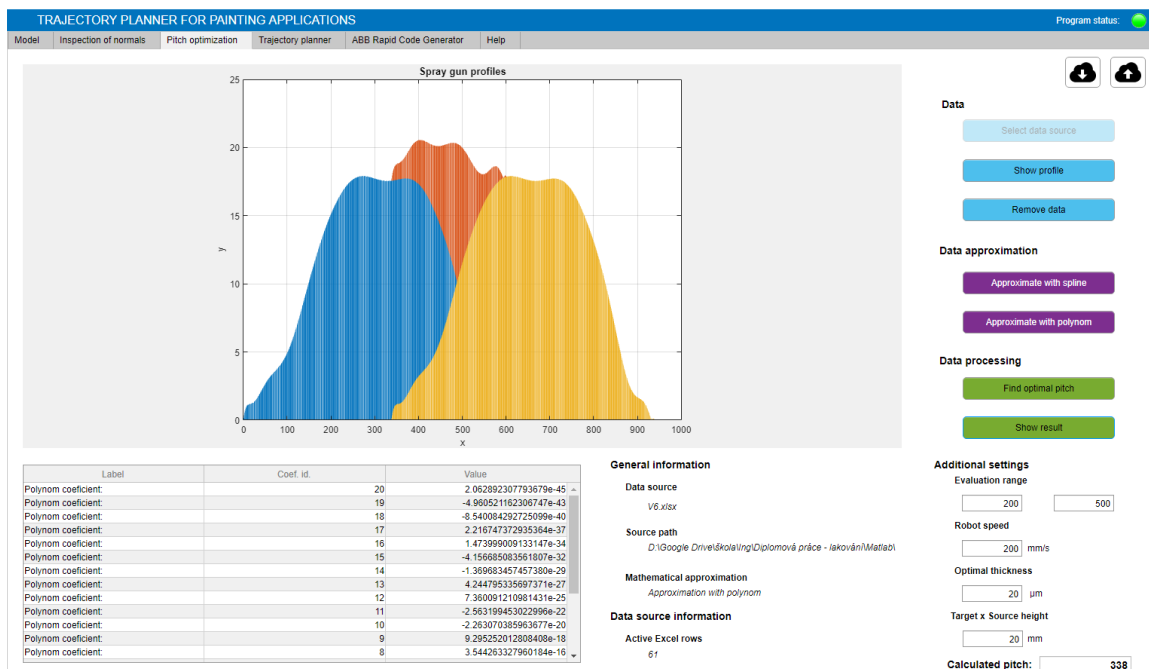
Na druhé kartě má uživatel možnost ověřit, zda model vykazuje standardní chování při výpočtu normál. Jelikož kontrola není automatická, uživatel má pro lepší inspekci opět možnost modelem otáčet, přibližovat a posouvat.

## 4.2.3 Záložka: Pitch optimization

Sekce *Pitch optimization* se zabývá hledáním optimální rozteče průjezdů robota od sebe. Uživateli se nabízí možnost importovat soubor formátu .xlsx obsahující výškové souřadnice vrstvy v bodech.

Uživatel může nastavit parametry jako: Vyhodnocovaný interval (Evaluation range), rychlost robota (Robot speed), požadovanou optimální tloušťku vrstvy (Optimal thickness) a vzdálenost mezi nástrojem a cílovým bodem na povrchu plochy (Target x Source height).

V záložce je také tabulka zobrazující hodnoty odpovídající zvolené aproximaci pro případ, že by si uživatel přál matematický popis aproximace zkopírovat.



Obr. 32: GUI - Analýza tloušťkových profilů

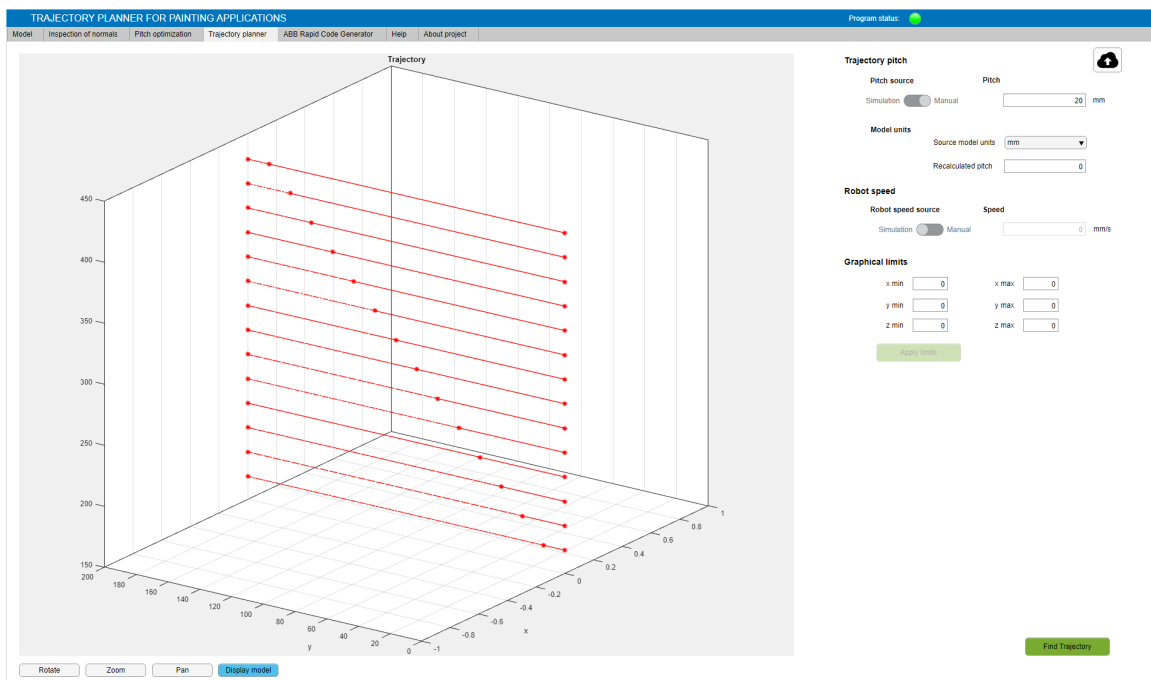
## 4.2.4 Záložka: Trajectory planner

Jak již anglický název napovídá, tento blok se zabývá hledáním optimální trajektorie. Pokud uživatel provedl analýzu tloušťky vrstvy a získal tak optimální rozestup mezi průřezdy, hodnota se automaticky propíše do tohoto okna. Aby nebylo vždy nutné analýzu rozestupu provádět, je možné hodnotu zadat manuálně.

Bohužel STL formát nedodrží jednotné měřítko a hodnoty souřadnic bodů se budou odvíjet od toho, v jakých souřadnicích byl model vytvořen a vygenerován. Z tohoto důvodu je do GUI přidán přepínač jednotek modelu (milimetry, centimetry a palce).

Z vizualizačních důvodů modelu je také možné měnit zobrazované intervaly os, modelem otáčet, posouvat a přibližovat.

Defaultně je trajektorie vykreslena do prostoru bez drátového modelu. Ten má uživatel možnost zobrazit kliknutím na tlačítko *Display model*.



Obr. 33: GUI - Ukázka nalezení optimální trajektorie na obdélníkové desce

## 4.2.5 Záložka: ABB Rapid code generator

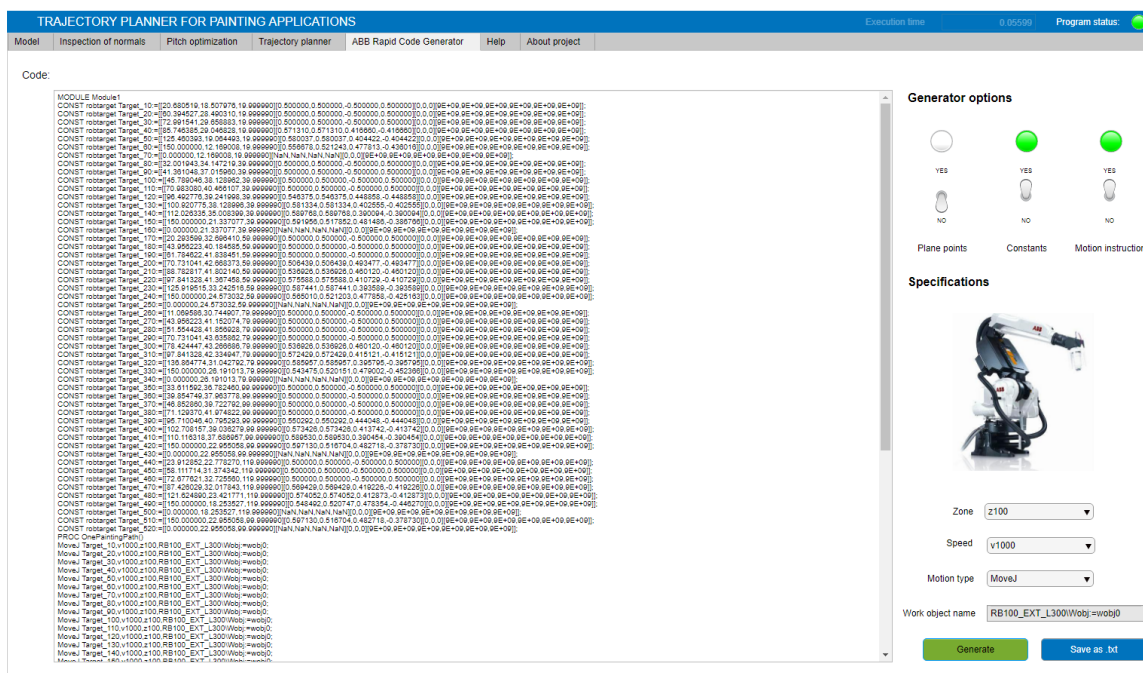
Jeden z cílů projektu bylo také automatické vygenerování pohybových instrukcí pro roboty od společnosti ABB. Právě tato úloha se odehrává na kartě ABB Rapid code



generator.

Uživatel má možnost vygenerovat celkem tři výstupy. Prvním z nich je kód obsahující souřadnice vodičích bodů spolu s natočením nástroje v kvaternionech. Druhá možnost je vygenerování bodů v RAPID struktuře. Jako poslední možností je vygenerování pohybových instrukcí opět v zápise RAPID syntaxe.

Samozřejmostí je také možnost zvolit rychlost robota, nastavit zónu, vybrat způsob pohybu a definovat název end-efektoru. V poslední řadě si uživatel může výstup uložit do textového souboru typu TXT.



Obr. 34: GUI - Ukázka karty pro generování výstupu

## 4.2.6 Záložka: Help

Asi každý sofistikovanější produkt by měl obsahovat přinejmenším stručný návod s nápovědou. Tato aplikace není výjimkou a je do ní implementována stručná nápověda, jež uživateli v řadě případů zodpoví kladené otázky.

### 4.3 Načtení a vizualizace modelu

Zvolený typ CAD souboru, s kterým aplikace pracuje je STL (Standard Tessellation Language). Ten se vyskytuje ve dvou formách, a to ASCII STL a Binary STL. Zápís ve formátu ASCII má tvar [12] [47]:

```
1     solid name
2         facet normal ni nj nk
3             outer loop
4                 vertex v1x v1y v1z
5                 vertex v2x v2y v2z
6                 vertex v3x v3y v3z
7             endloop
8         endfacet
9     .
10    .
11    .
12    endsolid name
```

Kde každý vrchol trojúhelníku je definován třemi souřadnicemi v kartézském souřadném systému. Tento zápis obsahuje také informaci o normále plochy **facet normal ni nj nk**, která je orientována podle posloupnosti vektorů v zápise. Normálu jsme ale schopni vypočítat ze znalosti pořadí bodů a pravidla pravé ruky. Soubor tedy nese redundantní informaci.

Normálový vektor můžeme vypočítat dle vztahu:

$$n = \frac{(\vec{v}_2 - \vec{v}_1) \times (\vec{v}_3 - \vec{v}_1)}{|(\vec{v}_2 - \vec{v}_1) \times (\vec{v}_3 - \vec{v}_1)|} \quad (11)$$

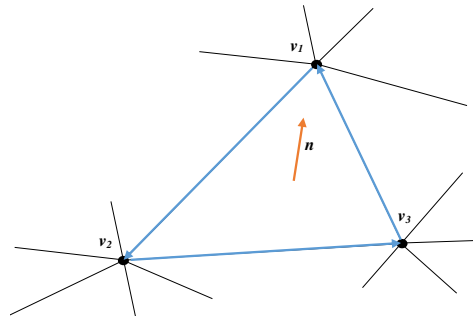
Tento typ zápisu STL formátu je velice přehledný, na druhou stranu, ale zabírá podstatně více místa, než je tomu u typu Binary STL. Ten je nositelem stejné informace, pouze v kompaktnější formě a jeho zápis má následující tvar [12]:

```

1  UINT8 [80] Header
2  UINT32 Number of triangles
3
4  foreach triangle
5      REAL32 [3] Normal vector
6      REAL32 [3] Vertex 1
7      REAL32 [3] Vertex 2
8      REAL32 [3] Vertex 3
9      UINT16 Attribute byte count
10 end

```

Prvních 80 znaků je vyhrazeno pro název souboru. Dále je uvedený počet trojúhelníku popisující soubor a následuje popis každého trojúhelníku, jež je popsán dvanácti 32bitovými čísly. Z důvodů kompaktnější velikosti byl zvolen vstupní soubor aplikace **Binary STL**.

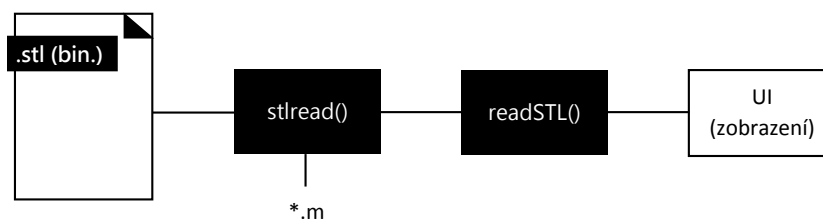


**Obr. 35:** Normálový vektor plochy trojúhelníku

Načtení modelu do aplikace je vyřešeno použitím STL File Readeru [32], který je napsán Erikem Johnsem. Tato funkce (`readstl()`) vrací data ve tvaru  $[f, v, n]$ , kde  $f$  - pořadové číslo trojúhelníku,  $v$  - souřadnice vrcholů a  $n$  - normálový vektor.

Vizualizace modelu je provedena funkcí `readSTL()`, která získává data z funkce `stlread()`, a samotné zobrazení je řešeno funkcí `patch()`, jež je vhodná pro zobrazování jednoho či více vyplněných polygonů.

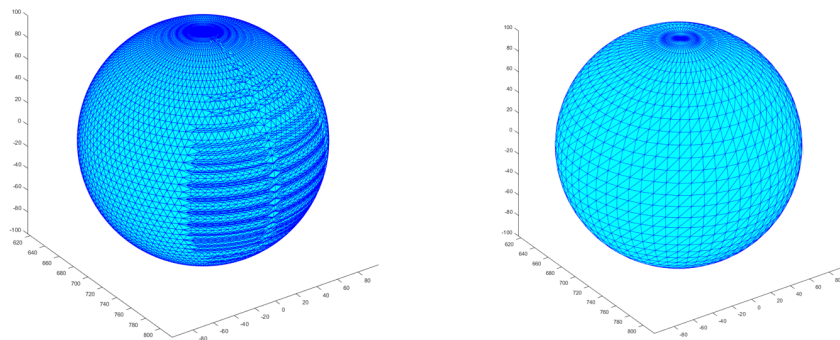
Datové toky mezi jednotlivými `.m` soubory jsou zobrazeny na následujícím diagramu.



**Obr. 36:** Diagram importu STL souboru

## 4.4 Výpočet normálových vektorů povrchu

V předcházející kapitole bylo řečeno, že ve zdrojovém souboru modelu jsou uvedené normálové vektory pro jednotlivé trojúhelníky. Tyto vektory jsou ale automaticky generované softwarem, který nemusí operace provést správně. Příkladem může být porovnání chybně vygenerované STL sítě pomocí softwaru FreeCAD, kde došlo k nevhodnému segmentování a zároveň ke špatně napočítaným normálovým vektorům. Pravým opakem je správně vygenerovaná síť softwarem Autodesk Inventor Professional 2019.



**Obr. 37:** Porovnání kvality exportu do STL. FreeCAD (vlevo), Autodesk Inventor Professional (vpravo).

Abychom si byli jisti, že jsou normálové vektory plochy správně vypočteny, je provedený vlastní výpočet a zároveň je provedena kontrola sekvence vrcholů. V případě, že vrcholy nejsou ve správném pořadí, jak je očekáváno, dojde k výpočtu vektoru opačného, jež souhlasí s referenčním směrem. Jako základ byl použit algoritmus od Adama H. Aitkenheada [6].

### 4.4.1 Algoritmus výpočtu normál a přidružených operací

Datový tok výpočtu normál v rámci projektu je zobrazený na obrázku 38. Zdroj dat pro výpočty, tedy funkci `ComputeNormals()` poskytuje `stlread()`. Dále funkce posílá vyhodnocená data do `DisplayNormals()`, kde dochází k překreslení jak samotného objektu, tak i normálových vektorů.

Po načtení dat je vhodné provést přeskupení do jednoho pole. To provedeme tak, že nejprve spočítáme počet ploch (trojúhelníků) a alokujeme pole o velikosti  $N \times 3 \times 3$ , kde  $N$  je počet trojúhelníků. Následně je pole naplněno souřadnicemi vrcholů a připraveno pro další výpočty.



**Obr. 38:** Datové toky v rámci výpočtu povrchových normál

**Kód 1:** Algoritmus přeskládání polí

```

1   ArrayOfVertices = zeros(size(TriangleFaces,1),3,3);
2
3   for i = 1:size(TriangleFaces,1)
4       ArrayOfVertices(i,:,1) = Vertex(TriangleFaces(i,1),:);
5       ...
6   end
  
```

V praxi se většina algoritmů na modelování těles omezuje na tělesa manifoldního typu. To znamená, že těleso existuje v reálném světě a nemá nulový objem. Lze také říci, že manifold těleso je takové, které má každý vrchol polygonu a hranu polygonu sdílenou s dalším polygonem či vrcholem tělesa. [49]

Vlastnost, zda se jedná o manifold těleso lze ověřit následujícím výpočtem.

$$V + F - E = 2 \tag{12}$$

kde  $V$  je počet vrcholů,  $F$  je počet stěn a  $E$  je počet hran. Tato podmínka nebude nikdy splněna v případě modelování ploch, jelikož jsou zakončeny "ostrým" koncem. Může se ale stát, že tato podmínka nebude splněna i u jiných těles, které byla chybně převedena do formátu STL, či chybně vymodelována.

Ověření, zda je těleso manifoldní je kontrolováno v `CalculateNormals()` následujícím algoritmem. Ten kontroluje, zda jedna hrana je současně hranou pro dvě stěny.

### Kód 2: Pseudokód pro ověřování STL modelu 1/2

```
1   % Alocate variables
2   Waiting a Checked
3
4   WHILE Cheked Every vertice (V)
5       % Choose point B and A
6       B = A + 1
7
8       Find point which match to other point A and B
9
10      IF match is not found THEN
11          Show warning and ask user if program should continue.
12          Set value ManifoldRisk = 1
13      END
14  END
```

Pokud importovaný model není manifold a zároveň se uživatel rozhodne pokračovat ve výpočtech, proces přeskočí verifikaci správnosti pořadí vrcholů (V) a provede výpočet normál. Modely, které uspějí v prvním testu jsou podrobeny testu, v kterém se kontroluje správnost pořadí vrcholů.

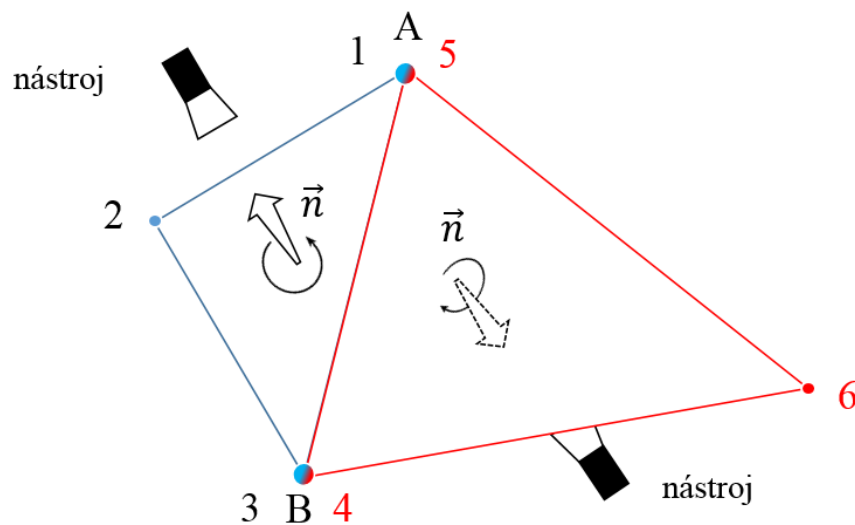
Test je založený na předpokladu referenčního směru, který určuje první analyzovaná stěna. Program si zvolí hranu, která přísluší bodu A a bodu B ( $B = A + 1$ ), "podívá" se zda tato kombinace bodů byla ověřena. Pokud ano, program pokračuje ověřením další kombinace. V opačném případě provede ověření podle následujícího algoritmu.

### Kód 3: Pseudokód pro ověřování STL modelu /2/2

```
1   % Get matched points A, B
2
3   IF A was not checked AND B was not checked THEN
4
5       % check if the edge is in the oposite direction
6       % to the original edge
7       IF B - A = 1 OR B - A = -2 THEN
8           Flipp direction and replace data in ArrazOfVertices
9       END
10  END
```

Ověření správné souslednosti vrcholů je bezpodmínečně nutné, jelikož vrcholy a jejich

pořadí určují směr normálového vektoru plochy. Ten následně slouží pro určení, z jakého směru má nástroj k ploše najet. Logicky by byla velká chyba, kdyby nástroj najížděl skrze těleso, jako je tomu na obrázku 39 a způsobil tak kolizi nástroje s cílovým objektem.



**Obr. 39:** Vliv souslednosti bodů na orientaci normálového vektoru

Poté co jsou vstupní data pro výpočet normálových vektorů `ArrayOfVertices` zkontrolována a opravena, je provedený jejich výpočet.

Výpočet normál probíhá ve smyčce, jejíž opakování je definováno počtem stěn modelu. Výstupem je tedy takový počet normálových vektorů v poli, jako je počet stěn modelu. Samotný výpočet je proveden pomocí výpočtu dvou vektorů, jež spolu svírají úhel na sousedních hranách stěny.

$$\text{Vector}AB = ab_1i + ab_2j + ab_3k \quad (13)$$

$$\text{Vector}AC = ac_1i + ac_2j + ac_3k$$

V předposledním kroku je provedený výpočet vektorového součinu, jehož výsledkem je normálový vektor.

$$\text{CrossProduct}AB \times AC = \text{Vector}AB \times \text{Vector}AC = \quad (14)$$

$$\begin{vmatrix} i & j & k \\ ab_1 & ab_2 & ab_3 \\ ac_1 & ac_2 & ac_3 \end{vmatrix}$$

Aby zobrazení vektorů bylo pro všechny stěny stejné, převedeme normálové vektory na jednotkové použitím vztahu:

$$CrossProductABxAC = \frac{CrossProductABxAC}{\|CrossProductABxAC\|} \quad (15)$$

#### 4.4.2 Vizualizace normál

Pohled na datový tok pro funkci `DisplayNormals()` je zobrazený níže. Funkce používá výstupní data z `ComputeNormals()` a zobrazuje na uživatelské rozhraní.



**Obr. 40:** Datové toky v rámci zobrazení povrchových normál

`DisplayNormals()` zobrazí STL model pomocí integrované MATLAB funkce `Patch()`. Následně se provede výpočet umístění vektoru na ploše. A to tak, že se vypočítá průměrná hodnota souřadnic ve směru x, y, a z. Tím se definuje bod, z kterého má normálový vektor vycházet. Koncový bod využije informaci o bodu na ploše a normálovém vektoru. Samotné zobrazení je realizované funkcí `plot3`, která slouží pro vykreslování křivek v 3D prostoru. Algoritmus, jakým způsobem je výpočet provedený je zobrazený níže.

**Kód 4:** Algoritmus výpočtu a zobrazení normálových vektorů

```

1   for i = 1:size(ArrayOfNormals,1)
2       NormalStart = mean(ArrayOfVertices(i, :, :), 3);
3       NormalEnd = NormalStart + 3*ArrayOfNormals(i, :);
4       plot3([NormalStart(1), NormalEnd(1)], _
5             [NormalStart(2), NormalEnd(2)], _
6             [NormalStart(3), NormalEnd(3)], ColourNormals, 'LineWidth', 2);
7   end
  
```

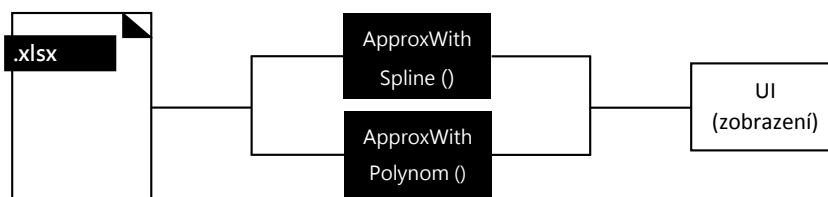


## 4.5 Aproximace tloušťkového profilu vrstvy

Součástí hledání optimálního lakovacího procesu je bezpochyby i nalezení vhodného přesahu tloušťkových profilů. Aby bylo možné hledat optimální rozteče mezi jednotlivými průjezdy atomizéru, je nejdříve nutné analyzovat naměřená data. Pro analýzu, respektive nalezení matematického popisu naměřených dat aplikace umožňuje provést interpolaci polynomem n-tého stupně a interpolaci pomocí kubického splinu.

Do systému není implementována kontrola, zda uživatel zvolil vhodnější variantu interpolace, jelikož je nutný jeho expertní odhad. Aplikace také neobsahuje předem definované charakteristiky atomizerů a je nutné importovat vlastní data, která mohou být výstupem měření nebo mohou být uměle vytvořena podle předpokladů distribuce barvy.

Skripty výpočtových funkcí komunikují přímo s uživatelským rozhraní, tak jak je uvedeno na diagramu 41.



Obr. 41: Datové toky pro aproximační funkce

Vstupní data musí splňovat určité náležitosti. Ty jsou specifikovány v tabulce 2.

Typ souboru	.xlsx
Sloupec A	Souřadnice (x) na zkoumaném vzorku, po jednom centimetru
Sloupec B	Tloušťky vrstvy v mikrometrech

Tab. 2: Požadavky na vstupní data pro analýzu tloušťkové vrstvy

### 4.5.0.1 Interpolace polynomem n-tého stupně

Interpolace polynomem n-tého stupně (polynomická regrese) je metoda, která prokládá zadané hodnoty polynomem n-tého stupně. Polynom je obecně definován jako:

$$p(x) = p_1x^n + p_2x^{n-1} + \dots + p_nx + p_{n+1} \quad (16)$$

kde  $p_n$  jsou koeficienty polynomu, který hledáme. Výpočet probíhá metodou nejmenších čtverců tak, že se snažíme minimalizovat odchylku mezi funkční hodnotou odpovídajícího polynomu a vstupní hodnotou. V softwaru MATLAB je implementace polynomicke interpolace možné provést vestavěnou funkcí `polyfit()`, použitou i v našem případě.

Funkce `polyfit()` vyžaduje zadat kromě vstupních dat i stupeň polynomu, pro který má koeficienty  $p_n$  hledat. Aplikace automaticky uživateli nabídne stupeň polynomu, který je odhadnut na základě počtu vstupních dat. Algoritmus, jak je stupeň polynomu odhadován je popsán níže. Je nutné brát ohled, že metoda vychází z ručního testování vhodnosti interpolačních modelů profilu s rozdílným počtem vstupů a hodnotami, takovými které od běžných atomizérů můžeme očekávat.

#### Kód 5: Postup pro hledání stupně polynomu

```
1      %Get array size
2      ArrayDim
3
4      if ArrayDim > 0
5          if ArrayDim > 24
6              PolDim = RoundUp(ArrayDim/3) - 1
7          else
8              PolDim = RoundUp(ArrayDim/2)
9          end
10     end
11
12     if PolDim > 40
13         PolDim = 40
14     end
```

Může se ale stát, že křivka nebude splňovat při doporučeném stupni polynomu požadovaných odchylek od naměřených dat. V tom případě má uživatel možnost nastavit požadovaný stupeň polynomu manuálně.

#### 4.5.0.2 Interpolace pomocí kubického splinu

Z vlastností polynomicke aproximace jedním polynomem je zřejmé, že v některých případech nemusí aproximace dosahovat takové přesnosti, jak bychom si přáli. Z tohoto důvodu je do aplikace implementována aproximace pomocí kubického splinu.

Kubický spline je definován jako křivka  $C$ , jež se skládá z dílčích polynomů třetího stupně. Každý z polynomů je definován na určitém intervalu a je navázán na polynom předcházející/nadcházející pomocí vazebních podmínek. Tento přístup aproximace vede k přesnějšímu proložení bodů. Na druhou stranu tato aproximace může vést k přeúčnění (overfittingu).

Matematicky je kubický spline definován jako [40]:

$$C(x) = \begin{cases} p_1(x), x_0 \leq x \leq x_1 \\ \dots \\ p_i(x), x_{i-1} \leq x \leq x_i \\ \dots \\ p_n(x), x_{n-1} \leq x \leq x_n \end{cases}$$

kde každý z polynomů má tvar:

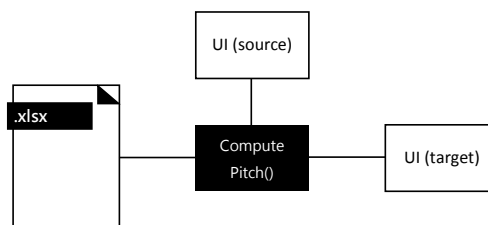
$$p_i = a_i + b_i x + c_i x^2 + d_i x^3 \quad (d_i \neq 0) \quad (17)$$

Implementace interpolace kubickým splinem je v aplikaci MATLAB vyřešena použitím integrované funkce `spline()` a její zobrazení je taktéž řešeno integrovanou funkcí.

## 4.6 Hledání optimální rozteče průjezdů

Hledání optimální rozteče jednotlivých průjezdů závisí na tloušťkovém profilu nanesené vrstvy, způsobem proložení, ale také i nastavení požadované výšky profilu. Všechny tyto parametry je nutné při výpočtech zohlednit. V dalších krocích simulace je také nutné znát rychlost pohybu aplikačního zařízení, při kterém vzorek vznikl a interval, na kterém má analýza vrstvy probíhat.

Funkce hledající optimální rozteč průjezdů `ComputePitch()` pracuje přímo se zdrojovým souborem XLSX, z kterého načte profil vrstvy. Dále funkce obdrží informaci z uživatelského rozhraní o tom, jaký způsob aproximace uživatel zvolil spolu s údajem o požadované optimální tloušťce a intervalu vyhodnocování. Výsledkem výpočtu je jediná hodnota určující rozteč v milimetrech.



Obr. 42: Diagram datových toků pro výpočet optimální rozteče

V našem případě má uživatel možnost zvolit ze dvou různých metod proložení dat. I přesto, že jsou metody podobné, výpočet se lehce liší a je nutné ho provést odlišným způsobem.

Metoda, která pro analýzu tloušťky vrstvy byla vyvinuta, je založena na minimalizaci objemové deviace mezi ideálním objemem a teoreticky naneseným objemem vrstvy.

Na obrázku 43 je znázorněna funkce  $f(x)$  na intervalu  $\langle a, b \rangle$ , která je výsledkem složení dvou totožných profilů  $g(x)$ , jako:

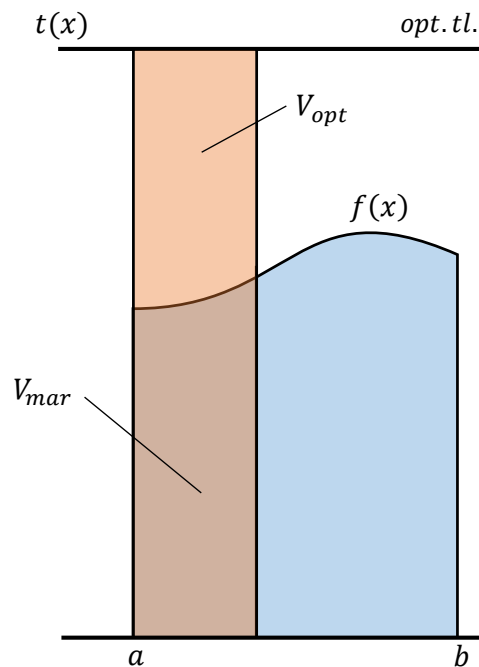
$$f(x) = g(x) + g(x + h) \quad (18)$$

Kde  $h$  je jejich vzájemné posunutí.

Cílem programu je nalézt takové posunutí, aby směrodatná odchylka objemových deviací mezi všemi objemy byla minimální. V matematickém zápise můžeme problém zapsat následovně:

$$\min_{i \rightarrow N} \left( \sqrt{\frac{1}{N-1} \sum_{i=1}^N (V_{mar_i} - V_{opt})^2} \right) \quad (19)$$

Realizace tohoto výpočtu ale vyžaduje získat nejdříve objemy pro jednotlivé intervaly křivky. Zároveň je nutné provést segmentaci křivky tak, aby odpovídala požadovanému měřítku. Za předpokladu, že vstupní datové body jsou od sebe vzdáleny rovnoměrně, lze provést segmentaci segmentů (pro případ spline) křivky  $C(x)$  na  $C'(x)$ .



**Obr. 43:** Teoreticky nanesená vrstva barvy

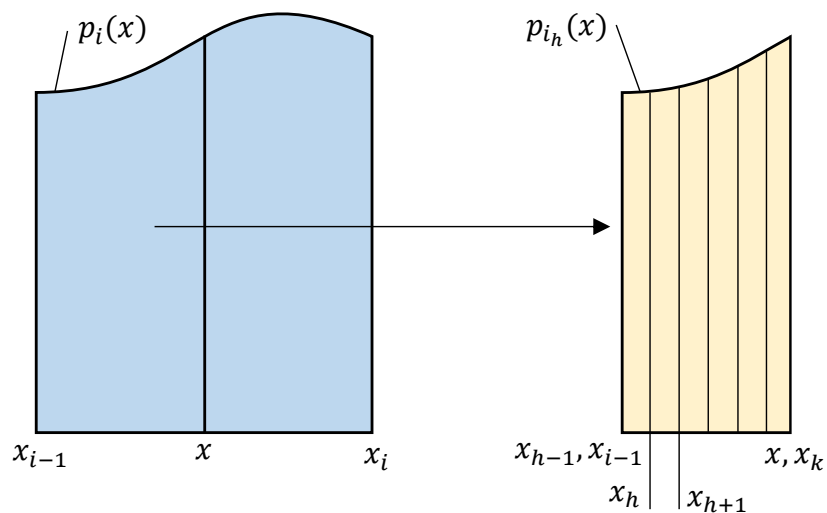
Pro segmentaci je zavedený parametr  $K$ , který definuje přesnost výpočtu. V případě, že body měření jsou snímány v intervalu 10 mm a požadavek výpočtové přesnosti činí jednotky milimetrů, zvolíme  $K = 10$ . Tím dojde k segmentaci křivky definované mezi dvěma body na 10 křivek definovaných mezi 11 body. Uměle vytvořené body jsou od sebe ve vzdálenosti  $s$ , kde

$$s = \frac{x - x_{i-1}}{K}. \quad (20)$$

Výsledkem rozdělení křivky je již zmíněná křivka  $C'(x)$  ve tvaru:

$$C'(x) = \left\{ \begin{array}{l} p_1(x), x_0 \leq x \leq x_1 \\ \dots \\ p_i(x), x_{i-1} \leq x \leq x_i \\ \dots \\ p_n(x), x_{n-1} \leq x \leq x_n \end{array} \right\} \left\{ \begin{array}{l} p_{1_1}(x), x_0 \leq x \leq x_0 + \frac{x - x_{i-1}}{K} \\ \dots \\ p_{1_h}(x), x_{h-1} \leq x_h \leq x_{h-1} + \frac{x - x_{i-1}}{K} \\ \dots \\ p_{1_K}(x), x_1 - \frac{x - x_{i-1}}{K} \leq x \leq x_1 \end{array} \right\}$$

Vizuálně je segmentování segmentů znázorněno na obrázku 44.



**Obr. 44:** Rozdělení intervalů

V dalším kroku jsou získány objemy segmentů  $V_{i_h}$  pomocí určité integrace křivek  $p_{i_h}$  jako:

$$V_{i_h} = \int_{x_{h-1}}^{x_{h-1} + \frac{x - x_{i-1}}{K}} p_{i_h}(x) dx \quad (21)$$

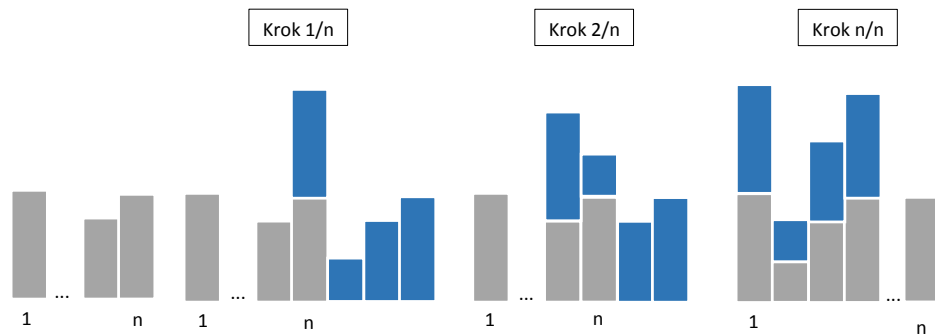
Prostředí MATLAB umožňuje postup implementovat pomocí cyklu, který začíná od prvního segmentu a končí posledním segmentem křivky definované na vyšetřovaném intervalu. Implementace algoritmu je detailně vysvětlena v následujících kapitolách. Zároveň je algoritmus výpočtu graficky znázorněn v diagramu na obrázku 47.

#### 4.6.1 Algoritmus výpočtu optimální rozteče překryvů

Výpočet rozteče je rozdělen do několika funkčních bloků, které vykonávají dílčí operace. V prvním bloku dojde k načtení surových dat z Excel souboru. Dále je vypočtený počet segmentů, na které je tloušťkový profil rozdělen a provede se alokace polí s obsahem nul. V okamžiku, kdy jsou pole připravená, program začne vykonávat první cyklus odpovídající postupu, jaký si uživatel zvolil.

Jak již bylo zmíněno, dle uživatelem zvolené strategie se provedou výpočty<sup>4</sup> objemů pro jednotlivé segmenty a uloží do pole. V ten samý okamžik proběhne výpočet ideálních objemů a uloží se do pole totožné velikosti, jako je tomu v případě objemů skutečných.

V dalším kroku dochází k prokládání tloušťkových profilů přes sebe a sčítání dílčích objemů tak, jako je tomu na obrázku 45. Aby bylo možné sčítat tato pole musíme zajistit, aby jejich velikost byla totožná.



Obr. 45: Průběh sčítání objemů

Je-li pole s objemy jednoho profilu velikosti  $[m \ n]$ , pak pole které pojme dvě pole  $[m \ n]$  vedle sebe, musí mít alokovanou velikost  $[2m \ n]$ . Pole s cílovou velikostí jsou ve funkci `ComputePitch()` nazvány jako *Range1*, *Range2*, *newRange1* a *NewRange2* a jejich obsahem jsou nuly. K těmto polím obsahující nuly jsou v každém kroku cyklu přičtena pole o stejných velikostech s posouvajícím se obsahem v každém kroku cyklu, jako je

<sup>4</sup>Způsob, jakým jsou objemy vypočteny je detailněji popsán v kapitole: Výpočet objemů spline křivky, Výpočet objemů polynomu n-tého stupně

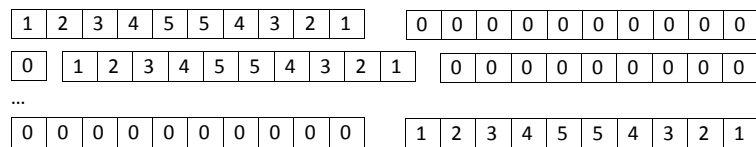
tomu na obrázku 46. Zároveň jsou hodnoty ukládány do pole `SumOfVal` a `ArrOfOptimal`, představující výstupy z tohoto logického bloku.

**Kód 6:** Algoritmus přípravy polí pro výpočet optimální rozteče průjezdů

```

1      % Work with arrays (regrouping, summation)
2      for i=1:1:ComputedSegments
3          Range1 = NewRange1 + [ArrayOfVolumesForProcessing ;
4                                 zeros(ComputedSegments,2)];
5          Range2 = NewRange2 + [zeros(i,2);ArrayOfVolumesForProces-
6                                 sing;zeros(ComputedSegments-i,2)];
7          SumOfVal(:,i) = Range1(:,2) + Range2(:,2);
8          ArrOfOptimal(:,i) = [ArrayOfOptimalVolumesForProcessing;
9                                ArrayOfOptimalVolumesForProcessing];
10     end

```



**Obr. 46:** Přeskupování pole pro jednotlivé kroky

Následuje výpočet přes všechny prvky pole `SumOfVal` a pole `ArrOfOptimal` podle vztahu:

$$V_{diff_{ij}} = V_{mar_{ij}} - V_{opt_{ij}} \quad (22)$$

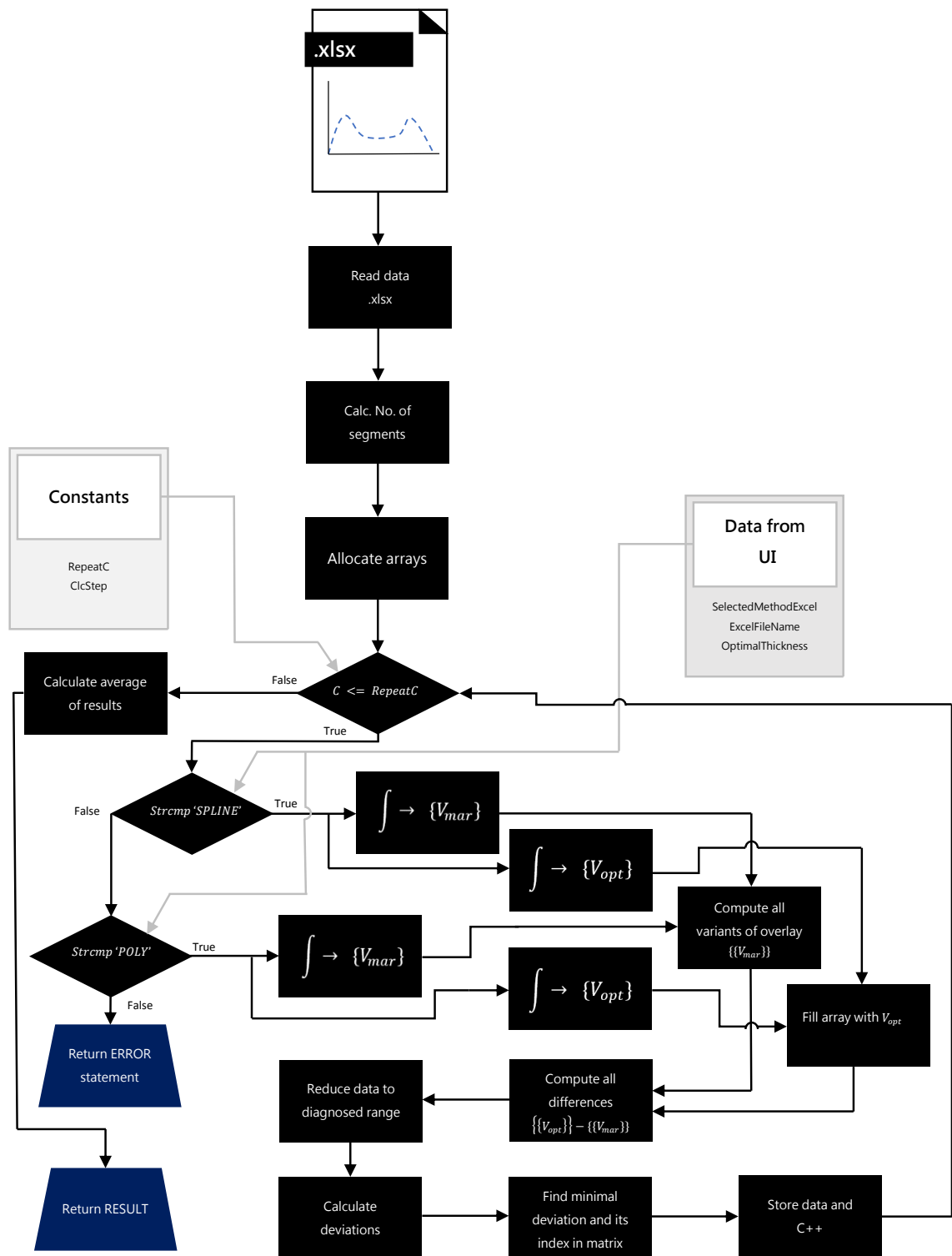
Výsledkem je pole obsahující objemové diference mezi ideálním objemem a teoretickým objemem tloušťkového profilu pro jednotlivé inkrementy. Dále je provedený výpočet směrodatné odchylky objemů od objemu ideálního z pole  $V_{diff_{ij}}$ , kde data jsou vyhodnocována po směru, jak je tomu na obrázku 48. Výstupem je pole obsahující směrodatné odchylky pro všechny případy překryvu.

V téměř posledním kroku celého výpočtu je vybrán výsledek s nejnižší možnou směrodatnou odchylkou pomocí příkazu `find(GetResult == min(GetResult(:)))`, kde `GetResult` je pole obsahující směrodatné odchylky pro všechny případy překryvu.

Z důvodů implementované možnosti aproximace pomocí polynomu vysokého řádu bylo do programu přidáno vylepšení, které minimalizuje nestabilitu<sup>5</sup> finálního výsledku.

<sup>5</sup>Funkce `polyfit()` využívá pro aproximaci metodu s použitím Alexandre-Théophil Vandermondovi





Obr. 47: Diagram algoritmu výpočtu optimální roztče

Řešení je provedeno pomocí cyklu, který celý výpočet opakuje do doby než  $i = \text{RepeatC}$

matic, u níž může pro vysoké stupně polynomu docházet ke špatné podmíněnosti Vandermondovi matice. To má za následek, že přidružený lineární systém nelze spolehlivě vyřešit. V extrémních případech může být Vandermondova matice singulární. Nestabilita může být také dána maximální rozlišovací schopností programu MATLAB. [50] [18]

a dílčí výsledky zprůměruje s přesností na jednotky. Přesto, že je tato metoda triviálním řešením, funguje velice spolehlivě <sup>6</sup>.

## Výpočet objemů spline křivek

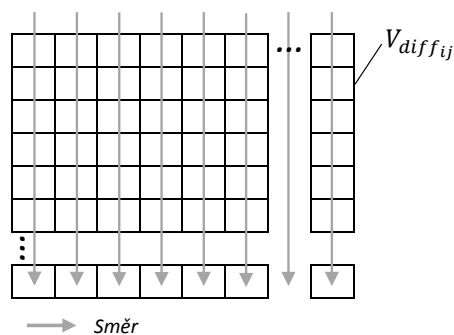
Pro výpočet objemu pod spline křivkou je použit výpočet integrace křivky P1 pomocí funkce `fnint` a funkce `fnval`. Zároveň jsou vypočteny ideální objemy pod přímkou optimální tloušťky  $t_{opt}$ . Výsledkem cyklu je pole `ArrayOfVolumesSpline` obsahující informace o polohách objemů v rámci profilu a vypočtené objemy. Druhým výstupem je pole `ArrayOfVolumesSpline` o totožné velikosti obsahující ideální objemy.

**Kód 7:** Hlavní část algoritmu pro výpočtu objemů pod spline křivkou

```

1   for i= 1:1:ComputedSegments
2       Rangels = RangeStart+(i-1)*CalcStep;
3       Rangele = RangeStart+i*CalcStep;
4       P1 = spline(xData,yData);
5       IntPolynom = diff(fnval(fnint(P1),[Rangels;Rangele]));
6       ArrayOfVolumesSpline(i,:) = [(Rangels+Rangele)/2,...
7           abs(IntPolynom)];
8       ArrayOfOptimalVolumes(i,:) = [(Rangele-Rangels)*...
9           abs(OptimalThickness)];
10  end

```



**Obr. 48:** Směr sčítání objemů v rámci pole  $V_{diff_{ij}}$

<sup>6</sup>Pro 4 různé data sety bylo nasimulováno 4 x 100 výpočtů se shodnými výsledky pro 100 případů

## Výpočet objemů polynomu n-tého stupně

Výpočet polynomu n-tého stupně kopíruje mechanismus výpočtu spline křivky s rozdílem použitých funkcí. Objemy jednotlivých segmentů jsou vypočteny pomocí vložené integrační funkce polynomu `polyval`. Výstupem cyklu jsou taktéž dvě pole obsahující typově stejná data.

**Kód 8:** Hlavní část algoritmu pro výpočtu objemů pod polynomem

```
1     for i= 1:1:ComputedSegments
2         Range1s = RangeStart+(i-1)*CalcStep;
3         Range1e = RangeStart+i*CalcStep;
4         P2 = polyfit(xData,yData,N);
5         PolFunc = polyint(P2);
6         IntPolynom = diff(polyval(PolFunc,[Range1s Range1e]));
7         ArrayOfVolumesSpline(i,:) = [(Range1s+Range1e)/2,...
8             abs(IntPolynom)];
9         ArrayOfOptimalVolumes(i,:) = [(Range1e-Range1s)*...
10            abs(OptimalThickness)];
11    end
```

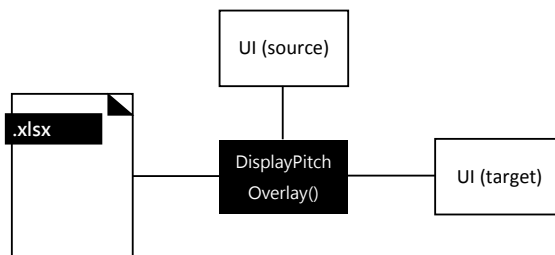
Jednou z nevýhod použití polynomu vysokého stupně je fakt, že při výpočtu dochází k jeho nestabilitě. To vede k tomu, že pro stejná vstupní data může být výsledek lehce odlišný. Během testování na vzorku V1 docházelo k maximální odlišnosti výsledku  $\pm 1$  mm. Tato hodnota přesnosti je v oblasti lakování sice zanedbatelná, rozhodně ale není chtěná. Z toho důvodu byl problém odstraněn opakováním výpočtu a následně zprůměrováním výsledků.

## 4.7 Vizualizace optimálního překryvu

Vizualizace analyzovaných dat je řešena obdobným způsobem jako je tomu při získávání optimální rozteče s rozdílem, že výsledná hodnota překryvu je již známá.

V první řadě jsou opětovně<sup>7</sup> získány objemy segmentů po celém profilu a načteny do předem alokovaných polí, tak jako je tomu u funkce `ComputePitch()`.

<sup>7</sup>Tento postup by bylo možné nahradit metodou, která by získávala data přímo z funkce `ComputePitch()` a uložila do RAM paměti. Toto elegantnější a výpočetně méně náročné řešení nebylo z důvodů úspory místa v paměti zvoleno. Počítač by byl totiž nucen držet pole v paměti i za stavu, kdy by to nebylo potřeba.



**Obr. 49:** Diagram datových toků pro funkci DisplayPitchOverlay()

Dále jsou vypočtené rozměry požadovaných polí `NewRange1` a `NewRange2`, které je nutné vyplnit nulami. Ty jsou bezpodmínečně nutné pro doplnění finálních polí `Range1` a `Range2`. Jak je vidět v kódu 9, pole se skládají ze dvou částí, které jsou spolu sečteny.

**Kód 9:** Procedura připravující pole pro finální vykreslení výsledků

```

1   AdditionalArray2 = OptimalPitchVal-ComputedSegments/2;
2   CompleteArray = ComputedSegments + AdditionalArray2;
3   NewRange1 = zeros(CompleteArray,2);
4   NewRange2 = zeros(CompleteArray,2);
5
6   Range1 = NewRange1 + [ArrayOfVolumesSpline ;
7       zeros(AdditionalArray2,2)];
8   Range2 = [zeros(AdditionalArray2,2) ; ArrayOfVolumesSpline] +
9       + NewRange2;
  
```

Aby velikost pole `ArrayOfVolumeSpline` vždy korespondovala s polem `Range1`, potažmo `NewRange1`, je pole `ArrayOfVolumeSpline` doplněno o pole velikosti  $M$ , kde  $M$  je definované jako:

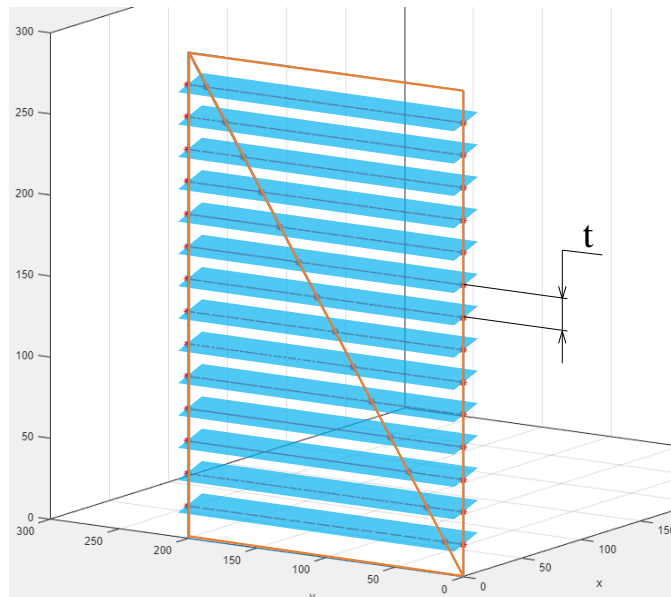
$$M = K - \frac{N}{2} \quad (23)$$

Kde  $N$  je celkový počet segmentů tloušťkového profilu a  $K$  představuje rozteč mezi optimálními průřezdy v segmentech. Obsahem pole  $M$  jsou nuly.

Pro zobrazení výsledku jsou použité dvě integrované MATLAB funkce `bar()` v režimu `'stacked'`.

## 4.8 Výpočet trajektorie a její zobrazení

Realizace výpočtu trajektorie je provedena aplikací myšlenky řezu v úrovních, jako je tomu u MFT algoritmu s rozdílem, že řezy jsou vždy paralelní k jedné ze základních rovin a jsou vedeny ve výšce, která představuje ideální rozteč průjezdů aplikátoru. Tento postup je také používán u konturových algoritmů v oblasti 3D tisku [39]. Na obrázku 50 je situace graficky znázorněna. I přesto, že se jedná na první pohled o triviální algoritmus, není tomu zcela tak. Celý postup se skládá z mnoha částí, které na sebe musí správně navazovat a systém musí být i do jisté míry robustní. Jednotlivé logické části celého postupu vytěžení trajektorie je rozděleno do následujících podkapitol, které procedury popisují.



**Obr. 50:** Naznačení metody řezu na STL síti obdélníku

Velmi zjednodušeně algoritmus nalezení trajektorie funguje následovně. Model je uživatelem orientován do polohy, kdy směr kladení trajektorie předpokládá paralelně s rovinou  $z$  (myšlena rovina definována přímkami  $x,y$ ). Dále je nalezen střed modelu na základě vyhodnocení polohy nejnižšího a nejvyššího bodu. Středem je vedena rovina paralelní k rovině  $z$ . Tato rovina slouží jako referenční a jsou od ní napočítány souřadnice dalších rovin, které protínají dané těleso. Průsečíkem roviny s tělesem jsou vypočteny průsečíky na hranách trojúhelníků a je definována jejich souslednost. Samotným výsledkem je pole obsahující pole pro jednotlivé vrstvy ve tvaru:

$$\left\{ \left\{ \begin{array}{ccc} x_{11} & y_{11} & z_{11} \\ x_{12} & y_{12} & z_{12} \\ \vdots & & \\ x_{1N} & y_{1N} & z_{1N} \end{array} \right\} \left\{ \begin{array}{ccc} x_{\dots 1} & y_{\dots 1} & z_{\dots 1} \\ x_{\dots 2} & y_{\dots 2} & z_{\dots 2} \\ \vdots & & \\ x_{\dots N} & y_{\dots N} & z_{\dots N} \end{array} \right\} \left\{ \begin{array}{ccc} x_{M1} & y_{M1} & z_{M1} \\ x_{M2} & y_{M2} & z_{M2} \\ \vdots & & \\ x_{MN} & y_{MN} & z_{MN} \end{array} \right\} \right\}$$

kde N je počet průsečíků pro danou plochu M. Po tomto kroku následuje vykreslení bodů do grafu v pořadí, které odpovídá směru kladení trajektorie od spodní vrstvy až po vrstvu horní.

Hlavní procedura celého výpočtu je `GeneratePath()`. Ta spolu s funkcí `FindTrianglePlaneIntersection()` získá požadovaný výstup. Algoritmus výpočtu je pro lepší přehlednost rozdělen do následujících dvou podkapitol. Na obrázku 51 je zobrazen diagram algoritmu hledání trajektorie.

## 4.9 Generování trajektorie `GeneratePath()`

Prvním krokem celého výpočtu je bezpochyby nalezení nejnižšího a nejvyššího bodu v celém modelu. To provedeme velice jednoduše použitím funkce `min()` a `max()` na souřadnicích z, a to přes všechny vrcholy. Tím získáme data pro výpočet počtu rovin, který je nezbytně nutný pro vedení řezu(ů).

Dále si připravíme pole `AllTrianglesInNewStruc` obsahující jak souřadnice všech vrcholů, tak informaci o minimální a maximální poloze bodů v modelu. Výsledná struktura je tedy pole o rozměru  $[N \times 14]$ , kde N je součet vrcholů v modelu.

Jak již bylo řečeno, jako první rovinu se snažíme nalézt rovinu, jež je ve středu tělesa. Tu nalezneme pomocí vzorce:

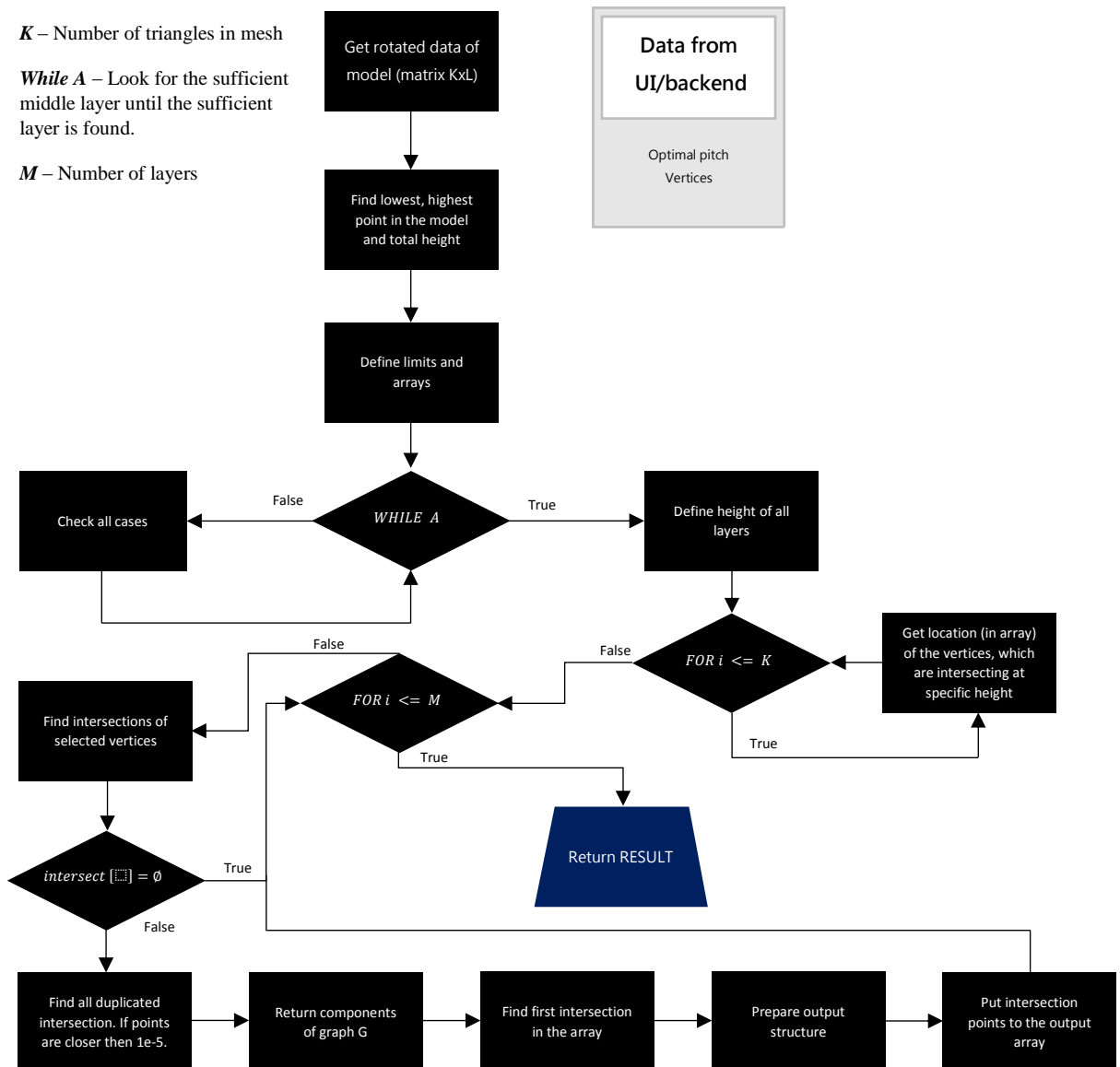
$$z = \frac{z_{max} - z_{min}}{2} \quad (24)$$

Od této roviny hledáme další roviny ve vzdálenosti násobku ideální rozteče průjezdů aplikátoru. Tím získáme dvě pole z souřadnic s hodnotami, které na sebe po logickém

uspořádání  $[array1, array2]$  nenavazují. Z tohoto důvodu jsou všechny hodnoty seřazeny funkcí  $sort([array1, array2])$  od nejmenší až po největší. Tím se také zaručí, že kladená trajektorie bude vedená ze spodní části modelu do horní části modelu.

V tento okamžik máme k dispozici všechny  $z$  souřadnice a musíme nalézt všechny trojúhelníky, u kterých předpokládáme, že bude docházet k průniku s rovinami.

Algoritmus hledání podezřelých trojúhelníků je částečně převzat z algoritmu, jehož autorem je Sunil Bhandari, který algoritmus publikoval na sociální síti MathWorks Community [11]. Popis funkce je následující:



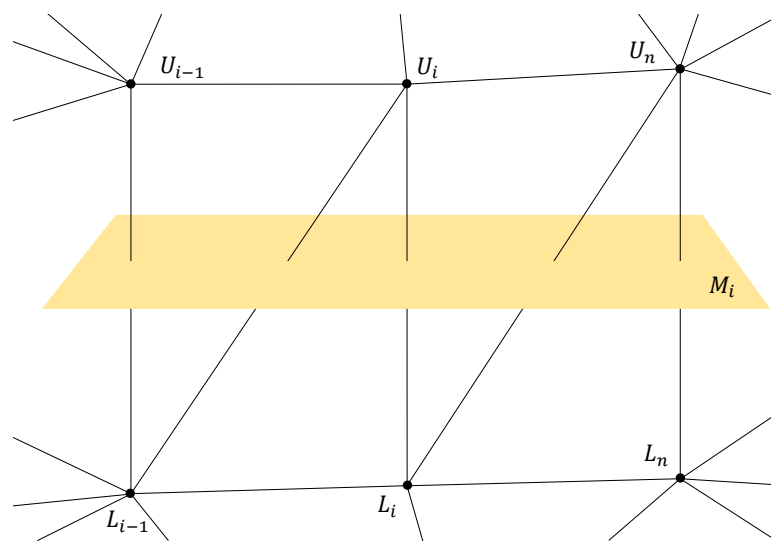
Obr. 51: Diagram algoritmu hledání trajektorie

Sunilův algoritmus je rozdělen na dvě části, které nezávisle na sobě provádějí logické operace a jejich výsledek je použit pro závěrečné vyhodnocení.

Můžeme říci, že celkem máme k dispozici počet trojúhelníků  $T$ . Pro každý trojúhelník z této množiny se provede analýza ve dvou krocích. V kroku prvním ověříme, zda nějaký bod trojúhelníku se nachází pod nějakou z řezných rovin, která je v jeho blízkosti. Pokud taková rovina existuje, případ dostane příslušné označení. Program dále pokračuje vyhodnocením, zda pro ten samý trojúhelník existuje zároveň bod, nacházející se nad řeznou rovinou. Pokud jsou obě dvě podmínky vyhodnoceny pozitivně, můžeme říci, že trojúhelník interaguje s danou rovinou.

Pokud je vyhodnocovaný trojúhelník interagujícím, uloží se jeho odpovídající poloha ve zdrojovém poli, do pole obsahující mezivýsledky. A to do takového řádku, jež odpovídá indexu řezné roviny (počet průsečíků je limitován na 5000).

Úplným výsledkem tohoto vyhodnocení je tedy pole obsahující indexy trojúhelníků, kterých se průnik týká. Každý řádek tohoto pole zároveň přiřazuje bodům v řádku příslušnou rovinu.



**Obr. 52:** Vizualizace problému hledání průniku roviny s trojúhelníky

V matematickém zápise má výsledná matice následující tvar:

$$Q = \begin{bmatrix} a_1 & a_2 & \dots & a_{n_1-1} & a_{n_1} \\ b_1 & b_2 & \dots & b_{n_1-1} & b_{n_2} \\ \vdots & \dots & \dots & \dots & \dots \\ M_1 & M_2 & \dots & M_{n_M-1} & M_{n_M} \end{bmatrix}$$



kde  $n_M$  je počet protátných trojúhelníků rovinou  $M$ . Zároveň můžeme říci, že  $M$  je celkový počet rovin.

Poté, co jsou data připravena k dalšímu zpracování, zapojí se do for cyklu, který svou vnitřní procedurou zpracovává jednotlivé roviny. Cílem tohoto cyklu je nalézt konkrétní souřadnice průsečíků přímk trojúhelníků s rovinami.

Výpočet průsečíků je ale prováděn funkcí `FindTrianglePlaneInterecton()`, která je v každém cyklu volána. Výsledek, který tato funkce vrátí, obsahuje souřadnice průsečíků v prostoru a zároveň informaci o velikosti obdrženého pole.

Bohužel síť STL modelů může obsahovat pro jeden a ten samý bod dva průsečíky, byť správně by měl být pouze jeden. Také může nastat situace, že síť modelu může být poškozená nebo může být v některých místech natolik hustá, že jednotlivé průsečíky spolu budou téměř splývat. Aby se tomuto jevu předešlo, je do programu implementována redukce nežádoucích bodů, jež jsou v menší blízkosti (parametry `SetTolerance` a `SetIsMemberTol`) odpovídající  $1e^{-5}$ .

Následuje přeskládání bodů podle správného pořadí, tak jak byly průsečíky nalezeny. Zároveň se vytvoří podgraf použitím funkce `graph()`, který obsahuje definici souslednosti bodů a tedy definuje trajektorii.

V úplně posledním kroku tohoto algoritmu dojde k naplnění výstupních polí, jež jsou označené jako `OutputArrayOfArray` a `DataForABB`.

Pole `OutputArrayOfArray` obsahuje další pole o rozměru  $Nx[x, y]$ , kde  $N$  je počet nalezených reálných průsečíků pro jednu rovinu. Písmena  $x$  a  $y$  jsou souřadnice reálných průsečíků v rovině  $z$ , tedy na ose  $x$  a  $y$ .

Pole `DataForABB` představuje stejný výstup jako `OutputArrayOfArray` s rozdílem, že je rozšířeno o souřadnici  $z$ .

### 4.9.1 FindTrianglePlaneIntersection()

Funkce FindTrianglePlaneIntersection patří k jedné z hlavních funkcí. Jejím úkolem je nalézt průsečíky trojúhelníků s plochou. Matice, kterou funkce zpracovává má následující tvar a je nositelem informace o všech trojúhelnících, které se podílí na průniku s danou rovinou:

$$Q = \begin{bmatrix} x_{11} & \dots & x_{N1} \\ y_{11} & \dots & y_{N1} \\ z_{11} & \dots & z_{N1} \\ x_{12} & \dots & x_{N2} \\ \vdots & \dots & \vdots \\ z_{13} & \dots & z_{N3} \\ z_H & \dots & z_H \end{bmatrix}$$

Kde  $x_{11}, y_{11}, z_{11}$  jsou souřadnice vrcholu A trojúhelníku 1,  $x_{12}, y_{12}, z_{12}$  jsou souřadnice vrcholu B trojúhelníku 1,  $x_{13}, y_{13}, z_{13}$  jsou souřadnice vrcholu C trojúhelníku 1,  $z_H$  představuje úroveň roviny na ose  $z$ .  $N$  představuje počet trojúhelníků podezřelých z průniku s danou rovinou.

Tento tvar, není ale dobře zpracovatelný a data jsou přeskupena do 4 matic ve tvaru

$$A = \begin{bmatrix} x_{11} & \dots & x_{N1} \\ y_{11} & \dots & y_{N1} \\ z_{11} & \dots & z_{N1} \end{bmatrix}, B = \begin{bmatrix} x_{11} & \dots & x_{N1} \\ y_{12} & \dots & y_{N2} \\ z_{12} & \dots & z_{N2} \end{bmatrix}, C = \begin{bmatrix} z_{13} & \dots & x_{N3} \\ z_{13} & \dots & y_{N3} \\ z_{13} & \dots & z_{N3} \end{bmatrix}$$

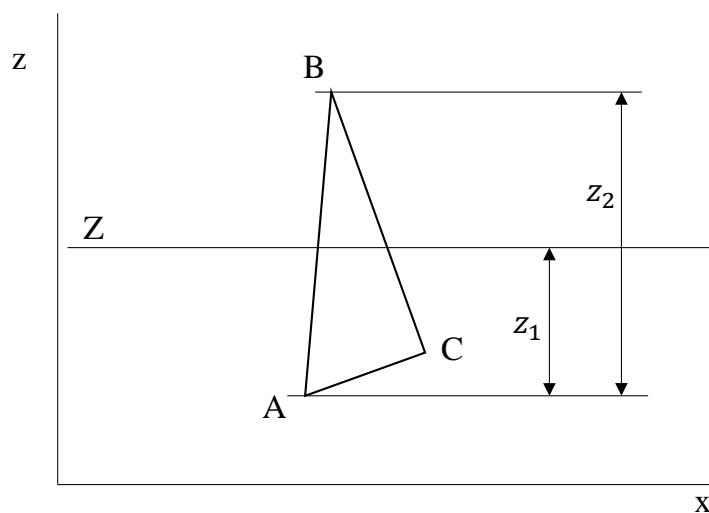
$$Z = \begin{bmatrix} z_H & \dots & z_H \\ z_H & \dots & z_H \\ z_H & \dots & z_H \end{bmatrix}$$

Jelikož průnik se analyzuje pouze s rovinou  $z$ , můžeme zbylé souřadnice eliminovat.

Eliminaci provedeme takovým způsobem, že si definujeme matici o rozměru  $[3 \times N]$ , kde  $k$  je opět počet trojúhelníků podezřelých z průniku s danou rovinou. Eliminační matice  $E$  má tvar:

$$E = \begin{bmatrix} 0(1) & \dots & 0(N) \\ 0(1) & \dots & 0(N) \\ 1(1) & \dots & 1(N) \end{bmatrix}$$

V tento okamžik máme všechna data připravena a můžeme provést výpočet průsečíků. Z podstaty věci víme, že v okamžiku, kdy nalezneme  $z$  souřadnici průniku, můžeme dopočítat umístění i ve zbývajících rozměrech. Úlohu tedy můžeme řešit jako 2D a její řešení použít následně pro dořešení úlohy 3D.



**Obr. 53:** Znárodnění problémů 2D úlohy

Prvním cílem je získat poměr mezi jednotlivými částmi  $z_1$  a  $z_2$ , a tím mít možnost dopočítat souřadnice průsečíků zbývajících rozměrů. Situace je znázorněna na obrázku 53 a matematický popis použitelný v prostředí MATLAB zapíšeme následovně:

$$\begin{aligned}
R_1 &= \frac{Z - \sum (N \cdot A)}{\sum (N \cdot (A - B))} \\
R_2 &= \frac{Z - \sum (N \cdot A)}{\sum (N \cdot (B - C))} \\
R_3 &= \frac{Z - \sum (N \cdot A)}{\sum (N \cdot (C - A))}
\end{aligned} \tag{25}$$

Výsledkem jsou poměry, které jsou následně použity pro výpočet souřadnic potenciálních průsečíků podle vtaů:

$$\begin{aligned}
I_1 &= A + (A - B) \cdot R_1 \\
I_2 &= B + (B - C) \cdot R_2 \\
I_3 &= C + (C - A) \cdot R_3
\end{aligned} \tag{26}$$

Získané souřadnice bodů bychom mohli získat ale pro jakýkoliv vrchol trojúhelníku v celém tělese, což by nezaručovalo 100% správnost výsledku. Následující postup vrátí logické hodnoty pro jednotlivé případy, které mohou nastat. Výsledkem tohoto postupu jsou tedy logické hodnoty (0,1), které jsou podle klíče<sup>8</sup> vyhodnoceny a je vytvořený výsledek.

$$\begin{aligned}
L_1 &= [I_1 < \max \{A, B\}] \wedge [I_1 > \min \{A, B\}] \\
L_2 &= [I_2 < \max \{B, C\}] \wedge [I_2 > \min \{B, C\}] \\
L_3 &= [I_3 < \max \{C, A\}] \wedge [I_3 > \min \{C, A\}]
\end{aligned} \tag{27}$$

---

<sup>8</sup> Klíč k získání finálních výsledků je uvedený v příloze ve funkci FindTrianglePlaneIntersection.

## 4.10 Generování výstupů

Z předcházejících kapitol víme, že výstupem z funkce `GeneratePath()` získáme souřadnice průsečíků, a tedy i trajektorii nástroje. Tato trajektorie ale nepředstavuje finální produkt, jelikož neobsahuje informace o polohách a natočení nástroje. Abychom získali tato doplňující data, je nutné provést další výpočty. Tyto výpočty jsou obsaženy ve funkci `PrepareDataForRAPID`.

Funkce obdrží na vstupu pole obsahující další pole, kde  $N$  je počet průsečíků pro danou plochu  $M$ . Pole má tedy podobu:

$$\left( \left( \begin{array}{ccc} x_{11} & y_{11} & z_{11} \\ x_{12} & y_{12} & z_{12} \\ \vdots & & \\ x_{1N} & y_{1N} & z_{1N} \end{array} \right) \left( \begin{array}{ccc} x_{\dots 1} & y_{\dots 1} & z_{\dots 1} \\ x_{\dots 2} & y_{\dots 2} & z_{\dots 2} \\ \vdots & & \\ x_{\dots N} & y_{\dots N} & z_{\dots N} \end{array} \right) \left( \begin{array}{ccc} x_{M1} & y_{M1} & z_{M1} \\ x_{M2} & y_{M2} & z_{M2} \\ \vdots & & \\ x_{MN} & y_{MN} & z_{MN} \end{array} \right) \right)$$

Jelikož vyvinutý algoritmus je vhodný primárně pro tělesa a plochy z jejichž stěny je možné nalézt kolmici k řezné ploše, můžeme si dovolit polohu nástroje stanovit "přímočarým" postupem.

Aby byla poloha nástroje přesně definovatelná ve 3D prostoru, potřebujeme celkem 7 parametrů. Konkrétně se jedná o 3 rozměry definující cílový bod (souřadnice  $x, y, z$ ), 3 směrové úhly a jeden úhel natočení v ose nástroje. V robotice se velice často popisují poslední čtyři parametry pomocí kvaternionů. Vzhledem k tomu, že i roboty ABB používají pro definici nástroje kvaterniony, je tento popis implementován v této práci.

Kvaterniony, také známé pod pojmem versory, jsou rozšířením komplexních čísel. Zajímavou vlastností kvaternionů je nekomutativnost a jsou reprezentovány podobnou formou jako komplexní čísla ve formě:

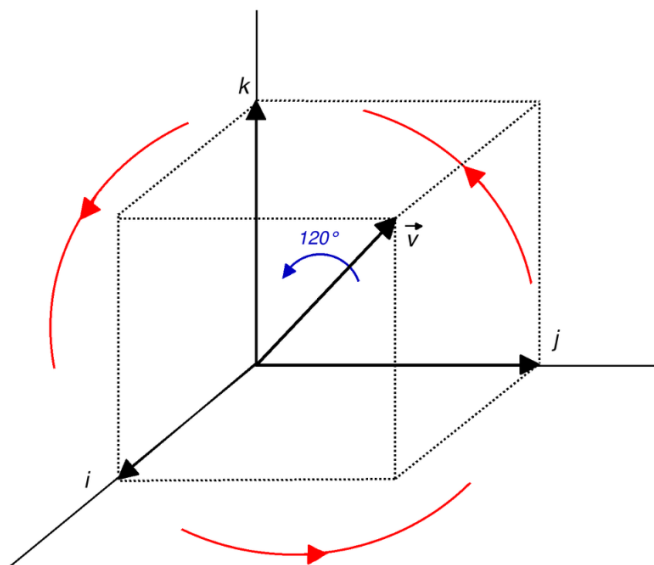
$$a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k} \tag{28}$$

kde  $a, b, c$  a  $d$  jsou reálná čísla a  $\mathbf{i}, \mathbf{j}$  a  $\mathbf{k}$  jsou fundamentální jednotky kvaternionu.

V robotice se spíše setkáváme se zápisem, který má tvar:

$$\vec{q} = [q_0, q_1, q_2, q_3]^T \quad (29)$$

Využití kvaternionu je velice široké, jak v matematice, fyzice, mechanice, počítačové grafice, tak i v dalších aplikacích, kde je nutné definovat rotace tělesa v prostoru [30].



**Obr. 54:** Grafické znázornění principu kvaternionu [2]

Vzhledem k tomu, že řezy provádíme v rovině, průsečíky logicky leží pro daný řez také v jedné rovině. Na základě tohoto a dalších předpokladů můžeme definovat vektory  $\vec{u}_i$  mezi průsečíky ( $A_N$ ) tak, že vedeme vektor mezi průsečíkem  $A_i$  a  $A_{i+2}$ . Tedy vektor  $\vec{u}_i$  dostaneme jako:

$$\vec{u}_i = (A_{i+2} - A_i) \quad (30)$$

K tomuto směrovému vektoru můžeme snadno dopočítat vektor k němu kolmý ( $\vec{n}_i$ ) dle vztahu:

$$\vec{n}_i = (\vec{u}_i(1, 2), -\vec{u}_i(1, 1), \vec{u}_i(1, 3)) \quad (31)$$

a posunout normálový vektor do bodu  $A_{i+1}$ , kterým musí vektor procházet. Výsledný normálový vektor  $\vec{n}$  je tedy definován jako:

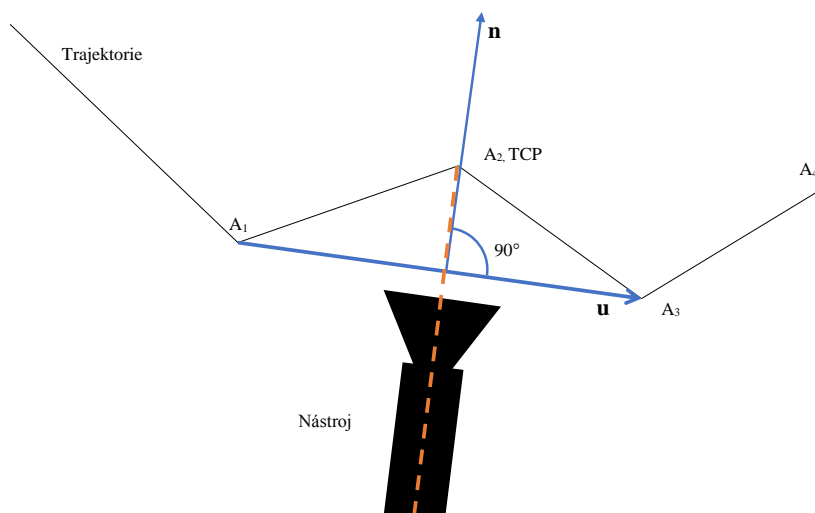
$$\vec{n} = [k_1 - k_2] \quad (32)$$

kde:

$$k_1 = [A_{i+1}(1, 1) + \vec{n}_i(1, 1), A_{i+1}(1, 2) + \vec{n}_i(1, 2), A_{i+1}(1, 3) + \vec{n}_i(1, 3)]$$

$$k_2 = [A_{i+1}(1, 1), A_{i+1}(1, 2), A_{i+1}(1, 3)]$$

Dále potřebujeme získat úhly směrového vektoru. Ty jsou nezbytné pro výpočet kvaternionů, a také jsou přímo jako jeden z výstupů.



**Obr. 55:** Naznačené řešení výpočtu polohy nástroje

Způsobů jakým můžeme směrové úhly získat je několik. V tomto programu je použitý přístup s použitím geometrické funkce arkus tangens. Výsledkem jsou jak jinak, než úhly v radiánech. Rovnice pro výpočet směrových úhlů jsou následující:

$$\alpha = \text{atan} \sqrt{\frac{(N_{1z} - N_{2z})^2 + (N_{1y} - N_{2y})^2}{N_{1x} + N_{2x}}}$$

$$\beta = \text{atan} \sqrt{\frac{(N_{1x} - N_{2x})^2 + (N_{1z} - N_{2z})^2}{N_{1y} + N_{2y}}}$$

$$\gamma = \text{atan} \sqrt{\frac{(N_{1x} - N_{2x})^2 + (N_{1y} - N_{2y})^2}{N_{1z} + u_{2z}}}$$
(33)

V rovnicích jsou definovány dva body které představují body definující směr normálového vektoru. Jedná se tedy o body odpovídající  $N_{1\dots} = k_1$  a  $N_{2\dots} = k_2$

Pro finální krok přípravy výstupů jsou vypočteny kvaterniony funkcí `angle2quat` (Tato funkce není obsažena ve standardní verzi MATLAB, pouze v MATLAB Aerospace Toolboxu).

Jako výstup této funkce jsou data uspořádaná do pole ve formě:

$$\left\{ \begin{array}{cccccccccc} A_{x_{i+2}} & A_{y_{i+2}} & A_{z_{i+2}} & q_{1_{i+2}} & q_{2_{i+2}} & q_{3_{i+2}} & q_{4_{i+2}} & \alpha_{i+2} & \beta_{i+2} & \gamma_{i+2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ A_{x_N} & A_{y_N} & A_{z_N} & q_{1_N} & q_{2_N} & q_{3_N} & q_{4_N} & \alpha_N & \beta_N & \gamma_N \end{array} \right\}$$

## 4.11 Generování RAPID kódu pro ABB roboty

Závěrečnou částí celého programu je vytvoření výstupů pro program Robot studio od společnosti ABB. Výstupy představují v tomto případě souřadnice průsečíků s úhly natočení nástroje v radiánech a kód pro roboty ABB.

První část je snadná, jelikož se jedná pouze o zobrazení dat na uživatelské rozhraní. Data jsou zobrazena v podobě pole, které je naplněno ve formátu:

$$\left\{ \begin{array}{cccccc} A_{x_i}; & A_{y_i}; & A_{z_i}; & \alpha_i; & \beta_i; & \gamma_i \\ \vdots; & \vdots; & \vdots; & \vdots; & \vdots; & \vdots \\ A_{x_N}; & A_{y_N}; & A_{z_N}; & \alpha_N; & \beta_N; & \gamma_N \end{array} \right\}$$

Jelikož se jedná z důvodů snadnějšího zobrazení o pole s jedním sloupcem, musí být data od sebe oddělena. Jako oddělovač je použitý středník.

V případě generování kódů v programovacím jazyku RAPID se jedná o závažnější problém, jelikož je nutné dosáhnout požadované struktury a zároveň zachovat pole s jedním sloupcem.



Požadovaná struktura se skládá ze dvou částí. Část první obsahuje výpis konstant ve struktuře: `CONST robtarget TargetName:=[[x,y,z],[q1,q2,q3,q4],(r1,r2,r3,r4)],[ax1,ax2,ax3,ax4,ax5,ax6]`

Pro naplnění této struktury je z důvodů přehlednosti generování textu rozděleno do několika bloků. V každém bloku je ke znakovému řetězci přidán řetězec další. Proces se opakuje do doby, než jsou všechny části řetězce vyplněny pro všechny potřebné body trajektorie.

Struktura a význam částí RAPID kódu je detailněji popsán v teoretické části práce.

V druhé části jsou generovány pohybové instrukce robota. Pro pohybové instrukce potřebujeme docílit struktury `TypeOfMovement, TargetName, Speed, Zone, Tool;`

Algoritmus přípravy tohoto bloku je totožný s blokem prvním. Do znakového řetězce jsou postupně přidávány data do doby, než jsou všechny části přidány pro všechny body trajektorie. Způsob jakým je řetězec vytvářen je znázorněn v kódu 10.

#### Kód 10: Způsob generování zdrojového kódu pro roboty

```
1  %=====
2  % Generate array of targets
3  % Generate first part: CONST robtarget Target_xxx
4  %=====
5  str = 'Target_';
6  str1 = 'CONST robtarget ';
7  str2 = ':[[';
8
9  for i=Targets:-1:1
10     Val = i*10;
11     ConstRobTarget{i,1} = sprintf('%s%d',str,Val);
12 end
13 for i=Targets:-1:1
14     StrBf = ConstRobTarget{i,1};
15     ConstRobTarget2{i,1} = sprintf('%s%s%s',str1,StrBf,str2);
16 end
```

V poslední části algoritmu jsou bloky zařazeny do sebe a odděleny příslušnými instrukčními příkazy.

## 4.12 Další vývoj aplikace

Vyvinutá aplikace funguje spolehlivě pouze pro určité případy použití a zároveň výsledek je velice ovlivněný vstupním modelem a jeho hustotou sítě. Zároveň hledání průsečíků nemusí probíhat správně pro tělesa, která nejsou manifoldní. Během vývoje aplikace bylo vymyšleno několik ideí, které by mohli vést ke zvýšení univerzálnosti a kvalitě výsledků. Idee jsou následující:

**Verifikace skutečné vzdálenosti mezi průjezdy.** Stávající řešení provede horizontální řezy v určité vzdálenosti od sebe. Neproběhne, ale zpětná verifikace, v jaké vzdálenosti jsou od sebe horizontální trajektorie skutečně. V případě, že by byla hodnota upravena na hodnotu skutečnou, došlo by k výraznému zvýšení přesnosti trajektorie pro obecně zakřivené plochy. Nalezení takové vzdálenosti je ale komplikované a bylo pravděpodobně nutné trajektorie v řezu rozdělit na menší trajektorie a provést řešení po částech.

**Optimalizace definice trajektorie.** Velkým problémem u zvoleného přístupu jsou také vytvořené průsečíky řezné roviny s trojúhelníky sítě. Pokud je model složen z velkého množství trojúhelníků, výsledek také obsahuje velké množství bodů. To by příliš nevyhovovalo v případě, že by výsledek byl finální a nemusel se upravovat. Takový případ ale jen tak nenastane a bylo by tedy vhodné omezit počet bodů definující trajektorii. To můžeme provést tak, že provedeme přesíťování modelu tak, aby síť modelu byla definována menším počtem trojúhelníků. Dalším způsobem by mohlo být proložení získaných bodů trajektorie křivkou, která by byla následně rozdělena podle zakřivení a získané body by definovaly trajektorii.

**Verifikace tloušťky vrstvy.** Mezi další příjemnou funkci aplikace by mohla být verifikace tloušťky nanosené vrstvy na povrchu s použitím distribuce ze zvoleného/vyhodnoceného atomizéru.

**Data pro produkci.** Dalším předmětem výzkumu by mohlo být získání dat nezbytně nutných pro plánování výroby (tj.: čas cyklu, spotřeba barvy, TE).

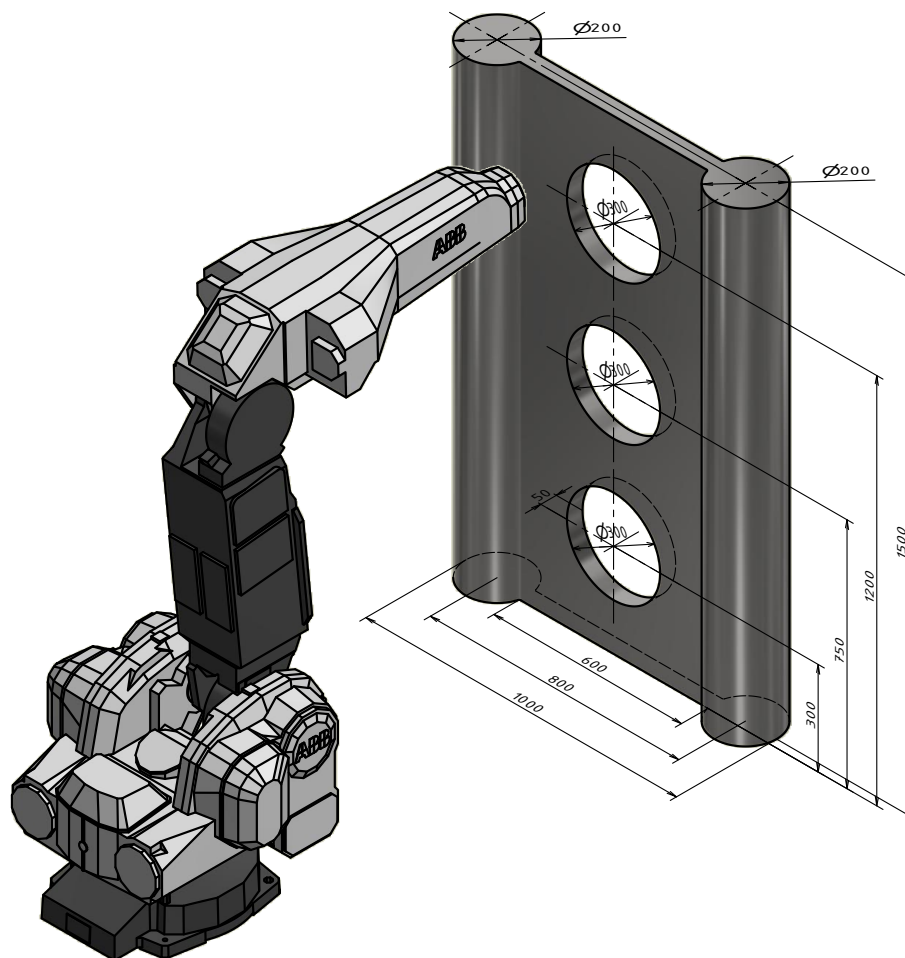
**Ukládání analýzy.** Aplikace by byla uživatelsky více příjemná, kdyby bylo možné načítat a ukládat měření. Při používání dosavadního řešení musí uživatel vždy provádět postup od začátku. Přinejmenším databáze atomizérů by práci velice usnadnila.

**Použití jiné metody hledání trajektorie.** Pokud by zvolený přístup hledání trajektorie nesplnil požadavky, bylo by nutné změnit koncept aplikace a přejít od metody řezání rovinami na jiný algoritmus, či kombinaci více algoritmů.

## 5 Část experimentální

Cílem experimentální části je představení postupu a použitelnosti vytvořeného nástroje. Za tímto účelem byl připravený virtuální vzorek části podpěrné konstrukce (obrázek: 56) o šířce 1000 mm a výšce 1500 mm, který se bude lakovat v pozici, jak je uvedeno na obrázku na tloušťku vrstvy 18  $\mu\text{m}$ . Konstrukce byla navržena tak, aby kombinovala několik různých typů ploch, jako je: zaoblení s velkým poloměrem, malým poloměrem a rovnou plochu jež je opatřena odlehčením. Díky této různorodosti se prokáží nejen výhody vytvořeného algoritmu, ale také i nevýhody, které mohou být použity jako předmět dalšího výzkumu.

Součástí experimentu je také reálná část, která se zabývá analýzou a výběrem vhodného nastavení procesu pro pohledovou vrstvu barvy.



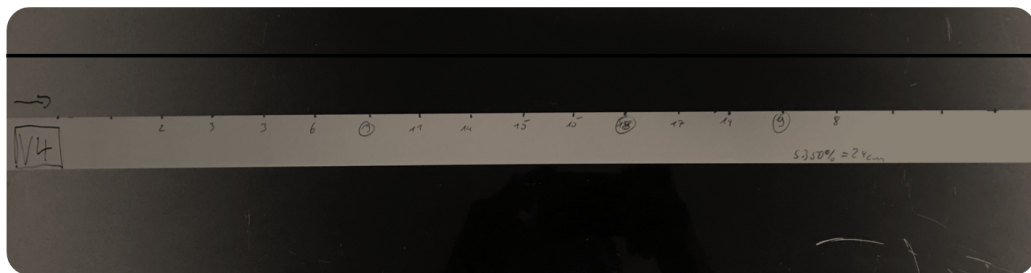
Obr. 56: Vzorek

## 5.1 Měření profilu tloušťky barvy

Pro experiment bylo od společnosti ABB poskytnuto celkem 24 vzorků pořízených lakovacím robotem IRB 5400 s lakovacím atomizérem RB1000-EXT. Jedná se o atomizér s externím nabíjením a vysokou efektivitou přenosu vodou ředitelných barev.

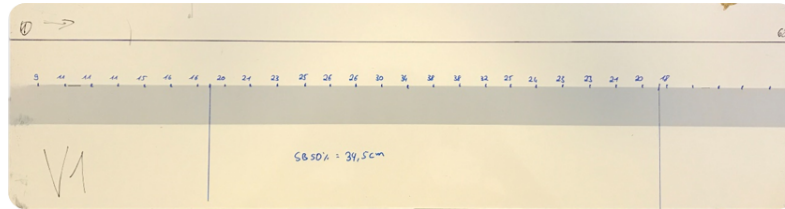
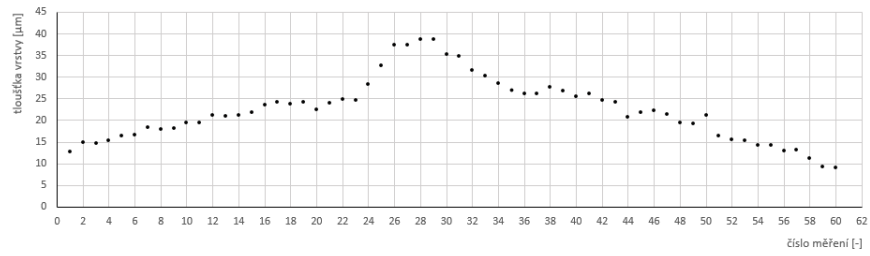
Z 24 vzorků bylo 12 se základovou barvou a 12 s pohledovou barvou. Z celku byly vybrány 3 vzorky se základovou barvou (V1, V2, V3) a 3 vzorky s pohledovou barvou (V4, V5, V6), jež byly podle pohledu nejvyšší kvality. Rozměr vzorků byl 600 x 160 mm. Jelikož se jedná o vzorky od zákazníka společnosti ABB, nemohou být detailní nastavení procesu veřejně publikovány, ani součástí přílohy.

Pro měření tloušťky vrstvy bylo použito zařízení Elcometr 456T a Defelsko Positector 6000. Vzorky byly nejdříve očištěny, odmaštěny, osušeny a následně označeny. Také byl každý ze vzorků opatřen šipkou určující směr měření a horizontální vodící linkou, jako je tomu u obrázku 57. Samotné snímání bodů Elcometrem bylo provedeno manuálně sondou na celé délce vzorku s rozstupem snímání 1 cm. Výsledek měření provedeným Positectorem 6000 nebylo možné použít, jelikož přístroj měří data kontinuálně s časovým rozstupem snímání minimálně 1 vteřina a nebylo možné dodržet konstantní rychlost. Aby bylo možné použít Positector, bylo by nutné umístit sondu na elektronicky řízený pojezd, u kterého by byla přesně definována rychlost a tím i přesná poloha měřených bodů.

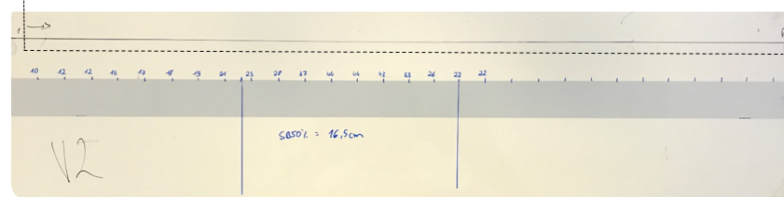
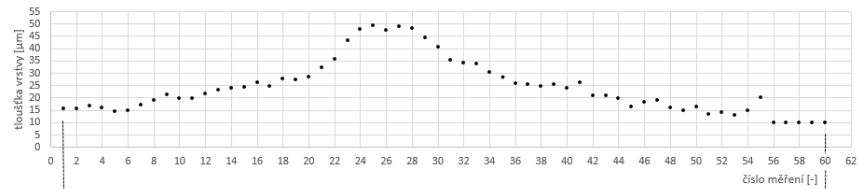


Obr. 57: Způsob měření vzorků

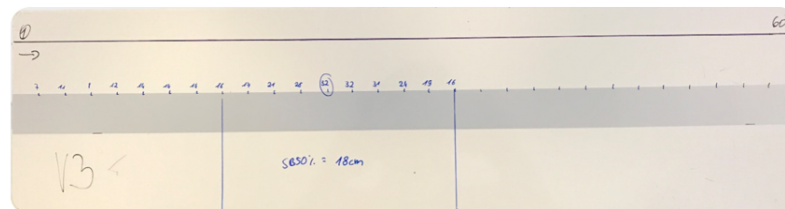
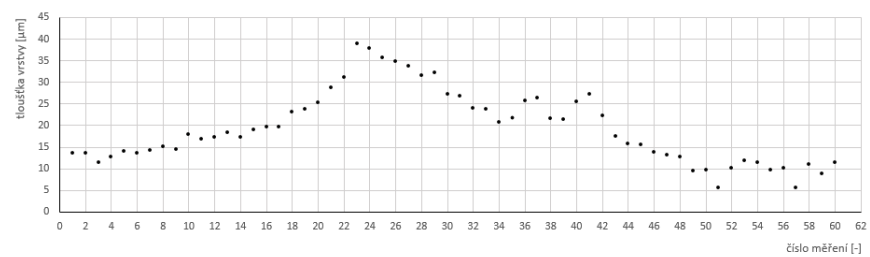
Dle specifikací výrobce je měření Elecometrem 456T zatíženo chybou  $\pm(1 - 3)\%$  /  $\pm 2.5\mu m$ , podle toho jaká hodnota je větší [24]. V obou případech je chyba měření natolik malá v porovnání s tolerancemi, které se pro tloušťky vrstvy v praxi používají, že není nutné se touto chybou příliš zabývat.



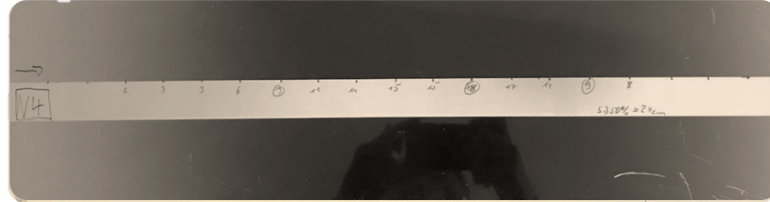
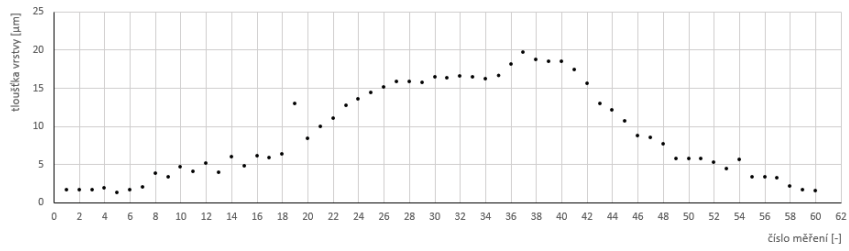
Obr. 58: Distribuce barvy - vzorek 1



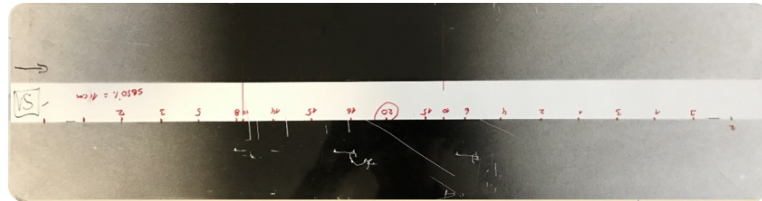
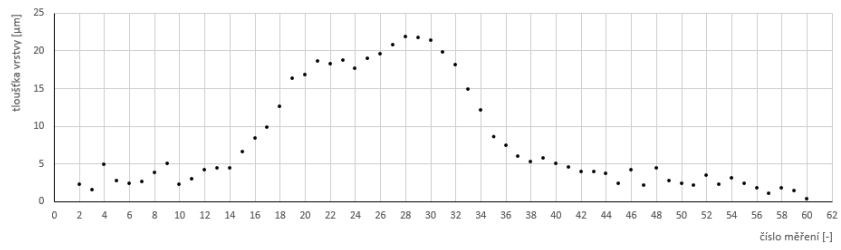
Obr. 59: Distribuce barvy - vzorek 2



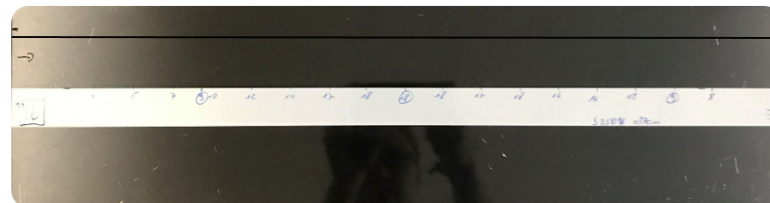
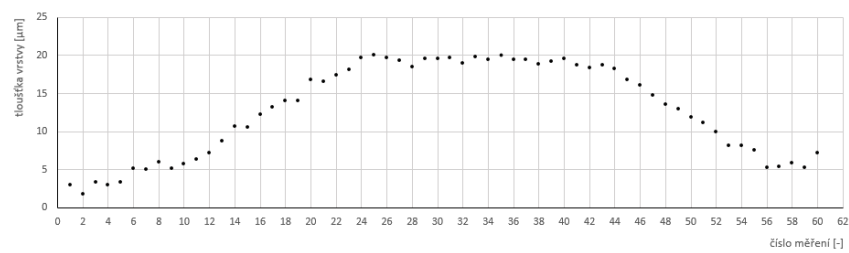
Obr. 60: Distribuce barvy - vzorek 3



Obr. 61: Distribuce barvy - vzorek 4

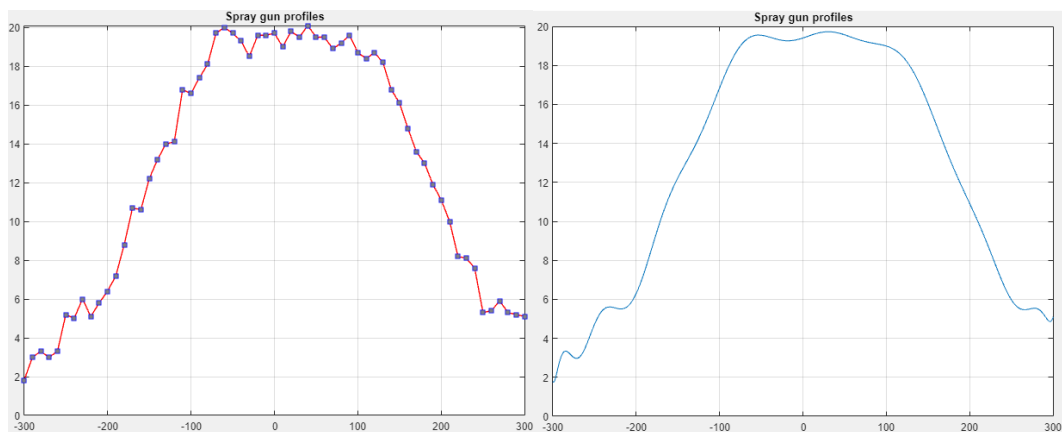


Obr. 62: Distribuce barvy - vzorek 5



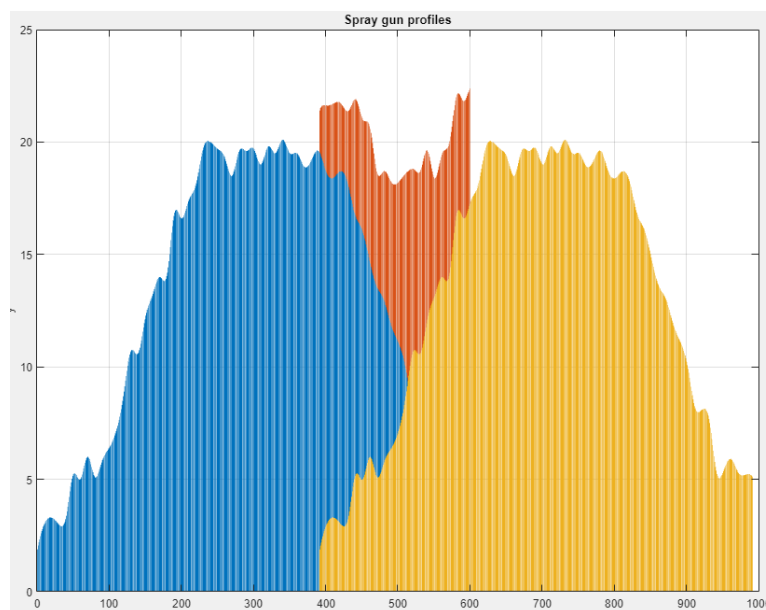
Obr. 63: Distribuce barvy - vzorek 6

Z naměřených vzorků pohledových barev byl vybrán jako nejvhodnější vzorek číslo 6, jelikož dosahuje nejrovnoměrnější distribuce barvy ve středu vzorku. Zároveň nejvíce odpovídá požadované tloušťce vrstvy  $18 \mu\text{m}$ . V dalším kroku vyhodnocení vzorků je použitý vytvořený nástroj. Distribuce barvy odpovídá grafu na obrázku 64. Z grafu je jisté, že vyhodnocovaný interval pro hledání ideálního překrytí bude někde mezi -70 až 300 jednotek. Po přičtení offsetu grafu, který tvoří vždy 1/2 rozlišení grafu víme, že vyhodnocovaný interval by měl být přibližně od 230 do 600 jednotek.



**Obr. 64:** Distribuce barvy zvoleného vzorku (vlevo), aproximace vzorku (vpravo)

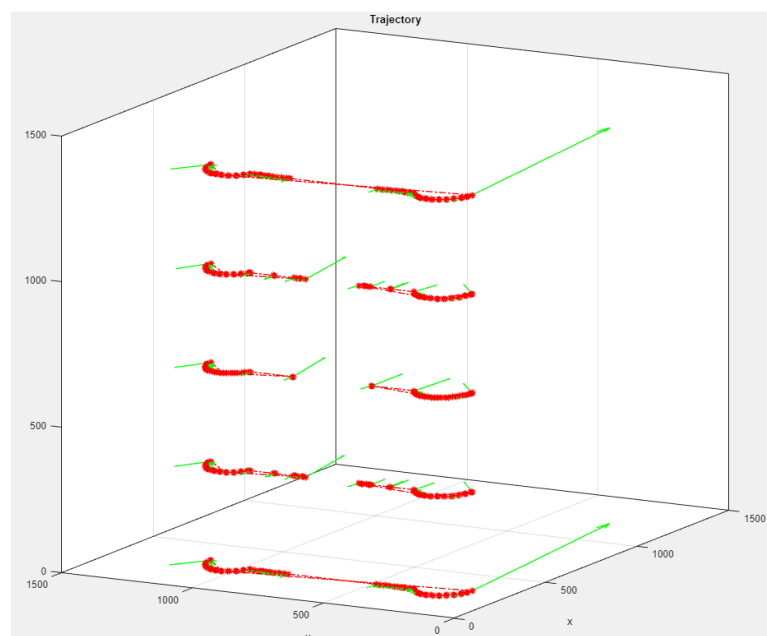
Pro vyhodnocení byla použita aproximace spline křivkou s výsledkem 349 jednotek a aproximace polynomem 20. stupně s výsledkem 350 jednotek. Jelikož výsledky jsou totožné, na volbě nezáleží. Pokud by ovšem byl výsledek velice rozdílný, bylo by vhodnější vybrat výsledek s menší hodnotou.



**Obr. 65:** Překryv pro aproximaci spline křivkou

## 5.2 Nalezení trajektorie na modelu a simulace

Model, který v této případové studii zpracováváme, je modelem hmotným. Tato skutečnost přináší, jak jisté výhody, tak i nevýhody. Z pohledu zpracování modelu se jedná o manifoldní těleso a může tedy proběhnout kontrola a oprava sítě bez problémů. Na stranu druhou dojde k vygenerování trajektorie i v takových místech, kterých nebude možné robotem dosáhnout. To by nebyl příliš velký problém v případě, že by trajektorie byla definována jednotkami bodů. Odstranění nežádoucích bodů by totiž zabralo nanejvýš pár minut. Tomu ale tak není a odstranění bodů by mohlo zabrat podstatně více času. Z tohoto důvodu byla provedena extrakce plochy, jež se zdá být na první pohled dosažitelná a dává smysl i z technologického hlediska.

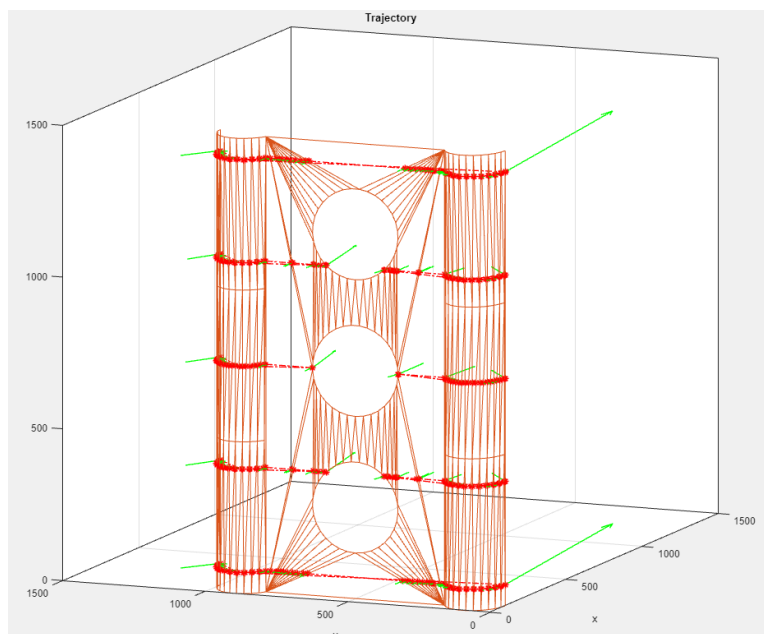


**Obr. 66:** Vygenerovaná trajektorie

Výsledkem analýzy ideální rozteče byla získána rozteč průjezdů 349 mm, což odpovídá pěti vodorovným průjezdům robota přes součást (výška součásti je 1500 mm). Pokud bychom tuto rozteč použili, spodní a horní průjezdy by byly pouhých pár desítek milimetrů od kraje, což není dobré z hlediska efektivity přenosu barvy. Z tohoto důvodu nastavíme rozteč na hodnotu 340 mm. Musíme si samozřejmě uvědomit, že nám tato změna může negativně ovlivnit kvalitu povrchu. V tomto případě se jedná o díl, u kterého větší tloušťka barvy bude pouze výhodou.

Z vytvořené trajektorie se v posledním kroku vygeneruje kód pro *Robot Studio*. Vygenerovaná trajektorie bohužel obsahuje podstatně více bodů, než je pro lakovací proces

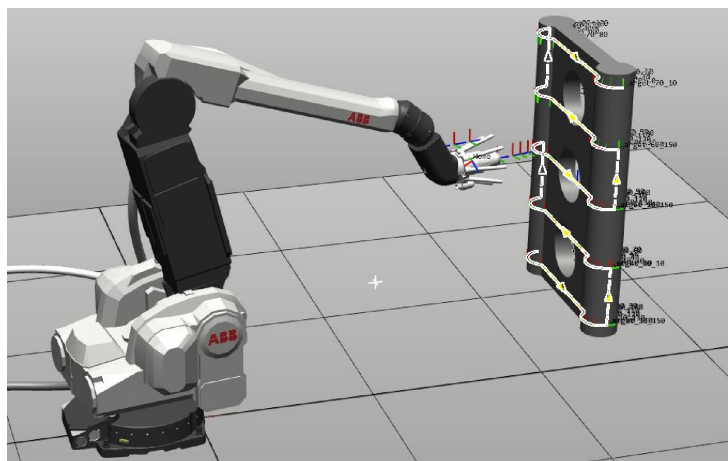




**Obr. 67:** Vygenerovaná trajektorie - obr. včetně modelu

nutné, a tak můžeme většinu bodů odstranit. Po odstranění bodů provedeme automatické nalezení konfigurací robota a rozdělení jedné procedury, jež byla automaticky vygenerována do více podprocedur. Tím získáme větší kontrolu nad optimalizací procesu. Dále pro každou lichou podtrajektorii změním směr, aby robot lakoal nejen zleva doprava, ale i zprava doleva.

Jelikož program nebere v úvahu dosažitelnost bodů, musíme některé z bodů natočit podle lokální nástrojové osy z tak, aby byly body dosažitelné. Zároveň je nutné přidat další pohybové instrukce, aby se mohl robot "vymotávat" a předešlo se singulárním, či nedosažitelným polohám. Po celkových úpravách, které trvaly zhruba 1 hodinu je simulace s časem cyklu 22.4 s spustitelná a připravena k nasazení v reálném testu.



**Obr. 68:** Simulace v programu Robot Studio

## 6 Závěr

Diplomová práce se zabývá vývojem algoritmu pro automatické hledání trajektorie v oblasti lakování obecných ploch definovaných síťovým modelem.

První polovina teoretické části práce představuje robotické lakovací systémy spolu s obecným popisem jejich řízení. Dále jsou rozvedeny postupy hledání optimálního nastavení lakovacích procesů. V této části jsou také zmíněny metody zjištění efektivity přenosu barvy, měření šířky paprsku, ale třeba i konvence polohy trajektorií.

Druhá polovina teoretické části se věnuje uvedení do problematiky plánovacích algoritmů, hledání vstupních a výstupních parametrů, které slouží pro návrh algoritmu. Tyto parametry jsou dále detailně rozvedeny a následně použity v praktické části. Součástí tohoto bloku je také popis vybraných přístupů hledání trajektorií a stručný průzkum trhu se softwary, které umožňují automatické hledání trajektorií v oblasti lakování. Mezi detailně popsané přístupy hledání trajektorie patří například algoritmus MFT (sledování povrchu), standardní analytická metoda a plánování trajektorie použitím rekonstrukce iso-parametrických ploch.

V praktické části práce jsou první dvě kapitoly věnovány návrhu procesu použití aplikace a návrhu uživatelského rozhraní. Následuje popis jednotlivých částí programu, z kterých se aplikace skládá.

Součástí práce je také experimentální část, v které byla prakticky otestována použitelnost programu ve virtuální případové studii s reálnými vzorky lakování. Na základě provedeného experimentu lze konstatovat, že navržený algoritmus byl pro daný problém velice efektivní a časová náročnost na přípravu trajektorie byla minimalizována. Aplikace však vyžaduje kvalifikovanou obsluhu, jelikož je nutné provést kontrolu výstupních dat. Především se jedná o kontrolu bodů na hranách nemanifoldního modelu, odstranění přebytečných bodů a natočení některých bodů z důvodů dosažitelnosti mechanismu.

## Seznam použitých značek, zkratek a symbolů

<b>Značka</b>	<b>Veličina</b>	<b>Jednotka</b>
I	Elektrický proud	[A]
U	Elektrické napětí	[V]

<b>Zkratka</b>	<b>Význam</b>
VN	vysoké napětí
HVLP	High Volume, Low Pressure
GUI	Graphical User Interface
UI	User Interface
CC	Cutter-Contact
IRF	International Federation of Robotics
ABB	Asea Brown Boveri
IPS	Integrated Process System
RZ	Rotační zvon
CBS	Cartridge Bell Paint System
CMYK	Cyan, Magenta, Yellow, Key
RGB	Red, Green, Blue
SW	Short Wave   Software
LW	Long Wave
DOI	Distinctness of image
PFR	Paint Flow Rate
TE	Transfer Efficiency
SC	Solid Content
ATPS	Automatic Trajectory Planning System
CAD	Computer Aided design
NURBS	Non-uniform rational B-spline
STL	Standard Tessellation Language, Standard Triangle Language
IGES	Initial Graphics Exchange Specification
VRML	Virtual Reality Modeling Language
CNC	Computer Numerical Control
MFT	Mesh following technique

## Seznam použité literatury a zdrojů

- [1] Computer history museum. Dostupné online z: <http://www.computerhistory.org/timeline/> (26. 7. 2018).
- [2] Quaternions and spatial rotation. Dostupné online z: [https://en.wikipedia.org/wiki/Quaternions\\_and\\_spatial\\_rotation](https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation) (24.12.2018), Publikováno 2018.
- [3] ABB. Operating manual - introduction to rapid (controller software irc5), abb se-721 68 västerås sweden. Dostupné online z: [http://www.oamk.fi/~eeroko/Opetus/Tuotantoautomaatio/Robotiikka/Introduction\\_to\\_RAPID\\_3HAC029364-001\\_rev-\\_en.pdf](http://www.oamk.fi/~eeroko/Opetus/Tuotantoautomaatio/Robotiikka/Introduction_to_RAPID_3HAC029364-001_rev-_en.pdf) (10.10.2018).
- [4] ABB. Operating manual - introduction to rapid (controller software irc5), abb se-721 68 västerås sweden. Dostupné online z: [http://www.oamk.fi/~eeroko/Opetus/Tuotantoautomaatio/Robotiikka/Introduction\\_to\\_RAPID\\_3HAC029364-001\\_rev-\\_en.pdf](http://www.oamk.fi/~eeroko/Opetus/Tuotantoautomaatio/Robotiikka/Introduction_to_RAPID_3HAC029364-001_rev-_en.pdf) (10.10.2018).
- [5] ABB. Robotware - os - rapid technical manual, abb se-721 68 västerås sweden. Dostupné online z: <http://developercenter.robotstudio.com/BlobProxy/manuals/RapidIFDTechRefManual/doc498.html> (10.10.2018).
- [6] Adam H. Aitkenhead. Compute mesh normals. Dostupné online z: [https://www.mathworks.com/matlabcentral/fileexchange/29585-compute-mesh-normals?s\\_tid=prof\\_contriblnk](https://www.mathworks.com/matlabcentral/fileexchange/29585-compute-mesh-normals?s_tid=prof_contriblnk) (13.10.2018), Publikováno 2010.
- [7] Mayur V. Andulkar, Shital S. Chiddarwar, and Akshay S. Marathe. Novel integrated offline trajectory generation approach for robot assisted spray painting operation. *Journal of Manufacturing Systems*, 37:201 – 216, 2015.
- [8] Mayur V. Andulkar, Shital S. Chiddarwar, and Akshay S. Marathe. Novel integrated offline trajectory generation approach for robot assisted spray painting operation. *Journal of Manufacturing Systems*, 37:202, 2017.
- [9] Mayur V. Andulkar, Shital S. Chiddarwar, and Akshay S. Marathe. Novel integrated

- offline trajectory generation approach for robot assisted spray painting operation. *Journal of Manufacturing Systems*, 37:203, 2017.
- [10] Robert Ayres and Miller Steve. The impacts of industrial robots, robotic institute, carnegie-mellon university, pittsburg, pa 15213, usa. pages 4–8.
- [11] Sunil Bhandari. Community profile - sunil bhandari. Dostupné online z: <https://www.mathworks.com/matlabcentral/profile/authors/7658874> (25.10.2018), sep 2018.
- [12] Marshall Burns. Automated fabrication: improving productivity in manufacturing, englewood cliffs, n.j. : Ptr prentice hall, 1993.
- [13] T. Chen, Heping; Fuhlbrigge and Xiongzi Li. Automated industrial robot path planning for spray painting process: A review, ieee. page 523, Publikováno 2008.
- [14] B.K. Choi, C.S. Lee, J.S. Hwang, and C.S. Jun. Compound surface modelling and machining. *Computer-Aided Design*, 20(3):127 – 136, 1988.
- [15] Carmelo Mineo; Stephen Gareth Pierce ; Pascual Ian Nicholson ; Ian Cooper. Introducing a novel mesh following technique for approximation-free robotic tool path trajectories, strana 192.
- [16] Carmelo Mineo; Stephen Gareth Pierce ; Pascual Ian Nicholson ; Ian Cooper. Introducing a novel mesh following technique for approximation-free robotic tool path trajectories, strana 195.
- [17] GRACO Corp. How a rotary bell atomizer works. Dostupné online z: [http://www.graco.com/content/dam/graco/ipd/literature/misc/mfg\\_sp\\_how\\_a\\_rotary\\_bell\\_works/How\\_a\\_rotary\\_bell\\_worksEN-A.pdf](http://www.graco.com/content/dam/graco/ipd/literature/misc/mfg_sp_how_a_rotary_bell_works/How_a_rotary_bell_worksEN-A.pdf) (8.9.2018).
- [18] Julien Cretel. Why is this polynomial equation badly conditioned? Dostupné online z: <https://stackoverflow.com/questions/32173321/why-is-this-polynomial-equation-badly-conditioned> (13.10.2018), Srpen 2015.
- [19] José M. Medina; José A. Díaz. Classification of batch processes in automotive metallic coatings using principal component analysis similarity factors from reflectance spectra.

pages 75–83, Publikováno 2015.

- [20] Tawfik Elmidany, Ahmed Elkeran, Ahmed Galal, and Mohammed Elkhateeb. Nurbs surface reconstruction using rational b-spline neural networks. 2014.
- [21] ABB 3HNT 00217-1 en Rev.02. Paint system overview t00217, manual identification file. page 16.
- [22] ABB 3HNA012856-001 en Rev.22. Unit description, paint, manual identification file. page 22.
- [23] ABB 3HNA012856-001 en Rev.22. Unit description, paint, manual identification file. page 27.
- [24] Gamin. Technický list elcometer 456 04. Dostupné online z: [https://www.elcometer.cz/fileadmin/user\\_upload/Elcometer\\_456\\_02.pdf](https://www.elcometer.cz/fileadmin/user_upload/Elcometer_456_02.pdf) (10.10.2018).
- [25] Mark Gundlach. Tolerancing in flexo and offset printing. Dostupné online z: <https://www.xrite.com/blog/tolerancing-in-flexo-and-offset-printing> (06.10.2018).
- [26] Brendan Hassett and Yuri Tschinkel. Weak approximation over function fields. *Inventiones mathematicae*, 163(1):171–190, 2006;2004;.
- [27] Wesley H. Huang. Optimal line-sweep-based decompositions for coverage algorithms, department of computer science. *Proceedings of the 2001 IEEE International Conference on Robotics and Automation Seoul, Korea , May 21-26, 2001*.
- [28] Wesley H. Huang. The minimal sum of altitudes decomposition for coverage algorithms. 2000.
- [29] 3D Molier International. Car door frame. Dostupné online z: <https://www.turbosquid.com/3d-models/3d-model-car-door-frame/956455> (04.10.2018).
- [30] Stanford University James Diebel. Representing attitude, euler angles, unit quaternions, and rotation vectors. page 13, 2016.

- [31] Antonio JK. Optimal trajectory planning for spray coating. *Proceedings of IEEE international conference on robotics and automation, San Diego, CA*, 2573:2570 – 2577, 1994.
- [32] Eric Johnson. Stl file reader. Dostupné online z: <https://www.mathworks.com/matlabcentral/fileexchange/22409-stl-file-reader?focused=5193625&tab=function> (14.10.2018), Publikováno 2011.
- [33] José A. Díaz José M. Medina. Advances in industrial color testing of automotive metallic finishes, universidad de granada, facultad de ciencias, departamento de Óptica, edificio mecenas, granada 18071, spain. Dostupné online z: <https://www.photonics.com/Article.aspx?AID=57311> (4.10.2018), Publikováno 2015.
- [34] Weihua Sheng; Ning Xi; Mumin Song; Yifan Chen; P. MacNeille. Automated cad-guided robot path planning for spray painting of compound surfaces. *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113)*, 3:1918–1923, 2000.
- [35] Bob Malone. George devol: A life devoted to invention, and robots, iee spectrum. Dostupné online z: <https://spectrum.ieee.org/automaton/robotics/industrial-robots/george-devol-a-life-devoted-to-invention-and-robots> (26.7.2018), Publikováno: 26. Srpna 2011.
- [36] Lucas Ariel Martinez. Automotive rotary-bell spray painting, modelling and simulation. *Degree project for Master of Science, Department of Physics, University of Gothenburg*, (4):1,4–5.
- [37] Rajesh Matai, Surya Singh, and M.L. Mittal. Traveling salesman problem: an overview of applications, formulations, and solution approaches. 11 2010.
- [38] Carmelo Mineo, Stephen Gareth Pierce, Pascual Ian Nicholson, and Ian Cooper. Robotic path planning for non-destructive testing – a custom matlab toolbox approach. *Robotics and Computer-Integrated Manufacturing*, 37:1 – 12, 2016.
- [39] Rodrigo Minetto, Neri Volpato, Jorge Stolfi, Rodrigo M.M.H. Gregori, and Murilo V.G. da Silva. An optimal algorithm for 3d triangle mesh slicing. *Computer-Aided Design*, 92:2, 2017.

- [40] Tom Lyche; Knut Mørken. Spline methods - lecture notes, kapitola: Spline approximation of functions and data, strana 97-102.
- [41] RU Nikita Kozin. Paint gun. Dostupné online z: <https://thenounproject.com/term/paint-gun/922219/> (04.10.2018).
- [42] International Federation of Robotics. How robots conquer industry worldwide, ifr.frankfurt main, germany. page 6, Publikováno 2017.
- [43] J.-F. Remacle, C. Geuzaine, G. Compère, and E. Marchandise. High-quality surface remeshing using harmonic maps. *International Journal for Numerical Methods in Engineering*, 83(4):407.
- [44] Jana Pospíšilová; Lenka Roušarová. Semestrální práce z předmětu kartografická polygrafie a reprografie, barvové prostory, Čvut. Dostupné online z: [http://geo3.fsv.cvut.cz/vyuka/kapr/SP/2008\\_2009/pospisilova\\_rousarova/index.html](http://geo3.fsv.cvut.cz/vyuka/kapr/SP/2008_2009/pospisilova_rousarova/index.html) (4.10.2018), Publikováno 2008.
- [45] Sung-Kee Noh Suk-Hwan Suh, In-Kee Woo. Development of an automatic trajectory planning system(atps)for spray painting robots. *Journal of Manufacturing Systems, Conference on Robotics and Automation*, 10/No.5(4):1948–1955, 1991.
- [46] Yuwen Sun, Dongming Guo, zhenyuan Jia, and Haixia Wang. Iso-parametric tool path generation from triangular meshes for free-form surface machining. *International Journal of Advanced Manufacturing Technology*, 28(7/8):721, 2006.
- [47] M. Szilvési-Nagy and Gy. Mátyási. Analysis of stl files. *Mathematical and Computer Modelling*, 38(7):945 – 960, 2003. Hungarian Applied Mathematics.
- [48] Pål Johan; Gravdahl; Jan Tommy. A real-time algorithm for determining the optimal paint gun orientation in spray paint applications. *IEEE Transactions on Automation Science and Engineering*, 7:803 – 816, 2010.
- [49] Loring W. Tu. An introduction to manifolds, springer science+ business media, llc 2011.
- [50] Vincent Mia Edie Verheyen. Matlab curve fitting - least squares method - wrong “fit” using high degrees, Publikováno 2016.



# Seznam obrázků, tabulek a kódů

## Seznam obrázků

Obrázek 1	První průmyslový robot Unimate (1961) . . . . .	11
Obrázek 2	Integrované průmyslové roboty od roku 1970 do roku 2017 . . . . .	12
Obrázek 3	Příklad lakovací sestavy IPS s čerpadlem na robotu IRB5400 . . . . .	13
Obrázek 4	Systémový diagram . . . . .	14
Obrázek 5	Procesní diagram vzduchového okruhu pro RB1000 . . . . .	15
Obrázek 6	Diagram toku barvy pro RB1000 s použitím jednotky 2K Mixer . . . . .	16
Obrázek 7	Typicky dosahované (používané) tvary stříkaných paprsků při robotickém lakování . . . . .	17
Obrázek 8	Vzorek před (vlevo) a po (vpravo) nástřiku barvou . . . . .	21
Obrázek 9	Příklad posuzování šířky paprsku . . . . .	21
Obrázek 10	Příprava stupnice . . . . .	22
Obrázek 11	Analýza vzorku . . . . .	22
Obrázek 12	Nástřik barvy na hliníkovou folii (vlevo); efektivita přenosu barvy na součást (vpravo) . . . . .	23
Obrázek 13	Model barevného prostoru CIE-LAB . . . . .	25
Obrázek 14	Příklady rozdílného způsobu položení trajektorie . . . . .	26
Obrázek 15	Schéma ATPS algoritmu . . . . .	28
Obrázek 16	Metodika hledání trajektorie - současnost . . . . .	28
Obrázek 17	Sít'ový model - trojúhelníková aproximace . . . . .	30
Obrázek 18	Poloha trysky vůči povrchu . . . . .	32
Obrázek 19	Konfigurace robota . . . . .	33
Obrázek 20	Znázornění zóny pohybu robota . . . . .	35
Obrázek 21	Aproximace polynomiální plochou, (a) STL model (b) polynom 3. stupně, (c) polynom 30. stupně . . . . .	37
Obrázek 22	Příklad vytvoření harmonické mapy z STL modelu pro kruhový výřez . . . . .	38
Obrázek 23	Body vzniklé průsečíkem roviny se sítí modelu . . . . .	39
Obrázek 24	Postup generování trajektorie . . . . .	40
Obrázek 25	Příklad způsobu položení trajektorie: horizontální (vlevo), vertikální (vpravo) . . . . .	40
Obrázek 26	Graf a jeho možnosti rozložení . . . . .	41
Obrázek 27	Diagram použití aplikace . . . . .	44
Obrázek 28	Diagram procesu pouze pro analýzu tloušťkového profilu . . . . .	45
Obrázek 29	Diagram procesu pouze pro generování optimální trajektorie . . . . .	45
Obrázek 30	Pohled na hlavní menu okna aplikace . . . . .	46
Obrázek 31	GUI - Správa modelu . . . . .	46

Obrázek 32	GUI - Analýza tloušťkových profilů . . . . .	47
Obrázek 33	GUI - Ukázka nalezení optimální trajektorie na obdélníkové desce . . .	48
Obrázek 34	GUI - Ukázka karty pro generování výstupu . . . . .	49
Obrázek 35	Normálový vektor plochy trojúhelníku . . . . .	51
Obrázek 36	Diagram importu STL souboru . . . . .	51
Obrázek 37	Porovnání kvality exportu do STL. FreeCAD (vlevo), Autodesk Inventor Professional (vpravo) . . . . .	52
Obrázek 38	Datové toky v rámci výpočtu povrchových normál . . . . .	53
Obrázek 39	Vliv souslednosti bodů na orientaci normálového vektoru . . . . .	55
Obrázek 40	Datové toky v rámci zobrazení povrchových normál . . . . .	56
Obrázek 41	Datové toky pro aproximační funkce . . . . .	57
Obrázek 42	Diagram datových toků pro výpočet optimální rozteče . . . . .	60
Obrázek 43	Teoreticky nanesená vrstva barvy . . . . .	61
Obrázek 44	Rozdělení intervalů . . . . .	62
Obrázek 45	Průběh sčítání objemů . . . . .	63
Obrázek 46	Přeskupování pole pro jednotlivé kroky . . . . .	64
Obrázek 47	Diagram algoritmu výpočtu optimální rozteče . . . . .	65
Obrázek 48	Směr sčítání objemů v rámci pole $V_{diff_{ij}}$ . . . . .	66
Obrázek 49	Diagram datových toků pro funkci DisplayPitchOverlay() . . . . .	68
Obrázek 50	Naznačení metody řezu na STL síťi obdélníku . . . . .	69
Obrázek 51	Diagram algoritmu hledání trajektorie . . . . .	71
Obrázek 52	Vizualizace problému hledání průniku roviny s trojúhelníky . . . . .	72
Obrázek 53	Znázornění problémů 2D úlohy . . . . .	75
Obrázek 54	Grafické znázornění principu kvaternionu . . . . .	78
Obrázek 55	Naznačené řešení výpočtu polohy nástroje . . . . .	79
Obrázek 56	Vzorek . . . . .	83
Obrázek 57	Způsob měření vzorků . . . . .	84
Obrázek 58	Distribuce barvy - vzorek 1 . . . . .	85
Obrázek 59	Distribuce barvy - vzorek 2 . . . . .	85
Obrázek 60	Distribuce barvy - vzorek 3 . . . . .	85
Obrázek 61	Distribuce barvy - vzorek 4 . . . . .	86
Obrázek 62	Distribuce barvy - vzorek 5 . . . . .	86
Obrázek 63	Distribuce barvy - vzorek 6 . . . . .	86
Obrázek 64	Distribuce barvy zvoleného vzorku (vlevo), aproximace vzorku (vpravo) . . . . .	87
Obrázek 65	Překryv pro aproximaci spline křivkou . . . . .	87
Obrázek 66	Vygenerovaná trajektorie . . . . .	88
Obrázek 67	Vygenerovaná trajektorie - obr. včetně modelu . . . . .	89
Obrázek 68	Simulace v programu Robot Studio . . . . .	89

## Seznam tabulek

Tabulka 1	Tabulka účinností aplikátorů . . . . .	18
Tabulka 2	Požadavky na vstupní data pro analýzu tloušťkové vrstvy . . . . .	57

## Seznam kódů

1	Algoritmus přeskládání polí . . . . .	53
2	Pseudokód pro ověřování STL modelu 1/2 . . . . .	54
3	Pseudokód pro ověřování STL modelu 1/2/2 . . . . .	54
4	Algoritmus výpočtu a zobrazení normálových vektorů . . . . .	56
5	Postup pro hledání stupně polynomu . . . . .	58
6	Algoritmus přípravy polí pro výpočet optimální rozteče průjezdů . . . . .	64
7	Hlavní část algoritmu pro výpočet objemů pod spline křivkou . . . . .	66
8	Hlavní část algoritmu pro výpočet objemů pod polynomem . . . . .	67
9	Procedura připravující pole pro finální vykreslení výsledků . . . . .	68
10	Způsob generování zdrojového kódu pro roboty . . . . .	81

## Seznam použitého SW

- Texmaker
- MiKTeX ( $\LaTeX$ )
- MATLAB
- BibTex Citation Machine - online citační nástroj pro BibTex
- PDFescape - online PDF editor
- ABB Robot Studio
- ABB Paint Power Pack
- Draw.io
- Autodesk Inventor 2019 Professional
- Microsoft Excel 2016
- Microsoft Word 2016
- Microsoft Paint Tool
- Microsoft Snipping Tool

## Seznam příloh

1. Zdrojové kódy aplikace
2. Protokoly z měření
3. Průzkum trhu

## Seznam souborů na přiloženém CD

1. Diplomová práce ve formátu PDF
2. Zdrojové soubory diplomové práce ve formátu  $\text{\LaTeX}$
3. Zdrojové kódy aplikace
4. Prezentace
5. Zdroje
6. Protokoly z měření
7. Simulace v Robot Studiu
8. Průzkum trhu
9. Fotodokumentace měření

## Tištěné přílohy (kódy)

V následující sekci práce jsou vytištěné klíčové kódy, které slouží jako stavební jednotka celé aplikace. Veškeré přílohy jsou přiloženy na přiloženém CD.

### **Abecedně jsou vloženy funkce:**

1. ApproxWithSpline
2. AproxWithPolynom
3. ChangeStructure
4. ComputeNormals
5. ComputePitch
6. DisplayNormals
7. DisplayPitchOverlay
8. FindTrianglePlaneIntersection
9. GeneratePath
10. GenerateRAPID
11. PlotTrajectoryFcn
12. PrepareDataForRAPID

## ApproxWithSpline

```
function [ApproximateWithSpl,Coef] = ApproxWithSpline(ExcelFileName)

% This function returns data containing approximation with spline curve

%=====
% Read data from excel and prepare for processing
%=====

    format long;
    data = xlsread(ExcelFileName);

    xData = data(:,1);
    yData = data(:,2);

    P2 = spline(xData,yData);
%   P2 = spapi(5,xData,yData);

%=====
% Approx. with spline and prepare data as output
%=====

    fnplt(P2);
    ApproximateWithSpl = fnplt(P2);
    Coef=P2.coefs;
    hold on;

end
```

## ApproxWithPolynom

```
function [ApproximateWithPol,Coef] = ApproxWithPolynom(Target,ExcelFileName)

% This function returns polynom aproximation data

%=====
% Get data
%=====

format long;
Data = xlsread(ExcelFileName);

x = Data(:,1);
y = Data(:,2);

DimSize = size(Data,1);

%=====
% if file is empty, do nothing
%=====
if DimSize > 0

    if DimSize > 24
        PolDim = ceil(DimSize/3)-1;
    else
        PolDim = ceil(DimSize/2);
    end

    if PolDim > 40
        PolDim = 40;
    end

    Prompt = {'Enter a degree of polynom'};
    Title = 'Polynom degree';
    DefInput = {int2str(PolDim)};
    opts.Interpreter = 'tex';
    UserInput = inputdlg(Prompt,Title,[1 40],DefInput,opts);

    PolDimOut = round(str2double(cell2mat(UserInput)));

    % Check for a valid integer.
    if isnan(PolDimOut)
        % User clicked Cancel, or entered a character, symbols, or something else not
allowed.
        PolDimOut = PolDim;
        Message = sprintf('You did not an integer.\nI will use sugested value: %d and
continue.', PolDimOut);
        uiwait(warndlg(Message));
    end

    PolDim=PolDimOut;

    FirstVal = Data(1,1);
    LastVal = Data(end,1);

    P = polyfit(x,y,PolDim);

    xfit = FirstVal:1>LastVal;
    yfit = polyval(P,xfit);
```

```
ApproximateWithPol = plot(Target,xfit,yfit);  
Coef=P;  
hold on;  
end  
end
```



## ChangeStructure

```
function [VerticesGroup1,VerticesGroup2,VerticesGroup3,LimitMax,LimitMin] = ChangeStructure(VerArr)

%=====
% Change structure of STL model to be easier plotable and more organized
%=====

    VerticesAll = reshape(VerArr',9,size(VerArr,1)/3)';
    VerticesAll = [VerticesAll(:,:),VerticesAll(:,1:3),ones(size(VerticesAll,1),1)*[NaN, NaN, NaN]];
    VerticesAll = reshape(VerticesAll',3,size(VerticesAll,1)*5)';

    VerticesGroup1 = VerticesAll(:,1)'; VerticesGroup2 = VerticesAll(:,2)'; VerticesGroup3 = VerticesAll(:,3)';

    LimitMax = max(max(VerArr));
    LimitMin = min(min(VerArr));

end
```

## ComputeNormals

```
function [ArrayOfNormals,varargout] = ComputeNormals(ObjectSourceData)

% BASED ON FILENAME:      COMPUTE_mesh_normals.m (Adam H. Aitkenhead)

%=====
% Read the input parameters from sthread()
%=====
if isstruct(ObjectSourceData)==1
    TriangleFaces      = ObjectSourceData.faces;
    Vertex             = ObjectSourceData.vertices;
    ArrayOfVertices    = zeros(size(TriangleFaces,1),3,3);

    for i = 1:size(TriangleFaces,1)
        ArrayOfVertices(i,:,1) = Vertex(TriangleFaces(i,1),:);
        ArrayOfVertices(i,:,2) = Vertex(TriangleFaces(i,2),:);
        ArrayOfVertices(i,:,3) = Vertex(TriangleFaces(i,3),:);
    end
else
    ArrayOfVertices = ObjectSourceData;
end

%=====
% Initialise array to hold the normal vectors
%=====
TrinagleFacetCount = size(ArrayOfVertices,1);
ArrayOfNormals = zeros(TrinagleFacetCount,3);

%=====
% Check the vertex ordering for each facet
%=====

if nargin==2
    StartFacet = 1;
    EdgePointA = 1;
    Checked = false(TrinagleFacetCount,1);
    Waiting = false(TrinagleFacetCount,1);
    ManiFoldRisk = 0;
    AnsC = 1;
    while min(Checked)==0

        Checked(StartFacet) = 1;

        EdgePointB = EdgePointA + 1;
        if EdgePointB==4
            EdgePointB = 1;
        end

%=====
% Find mathpoints A
%=====
        CoX = ArrayOfVertices(:,1,:)==ArrayOfVertices(StartFacet,1,EdgePointA);
        CoY = ArrayOfVertices(:,2,:)==ArrayOfVertices(StartFacet,2,EdgePointA);
        CoZ = ArrayOfVertices(:,3,:)==ArrayOfVertices(StartFacet,3,EdgePointA);
        [TempA,TempB] = find(CoX & CoY & CoZ);
        MatchPointA = [TempA,TempB];
        MatchPointA = MatchPointA(MatchPointA(:,1)~=StartFacet,:);

%=====
% Find mathpoints B
```

```

=====
CoX = ArrayOfVertices(:,1,:)==ArrayOfVertices(StartFacet,1,EdgePointB);
CoY = ArrayOfVertices(:,2,:)==ArrayOfVertices(StartFacet,2,EdgePointB);
CoZ = ArrayOfVertices(:,3,:)==ArrayOfVertices(StartFacet,3,EdgePointB);
[TempA,TempB] = find(CoX & CoY & CoZ);
MatchPointB = [TempA,TempB];
MatchPointB = MatchPointB(MatchPointB(:,1)~=StartFacet,:);

=====
% Find edge which math both
=====

[MemberA,MemberB] = ismember(MatchPointA(:,1),MatchPointB(:,1));
MatchFacet = MatchPointA(MemberA,1);

if numel(MatchFacet)~=1
    if exist('warningdone','var')==0
        if ManiFoldRisk == 0
            answer = questdlg('Model Mesh is not manifold. Do you want to continue? Calculations can fail or can be inaccurate.','Continue','Quit');
        else
            AnsC = 2;
        end
        ManiFoldRisk = 1;
        switch answer
            case 'Continue'
                AnsC = 1;
            case 'Quit'
                AnsC = 2;
        end
    end
end
end

=====
% Flipping
=====

if ManiFoldRisk == 1
    % ans2 = maniFoldRisk; skip this part
else
    if AnsC == 1
        MatchPointA = MatchPointA(MemberA,2);
        MatchPointB = MatchPointB(MemberB(MemberA),2);
        if Checked(MatchFacet)==0 && Waiting(MatchFacet)==0

            %Ensure the adjacent edge is traveled in the opposite direction to the original edge
            if MatchPointB-MatchPointA==1 || MatchPointB-MatchPointA==-2

                %Direction needs to be flipped
                [ ArrayOfVertices(MatchFacet,:,MatchPointA) , ArrayOfVertices(MatchFacet,:,MatchPointB) ] = deal( ArrayOfVertices(MatchFacet,:,MatchPointB) , ArrayOfVertices(MatchFacet,:,MatchPointA) );
            end
        end
        elseif AnsC == 2
            return
        end
    end
end

Waiting(MatchFacet) = 1;

if EdgePointA<3

```

```

        EdgePointA = EdgePointA + 1;
    elseif EdgePointA==3
        EdgePointA = 1;
        Checked(StartFacet) = 1;
        StartFacet = find(Waiting==1 & Checked==0,1,'first');
    end

end

end

%=====
% Compute the normal vector for each face
%=====

for loopFace = 1:TrinagleFacetCount

    %Find the coordinates for each vertex.
    VerA = ArrayOfVertices(loopFace,1:3,1);
    VerB = ArrayOfVertices(loopFace,1:3,2);
    VerC = ArrayOfVertices(loopFace,1:3,3);

    %Compute the vectors AB and AC
    VectorAB = VerB-VerA;
    VectorAC = VerC-VerA;

    %Determine the cross product AB x AC
    CrossProductABxAC = cross(VectorAB,VectorAC);

    %Normalise to give a unit vector
    CrossProductABxAC = CrossProductABxAC / norm(CrossProductABxAC);
    ArrayOFNormals(loopFace,1:3) = CrossProductABxAC;

end

%=====
% Prepare the output parameters
%=====

if nargout==2
    meshdataOUT = ArrayOfVertices;
    varargout(1) = {meshdataOUT};
end

```

## ComputePitch

```
function [BestPitch] = ComputePitch(FileName, Method, OptThickness, ValRaStartNum, ValRaEndNum)

% This function returns best pitch

ExcelFileName = FileName;
OptimalThickness = OptThickness;
EvalStart = ValRaStartNum;
EvalEnd = ValRaEndNum;

%=====  
% Default Thinkness  
%=====

ClcStep = 10;  
RepeatC = 5;  
SelectedMethod = Method;

%=====  
% Read data from excel and prepare for processing  
%=====

data = xlsread(ExcelFileName);  
xData = data(:,1);  
yData = data(:,2);

ArraySize = size(xData,1);  
RangeStart = xData(1,1);  
RangeEnd = xData(ArraySize,1);

ArrayOfVolumesForProcessing = zeros(ArraySize-1,2);

CalcStep = (abs(RangeStart)+abs(RangeEnd))/(ArraySize-1)*(1/ClcStep);

DimSize = size(data,1);

if DimSize > 24  
    PolDim = ceil(DimSize/3)-1;  
else  
    PolDim = ceil(DimSize/2);  
end

if PolDim > 40  
    PolDim = 40;  
end

N = PolDim;

%=====  
% Find optimal arrays size  
%=====

P1 = spline(xData,yData);  
Coef = P1.coefs;  
ComputedSegments = size(Coef,1)*ClcStep;  
BestPitchStPreP = zeros(1,RepeatC);  
TestArr = zeros(1,100); % Testting porpuse only

%=====  
% Repeat operation to reduce numerical instability and increase accuracy  
%=====

% gui_active(3); % will add an abort button  
% h = progressbar( [],0,'Finding best pitch' );
```

```

% for t=1:1:100                                TESTING PORPUSE ONLY
for c=1:1:RepeatC
    %=====
    % Proceeds data according to the user choosed option
    %=====

    if 1 == strcmp(SelectedMethod,'Approximation with spline')

        %=====
        % Get spline and
        %=====

        for i= 1:1:ComputedSegments
            Range1s = RangeStart+(i-1)*CalcStep;
            Range1e = RangeStart+i*CalcStep;
            P1 = spline(xData,yData);
            IntPolynom = diff(fnval(fnint(P1),[Range1s;Range1e]));
            ArrayOfVolumesForProcessing(i,:)= [(Range1s+Range1e)/2,abs(IntPolynom)
];
            ArrayOfOptimalVolumesForProcessing(i,:)= [(Range1e-Range1s)*abs(Optima
lThickness)];
        end

    elseif 1 == strcmp(SelectedMethod,'Approximation with polynom')

        %=====
        % Get polynom and
        %=====

        for i= 1:1:ComputedSegments
            Range1s = RangeStart+(i-1)*CalcStep;
            Range1e = RangeStart+i*CalcStep;
            P2 = polyfit(xData,yData,N);
            PolFunc = polyint(P2);
            IntPolynom = diff(polyval(PolFunc,[Range1s Range1e]));
            ArrayOfVolumesForProcessing(i,:)= [(Range1s+Range1e)/2,abs(IntPolynom)
];
            ArrayOfOptimalVolumesForProcessing(i,:)= [(Range1e-Range1s)*abs(Optima
lThickness)];
        end

    else
        f = warndlg('The preferred method was not selected. Please select your pre
ferred option','Warning');
        return;
    end

    %=====
    % Define ranges
    %=====

    CompleteArray = ComputedSegments+ComputedSegments;
    NewRange1 = zeros(CompleteArray,2);
    NewRange2 = zeros(CompleteArray,2);
    RangeToDiagnostic = zeros(ComputedSegments*2,ComputedSegments*2);
    SumOfVal = zeros(CompleteArray,ComputedSegments);
    DiffForS = zeros(CompleteArray,ComputedSegments);
    GetResult = zeros(RangeEnd-RangeStart,1);

    %=====
    % Calculate volume of all segments [full arrays]
    %=====

```

```

for i=1:1:ComputedSegments
    Range1 = NewRange1 + [ArrayOfVolumesForProcessing ; zeros(ComputedSegments,2)]
;
    Range2 = NewRange2 + [zeros(i,2);ArrayOfVolumesForProcessing ; zeros(ComputedS
egments-i,2)];
    SumOfVal(:,i) = Range1(:,2) + Range2(:,2);
    ArrOfOptimal(:,i) = [ArrayOfOptimalVolumesForProcessing;ArrayOfOptimalVolumesF
orProcessing];
end

%=====
% Calculate difference between optimal and measured segments
%=====

for i=1:1:CompleteArray
    for j=1:1:ComputedSegments
        DiffForS(i,j) = SumOfVal(i,j) - ArrOfOptimal(i,j);
    end
end

%=====
% Reduce diagnosed range option 1) [1/4][1/2][1/4] [throw][keep][throw]
% - testing porpuse
% Reduce diagnosed range option 2) [0/6][4/6][2/6] [throw][keep][throw]
% - testing porpuse
% Reduce diagnosed range option 3) MANUAL - defined by user
%=====

Option = 3;

if Option == 1
    offset = ComputedSegments/2;
    for i=1:1:ComputedSegments
        for j=1:1:ComputedSegments
            RangeToDiagnostic(i,j)=DiffForS(i+offset,j);
            ChoosedOption = 1;
        end
    end
elseif Option == 2
    offset = 0;
    for i=1:1:(ComputedSegments+ComputedSegments/3)
        for j=1:1:ComputedSegments
            RangeToDiagnostic(i,j)=DiffForS(i+offset,j);
            ChoosedOption = 2;
        end
    end
elseif Option == 3
    offset = 0;
    for i=1:1:ComputedSegments
        for j=EvalStart:1:EvalEnd
            RangeToDiagnostic(i,j)=DiffForS(i+offset,j);
            ChoosedOption = 3;
        end
    end
end

%=====
% calc deviation and put in one Result array
%=====

```

```

if ChoosedOption == 1
    for z=1:1:ComputedSegments
        sum = 0.0;
        for p=1:1:ComputedSegments
            sum = sum + power(RangeToDiagnostic(p,z),2);
        end
        GetResult(z,1) = power((1/(ComputedSegments-1))*sum,0.5);
    end

elseif ChoosedOption == 2
    for z=1:1:ComputedSegments
        sum = 0.0;
        for p=1:1:(ComputedSegments+ComputedSegments/3)
            sum = sum + power(RangeToDiagnostic(p,z),2);
        end
        GetResult(z,1) = power((1/((ComputedSegments+ComputedSegments/3)-1))*sum,0
.5);
    end

elseif ChoosedOption == 3
    for z=1:1:ComputedSegments
        sum = 0.0;

        for p=EvalStart:1:(EvalEnd-1)
            sum = sum + power(RangeToDiagnostic(p,z),2);
        end

        if sum == 0
            sum = 10.^10;
        end

        GetResult(z,1) = power((1/((EvalEnd - EvalStart)-1))*sum,0.5);
    end
end

%=====
% Find best pitch (minimal deviation)
%=====

LocInRedRan = find(GetResult == min(GetResult(:)));

BestPitchStPreP(1,c) = LocInRedRan(1,1);

%i++ progressBar
% h = progressBar(h,1/RepeatC );
end
%Close progressBar

Suma = BestPitchStPreP(1,1)+BestPitchStPreP(1,2)+BestPitchStPreP(1,3)+BestPitchStPreP(
1,4)+BestPitchStPreP(1,5);

if ChoosedOption == 1
    OutPut = round(Suma/RepeatC-ComputedSegments/2);

elseif ChoosedOption == 2
    OutPut = round(Suma/RepeatC);

elseif ChoosedOption == 3
    OutPut = round(Suma/RepeatC);
end

```



```
%     TestArr(1,t) = OutPut     TESTING PORPUSE ONLY  
  
% end                           TESTING PORPUSE ONLY  
  
    BestPitch = OutPut;  
  
%     progressbar( h,-1 );  
end
```

## DisplayNormals

```
function [] = DisplayNormals(Target,coordVerticesOutput,coordNormalsOutput)
%=====
% prepare data sets
%=====

    x = squeeze(coordVerticesOutput(:,1,:))';
    y = squeeze(coordVerticesOutput(:,2,:))';
    z = squeeze(coordVerticesOutput(:,3,:))';

    % if nargin==2
    %   ArrayOfNormals = varargin{3};
    % end

    ArrayOfNormals =coordNormalsOutput;

%=====
% Color settings
%=====

    ColorPatchEdge = 'b';
    ColorPatchFace = 'c';
    ColorNormals   = 'r';

%=====
% plot normals
%=====

    % if nargin==2
    hold on;
        for i = 1:size(ArrayOfNormals,1)
            NormalStart = mean(coordVerticesOutput(i, :, :),3);
            NormalEnd = NormalStart + 3*ArrayOfNormals(i,:);
            plot3(Target, [NormalStart(1),NormalEnd(1)], [NormalStart(2),NormalEnd(2)], [Normal
Start(3),NormalEnd(3)],ColorNormals, 'LineWidth',2);
            % hold(app.UIAxesNormals);
            hold on;
        end

%=====
% plot all faces
%=====

% [PlotAllTogether] = patch(Target,x,y,z,ColorPatchEdge,'EdgeColor',ColorPatchEdge,'FaceCo
lor',ColorPatchFace);
% hold(app.UIAxesNormals);

hold on;
patch(Target,x,y,z,ColorPatchEdge,'EdgeColor',ColorPatchEdge,'FaceColor',ColorPatchFace);

% FOLLOWING PART USED TO BE USED DURING TESTING

% axis equal tight

% camlight('headlight');
% material('dull');
% end

%=====
% prepare outputs
```

```
%=====
% [varargout] = {PlotAllTogether};
% if nargin == 1
%     varargout(1) = {PlotAllTogether};
% end
```

## DisplayPitchOverlay

```
function [] = DisplayPitchOverlay(Target,FileName, ChooosedMethod, OptThickness, Pitch)
%UNTITLED Summary of this function goes here
% Detailed explanation goes here

%=====
% Data from UI and constants
%=====

ExcelFileName = FileName;
OptimalThickness = OptThickness;
OptimalPitchVal = Pitch;
SelectedMethod = ChooosedMethod;
ClcStep = 10;

%=====
% Read data from excel and prepare for processing
%=====
data = xlsread(ExcelFileName);
xData = data(:,1);
yData = data(:,2);

ArraySize = size(xData,1);
RangeStart = xData(1,1);
RangeEnd = xData(ArraySize,1);

ArrayOfVolumesSpline = zeros(ArraySize-1,2);

CalcStep = (abs(RangeStart)+abs(RangeEnd))/(ArraySize-1)*(1/ClcStep);

%=====
% Find optimal arrays size
%=====
P1 = spline(xData,yData);
Coef = P1.coefs;
ComputedSegments = size(Coef,1)*ClcStep;

% gui_active(3); % will add an abort button
% h = progressbar( [],0,'Finding best pitch' );

%=====
% Procees data according to the user choosed option
%=====

if 1 == strcmp(SelectedMethod,'Approximation with spline')

%=====
% Get spline and array of optimal volumes
%=====

for i= 1:1:ComputedSegments
RangeIs = RangeStart+(i-1)*CalcStep;
RangeIe = RangeStart+i*CalcStep;
P1 = spline(xData,yData);
IntPolynom = diff(fnval(fnint(P1),[RangeIs;RangeIe]));
ArrayOfVolumesSpline(i,:) = [(RangeIs+RangeIe)/2,abs(IntPolynom)];
ArrayOfOptimalVolumes(i,:) = [(RangeIe-RangeIs)*abs(OptimalThickness)];
end
```

```

elseif 1 == strcmp(SelectedMethod,'Approximation with polynom')

%=====
% Get polynom and array of optimal volumes
%=====

P2 = polyfit(xData,yData,15);

for i= 1:1:ComputedSegments
    Rangels = RangeStart+(i-1)*CalcStep;
    Rangele = RangeStart+i*CalcStep;
    PolFunc = polyint(P2);
    IntPolynom = diff(polyval(PolFunc,[Rangels Rangele]));
    ArrayOfVolumesSpline(i,:) = [(Rangels+Rangele)/2,abs(IntPolynom)];
    ArrayOfOptimalVolumes(i,:) = [(Rangele-Rangels)*abs(OptimalThickness)];
end
else
    f = warndlg('Something went wrong. Repeat simulation','Warning');
    return;
end

%=====
% Claculate scale factor and prepare arrays
%=====
% OLD APPROACH
% FirstDataPoint = data(1,2);
% FirstVolume = ArrayOfVolumesSpline(1,2);
MultScaleFactCoef = 1;

AdditionalArray2 = ComputedSegments/2 + (OptimalPitchVal-ComputedSegments/2);
CompleteArray = ComputedSegments + AdditionalArray2;
NewRange1 = zeros(CompleteArray,2);
NewRange2 = zeros(CompleteArray,2);

Range1 = NewRange1 + [ArrayOfVolumesSpline ; zeros(AdditionalArray2,2)];
Range2 = [zeros(AdditionalArray2,2) ; ArrayOfVolumesSpline] + NewRange2;

%=====
% Plot results
%=====

PlotProfile = [Range1(:,2)*MultScaleFactCoef Range2(:,2)*MultScaleFactCoef];
bar(Target,PlotProfile,'stacked');

hold on;
hold(Target);
bar(Target,Range2(:,2)*MultScaleFactCoef)
hold on;

%=====
% Old approach for Guide UI builder
%=====
% plot(Target,xlim,[OptimalThickness OptimalThickness], 'b','LineWidth',1.5)
% plot(xlim,[MinimalThickenss MinimalThickenss], 'r','LineWidth',1.5)
% ylabel('Thickness profile [not in scale!])
% xlabel(' [mm]')
% legend('1st pass','Overlay','2nd pass','Optimal Thickness','Minimal Thickness')

% bar(MargedArray(:,2),MargedArray(:,1))
% ArrayOfVolumes
% PitchDist

```

end

---

*Published with MATLAB® R2018a*

## FindTrianglePlaneIntersection

```
function [OutputPointsTrans,SizeOfOutput,OutputPoints2] = FindTrianglePlaneIntersection(Tr
ianglesToAnal,ZPlanesLevel, logicValC)

% Calculate intersection of triangles and plane

%=====
% Triangles which will be analyzed
%=====
TrianglesToAnalTrans = TrianglesToAnal';

%=====
% Devide array of triangles to coord of vertices (Vert for each triangle)
%=====
PointSet1 = TrianglesToAnalTrans(1:3,:);
PointSet2 = TrianglesToAnalTrans(4:6,:);
PointSet3 = TrianglesToAnalTrans(7:9,:);

%=====
% Prepare matrix of Z planes at the specific level recieved from function
% Generate path
%=====
ConstArrOfZlevel = ones(1,size(PointSet1,2))*ZPlanesLevel;

%=====
% Coord X, and Y are not interesting for us. Do elimination.
%=====
ConstOfArrZerosOnes = [ zeros(1,size(PointSet1,2));
                        zeros(1,size(PointSet1,2));
                        ones(1,size(PointSet1,2))];

% OLD SOLUTION - VERY DIF. TO UNDERSTAND
% intersect1 = PointSet1+bsxfun(@times,PointSet1-PointSet2,rdivide((ConstArrOfZlevel-sum(t
imes(ConstOfArrZerosOnes,PointSet1))),sum(times(ConstOfArrZerosOnes,(PointSet1-PointSet2)
)))));
% intersect2 = PointSet2+bsxfun(@times,PointSet2-PointSet3,rdivide((ConstArrOfZlevel-sum(t
imes(ConstOfArrZerosOnes,PointSet2))),sum(times(ConstOfArrZerosOnes,(PointSet2-PointSet3)
)))));
% intersect3 = PointSet3+bsxfun(@times,PointSet3-PointSet1,rdivide((ConstArrOfZlevel-sum(t
imes(ConstOfArrZerosOnes,PointSet3))),sum(times(ConstOfArrZerosOnes,(PointSet3-PointSet1)
)))));

TSet1 = rdivide((ConstArrOfZlevel-sum(times(ConstOfArrZerosOnes,PointSet1))),sum(times(Con
stOfArrZerosOnes,(PointSet1-PointSet2))));
TSet2 = rdivide((ConstArrOfZlevel-sum(times(ConstOfArrZerosOnes,PointSet2))),sum(times(Con
stOfArrZerosOnes,(PointSet2-PointSet3))));
TSet3 = rdivide((ConstArrOfZlevel-sum(times(ConstOfArrZerosOnes,PointSet3))),sum(times(Con
stOfArrZerosOnes,(PointSet3-PointSet1))));

    % pause program to make take some time to calm down, when there is big
    % amount of data processed, app is crashing
    pause(0.01);

%=====
% Using bsxfun to allow @times of {PointSet1-PointSet} with t1;
%=====
intersect1 = PointSet1+bsxfun(@times,PointSet1-PointSet2,TSet1);
intersect2 = PointSet2+bsxfun(@times,PointSet2-PointSet3,TSet2);
intersect3 = PointSet3+bsxfun(@times,PointSet3-PointSet1,TSet3);
```

```

pause(0.01);

%=====
% EVALUATION KEY | get data which meets all logical conditions
%=====
Logic1 = intersect1(3,:) <max(PointSet1(3,:),PointSet2(3,:)) & intersect1(3,:) > min(PointSet1(
3,:),PointSet2(3,:));
Logic2 = intersect2(3,:) <max(PointSet2(3,:),PointSet3(3,:)) & intersect2(3,:) > min(PointSet2(
3,:),PointSet3(3,:));
Logic3 = intersect3(3,:) <max(PointSet3(3,:),PointSet1(3,:)) & intersect3(3,:) > min(PointSet3(
3,:),PointSet1(3,:));
OutputPoints = [[intersect1(:,Logic1&Logic2);
                intersect2(:,Logic1&Logic2)], [intersect2(:,Logic2&Logic3);
                intersect3(:,Logic2&Logic3)], [intersect3(:,Logic3&Logic1);
                intersect1(:,Logic3&Logic1)]];
if ~isempty(OutputPoints)
    ZPlanesLevelArr = ones(size(OutputPoints,1),1)*ZPlanesLevel;
    OutputPoints2 = [OutputPoints, ZPlanesLevelArr];
end
OutputPointsTrans = OutputPoints';
SizeOfOutput = size(OutputPoints,1);

end

```



## GeneratePath

```
function [OutputArrayOfArray,ZplaneHeig, DataForABB] = GeneratePath(AllTriangles,Pitch)

%=====
% Find minimal and maximal Z coord. Use +1e-5 to reduce fuzzy problems
%=====

% MinModelLevel = min([AllTriangles(:,3); AllTriangles(:,6);AllTriangles(:,9)]);
% MaxModelLevel = max([AllTriangles(:,3); AllTriangles(:,6);AllTriangles(:,9)]);

MinModelLevel = min([AllTriangles(:,3);
                    AllTriangles(:,6);
                    AllTriangles(:,9)])-1e-5;
MaxModelLevel = max([AllTriangles(:,3);
                    AllTriangles(:,6);
                    AllTriangles(:,9)])+1e-5;

%=====
% Find layers WARNING: There are two different approaches available
%=====
% Sometimes is program causing instability, because number of layers is
% higher then expected. This problem is probably cause by Fuzzy states. For
% example if height of the object is 500 mm and pitch is 100 mm, then there
% will be two edges causing fuzzy problems. In this case is better to use
% simplified slicing from the lowest layer. (Simplified approach is bellow)
% ZplaneHeig = MinModelLevel : Pitch : MaxModelLevel;

MiddleL = (MaxModelLevel-MinModelLevel)/2;
PartOfSecBottom = (MiddleL-Pitch): -Pitch : MinModelLevel;
PartOfSecTop = MiddleL : Pitch : MaxModelLevel;
ZplaneHeig = sort([PartOfSecBottom,PartOfSecTop]);

OutputArrayOfArray = {};
DataForABB = {};
%=====
% Define new strucure with all coord points of vertices and on the last put
% one column with MIN value and one column with MAX Value. Final structure
% of AllTrianglesInNewStruc is [Number of triangles, Min, Max]
%=====
AllTrianglesInNewStruc = [AllTriangles(:,1:12),min(AllTriangles(:,[3 6 9]),[],2), max(AllT
riangles(:,[ 3 6 9]),[],2)];

%=====
% IFollowing part of code is used for finding intersection between observed
% layer and triangles
%=====

ModelSlices = ZplaneHeig;
MaxPlanes = 4000;
ArrOfTrangles = zeros(size(ZplaneHeig,2),MaxPlanes);
ArrOfTrianglesSize=zeros(size(ZplaneHeig,2),1);

for i = 1:size(AllTrianglesInNewStruc,1)

    %=====
    % Define first lyer which is in th emiddle of model heigh
    % Next: Analyze which case is going to be solved (All cases are
    % mentioned in the Thesis
```

```

% Algorithmn finding solution for AllTrianglesInNewStruc(i,12)
%=====

TraingleVertices13 = AllTrianglesInNewStruc(i,13);
ModelHeight= size(ModelSlices,2);
MinimumPitchLimit = 1; IntersecIsNotMatched = true;
LogicValA = true; LogicValB = true;

%=====
% Whole While part is taken over from: Sunil Bhandari
%=====
while IntersecIsNotMatched

    StartPointInMid = MinimumPitchLimit + floor((ModelHeight - MinimumPitchLimit)/2);

    if StartPointInMid == 1 && ModelSlices(StartPointInMid) >= TraingleVertices13
        SolutionCase = 2;
    elseif ModelSlices(StartPointInMid) <= TraingleVertices13 && StartPointInMid == si
ze(ModelSlices,2)
        SolutionCase = 2;
    elseif ModelSlices(StartPointInMid) > TraingleVertices13 && ModelSlices(StartPoint
InMid-1) < TraingleVertices13
        SolutionCase = 0;
    elseif ModelSlices(StartPointInMid) > TraingleVertices13
        SolutionCase = -1;
    elseif ModelSlices(StartPointInMid) < TraingleVertices13
        SolutionCase = 1;
    end

%=====
% Ceck output case
%=====
    if SolutionCase == -1
        ModelHeight = StartPointInMid - 1;
    elseif SolutionCase == 1
        MinimumPitchLimit = StartPointInMid + 1;
    elseif SolutionCase == 0
        TraingleVertices13 = StartPointInMid;
        IntersecIsNotMatched = false;
    elseif ModelHeight > MinimumPitchLimit || SolutionCase == 2
        LogicValA = false;
        IntersecIsNotMatched = false;
    end
end

ZPlaneLowInd = StartPointInMid;

%=====
% THE SAME APPROACH AS ABOVE! Only difference is evaluating column.
% In this cese is solved problem for MAX value in
% AllTrianglesInNewStruc(i,14)
%=====

TraingleVertices14 = AllTrianglesInNewStruc(i,14);

ModelHeight= size(ModelSlices,2);
MinimumPitchLimit = 1;
IntersecIsNotMatched = true;

while IntersecIsNotMatched
    StartPointInMid = MinimumPitchLimit + floor((ModelHeight - MinimumPitchLimit)/2);

```

```

    if StartPointInMid <=1
        DetectPosERROR = 0;
    end
    % try
        if StartPointInMid == 1 && ModelSlices(1) <= TraingleVertices14
            SolutionCase = 2;
        elseif StartPointInMid == size(ModelSlices,2) && ModelSlices(StartPointInMid)
<= TraingleVertices14
            SolutionCase = 2;
        elseif ModelSlices(StartPointInMid) > TraingleVertices14 && ModelSlices(StartP
ointInMid-1) < TraingleVertices14
            SolutionCase = 0;
        elseif ModelSlices(StartPointInMid) > TraingleVertices14
            SolutionCase = -1;
        elseif ModelSlices(StartPointInMid) < TraingleVertices14
            SolutionCase = 1;
        end
    % catch
    %     SolutionCase = 2;
    % end

    %=====
    % Ceck output case
    %=====
    if SolutionCase == -1
        ModelHeight = StartPointInMid -1;
    elseif SolutionCase == 1
        MinimumPitchLimit = StartPointInMid + 1;
    elseif SolutionCase == 0
        TraingleVertices14 = StartPointInMid;
        IntersecIsNotMatched = false;
    elseif ModelHeight > MinimumPitchLimit || SolutionCase == 2
        LogicValB = false;
        IntersecIsNotMatched = false;
    end
end

%=====
% Prepare output if conditions are meetig requirments
% i.e. line defined between two points intersects plane at the
% specific level
%=====
PlaneZHeightI = StartPointInMid;
if PlaneZHeightI > ZPlaneLowInd
    for j = ZPlaneLowInd:PlaneZHeightI -1
        ArrOfTrianglesSize(j) = ArrOfTrianglesSize(j) + 1;
        ArrOfTrangles(j,ArrOfTrianglesSize(j)) = i;
    end
end
end

ArrOfTrianglesToBeCheckForIntersection = ArrOfTrangles;

for CutNo = 1:size(ZplaneHeig,2)
    log = false;
    TrianglesToBeCheckedforK = ArrOfTrianglesToBeCheckForIntersection(CutNo,1:ArrOfTriangl
esSize(CutNo));

    % if k == 1
    %     log= true;
    % elseif k == size(ZplaneHeig,2)

```

```

%         log =false;
%     end

%=====
% Find intersections and get exact coordinates
%=====
[lines,linesize] = FindTrianglePlaneInterrection(AllTrianglesInNewStruc(TrianglesToBeCheckedforK,:), ZplaneHeig(CutNo),log);

if linesize ~= 0

%=====
% This procedure will find all matched points with its nodes
% and remove duplicated matches
%=====
StartNodes = lines(1:linesize,1:2);
EndNodes = lines(1:linesize,4:5);
AllNodes = [StartNodes; EndNodes];
connectivity = [];
SetTolerance = 1e-10;
SetIsMemberTol = 1e-8;

%=====
% C = uniquetol(A,tol) returns the unique elements in A using tolerance tol.
% Two values, u and v, are within tolerance if
% abs(u-v) <= tol*max(abs(A(:)))
% That is, uniquetol scales the tol input based on the magnitude of the data
.
% uniquetol is similar to unique. Whereas unique performs exact comparisons,
% uniquetol performs comparisons using a tolerance.
% Mathworks description
%=====
AllNodes = uniquetol(AllNodes,SetTolerance,'ByRows',true);
AllNodes = sortrows(AllNodes,[1 2]);

%=====
% LIA = ismembertol(A,B,tol) returns an array containing logical 1 (true)
% where the elements of A are within tolerance of the elements in B.
% Otherwise, the array contains logical 0 (false). Two values, u and v,
% are within tolerance if abs(u-v) <= tol*max(abs([A(:);B(:)]))
% That is, ismembertol scales the tol input based on the magnitude of the data.
ta.
% ismembertol is similar to ismember.
% Whereas ismember performs exact comparisons,
% ismembertol performs comparisons using a tolerance.
% Mathworks description
%=====
[~, n1] = ismembertol(StartNodes, AllNodes, SetIsMemberTol, 'ByRows',true);
[~, n2] = ismembertol(EndNodes, AllNodes,SetIsMemberTol, 'ByRows',true);
CoA = [n1 n2];
CoB = [n2 n1];

%=====
%Check for: repeated edges
%         too thin surfaces
%         unclosed loops
%=====

SolutionCase = ismember(CoB,CoA,'rows');
CoA(SolutionCase == 1,:)=[];
G = graph(CoA(:,1),CoA(:,2));

```

```

=====
% bins = conncomp(G) returns the connected components of graph G as bins.
% The bin numbers indicate which component each node in the graph belongs to
.
% If G is an undirected graph, then two nodes belong to the same component
% if there is a path connecting them.
% If G is a directed graph, then two nodes belong to the same strong
% component only if there is a path connecting them in both directions.
% Mathworks description
=====
bins = conncomp(G);

ListOfAwesomeCoordOUT = [];

for i = 1: max(bins)

    =====
    % Find first intersection
    % path = [];
    % dfsearch(G,s) applies depth-first search to graph G
    % starting at node s.
    % The result is a vector of node IDs in order of their
    % discovery.
    =====
    StartIntersectNode = find(bins==i, 1, 'first');
    AwesomePath = dfsearch(G, StartIntersectNode);
    AwesomePath = [AwesomePath; AwesomePath(1)];

    % Coordinates for x and y
    ListOfAwesomeCoord = [AllNodes(AwesomePath,1) AllNodes(AwesomePath,2)]
;
    if ~isempty(AwesomePath)
        if ListOfAwesomeCoord(1,1)>ListOfAwesomeCoord(2,1) || ListOfAwesom
eCoord(1,2)> ListOfAwesomeCoord(2,2)
            ListOfAwesomeCoord = ListOfAwesomeCoord(end:-1:1,:);
        end
    end
    % Connect to the first point
    ListOfAwesomeCoordOUT = [ListOfAwesomeCoordOUT;ListOfAwesomeCoord; [Na
N NaN]];

    ListOfAwesomeCoordOUTsize = size(ListOfAwesomeCoordOUT,1);
end
OutputArrayOfArray(CutNo) = {ListOfAwesomeCoordOUT};
ArrayOfZcoord = ones(length(ListOfAwesomeCoordOUT),1)*ZplaneHeig(CutNo);
ListOfAwesomeCoordOUTABB = [ListOfAwesomeCoordOUT, ArrayOfZcoord];
DataForABB(CutNo)={ListOfAwesomeCoordOUTABB};
end
end
end

```

## GenerateRAPID

```
function [RAPIDgoOUT] = GenerateRAPID(MainDataSet,ZoneData,VelocityData,MotionData,WorkObjectData,Points,Const,MoveInst)

% This function generates RAPID code for ABB robots
% Following structure is according all rules mentioned in the thesis
% As this is a complex structure, chain method has been used to make all
% steps clear and easy to understand

%=====
% Define and reallocate variables
%=====
MainDataSetArray = MainDataSet;
Zone = ZoneData;
Veloc = VelocityData;
Mot = MotionData;
Wobj = WorkObjectData;
Targets=0;

%=====
% Count number of recieved arrays
%=====
size(MainDataSetArray,2)
for i=1:size(MainDataSetArray,2)
    for j=1:size(MainDataSetArray{i})
        Targets = Targets + 1;
    end
end

%=====
% Count number of recieved arrays
%=====
% Prepare data with special content
T = 0;
for i=1:size(MainDataSetArray,2)
    GetData=MainDataSetArray{i};
    for j=1:size(GetData,1)
        T = T + 1;
        SpecialArr(T,:) = GetData(j,:);
    end
end

%=====
% Generate array of targets
% Generate first part: CONST robtarget Target_XXX
%=====
str = 'Target_';

for i=Targets:-1:1
    Val = i*10;
    ConstRobTarget{i,1} = sprintf('%s%d',str,Val);
end

%=====
% Generate second part: :=[[
%=====
str1 = 'CONST robtarget ';
str2 = ':=[[';
for i=Targets:-1:1
    StrBf = ConstRobTarget{i,1};
```

```

ConstRobTarget2{i,1} = sprintf('%s%s%s',str1,StrBf,str2);
end

%=====
% ADD point X,Y,Z coord
%=====
format short g
str3 = ',';
str4 = '],';

for i=Targets:-1:1
    Xcoord = SpecialArr(i,1);
    Ycoord = SpecialArr(i,2);
    Zcoord = SpecialArr(i,3);
    TextBf = ConstRobTarget2{i,1};
    ConstRobTarget3{i,1} = sprintf('%s%f%s%f%s%f%s',TextBf,Xcoord,str3,Ycoord,str3,Zcoord,
str4);
end

%=====
% ADD point Q1, Q2, Q3 and Q4
%=====
str5 = '[';
for i=Targets:-1:1
    Quaternion1 = SpecialArr(i,4);
    Quaternion2 = SpecialArr(i,5);
    Quaternion3 = SpecialArr(i,6);
    Quaternion4 = SpecialArr(i,7);
    TextBf = ConstRobTarget3{i,1};
    ConstRobTarget4{i,1} = sprintf('%s%f%s%f%s%f%s%f%s',TextBf,str5,Quaternion1,str3,Qua
ternion2,str3,Quaternion3,str3,Quaternion4,str4);
end

%=====
% ADD point Rob Config
% Deufalt config is [0,0,0,0]
%=====
config = '[0,0,0,0]';
for i=Targets:-1:1
    StrBf = ConstRobTarget4{i,1};
    ConstRobTarget5{i,1} = sprintf('%s%s',StrBf,config);
end

%=====
% ADD point External Axis
%=====
ExAx = ', [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]';
for i=Targets:-1:1
    StrBf = ConstRobTarget5{i,1};
    ConstRobTarget6{i,1} = sprintf('%s%s',StrBf,ExAx);
end

%=====
% Generate move instructions
%=====
gap = ' ';
coma = ',';
coma2 = '!';
for i=Targets:-1:1
    TargMove = ConstRobTarget{i,1};
    MoveRobTarget{i,1} = sprintf('%s%s%s%s%s%s%s%s%s',Mot,gap,TargMove,coma,Veloc,coma,Z

```

```
one, coma, Wobj, coma2);
end

%=====
% Define procedure keywords ROBOT STUDIO
%=====
StartProgram{1,1} = 'MODULE Module1';
EndProgram{1,1} = 'ENDMODULE';
StartProcedure{1,1} = 'PROC OnePaintingPath()';
EndProcedure{1,1} = 'ENDPROC';

RAPIDgoOUT=[StartProgram;ConstRobTarget6;StartProcedure;MoveRobTarget;EndProcedure;EndProgram];
```



## PlotTrajectory

```
function []= PlotTrajectoryFcn(TargetFig,ArrayOfArraysWithAllPoints, ZetLevelOfPlane)

%=====
%   Plots Trajectory between point in specific layers
%=====

    for i = 1: size(ArrayOfArraysWithAllPoints,2)
        PointsToProcess = ArrayOfArraysWithAllPoints{i};
        if ~isempty(PointsToProcess)
            for j = 1:size(PointsToProcess,1)-1
                Obrazek = plot3(TargetFig,PointsToProcess(j:j+1,1),PointsToProcess(j:j+1,2)
),ones(2,1)*ZetLevelOfPlane(i),'-r*');
                Obrazek.LineWidth= 1;
                hold(TargetFig,'on');
            end
        end
    end
end
```

## PrepareDataForRAPID

```
function [OutputForRAPIDGen] = PrepareDataForRAPID(SourceData,TargetFig)

%=====
% This function is used for postprocessing and preparing data for
% creating code for ABB robots and other outputs which are necessary
% for robot programming
%=====

OutputForRAPIDGen = {};

for i = 1:size(SourceData,2)
%=====
%   select one array of array
%=====
EvalData = SourceData{i};

%   Skip empty arrays
if size(EvalData,1) ~= 0
    AvgV = 0;
    for j=(1:size(EvalData,1)-2)

        A1 = [EvalData(j,1),EvalData(j,2),EvalData(j,3)];
        A2 = [EvalData(j+1,1),EvalData(j+1,2),EvalData(j+1,3)];
        A3 = [EvalData(j+2,1),EvalData(j+2,2),EvalData(j+2,3)];

%=====
% Get data for directional vector
%=====
U1x = A1(1,1);  U3x = A3(1,1);
U1y = A1(1,2);  U3y = A3(1,2);
U1z = A1(1,3);  U3z = A3(1,3);

%=====
% Define directional vector
%=====
u1 = [(U3x - U1x), (U3y - U1y), (U3z-U1z)];

%=====
% Get normal vector to directional vector
%=====
n1dir = [u1(1,2), -u1(1,1), u1(1,3)];

%=====
% Move normal vector to point A2
%=====
n1 = [A2(1,1)+n1dir(1,1), A2(1,2)+n1dir(1,2), A2(1,3)+n1dir(1,3)];
n2 = [A2(1,1), A2(1,2), A3(1,3)];

%=====
%Organise data
%=====
Normal2Store = [n1,n2];

%=====
%Do not show longer vectors then: DifferEnce/2 <= Avg
%=====
DifferEnce = n1-n2;
DifLenght= length(DifferEnce);
AvgV = AvgV + DifLenght;
```

```

    AvgOut = AvgV/j;
    if DifLenght/2 <= AvgOut
        % Difference
        Obrazek = quiver3(TargetFig,n2(1,1),n2(1,2),n2(1,3),DifferEnce(1),
DifferEnce(2),DifferEnce(3),0);
        Obrazek.LineWidth = 1;
        Obrazek.Color = 'g';
        hold(TargetFig,'on');
    end

%=====
% Prepare data for quaterion calculations
%=====
%-----
%          POINT                POINT2
%-----
N1x = Normal2Store(1,1);    N2x = Normal2Store(1,4);
N1y = Normal2Store(1,2);    N2y = Normal2Store(1,5);
N1z = Normal2Store(1,3);    N2z = Normal2Store(1,6);

%=====
% Qauterion extenstion
%=====
Alpha = real(atan(sqrt(((N1z-N2z)^2+(N1y-N2y)^2)/(N1x-N2x))));
Beta = real(atan(sqrt(((N1x-N2x)^2+(N1z-N2z)^2)/(N1y-N2y))));
Gamma = real(atan(sqrt(((N1x-N2x)^2+(N1y-N2y)^2)/(N1z-N2z))));

%=====
% Convert angles to quaterions
%=====
QuaternionABB = angle2quat(Alpha, Beta, Gamma);
Output(j,:) = [A2,QuaternionABB,Alpha, Beta, Gamma];
end
OutputForRAPIDGen(i)={Output};
end
end

```