FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Návrh datového postprocesu pro programování průmyslového robota pomocí zadaného pohybu v CAD nástroji

Bakalářská práce

PLZEŇ, 2022 **Veronika Endrštová**

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Veronika ENDRŠTOVÁ**
Osobní číslo:	**A19B0282P**
Studijní program:	**B0714A150005 Kybernetika a řídicí technika**
Specializace:	**Automatické řízení a robotika**
Téma práce:	**Návrh datového postprocesu pro programování průmyslového robota pomocí zadaného pohybu v CAD nástroji**
Zadávající katedra:	**Katedra kybernetiky**

Zásady pro vypracování

1. Seznamte se s možnostmi programování průmyslového robotu Stäubli –využijte simulační prostředí „Stäubli Robotics Suite“ (SRS).
2. Seznamte se s prostředím vybraného CAD systému a modelujte zvolený obrobek pro testování plánovače trajektorie pohybu.
3. Navrhněte postup pro nakreslení požadované trajektorie robotu s ohledem na obrobek a požadovanou zvolenou aplikaci (např. ostřik obrobku v průmyslových mycích linkách).
4. Navrhněte postup pro export zájmových bodů pohybové trajektorie.
5. Exportované body trajektorie využijte pro naplánování pohybu robotu v prostředí SRS a pohyb robotu simulujte.
6. Diskutujte výsledky a další možnosti práce.

Rozsah bakalářské práce: **30 – 40 stránek A4**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

1. W. Khalil, E. Dombre: Modelling, Identification and Control of Robots.
2. L. Sciavicco, B. Siciliano: Modelling and Control of Robot Manipulators.
3. Přednášky k předmětu URM.

Vedoucí bakalářské práce: **Ing. Martin Švejda, Ph.D.**
Výzkumný program 1

Datum zadání bakalářské práce: **15. října 2021**
Termín odevzdání bakalářské práce: **23. května 2022**

Západočeská univerzita v Plzni
Fakulta aplikovaných věd
L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Prof. Ing. Josef Psutka, CSc.
vedoucí katedry

V Plzni dne 15. října 2021

PROHLÁŠENÍ

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem bakalářskou práci vypracovala samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne 15. srpna 2022

. . . . . .
Veronika Endrštová

PODĚKOVÁNÍ

V první řadě bych chtěla poděkovat vedoucímu bakalářské práce Ing. Martinu Švejdovi Ph.D. za jeho veškerou pomoc ohledně práce. Především za čas, který mi věnoval a za všechny jeho přínosné rady a doporučení. Dále bych chtěla poděkovat zaměstnancům firmy ZF Electronics Klášterec s.r.o., kteří mi umožnili využívat jejich školícího robota TX2_60 a také za jejich odborné rady. V neposlední řadě bych ráda poděkovala své rodině a kamarádům za jejich podporu při studiu.

ANOTACE

Hlavním cílem této práce bylo vytvořit způsob vytvoření trajektorie pro robota používaného pro tlakové mytí obrobku během jeho výrobního procesu. Tato trajektorie má být nastavena v 3D CAD softwaru Autodesk Inventor. Zadáním je navrhnout způsob exportu bodů trajektorie z výkresu a následně ji importovat do prostředí Staubli Robotics Suite jako pohyb robotického ramene. K tomu byla použita aplikace s názvem „AItoSRS“, která byla vyvinuta speciálně pro tento účel.

Klíčová slova: Stäubli, SRS, Autodesk Inventor, plánování trajektorie, robotický manipulátor

ANNOTATION

The main objective of this thesis was to create a way of setting trajectory for a robot manipulator used for pressure washing of a workpiece during its manufacturing process. This trajectory is to be set in 3D CAD software Autodesk Inventor. The assignment is to design a way of exporting the trajectory points from the drawing and then import it to Staubli Robotics Suite environment as a motion of the robotic arm. This was done with the use of application called „AItoSRS“ which was developed specifically for this reason.

Key words: Stäubli, SRS, Autodesk Inventor, trajectory planning, robotic manipulator

ZADÁNÍ

1. Seznamte se s možnostmi programování průmyslového robotu Stäubli – využijte simulační prostředí „Stäubli Robotics Suite“ (SRS).
2. Seznamte se s prostředím vybraného CAD systému a modelujte zvolený obrobek pro testování plánovače trajektorie pohybu.
3. Navrhněte postup pro nakreslení požadované trajektorie robotu s ohledem na obrobek a požadovanou zvolenou aplikaci (např. ostřik obrobku v průmyslových mycích linkách).
4. Navrhněte postup pro export zájmových bodů pohybové trajektorie.
5. Exportované body trajektorie využijte pro naplánování pohybu robotu v prostředí SRS a pohyb robotu simulujte.
6. Diskutujte výsledky a další možnosti práce.

ASSIGNMENT

1. Get familiar with the programming possibilities of Stäubli - an industrial robot using the „Stäbli Robotics Suite“ (SRS) simulation environment.
2. Get familiar with the environment of the selected CAD system and model the selected workpiece to test the motion trajectory planner.
3. Design a procedure to draw the required robot trajectory with respect to the workpiece and the required selected application (e.g. spraying of the workpiece in industrial washing lines).
4. Design a procedure to export the points of interest of the motion trajectory.
5. Use the exported trajectory points to plan the robot motion in the SRS environment and simulate the robot motion.
6. Discuss the results and other work options.

Obsah

1 Úvod

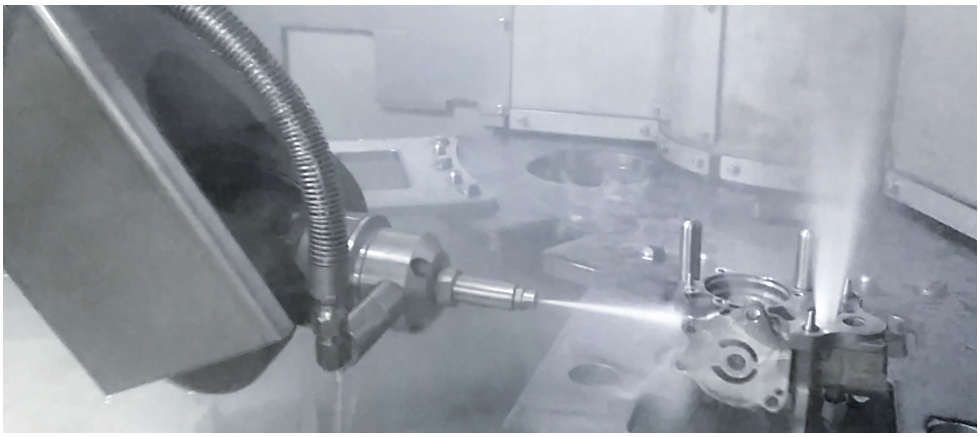
Obsahem této bakalářské práce bylo především vymyslet způsob, jak v prostředí aplikace Autodesk Inventor (AI) nakreslit požadovanou trajektorii pro ostřik obrobku v průmyslových mycích linkách a navrhnout, jak jednotlivé zájmové body trajektorie z této aplikace exportovat. Exportované zájmové body poté využít k naplánování pohybu robotu v prostředí SRS (Staubli Robotics Suite) a pohyb simulovat. Simulační prostředí SRS je aplikace pro správu robotů značky Stäubli ve které můžeme kromě základního i pokročilejšího nastavení robotů, navrhnout i různé trajektorie s ohledem na daný obrobek, který lze do simulačního prostředí nahrát a následně zobrazit ve 3D zobrazení. Bakalářská práce tedy zahrnuje jak seznámení s aplikací Autodesk Inventor, simulačním prostředím SRS a samozřejmě i s řešením převodu dat z jedné aplikace do druhé.

K čemu je vůbec navrhnutí takového datového postprocesu pro programování robotů pomocí zadané trajektorie v CAD nástroji dobré? Hlavní důvod a především i výhoda je ušetření času programátorovi při plánování trajektorie pohybu těchto průmyslových robotů. Ve většině firem v České republice, které mají průmyslové roboty nějakým způsobem zahrnuté do chodu výroby daných komponentů jsou totiž lidé pro práci s roboty proškolováni především pro návrh trajektorie pouze přímo z bezprostřední vzdálenosti u robota. Znamená to, že operátor vytváří program pro projetí trajektorie přímo pomocí controlleru ovládající robot, kdy si s robotem vždy najede do pozice ve které například chce, aby něco uchopil, pozici uloží a takto projde celou trajektorii, čímž vytvoří program. Takovýto operátor umí nastavit trajektorii a odzkoušet projetí tak, aby nedocházelo ke kolizi s obrobkem nebo k najetí do bezpečné zóny. Zároveň ale tento operátor nemá vůbec ponětí o tom, jaké hodnoty jsou tímto způsobem plánování trajektorie do robota ukládány. Tento způsob je samozřejmě v pořádku, ale zabere poměrně dost času tyto body všechny projet, uložit je a ještě odzkoušet naplánovaný pohyb. Problém poté může nastat například, když operátor nebude mít k dispozici obrobek, ale pouze souřadnice a natočení jednotlivých zájmových bodů trajektorie. Další problém může nastat, když se bude od robota očekávat nějaká větší přesnost, například pájení součástek na desce plošných spojů (PCB). V tomto případě ne jenom že by člověk strávil spoustu času ukládáním poloh všech bodů, ale asi by to nedokázal ani tak přesně, jak bychom si u takového pájení představovali.

Právě teď se dostáváme k původní otázce, proč je dobé navrhnout datový postproces pro plánování trajektorie robotů pomocí zadané trajektorie v CAD nástroji. Pojďme si to tedy vysvětlit na následujícím příkladu. Máme firmu která vyrábí díly do převodovky a zaměstnáváme jednoho zaměstnance, který nám právě tyto díly navrhuje a modeluje například v již zmíněné aplikaci Autodesk Inventor. Tento zaměstnanec vytvoří 3D model a podle tohoto modelu se spustí sériová výroba. Jelikož každý vyrobený kus je znečištěný od špon a má spoustu děr pod různými úhly, tak nestačí použít k mytí klasickou průmyslovou myčku, ale použijeme průmyslového robota, který je opatřen tryskou na koncovém efektoru a prostříkává vždy každou díru zvlášť pod přesným úhlem, aby bylo čištění co nejvíce úspěšné. Abychom mohli takového robota rozpohybovat, potřebujeme někoho, kdo nám naplánuje trajektorii pohybu. V naší firmě opět zaměstnáváme pouze operátora se základním školením, který bude plánovat trajektorii pomocí ukládání jednotlivých bodů pomocí pohybu robotem a stráví s tím poměrně dost času. Proč tedy rovnou nevyužít toho, že zaměstnáváme člověka, který nám modeluje výrobky v CAD nástroji a zná souřadnice

jednotlivých děr a dokonce i jejich úhly. Z modelu obrobku vychází veškeré informace které potřebujeme pro naplánování trajektorie čištění a již stačí pouze vymyslet způsob, aby ten stejný člověk který nám vytvořil model, rovnou zakreslil i trajektorii pro robota. Musíme již tedy pouze vymyslet způsob, jak trajektorii z Inventoru exportovat, navrhnout přepočet souřadnic zájmových bodů do souřadnicového systému robota a vypočítat jednotlivé postupné rotace Rx, Ry, Rz, které jsou potřebné pro správné natočení koncového efektoru robotu. Na závěr stačí vymyslet kód, který tyto body bude poté automaticky projíždět. Když poté naučíme našeho CAD designéra vložit výstupní data do SRS, aby uměl zkontrolovat zda nedochází ke kolizi a aby to poté uměl vše nahrát do reálného robota. Ušetříme jak celkový čas plánování celé trajektorie, tak i za placení dalšího člověka, který by plánoval trajektorii ručně a především touto změnou dosáhneme zvýšení kvality čištění, protože robot bude znát přesné body a úhly jednotlivých děr.

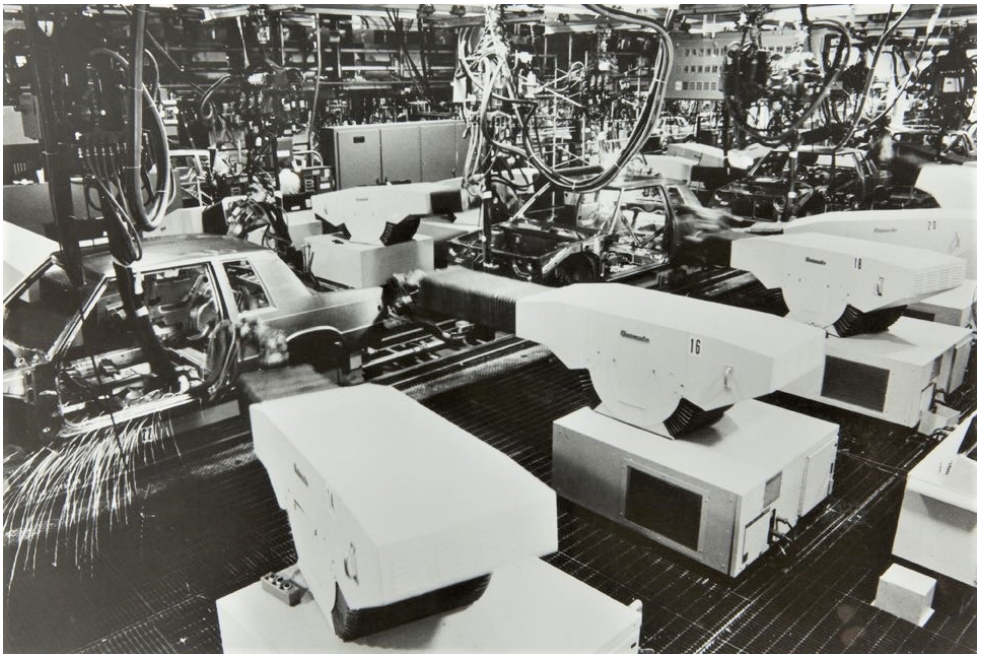
Přesně toto řešení jsme navrhli v rámci této bakalářské práce, kdy máme vymyšlený způsob zakreslení trajektorie v Inventoru, máme vytvořené makro pro export souřadnic zájmových bodů trajektorie a vytvořili jsme aplikaci Autodesk Inventor to Staubli Robotics Suite (AItoSRS), která slouží právě pro přepočet dat zvolené trajektorie v aplikaci Autodesk Inventor předepsaným způsobem do Staubli Robotics Suite a rovnou nám vytvoří i program, který je po zkopírování do SRS funkční.



Obrázek 1: Ukázka čištění průmyslovým robotem

1.1 Historie průmyslových robotů a automatizace

Robot je stroj, schopný automaticky provádět složité série činností. Slovo „robot“ se poprvé objevilo v tisku ve hře Karla Čapka R.U.R. (Rossum’s Universal Robots), napsané v roce 1920 a poprvé uvedené v roce 1921 při premiéře již zmíněné hry v pražském Národním divadle [9]. Slovo pochází z českého slova robota, což znamená „nucená práce“ a vymyslel ho bratr Karla Čapka, který se jmenoval Josef Čapek. Průmyslové roboty se hojně využívají v automatizované výrobě, kde šetří pracovní sílu, ale také v nejrůznějších aplikacích, jako je svařování a montáž, manipulace, balení a různé další aplikace. První průmyslový robot byl instalován společností General Motors (GM) v roce 1961, aby zvedal horké kusy kovu ze stroje na tlakové lití. Tento robot Unimate (na obrázku 2), jak se jmenoval, byl vyvinut Georgem Devolem a Josephem F. Engelbergerem, zakladateli robotické společnosti Unimation [7]. Několik let po instalaci prvního robota se společnost GM stala nejautomatizovanější automobilkou na světě a způsobila tak revoluci v automobilovém průmyslu.



Obrázek 2: První průmyslový robot UNIMATE

V 60. a 70. letech 20. století se objevily pokročilejší robotické paže, které již využívaly kamery nebo senzory. Navržením robota Shakey v roce 1966 jsme navíc dosáhli dalšího důležitého milníku v robotice a to především pro vývoj mobilní robotiky a díky tomu se souběžně objevily také i první mobilní průmyslové roboty [8]. Hlavní rozdíl mezi stacionárním a mobilním robotem je v pohybu a v prostorové orientaci. Stacionární roboty jsou ukotveny a jejich poloha základny se tedy nemění.

Většina robotů, která se používá v průmyslu jsou stacionárními roboty neboli manipulátory a mají za úkol například montování, sváření, lakování nebo vizuální kontrolu pomocí kamer. Mobilní roboty jsou většinou menší a obratnější než průmyslové stacionární roboty a používají se často v nebezpečných prostředích, například při zneškodňování bomb, a mohou být vybaveny senzory a kamerami, které shromažďují informace o jejich okolí. Hlavní rozdělení těchto robotů je podle ovládání a to na autonomní a dálkově ovládané [11]. Autonomní roboti se používají většinou v logistice, kdy například jezdí po výrobní hale, kontrolují dostupnost materiálu na jednotlivých výrobních linkách a v případě, že zjistí nedostupnost, přivezou nový materiál ze skladu. S roboty na dálkové ovládání se pak můžeme setkat například při kontrole těsnosti potrubí či průzkumu mořského dna.

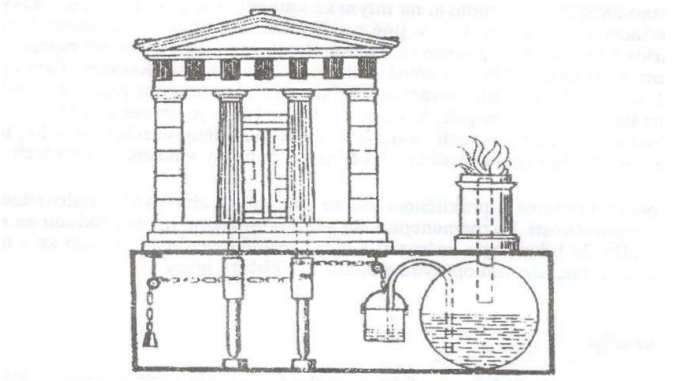
S roboty přímo souvisí i pojem automatizace. Automatizace je proces, pomocí kterého se mohou provádět opakované úkoly bez lidského zásahu. Může být provedena na základě řady faktorů, jako je čas, pořadí, počet nebo podmínka a je používána například k řízení výroby, logistiky, finančních transakcí nebo jiných procesů. Lidé se často obávají, že jim automatizace sebere práci, ale to je pouze iluze. Ve skutečnosti automatizace pomáhá lidem vykonávat práci rychleji, efektivněji a vytvoří spoustu nových pracovních příležitostí jako je údržba strojů, programování a lean management.

Mezi výhody automatizace pak patří:

- Zvýšení efektivity - práce se opakuje stejným způsobem každý den a cyklus může běžet nepřetržitě s výjimkou času pro potřebné opravy a údržby. Šetří sílu zaměstnancům a vyrábíme rychleji.
- Zvýšení kvality - vždy odvedená stejná úroveň práce.
- Snížení nákladů - je potřeba méně pracovních hodin a firma tak ušetří peníze, které by jinak platila zaměstnancům.
- Zvýšení bezpečnosti - zaměstnanci jsou vystaveni méně nebezpečí (např.práce s těžkými kusy nebo při vysoké teplotě)
- Zvýšení spolehlivosti - je méně pravděpodobné, že dojde k chybám, protože pracovní kroky jsou přesně definovány a na stroj nepůsobí žádný stres ani únava jako na běžné zaměstnance.

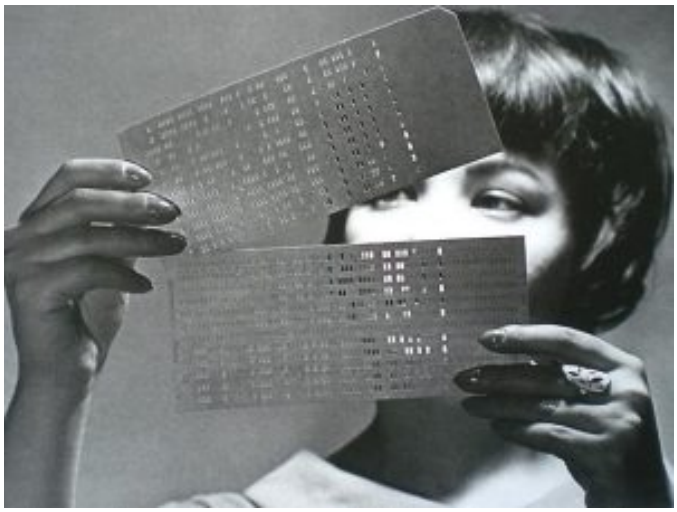
Automatizace se objevuje již od počátků civilizace. První zaznamenané použití automatizace mělo podobu mechanických zařízení, která sloužila k automatizaci jednoduchých úkonů [1]. Tato zařízení byla původně poháněna vodou nebo větrem, ale později byla poháněna parními stroji. Jedny z prvních známých náznaků automatice pocházely od vynálezce Héróna Alexandrijského a jednalo se například o vrata chrámu, které se samy otevíraly v závislosti na zapáleném obřadním ohni [1][2][4]. Využívala se zde znalost teplotní roztažnosti vzduchu a tlaku vyvíjeného na kapalinu [2].

Stejný vynálezce také ve svých spisech popisuje automat pro dávkování vždy stejného množství svěcené vody, nebo Héronóvu baňku, která dokázala s využitím páry roztočit kovový kulový kotel. Jednalo se o první náznak primitivního parního stroje, ale vynálezce již nedokázal vymyslet způsob, jak tuto baňku připojit k jiné části stroje aby fungovala jako pohon [4].



Obrázek 3: Mechanismus otevírání dveří chrámu

Za zmínku určitě stojí i vynálezce James Watt, který vynalezl a sestrojil Wattův odstředivý regulátor, který se používal ke stabilizaci otáček parního stroje. Ten byl vynalezen roku 1765 stejnojmenným vynálezcem [6]. Na počátku 19. století se poté parní zařízení začala používat v továrnách a výrobních linkách k provádění opakujících se úkonů, jako je montáž a balení, a objevily se tak první pokusy o automatizaci výrobních procesů. Byl také vytvořen první programovatelný stroj, který se nazýval žakárový tkalcovský stav a bylo ho možné naprogramovat tak, aby tkal dané vzory do látky. Žakárový tkalcovský stav byl prvním strojem, který používal systém děrných štítků, což je systém karet s otvory, které může stroj číst [5].



Obrázek 4: Děrný štítek

Teprve v 20. století se však automatizace stala běžnou součástí průmyslové výroby. Začaly vznikat například první digitální počítače a stroje s číslicovým řízením (NC - Numerical control). První stroj s NC sestrojil koncem 40. let minulého století John T. Parsons se svým týmem na MIT [10]. Tento stroj, nazvaný Servo Mechanism, byl navržen tak, aby automaticky řídil dráhu nástroje pomocí souboru čísel zaznamenaných na papírové pásce. Stroj byl schopen přečíst pásku a řídit se pokyny, aby dosáhl konzistentních výsledků. Stroje s NC řízením byly vybaveny motory, které byly řízeny právě instrukcemi na děrované pásce, kdy se pomocí čtečky přečetla právě jedna řádka (věta) pásky, uložila se do paměti a provedla. Po provedení této věty, se načetla další řádka a paměť se přepsala → mohla se tedy vždy vykonat pouze jedna instrukce. V 50. letech 20. století se NC stroje začaly používat v průmyslu, nejprve v leteckém průmyslu, kde se používaly k frézování leteckých dílů. NC stroje umožnily vyrábět díly s větší přesností a konzistencí, než bylo možné při ručním obrábění. NC stroje se rychle staly základními nástroji v mnoha průmyslových odvětvích, včetně automobilového, elektronického a lodního průmyslu.

Mezi jedny z prvních digitálních počítačů pak patří například digitální počítač ENIAC (1946), jehož vývojem dosáhli tvůrci také poprvé elektronického zpracování dat a o pár let později v roce 1951 byl vytvořen další digitální počítač UNIVAC I. Tyto počítače zajistily, aby se řídicí funkce v automatizaci stala mnohem sofistikovanější a související výpočty byly prováděny mnohem rychleji, než bylo dříve možné [5].

NC stroje jsou dnes již řízeny počítačovými programy, které jim umožňují vyrábět díly s extrémně vysokou přesností a takovéto stroje nazýváme jako stroje s číslicovým řízením počítačem (CNC - Computer Numerical Control). V dnešní době jsou CNC stroje používány především při broušení, řezání, frézování, vrtání a obrábění jednotlivých výrobků. Používají se zejména pro jejich rychlost, přesnost a efektivitu, díky čemuž tyto stroje najdete skoro v každé větší výrobě. CNC stroje pracují podle zadaného kódu a výsledek je tedy vždy stejných rozměrů a kvality. Tyto stroje se navíc programují jednoduše vytvořením ručně seskládaných instrukcí (kódu) a nebo pomocí CAD/CAM programů. Programy CAD (Computer Aided Design = počítačem podporované navrhování) slouží pro navrhnutí modelu ve 2D či 3D zobrazení a v návaznosti na CNC nám tento program pomůže s navrhnutím výsledného tvaru obrobku, který budeme chtít pomocí CNC vyrobit. Příklady CAD programů jsou například SolidWorks, AutoCAD a Inventor. CAM programy už jsou pak specifické programy pro vytvoření kódu pro CNC stroje. Do tohoto programu se vloží námi vytvořený model obrobku v CAD nástroji a poté jednotlivými nastaveními např. volba nástroje, volba polotovaru, uchycení polotovaru a vytvoření jednotlivých operací (vrtání, frézování atd.) vytvoříme základní projekt, jehož exportování z tohoto programu získáme funkční kód pro CNC stroj. Příklady CAM nástrojů jsou například Inventor CAM a SolidCAM. Některé programy jako je například Fusion 360 jsou pak CAD/CAM programy, ve kterém zvládnete jak obrobek navrhnout, tak i vytvořit instrukce pro CNC. Výhodou těchto programů je jednoznačně propojení CAD a CAM nástrojů a můžeme tedy vytvořit všechno potřebné pomocí jednoho programu.

V naší bakalářské práci budeme pracovat pouze s CAD nástrojem, přesněji s Autodesk Inventor o kterém si řekneme více v následujícím bodě.

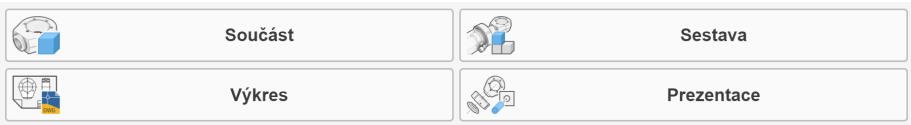
2 Autodesk Inventor

Autodesk Inventor je 3D CAD software pro strojírenský návrh, simulaci, vizualizaci a dokumentaci. Používají ho konstruktéři a designéři k vytváření, úpravám a optimalizaci výrobků. Inventor slouží k vytváření 3D modelů součástí, sestav a 2D výkresů. Obsahuje prostředí pro parametrické modelování, které umožňuje konstruktérům upravovat své návrhy změnou rozměrů. Díky tomuto jsou úpravy rozměrů v tomto programu jednodušší než například v programu AutoCAD, protože stačí například změnit hodnotu kóty výšky kvádru a kvádr se tak automaticky změní. V již zmíněném programu AutoCAD bychom museli vytvořený kvádr smazat a vytvořit nový. Inventor obsahuje také výkonný simulační nástroj, který lze použít k testování výkonnosti výrobků v reálných podmínkách [3].

2.1 První spuštění

Po spuštění aplikace, v našem případě přesně verzi Inventor Professional 2022 se nám zobrazí okno, kde najdeme kromě náhledu posledních vytvořených souborů i nabídku pro vytvoření nového souboru. V této nabídce můžeme najít:

- Součást - vytvoření samostatné součásti nebo jedné komponenty z celku pomocí existujících tvarů nebo za pomoci náčrtu a jeho následného vysunutí, rotace, tažení atd.
- Výkres - slouží pro okótování jednotlivých pohledů vytvořené součásti, přidání poznámek, symbolů a samozřejmě to vše na výkresu s razítkem.
- Sestava - zde můžeme vytvořit celek z jednotlivých součástí a tyto soustavy i spojit pomocí různých šroubových spojů, hřídelí, ložiskovými spoji, ozubenými koly a lze zde vytvořit i rámová konstrukce.
- Prezentace - prostředí ve kterém si můžeme vytvořit scénář z různých pohledů naší sestavy a taky můžeme simulovat pohyb jednotlivých součástí a to vše zaznamenat na video.

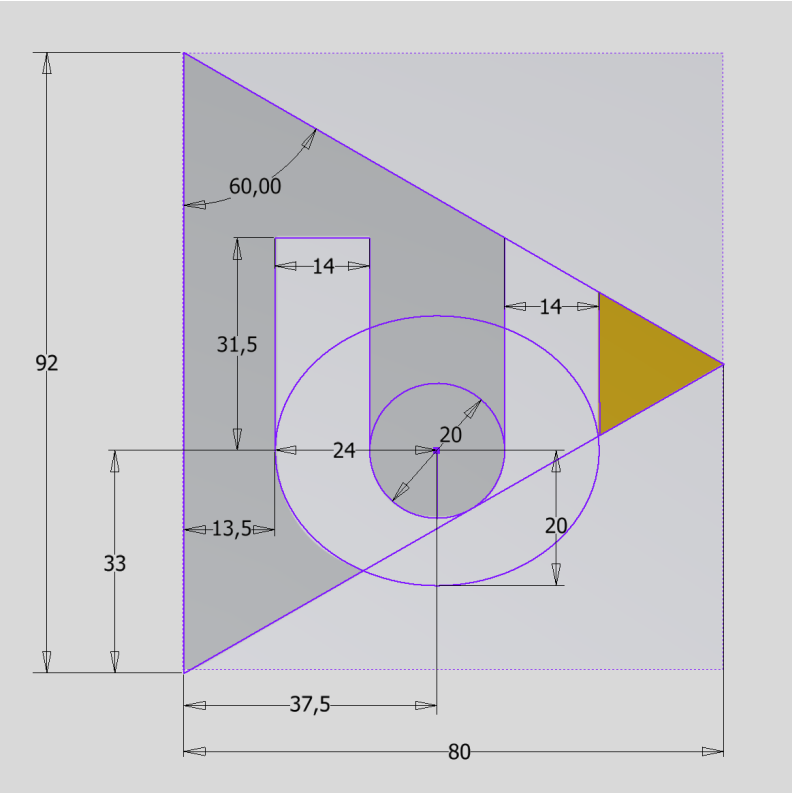


Obrázek 5: Nabídka nového vytvoření

Navíc si můžeme vytvořit například pevnostní analýzu, analýzu křivosti, pevnosti a spoustu dalších analýz. Můžeme nastavovat různé vzhledy součástí, vybírat z nabídky materiálů a v neposlední řadě také můžeme vytvářet různá makra.

2.2 Návrh loga

Abychom mohli naši testovací krychli označit logem školy, museli jsme si jako první vytvořit toto logo jako náčrt. Vytvořili jsme si tedy nový náčrt, do kterého jsme přes záložku *Vložení* → *Obrázek* vložili stažené a oříznuté logo naší školy. Pomocí záložky *Vytvořit* jsme pak pomocí čar, kružnic a elips obkreslili vložený obrázek. Výsledný okótovaný návrh loga lze vidět níže na obrázku 6.



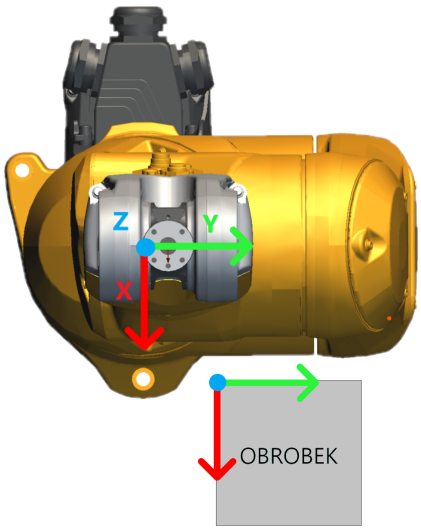
Obrázek 6: Logo školy

2.3 Vytvoření potřebných součástí

Pro naši práci potřebujeme vytvořit 3 základní součásti. První z nich bude zkušební krychle s deseti dírami. Na této krychli budeme zkoušet funkčnost našeho datového postprocesu při simulaci pohybu robota v simulačním prostředím SRS. Zbylé dvě součásti budou potřebné pro zakreslení trajektorie a pro následný export zájmových bodů z Inventoru k jejichž významu se dostaneme později.

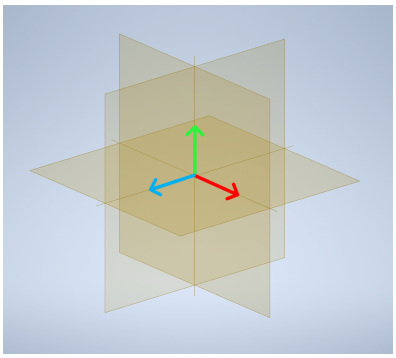
2.3.1 Zkušební krychle

Vytvoříme si nový pracovní soubor *Součást* a hned prvním nejdůležitějším krokem je volba správného natočení UCS (Unified Computing System). Aby totiž fungovaly všechny navazující kroky exportu dat, potřebujeme mít stejné natočení UCS jak v Inventoru tak i v simulačním prostředí SRS. Natočení UCS robota v SRS je zobrazeno na obrázku 7.

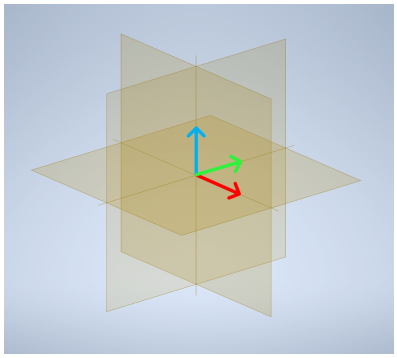


Obrázek 7: Natočení UCS robota

Na kartě *3D model* vybereme ze záložky *Náčrt* → *Zahájit 2D náčrt*. Otevře se nám okno náčrtu, kde si musíme zvolit rovinu pro vytvoření náčrtu. Potřebujeme ale mít takové natočení UCS jako v SRS a toho docílíme pomocí správné volby roviny k vytvoření náčrtu a jejího správného natočení. Ke správnému natočení použijeme ViewCube a až budeme mít správné natočení UCS (viz obrázek 9), tak zvolíme rovinu XY. Aby se nám s aktuálním natočením pracovalo lépe, klikneme na ViewCube a vybereme *Nastavit aktuální pohled jako výchozí*, díky čemuž nám budou odpovídat jednotlivé pohledy ViewCube našemu natočení UCS.

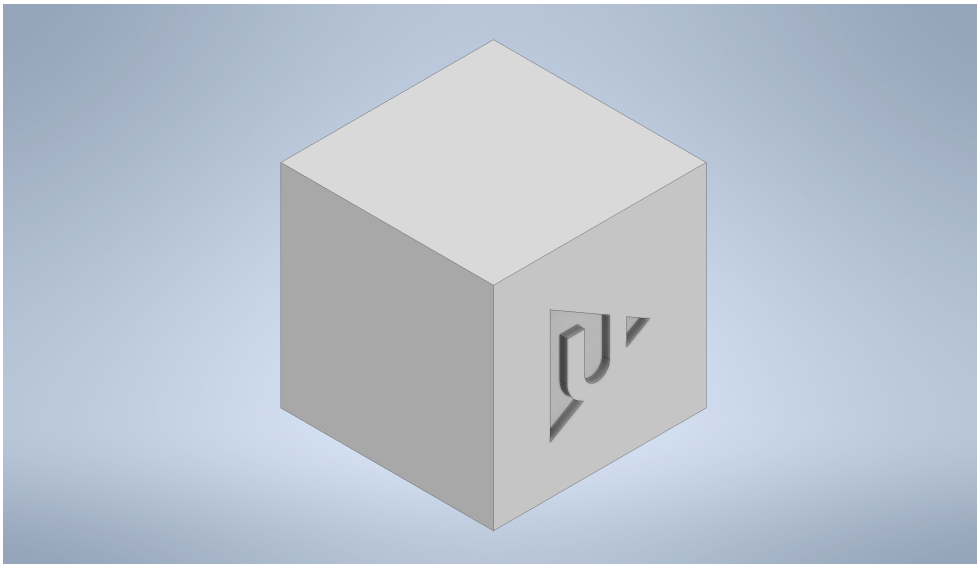


Obrázek 8: Původní natočení UCS

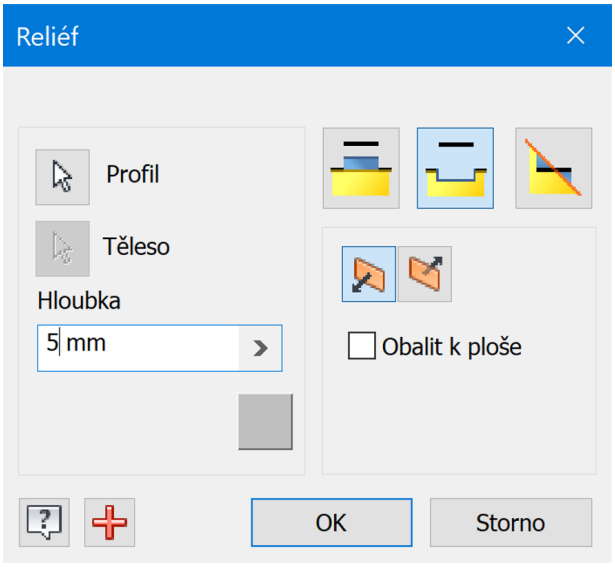


Obrázek 9: Finální natočení UCS

V otevřeném náčrtu roviny XY si vytvoříme čtverec o rozměrech 150x150mm a jeho levý horní roh bude umístěn v počátku našeho UCS. Po dokončení klikneme na *Dokončit náčrt*. Máme již vytvořenou podstavu naší krychle a nyní již stačí pouze použít příkaz *Vysunutí* v záložce *Vytvoření* a vysunutím vytvořit krychli o rozměrech 150x150x150mm. Zkopírováním a umístěním náčrtu loga na přední stranu naší krychle můžeme následně vytvořit gravírování. Gravírování loga do krychle vytvoříme pomocí příkazu *Reliéf* v záložce *Vytvoření*, kdy po kliknutí na příkaz *Reliéf* vybereme náčrt našeho loga, zvolíme *Gravírování z plochy*, požadovaný směr a nastavíme hloubku, která je v našem případě 5mm. Potvrzením nastavení se nám vytvoří gravírování loga viz obrázek 10. Náhled nastavení gravírování je pak zobrazeno na obrázku 11.

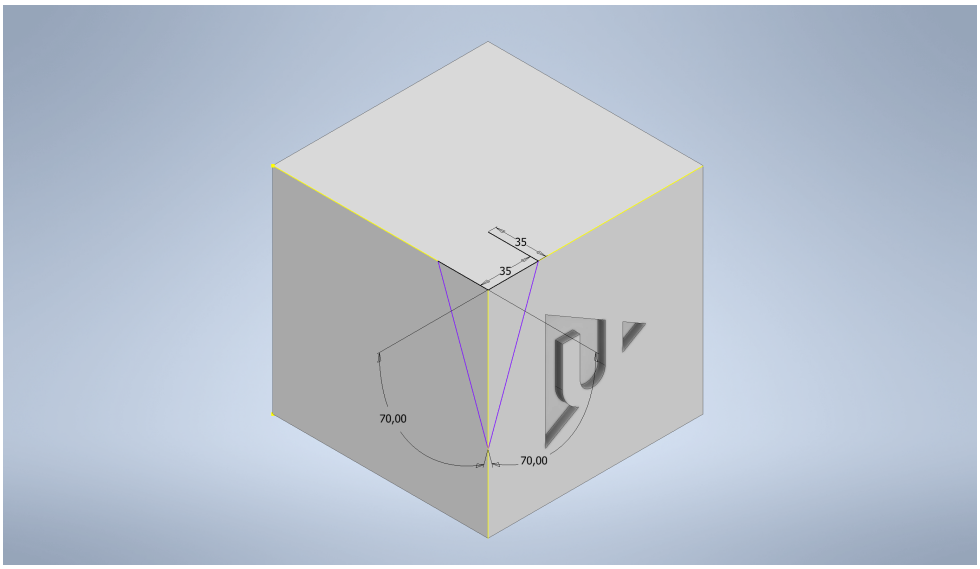


Obrázek 10: Gravírované logo školy



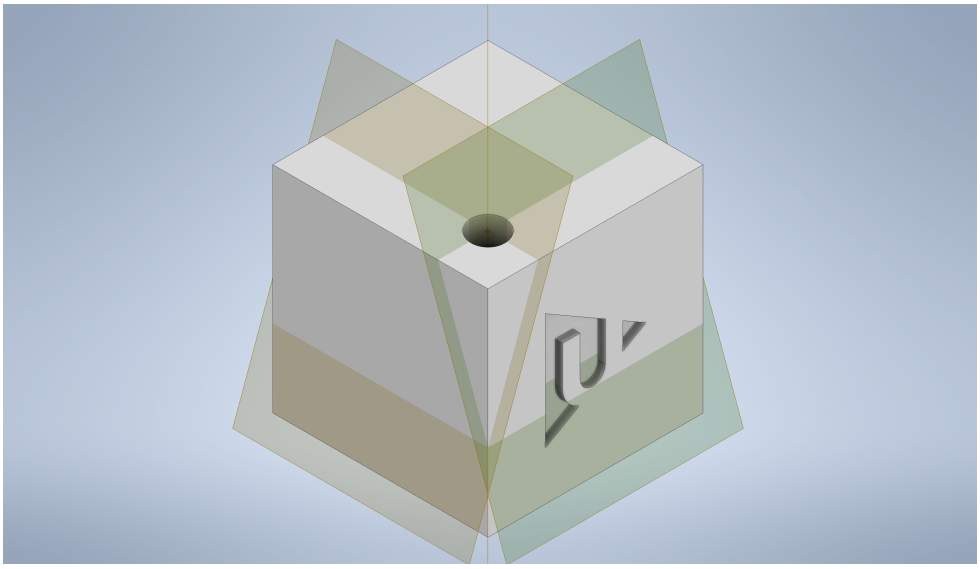
Obrázek 11: Nastavení gravírování

Nyní máme připravený základ pro tvorbu zkušebních děr v naší krychli. S ohledem na velikost zvoleného robota v SRS, jeho pracovního prostoru a možnostech natočení koncového efektoru jsme vytvořili deset děr na 3 stranách krychle. Všechny tyto díry mají jiný úhel, abychom mohli následně otestovat správnou funkčnost datového postprocesu. Princip vytváření děr pod různými úhly si vysvětlíme na jedné z vytvořených děr. Zvolíme stranu krychle na které budeme chtít díru vytvořit. V našem případě tedy klikneme levým tlačítkem myši na horní stranu krychle a vybereme *Vytvořit náčrt*. Otevře se nám okno náčrtu a vytvoříme si pomocí čar střed pro díru a náčrt zavřeme. Následně si vytvoříme další dva náčrty s požadovanými úhly díry a to vždy pro dva různé pohledy díry, tedy úhel díry z předního a z levého pohledu. Již vytvořené a okótované tři potřebné náčrty jsou zobrazeny na obrázku 12.

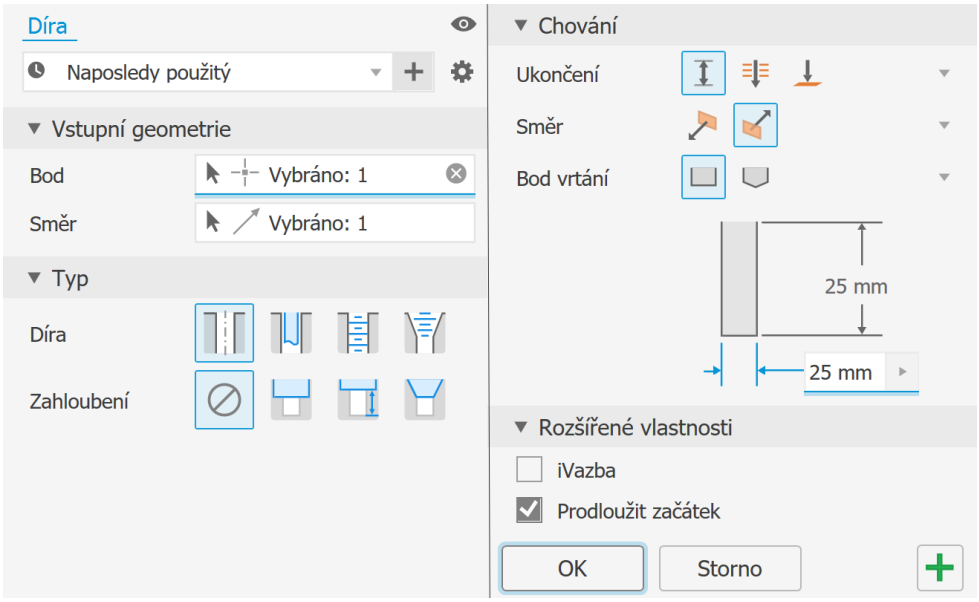


Obrázek 12: Okótované náčrty díry

Z dvou vytvořených náčrtů úhlu díry z různých pohledů musíme následně vytvořit roviny. Rovinu vytvoříme pomocí záložky *Pracovní konstrukční prvky* → *Rovina* a po vybrání jednoho z vytvořeného úhlu a potvrzením 90° natočení roviny vůči vybrané čáře vytvoříme první rovinu. Postup opakujeme a vytvoříme i rovinu podle druhého úhlu. Vytvořené roviny jsou zobrazeny na obrázku 13. Dalším přípravným krokem pro vytvoření díry je vložení pracovního bodu na vytvořený střed díry. Tento bod najdeme stejně jako rovinu pod záložkou *Pracovní konstrukční prvky* → *Bod*. Posledním přípravným krokem je vytvoření osy díry. Tu vytvoříme jako průsečík našich dvou vytvořených rovin *Pracovní konstrukční prvky* → *Osa* → *Průsečík dvou rovin*. Již máme vytvořené všechny potřebné věci pro vytvoření díry a můžeme ji pomocí záložky *Upravit* → *Díra* vytvořit. Po kliknutí na příkaz *Díra* se nám otevře okno s nastavením. Prvním kliknutím vybereme náš středový bod díry a hned poté vytvořenou osu díry. Dále zvolíme typ díry a zahloubení jako *Jednoduchá díra* bez zahloubení a typ ukončení jako *Vzdálenost*, směr díry, způsob vrtání *Ploché* a nastavíme hodnotu hloubky na 25mm a průměr díry taktéž na 25mm. V posledním kroku ještě zaškrtneme *Prodloužit začátek* a potvrzením nastavení vytvoříme díru (obrázek 13). Samotné nastavení pro vytvoření díry je zobrazeno na obrázku 14.

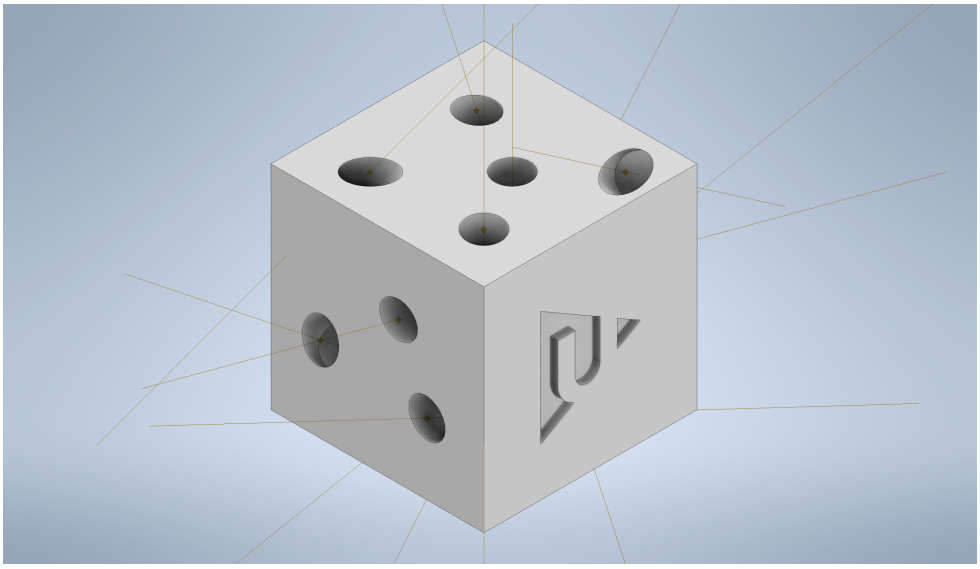


Obrázek 13: Vytvořená díra s viditelnými rovinami a osou

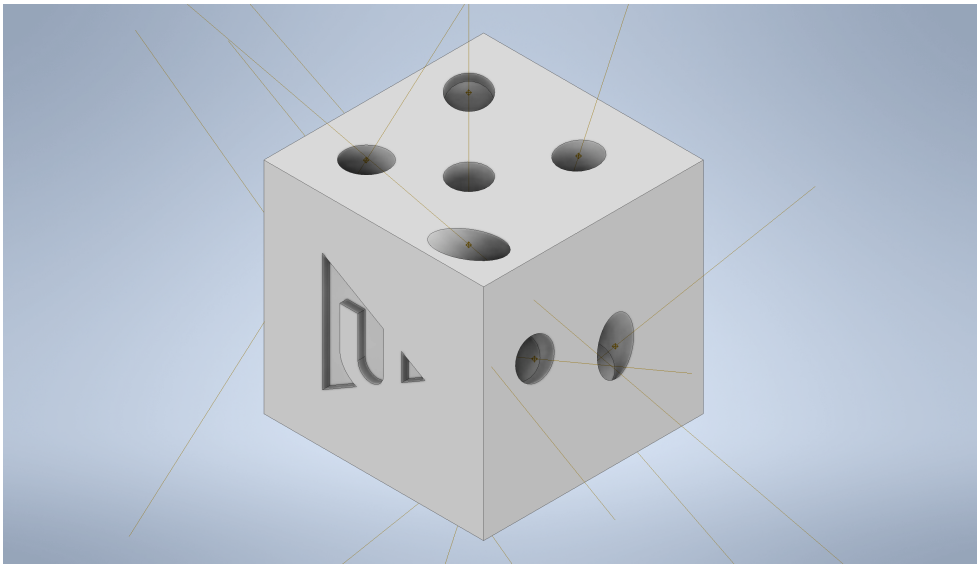


Obrázek 14: Nastavení pro vytvoření díry

Stejným postupem jsme vytvořili všech deset děr, které se rozkládají na horní, levé a pravé straně naší krychle. Všechny tyto díry mají jiné úhly natočení a jejich umístění je zobrazeno na obrázcích 15 a 16 níže.



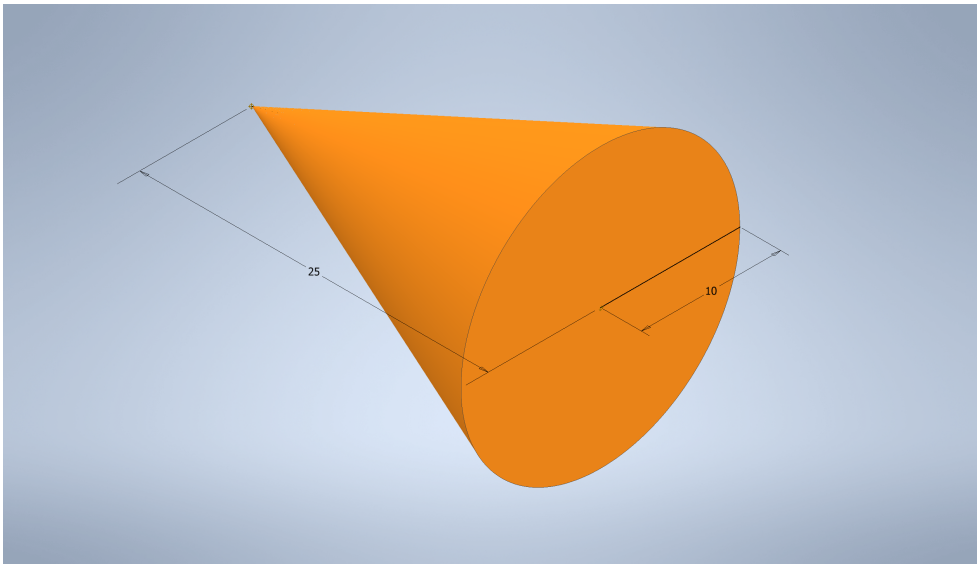
Obrázek 15: Izometrický pohled přední levý



Obrázek 16: Izometrický pohled přední pravý

2.3.2 Kužel pro export

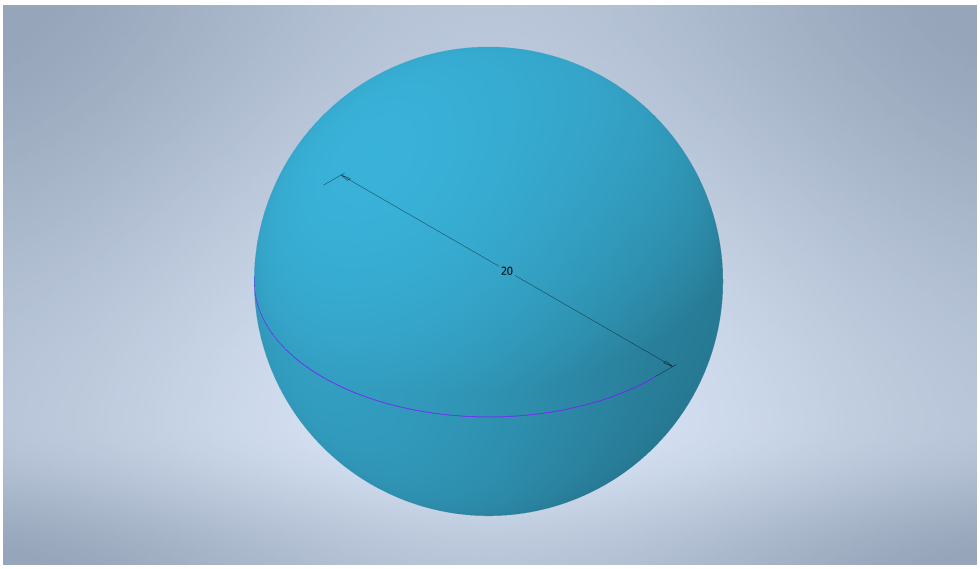
Vytvoříme si nový pracovní soubor *Součást* s novým náčrtem do kterého si vytvoříme rovinný útvar rotačního tělesa. V případě našeho kuželu se jedná o pravoúhlý trojúhelník s rozměry stran 25x10mm. Po dokončení náčrtu rovinný útvar orotujeme čímž získáme kužel. Rotaci vytvoříme pomocí *Vytvoření* → *Rotace*. Po vybrání *Rotace* klikneme na plochu našeho trojúhelníku a následně na stranu trojúhelníku, která bude sloužit jako osa pro rotaci. Následně nastavení (obrázek 19) potvrdíme a máme vytvořený kužel viz obrázek 17. Pro později vysvětlený export souřadnic z prostředí Inventoru potřebujeme ještě přidat dva pracovní body. Jeden pracovní bod *Pracovní konstrukční prvky* → *Bod* umístíme na vrchol kuželu a druhý pracovní bod umístíme na střed podstavy. Tyto pracovní body si pojmenujeme jako *ConeTop* a *ConeBottom*. Pro přehlednost mezi ostatními součástmi si změníme barvu kuželu na oranžovou a to kliknutím pravým tlačítkem myši na vytvořený kužel a zvolením *Vlastnosti* ve kterých vybereme oranžovou barvu.



Obrázek 17: Kužel pro export

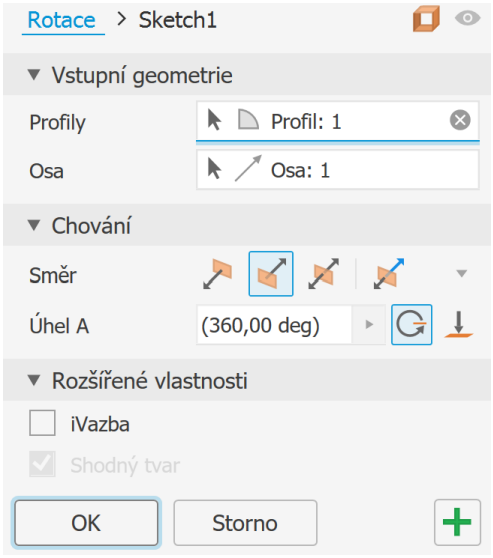
2.3.3 Koule pro export

Koule lze vytvořit dvěma způsoby. První způsob je vytvoření přes záložku *Vytvořit volný tvar* → *Koule*. Pro takto vytvořenou kouli ale nejde přidat pracovní bod na střed koule, protože se střed koule nezobrazuje a nemáme se tedy k čemu přichytit. Použijeme proto druhý způsob a tím je vytvoření opět pomocí rotace rovinného útvaru, což je pro naši kouli půlkružnice s průměrem 20mm. Po vytvoření koule umístíme pracovní bod na střed koule, který pojmenujeme jako *Center* a přes *Vlastnosti* změníme barvu na tyrkysovou. Vytvořená koule je zobrazena na obrázku 18.



Obrázek 18: Koule pro export

Proč právě tyto tělesa? Kužel nám dává směr natočení díry pomocí dvou bodů *ConeTop* a *ConeBottom* z kterých dokážeme určit směrový vektor. Právě kužel je ideální svým tvarem, který připomíná šipku pro určení směru. Koule pak slouží pro ukotvení v jednom bodě a používáme ji, protože je více přehlednější pracovat s koulí než se samotným jedním bodem.



Obrázek 19: Nastavení rotace

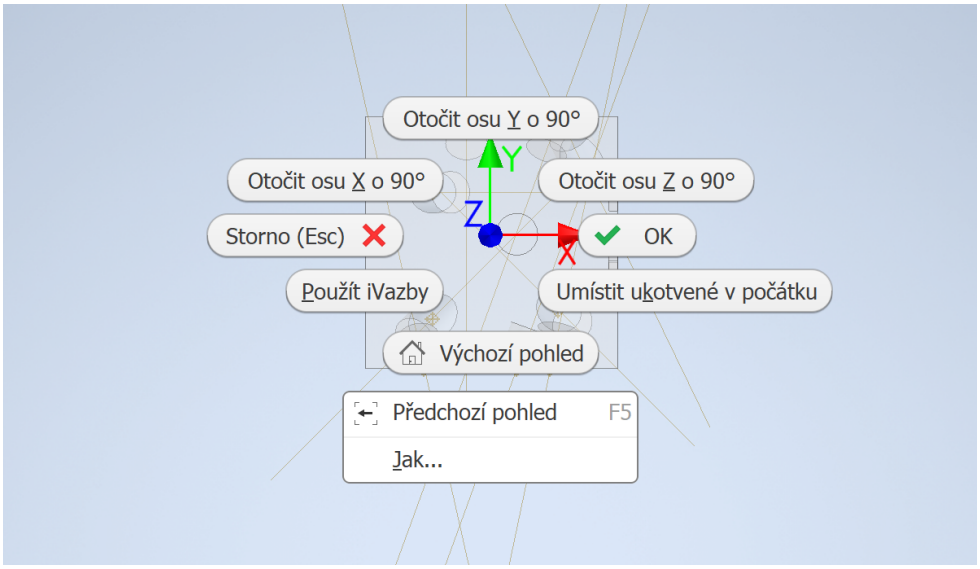
3 Zakreslení trajektorie a její export

Jednou z nejdůležitější částí naší práce bylo navrhnutí postupu pro nakreslení požadované trajektorie a taktéž navrhnutí způsobu následného exportu zájmových bodů této trajektorie. S ohledem na již známou aplikaci, ostřik obrobku v průmyslových mycích linkách si dokážeme představit jak takový pohyb robota může vypadat. Po spuštění aplikace robot najede koncovým efektorem ve směru osy díry blíže k díře, provede čištění a ve stejném natočení se poté od díry zase vzdálí. Přejede nad další díru a takto svůj postup zopakuje pro všechny díry. Pro návrh trajektorie nás tedy zajímají především souřadnice středů jednotlivých děr na ploše stran krychle a samozřejmě úhel natočení těchto děr. Když toto budeme znát, dokážeme již vytvořit požadovaný pohyb přímo v simulačním prostředí SRS.

3.1 Zakreslení trajektorie

Pro zakreslení trajektorie budeme používat již vytvořené součásti, tedy kužel a koule, jejichž soubory máme pojmenované jako **cone** pro kužel a **transit** pro kouli a od tohoto momentu je budeme nazývat čistě jejich pracovními názvy cone a transit. Cone budeme používat pro export zájmových bodů na ose děr a transit poté v případě, že budeme potřebovat do naší trajektorie zahrnout i nějaké průjezdné body.

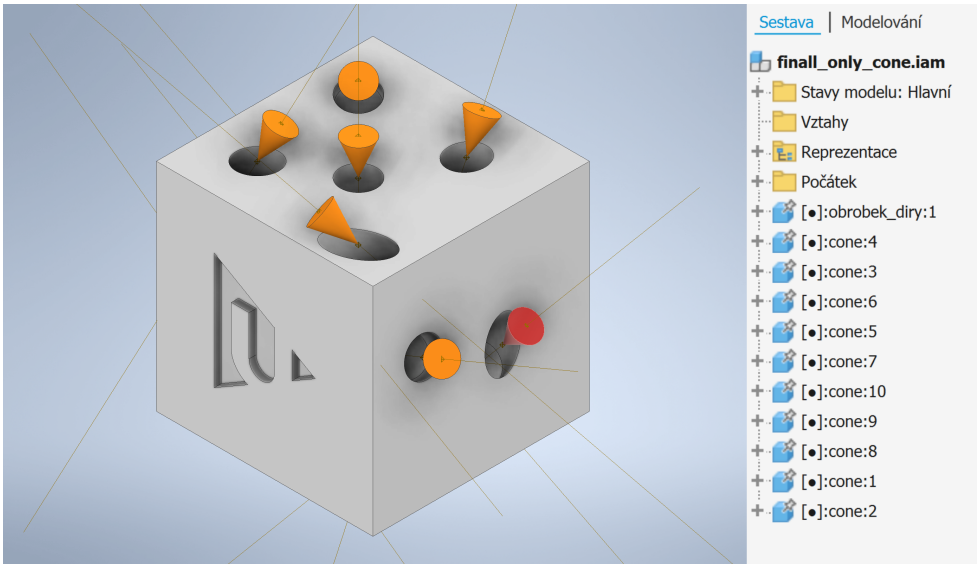
Abychom mohli začít se zakreslováním trajektorie, musíme si nejprve vytvořit nový pracovní soubor *Sestava* a načíst naší zkušební krychli s dírami. Na kartě *Sestavaní*, v záložce *Komponenta* → *Umístit* vybereme naší krychli a výběr potvrdíme. Krychli musíme umístit do počátku a to provedeme tak, že klikneme pravým tlačítkem myši a vybereme z nabídky (na obrázku 20) *Umístit ukotvené v počátku*.



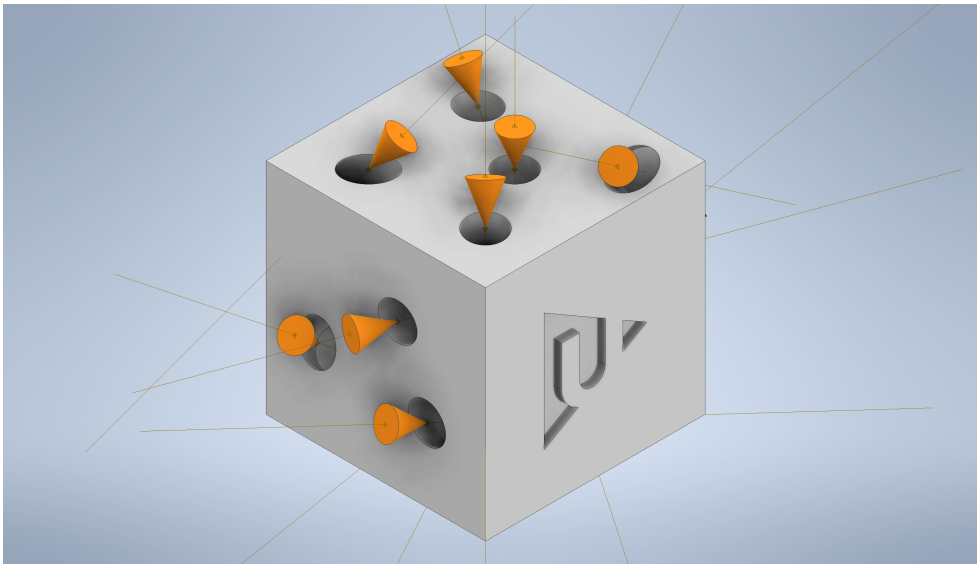
Obrázek 20: Zobrazení nabídky umístění

Je potřeba vždy načíst jako první zkušební krychli se správně natočeným UCS, protože UCS celé soustavy se nastaví právě podle první načtené součásti. Kdybychom jako první načetli například součást cone a až poté zkušební krychli. Měli bychom krychli umístěnou do počátku UCS převzatého z nastavení součásti cone.

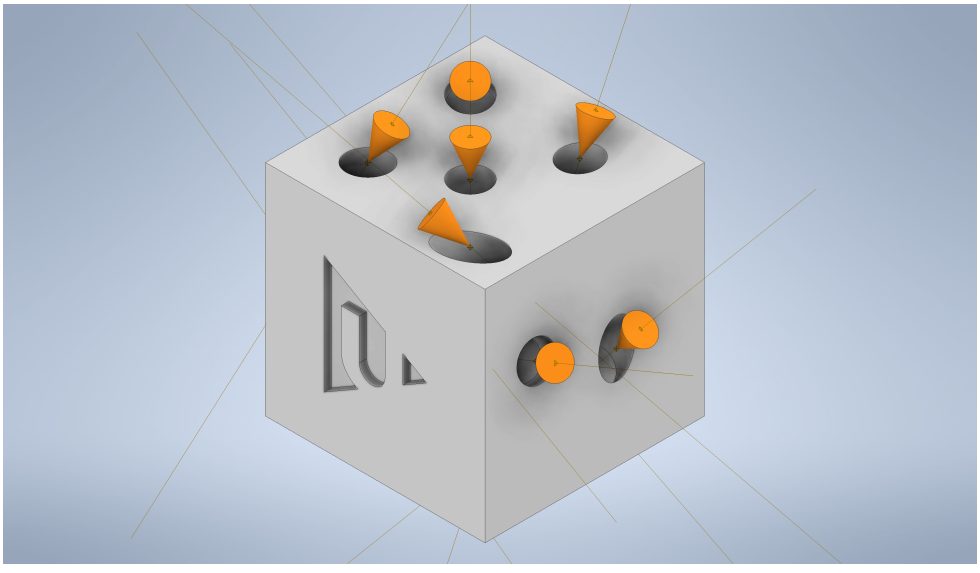
Zkušební krychli máme již načtenou a můžeme začít se zakreslováním trajektorie. Základní trajektorii zakreslíme pomocí součásti cone a to tak, že ke každé díře, kterou budeme chtít projet, musíme tuto součást ukotvit. Jako první si tedy musíme načíst součást cone a po načtení ji umístíme kdekoliv do prostoru. Následně na ni klikneme pravým tlačítkem myši a v otevřené nabídce přes *Komponenta* vybereme *Uchopení uzlů*. Klikneme na pracovní bod na vrcholu (ConeTop), zvolíme první obrázek *Volně přetáhnout* a přichytíme tento bod k bodu ve středu naší díry. Potvrdíme nulové odsazení a pokračujeme výběrem druhého bodu na podstavě (ConeBottom) a stejným způsobem jako první bod ho tentokrát přichytíme k ose dané díry. Na závěr opět klikneme pravým tlačítkem myši na právě umístěný cone a volbou *Provést a ukotvit* ho pevně ukotvíme k díře. Stejný postup opakujeme pro všechny díry a náhled sestavy najdeme na obrázku 22 a 23. Finální sestava pak bude tedy kromě finální krychle obsahovat i deset součástí cone, které jsou automaticky i očíslovány od 1 do 10 a jejich seznam lze vidět na obrázku 21. Přesně tyto čísla nám budou značit v jakém pořadí nám robot díry projede. Pokud chceme například aby robot projel jako první díru vyznačenou na obrázku 21 červeným cone, musíme tento cone přejmenovat na cone:1. První varianta je tedy přiřadit jednotlivé cone náhodně a poté je přejmenovat v závislosti na pořadí projetí. A nebo použít druhou variantu, že budeme cone nahrávat postupně a rovnou je přiřazovat podle toho, jak budeme chtít jednotlivé díry projet.



Obrázek 21: Seznam použitých součástí - základní pohyb

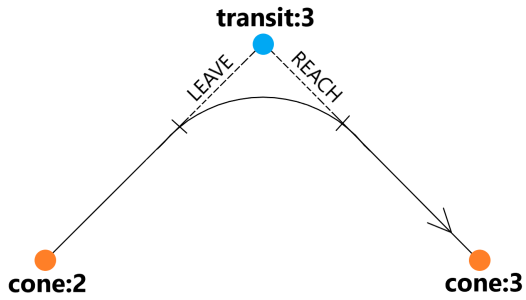


Obrázek 22: Izometrický pohled přední levý



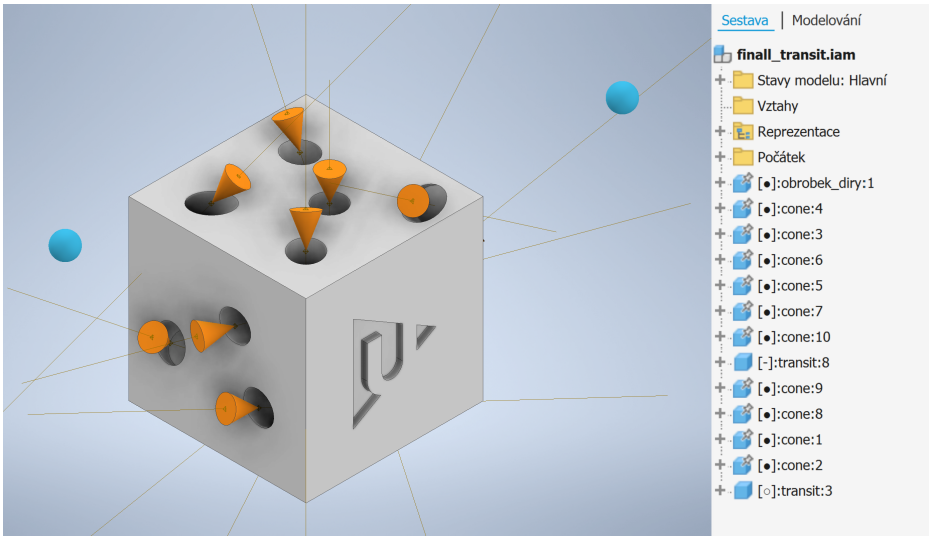
Obrázek 23: Izometrický pohled přední pravý

Jak zakreslit základní projetí vybraných děr už víme, ale co když následně v simulaci zjistíme, že nám nějaký automaticky vytvořený pohyb, například mezi dvěma dírami nevyhovuje? Například z důvodu kolize jakékoliv části robota s naším obrobkem nebo s objekty umístěnými v okolí? V tomto případě lze použít námi dříve vytvořenou součást transit, která bude sloužit jako bod blízkého průjezdu. Tímto bodem robot neprojede přímo, ani se zde nebude zastavovat, ale projede ho v jeho těsné blízkosti dle nastavených hodnot *leave* a *reach*. V rámci naší simulace byla tato vzdálenost nastavena na 30mm pro obě hodnoty a projetí robota kolem takového bodu je zobrazeno na obrázku 24.



Obrázek 24: Pohyb kolem průjezdného bodu

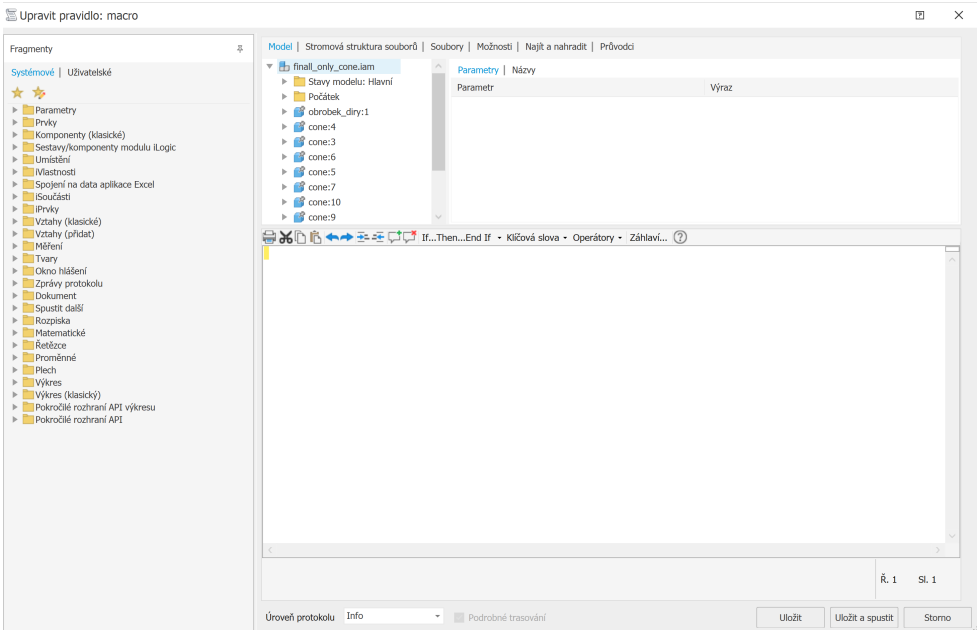
Tuto součást, neboli průjezdný bod můžeme pro každou díru použít maximálně jednou s výjimkou první díry, před kterou tento průjezdný bod nemůžeme umístit, protože by nešel pro pohyb robota definovat. Mezi dvěma dírami můžeme mít tedy vždy umístěnu jednu součást transit a celkově těchto součástí můžeme mít pro deset děr devět. U této součásti je taktéž důležité číslování, značíme ji vždy stejným číslem jako díru (cone), nad kterou má robot dojet ihned za průjezdným bodem. Číslování součástí cone a transit ukázáno na obrázku 24.



Obrázek 25: Seznam použitých součástí - pohyb s průjezdnými body

3.2 Export zájmových bodů

Export zájmových bodů, tak abychom s nimi mohli dále pracovat, bude řízen makrem, které jsme si vytvořili. Toto makro nám vytvoří .csv soubor, ve kterém najdeme souřadnice všech zájmových bodů (ConeTop, ConeBottom, Center). Makro se v Inventoru vytvoří přes kartu *Správa*, záložku *iLogic* → *Přidat pravidlo* a vytváří se v makro verzi programovacího jazyka Visual Basic (VBA). Na obrázku 26 můžeme vidět náhled okna ve kterém se makro programuje. V levé části okna jsou záložky základních předdefinovaných příkazů, které se dvojklikem zobrazí rovnou v programovacím okně a díky tomu je programování usnadněné a velmi přehledné.



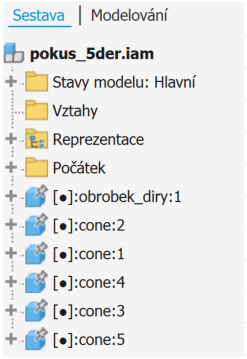
Obrázek 26: Náhled okna pro vytváření makra

Již vytvořené makro nahrajeme do Inventoru opět přes kartu *Správa*, záložku *iLogic* → *Prohlížeč iLogic*. Otevře se nám nová karta *iLogic*, zvolením *Externí pravidla* a pravým kliknutím myši na *Standardní adresáře* → *Přidat externí pravidlo* a vybereme námi vytvořené makro. Nyní když budeme mít pomocí součástí cone a transit navrhnutou trajektorii, můžeme toto makro spustit (pravým kliknutím myši na naše makro → *Spustit pravidlo*). Makro nám následně vytvoří již zmíněný .csv soubor, který bude mít stejný název jako je název sestavy ve které jsme makro spustili. Abychom mohli makro spustit, je potřeba mít vždy již vytvořenou sestavu uloženou a pojmenovanou.

3.2.1 Algoritmus makra

Naše vytvořené makro po jeho spuštění projede všechny komponenty (součásti) aktuální sestavy a bude pracovat vždy pouze s komponenty, které začínají názvem **cone:** nebo **transit:** a v případě neshody komponenty přeskočí. Pokud tedy budeme mít špatně pojmenované součásti pro vyznačení trajektorie, makro nebude fungovat. Makro načítá komponenty postupně od shora, tak jak je máme zobrazeny například v seznamu na obrázku 27. Když makro narazí na požadovanou součást, projede všechny pracovní body této součásti a budou ho opět zajímat pouze pracovní body, které jsme si nastavili, tedy **ConeTop**, **ConeBottom** a **Center**. Je tedy jasné, že pro každou součást cone, nás budou zajímat právě dva body, zatímco pro součást transit pouze jeden. Makro projede pracovní body a když narazí na jeden z výše zmíněných pracovních bodů, musí jako první přepočítat souřadnicový systém dané komponenty na souřadnicový systém celé soustavy. Kdybychom toto neudělali, získali bychom pro všechny komponenty pokaždé stejné souřadnice a to ty, ve kterých jsou cone a transit umístěny ve svém projektu součásti, nikoliv souřadnice umístění kolem naší krychle.

Makro nám našlo potřebnou komponentu, pracovní bod na této komponentě, převedlo souřadnicový systém a již stačí pouze načíst jednotlivé souřadnice x, y, z pracovního bodu a pomocí zřetězení získaných informací pomocí středníku, vytvořit jeden řádek našeho .csv souboru. Získané souřadnice x, y, z musíme ještě vynásobit deseti, protože z nějakého důvodu, i když nastavené jednotky Inventoru máme milimetry, příkazy v makru získávají souřadnice v centimetrech. Do .csv souboru ukládáme ke každému pracovnímu bodu název součásti, název pracovního bodu a hodnoty x, y, z v milimetrech. Řádek může vypadat například takto → `cone:5;ConeTop;100;100;100` a první řádek souboru slouží vždy jako legenda jednotlivých sloupců a to ve tvaru → `Component;Point;X;Y;Z`.



Component	Point	X	Y	Z
cone:2	ConeTop	90	80	150
cone:2	ConeBottom	90	80	175
cone:1	ConeTop	130	120	150
cone:1	ConeBottom	138,6981756	102,0454193	165,065682
cone:4	ConeTop	40	30	150
cone:4	ConeBottom	51,18033989	41,18033989	169,3649167
cone:3	ConeTop	40	105	150
cone:3	ConeBottom	29,43454346	105	172,6576947
cone:5	ConeTop	115	35	150
cone:5	ConeBottom	106,9096106	43,09038943	172,2281623

Obrázek 27: Ukázka vizualizace exportovaných dat pomocí makra

3.2.2 Ukázka kódu makra

```
1       Dim coneSelector As String = "cone:"
2       Dim transitSelector As String = "transit:"
3
4       Dim oAppend As System.IO.StreamWriter
5       oFile = ThisDoc.PathAndFileName(False) & ".csv"
6       IO.File.WriteAllText(oFile, "")
7       oAppend = IO.File.AppendText(oFile)
8       Dim legend = New String() {"Component", "Point", "X", "Y", "Z"}
9       oAppend.WriteLine(Join(legend, ";"))
10
11      Dim oAsmCompDef As AssemblyComponentDefinition
12      oAsmCompDef = ThisApplication.ActiveDocument.ComponentDefinition
13
14      Dim oName As String
15      Dim wpName As String
16      Dim X,Y,Z As Double
17      Dim oOcc As ComponentOccurrence
18      For Each oOcc In oAsmCompDef.Occurrences
19          oName = oOcc.Name
20          If oName.Contains(coneSelector) Or oName.Contains(transitSelector) Then
21              oDef = oOcc.Definition
22              defWPs = oDef.WorkPoints
23              Dim wp As WorkPoint
24              For Each wp In defWPs
25                  wpName = wp.Name
26                  If wpName = "ConeTop" Or wpName = "ConeBottom" Or wpName = "Center" Then
27                      Dim proxy As WorkPointProxy
28                      Call oOcc.CreateGeometryProxy(wp, proxy)
29                      X = proxy.Point.X * 10
30                      Y = proxy.Point.Y * 10
31                      Z = proxy.Point.Z * 10
32                      Dim coords = New String() {oName, wpName, X, Y, Z}
33                      oAppend.WriteLine(Join(coords, ";"))
34                  End If
35              Next
36          End If
37      Next
38
39      oAppend.Flush()
40      oAppend.Close()
```

3.2.3 Komentáře kódu

```
4       vytvoření proměnné zapisovače do textového souboru
5       vytvoření proměnné s názvem souboru do kterého budeme zapisovat
        pomocí získání názvu aktuální sestavy + nastavení koncovky souboru
6       vyčistí již existující soubor - důležité při ukládání změn
7       přiřazení souboru do zapisovače
8       pojmenování sloupců v našem souboru (legenda) - pole stringů
9       vypsání legendy na první řádek souboru zřetězením pole stringů pomocí středníku
11      nastavení typu proměnné jako komponenta inventoru -
        - kompletní komponenta se všemi prvky (definice komponenty)
12      načtení existujících komponentů v sestavě
17      nastavení typu proměnné jako výskyt komponentů (10 definic = 10 výskytů)
18      projetí všech komponentů
19      získání názvu komponenty
20      vybrání pouze komponenentů, které se jmenují "cone:" nebo "transit:"
21      získání definice daného komponentu
22      získání všech pracovních bodů z definice komponenty
24      projetí všech získaných pracovních bodů
25      získání názvu daného pracovního bodu
26      vybrání pouze pracovních bodů, které se jmenují "ConeTop", "ConeBottom" nebo "Center"
28      přepočet souřadnicového systému dané součásti na celou součást
29      získání hodnoty souřadnice X pracovního bodu v mm
30      získání hodnoty souřadnice Y pracovního bodu v mm
31      získání hodnoty souřadnice Z pracovního bodu v mm
32      vytvoření pole stringů potřebných informací pro vypsání
33      vypsání do řádku souboru zřetězením pole stringů pomocí středníku
39      vyčištění zásobníku
40      zavření souboru
```

3.2.4 Souhrn pravidel pro spuštění makra

1. Vytvořená soustava musí být uložena.
2. Komponenty pro vyznačení trajektorie musejí mít názvy **cone** a **transit**
3. Komponenty cone musí být číslovány **celými kladnými čísly** $\mathbb{Z}^+$ a to od 1 do n bez vynechaných číslic.
4. Komponenty transit jsou číslovány opět čísly $\mathbb{Z}^+ - \{1\}$ podle čísla součásti cone, kam chceme přes průjezdný bod dojet (obrázek 24).
5. Může bý použit vždy nanejvýš jeden komponent transit na jeden komponent cone s výjimkou umístění před cone:1.
6. Pracovní body součásti cone musí být pojmenovány **ConeTop** a **ConeBottom**.
7. Pracovní bod součásti transit musí být pojmenován **Center**.

4 AItoSRS

Víme jak zakreslit trajektorii v prostředí Inventoru, pomocí makra exportovat zájmové body zakreslené trajektorie a nyní musíme tyto exportované body využít k naplánování pohybu robotu v přostředí SRS. Pro tento krok jsme si vytvořili aplikaci s pracovním názvem AItoSRS.

4.1 Představení aplikace

Jedná se o aplikaci sloužící pro přepočet dat zvolené trajektorie v aplikaci Autodesk Inventor předepsaným způsobem do Staubli Robotics Suite a jak již bylo zmíněno, byla pojmenována jako AItoSRS. Jedná se o zkratku Autocad Inventor to Staubli Robotics Suite a její logo (na obrázku 28) bylo vytvořeno z loga Inventoru a barev používaných v ikoně aplikace SRS.

Obrázek 28: Logo aplikace AItoSRS

Tato aplikace nám ze vstupních dat (data .csv souboru získaná pomocí makra v AI) vytvoří nový .csv soubor obsahující již přepočtená data pro robota ve tvaru souřadnic x, y, z a úhlů postupných rotací Rx, Ry, Rz. Stejně tak nám vytvoří i funkční kód pro robota, který zajistí projetí naplánované trajektorie. Aplikace je vytvořena jako Windows Form Application a naprogramována v programovacím jazyce C#. K vytvoření aplikace jsme použili vývojové prostředí Visual Studio 2019 a aplikace je vytvořená jako samostatně spustitelný soubor s povinnou předchozí instalací.

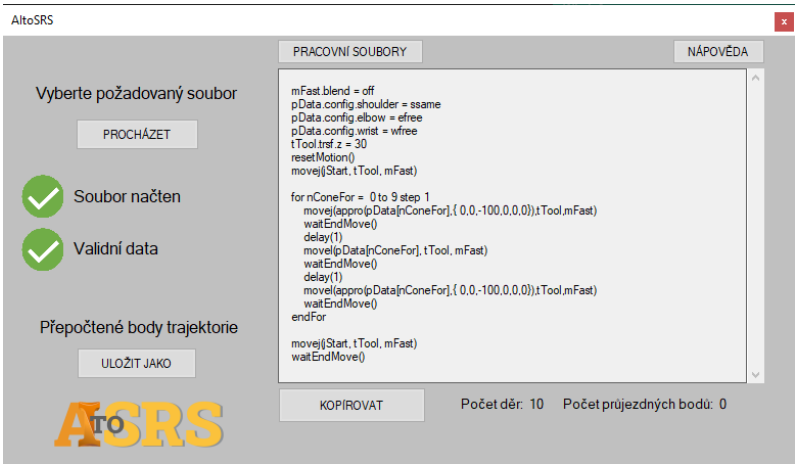
Aplikace je vytvořena tak, aby byla uživatelsky velmi jednoduchá k používání a samozřejmě také přehledná. Navíc obsahuje i PDF soubor s nápovědou, ve které se uživatel s aplikací seznámí a kde je i seznam chybových hlášek.

V této kapitole si jako první ukážeme aplikaci z uživatelského pohledu, kde aplikaci popíšeme a vysvětlíme její používání. Následně si probereme způsob přepočtu směrového vektoru na úhly postupné rotace Rx, Ry, Rz a v závěru si popíšeme algoritmus naší aplikace AItoSRS.

4.2 Popis uživatelského rozhraní

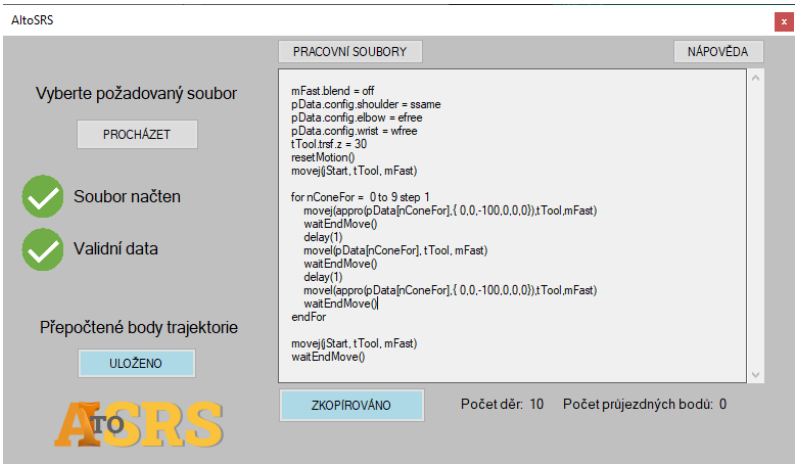
Aplikace jak již bylo zmíněno, je velmi jednoduchá na používání a přehledná. Tomu odpovídá i její design, který můžeme vidět na obrázku 29 , kdy v ní najdeme pouze pětici tlačítek a jedno okno pro vypsání vygenerovaného kódu pro robota. Tato aplikace funguje na principu, kdy vybereme požadovaný soubor k přepočtu dat pomocí tlačítka *PROCHÁZET*. Jedná se o soubor, který jsme si již dříve vyexportovali z prostředí Inventoru pomocí našeho makra. Po vybrání souboru z vyskakovacího okna svou volbu potvrdíme a vyčkáme až se data načtou a program zkontroluje jejich validitu. V případě, že je soubor v pořádku načten a data jsou validní, budeme o této skutečnosti informováni pomocí zelených ikon (viz Obrázek 30 níže). Po načtení souboru a kontrole dat se uživateli zobrazí ihned kód programu, který po zkopírování do SRS zajistí projetí zájmových bodů, které uživatel vyznačil v aplikaci Inventor. Společně s programem se uživateli zobrazí i informace o zjištěném počtu zájmových bodů (komponenty cone) a počet průjezdných bodů (komponenty transit).

Obrázek 29: Náhled po spuštění aplikace

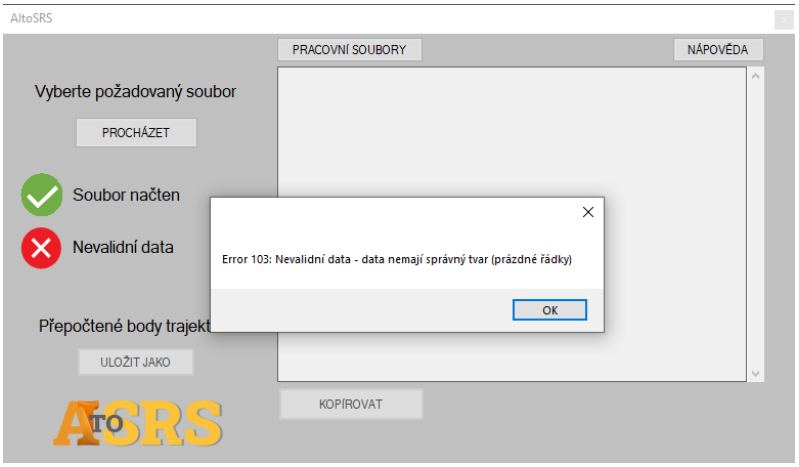


Obrázek 30: Zobrazení kódu k nahrání do SRS

Nyní můžeme již zmíněný kus kódu zkopírovat jednoduše pomocí tlačítka *KOPÍROVAT*, kdy se po kliknutí na toto tlačítko uloží kód do schránky systému. Přepočtené zájmové body trajektorie pak uložíme po kliknutí na tlačítko *ULOŽIT JAKO*, které otevře vyskakovací okno s výběrem umístění ukládaného souboru. Uložený soubor obsahuje data jednotlivých zájmových bodů ve tvaru souřadnic x, y, z a jednotlivých úhlů postupné rotace Rx, Ry, Rz. Pokud máme více souborů k přepočtu, můžeme opět kliknout na tlačítko *PROCHÁZET*, vybrat další soubor a celý postup opakovat. Pro zlepšení přehlednosti se nám také po kliknutí na tlačítko *KOPÍROVAT* a *ULOŽIT JAKO* změní barva daného tlačítka. Zbarvená tlačítka jsou zobrazena na obrázku 31.



Obrázek 31: Zbarvená tlačítka po kliknutí



Obrázek 32: Zobrazení chybové hlášky

V různých případech nám také může program skončit chybou. Jedná se o problémy s načtením souboru nebo validitou vstupních dat, kdy každá chyba je zobrazena pomocí vyskakovacího okna a je označena číslem chyby (viz obrázek 32). Pokud uživatel vytvoří trajektorii v Inventoru pomocí komponentů, tak jak je uvedeno v bodě 3.2.4 a nebude jakýmkoliv způsobem následně přepisovat exportovaný .csv soubor, tak nikdy nedojde k problému s validitou dat. Pokud však uživatel soubor upraví, může být následně jeho soubor odmítnut a to například kvůli špatnému popisu sloupců na první řádce, což je základní ukazatel toho, že se jedná o soubor vyexportovaný naším makrem. Dále například kontrolujeme, zda soubor není prázdný nebo neobsahuje náhodně prázdné řádky či zda číslování komponentů transit odpovídá zadaným předpokladům. To znamená, že pokud máme použitých 10 komponent cone a jeden komponent transit máme pojmenovaný jako transit:20, data nebudou akceptována. Všechny existující chybové hlášky a jejich možné příčiny jsou vypsány v tabulce 1.

Označení	**Možná příčina**
Error 100	Soubor je využíván jiným procesem.
Error 101	Soubor je prázdný.
Error 102	Soubor nemá správnou legendu prvního řádku.
Error 103	Soubor obsahuje náhodně prázdné řádky.
Error 104	V souboru je použité špatné číslování komponenty transit. → číslo je např. větší než počet komponent cone, nebo je rovno nule.
Error 105	V souboru je použitý zákázaný průjezdný bod transit:1 Průjezdný bod před prvním komponentem cone **nelze definovat!**
Error 106	Komponent cone nemá svoji dvojici (ConeTop nebo ConeBottom)

Tabulka 1: Tabulka chybových hlášek

Přímo v aplikaci ještě najdeme dvě tlačítka, o kterých jsme se zatím nezmínili. První tlačítko *PRACOVNÍ SOUBORY* slouží pro stažení potřebných souborů pro vytvoření trajektorie v Inventoru, tedy přímo dva projekty součásti pro Inventor, kdy jeden je součást cone a druhý transit. V tomto souboru pak najdeme ještě i makro potřebné pro export zájmových bodů, jehož soubor vkládáme do aplikace AItoSRS. Kliknutí na druhé tlačítko *NÁPOVĚDA* se nám poté zobrazí nápověda k programu v PDF formátu.

Po úspěšném nahrání souboru a získání potřebných dat a kódu pro robota, stačí již pouze tyto data nakopírovat na příslušná místa do SRS. Získaný kód nahrajeme do funkce `start()` námi vytvořené aplikace v SRS a data z .csv souboru nakopírujeme do pole nastavené proměnné opět v SRS. Potřebnou konfiguraci aplikace pro spuštění získaného kódu, jak vytvořit aplikaci, spustit simulaci a spoustu dalšího bude vysvětleno v kapitole 5.

4.2.1 Generovaný kód

Máme pouze dva druhy generovaného kódu a to jeden pro trajektorii bez průjezdných bodů a druhý pro trajektorii s průjezdnými body. Oba dva kódy jsou téměř totožné, protože mají stejné základní nastavení proměnných a to vypnuté prolnutí u proměnné `mFast` nastavující parametry pohybu mezi komponenty cone. Dále pak nastavení konfigurace jednotlivých kloubů proměnné `pData` a nastavení délky nástroje `tTool` ve směru osy z koncového efektoru na 30mm. Nástroj sice nastavujeme, ale na robota reálně žádný nástroj nepřidáváme a to z toho důvodu, že nástroj používáme spíše jako nastavení bezpečného odstupu od obrobku. Ukázka kódu tohoto nastavení níže.

```
mFast.blend = off
pData.config.shoulder = ssame
pData.config.elbow = efree
pData.config.wrist = wfree
tTool.trsf.z = 30
```

Pro kód s trajektorií přes průjezdné body pak k tomuto základnímu nastavení přidáme ještě nastavení hodnot proměnné `nTransit`, kdy indexujeme vždy podle počtu použitých komponentů transit. Pokud například naplánujeme trajektorii s dvěma průjezdovými body (komponenty transit), tak budeme mít proměnnou s dvěma indexy 0 a 1. Tato proměnná slouží pro uchování informace o návaznosti pohybu z komponentu cone na další tuto komponentu přes průjezdný bod. Zvolíme si dva průjezdné body před cone:3 a cone:8 → do indexu od nejmenšího čísla cone nastavíme hodnoty 2 a 7, tedy hodnoty cone ze kterých máme jet přes průjezdný bod. V tomto nastavení dále ještě nalezneme definování pohybových vlastností proměnné `mDesc`, ve které nastavíme prolnutí se souřadnicově zadanými hodnotami *leave* a *reach* na 30mm a budeme ji používat pro pohyb kolem průjezdných bodů. Vysvětlení prolnutí a nastavení těchto hodnot je zobrazeno na obrázku 24 a v textu v jeho blízkosti.

```
nTransit[0]=2
nTransit[1]=7
mDesc.blend = Cartesian
mDesc.leave = 30
mDesc.reach = 30
```

Základní princip obou kódů je pak stejný, s tím rozdílem, že v případě kódu s použitými komponenty transit máme přidaný navíc jeden cyklus, který kontroluje a nastavuje přejezd přes průjezdný bod ve správné chvíli. Vysvětlíme si tedy jako první základní princip pohybu mezi komponenty cone.

Najedeme s robotem do startovací pozice `jStart`, kterou definuje vždy sám uživatel a následně budeme vykonávat stále se opakující sled pohybových instrukcí *movej* a *movel*, na základě počtu zjištěných komponentů cone. Rozdíl mezi zmíněnými pohybovými instrukcemi je vysvětlen v bodě 46. Budeme projíždět označené díry postupně podle čísla jejich cone a potřebná data těchto děr máme taktéž v proměnné `pData` seřazena podle číslování cone.

První cyklus = projetí první díry (nConeFor = 0). Ze startovní pozice jedeme příkazem *movej* na pozici, která je vzdálená od středu dané díry 100mm ve směru její osy. Již v této fázi má robot osu **Z** koncového efektoru srovnanou ve směru osy naší první díry. V této pozici setrvá 1s a příkazem *movel* jede ve směru osy díry blíže k souřadnicím středu díry. Kdybychom neměli nastavený nástroj, tak by jel přesně až na dané souřadnice a do obrobku by naboural. Díky zvolenému nástroji se ale zastaví o 30mm dříve. Opět setrvá 1s a stejným pohybem se opět vrací ve směru osy díry na předchozí pozici. Přejíždí ihned nad druhou díru a stejný postup opakuje pro všechny díry. Po projetí tohoto sledu příkazů i u poslední díry se opět robot vrátí na startovací pozici, kde dojde k vypnutí aplikace.

```
resetMotion()
movej(jStart, tTool, mFast)

for nConeFor =  0 to 9 step 1
    movej(appro(pData[nConeFor],{ 0,0,-100,0,0,0}),tTool,mFast)
    waitEndMove()
    delay(1)
    movel(pData[nConeFor], tTool, mFast)
    waitEndMove()
    delay(1)
    movel(appro(pData[nConeFor],{ 0,0,-100,0,0,0}),tTool,mFast)
    waitEndMove()
endFor

movej(jStart, tTool, mFast)
waitEndMove()
```

Princip druhého typu kódu spočívá v tom, že stejně jako v přechozím kódu najede robot do startovní pozice a následně provede první sled pohybových instrukcí pro první díru. Zde je ale změna oproti prvnímu kódu a robot nejede ihned nad druhou díru, ale zkontroluje, zda momentálně nemá jet nad druhou díru přes průjezdný bod. To zjišťuje pomocí cyklu nTransitFor ve kterém porovnává pořadí cyklu s hodnotami uloženými v proměnné nTransit. Když nalezne shodu, robot vykoná navíc jednu pohybovou instrukci *movej*, kterou projede s prolnutím kolem průjezdného bodu nad druhou díru. V případě neshody, jede nad druhou díru rovnou a stejným sledem instrukcí tímto způsobem projede všechny díry. Po projetí poslední díry se opět vrátí na startovací pozici a dojde k vypnutí aplikace.

```
resetMotion()
movej(jStart, tTool, mFast)

for nConeFor =  0 to 9 step 1
    for nTransitFor = 0 to 1 step 1
        if nConeFor == nTransit[nTransitFor]
            movej(pData[10 +nTransitFor], tTool, mDesc)
        endIf
    endFor
    movej(appro(pData[nConeFor],{ 0,0,-100,0,0,0}),tTool,mFast)
    .....
endFor
.....
```

4.3 Algoritmus kódu apikace AItoSRS

Jak již bylo zmíněno, tak jsme naší aplikaci vytvořili v jazyce C# pomocí frameworku Windows Form Application a využívali jsme zde především znalosti objektového programování. Nyní si zkráceně popíšeme algoritmus aplikace chronologicky ihned od načítání souboru. Kliknutím na tlačítko *PROCHÁZET* otevřeme dialogové okno, kde si uživatel zvolí požadovaná data k přepočtu a svůj výběr potvrdí tlačítkem *Otevřít*. Tímto získáme náš vstupní soubor, který program po jednotlivých řádcích přečte a zkontroluje pomocí formátu první řádky, zda se jedná o soubor exportovaný pomocí našeho makra. Průběžně také vyhodnocujeme zobrazení jednotlivých chybových hlášek a to přesněji Error 100 až Error 102. V tuto chvíli máme soubor zkontrolovaný z hlediska správnosti souboru, nikoliv dat a na základě výsledků kontroly zobrazíme první informaci o načtení souboru.

Nyní bude program číst soubor opět po řádcích a také již vyhodnocovat správnost dat, které si budeme průběžně ukládat do potřebných proměnných. V rámci tohoto bloku budeme opět vyhodnocovat zobrazení chybových hlášek a to tentokrát pro Error 103 až Error 106. Začneme načítat jednotlivé řádky souboru a každý řádek si rozdělíme podle středníků na pole hodnot. Z každého řádku nás zajímá celý název (cone:10), který následně rozdělíme podle dvojtečky na název, číslovku a dále nás pak zajímají ještě jednotlivé hodnoty x, y, z. Na základě určeného názvu (cone, transit) se pak podmínkou if dostaneme na kód určený pro daný typ zjištěné komponenty. Rozdíl kódu je pouze v tom, že u komponenty cone budeme vždy vyžadovat dvojici dvou hodnot, tedy ConeTop a ConeBottom zatímco u komponenty transit nás zajímá pouze jedna hodnota Center. Předpokládáme správného seřazení dat, tak že na prvním řádku je vždy ConeTop a na řádku pod ním ConeBottom stejného čísla dané komponenty, nikoliv opačně. Toto řazení nám zajišťuje makro, takže bychom s tímto předpokladem neměli mít problém. Vytvoříme si dvě třídy představující naše komponenty a které jsou pojmenované stejně jako Cone a Transit.

V případě zjištění komponenty cone si uložíme zjištěné hodnoty x, y, z do proměnné vektoru **TOP** a ihned v této podmínce přečteme další řádek, který opět rozdělíme podle středníků. V případě že se jedná o stejnou komponentu se stejným číslem, hodnoty x, y, z uložíme do proměnné vektoru **BOTTOM**. Z obou těchto vektorů a čísla komponenty si nyní vytvoříme objekt `Cone cone = new Cone(number,TOP,BOTTOM)`, který následně vložíme do listu. Vytvoříme si i `Transit transit = new Transit(number,CENTER,OBJECT-CONE)`, tedy objekt pro komponentu transit, kde **CENTER** je opět vektor souřadnic x, y, z a **OBJECT-CONE** je objekt cone, který má číslo o jedna menší než je číslo aktuální komponenty transit. U komponenty transit budeme ještě vytvářet list s postupně přidávanými číselnými hodnotami z názvu jednotlivých komponent transit. Stejným postupem projedeme celý soubor a na konci budeme mít list všech komponent cone, transit a list číselných hodnot z názvů komponent transit. Následně si všechny tři listy seřadíme podle číselných hodnot z názvů jednotlivých komponent (number). Pokud v této části nenalezneme žádný problém s validitou dat, zobrazíme i druhou informaci o validních datech.

Nyní si z již známých proměnných a příkazů připravíme dva druhy kódů, kdy následně jeden z nich na základě počtu zjištěných komponent transit vypíšeme do textboxu naší aplikace. V případě že nenalezneme žádný komponent transit, vypíšeme kód bez řešení přejezdových bodů, v opačném případě vypíšeme druhý kód. Spolu s výpisem tohoto kódu si vypíšeme do dalších textboxů počet nalezených komponentů cone a transit, které se budou zobrazovat vedle textu „Počet děr“ a „Počet průjezdných bodů“.

Vstupní data začneme do výstupního formátu přepočítávat až v případě, kdy klikneme na tlačítko *ULOŽIT JAKO*. Po kliknutí na toto tlačítko zvolíme požadované umístění, název souboru a potvrzením své volby tlačítkem *Uložit* spustíme proces přepočtu dat a jejich průběžné ukládání do .csv souboru.

V cyklu si postupně projedeme všechny objekty cone uložené v příslušném listu, na kterých vždy vyvoláme metodu **ANGLE**, která nám ze dvou vektorů určí směrový vektor. Z tohoto směrového vektoru nám následně určí postupné rotace Rx, Ry, Rz a vrátí nám je jako parametry daného objektu. Následně si vytvoříme pole hodnot ve tvaru `{X,Y,Z,Rx,Ry,Rz}`, kde hodnoty x, y, z budou souřadnice ConeTop daného komponentu a postupné rotace budou hodnoty vrácené naší metodou angle. Z tohoto složeného pole pak vytvoříme textový řetězec s oddělovači jako středník a zapíšeme jej na aktuální řádku do souboru. Stejným postupem přepočítáme a nahrajeme všechny vektory cone a v případě že již nemáme žádné komponenty transit, program zde končí a čeká na další instrukce. Pokud ale máme nějaké komponenty transit, projedeme je stejným způsobem v cyklu jako komponenty cone s tím rozdílem, že pro zjištění postupných rotací nyní budeme brát směrový úhel mezi naším bodem Center a bodem ConeTop objektu cone, který máme již v daném objektu transit uložený. Pro výpočet zde používáme metodu **ANGLE-TRANSIT** a při ukládání do souboru pokračujeme za posledním přidaným řádkem s komponentou cone. Pokud tedy máme 10 komponent cone a 2 komponenty transit, prvních 10 řádků našeho .csv souboru budou hodnoty komponent cone a řádky 11 a 12 budou hodnoty komponent transit.

Princip přepočtu směrového vektoru na postupné rotace, který je používán v obou metodách je vysvětlen v následujícím bodě 4.4.

4.4 Výpočet úhlů rotace

Abychom mohli aplikaci naprogramovat, museli jsme jako první vymyslet způsob jak přepočítat souřadnice směrového vektoru na úhly postupné rotace.

Postupná rotace XYZ je rotace vždy kolem aktuálních os souřadnicového systému. Ve schéma rotace XYZ provedeme jako první rotaci kolem osy x o úhel α. Další rotaci o úhel β poté vytváříme už kolem osy y', tedy osy y nového souřadnicového systému (s.s.). Na závěr uděláme ještě rotaci kolem osy z" o úhel γ a máme vytvořenou postupnou rotaci. Úhly α , β , γ nazýváme Eulerovými úhly. Obecně u rotace musíme ještě rozlišovat zda máme pravotočivý nebo levotočivý systém. V případě našeho robota je tento údaj stejně jako schéma rotace (XYZ) daný výrobcem.

Matice rotací okolo jednotlivých os pravotočivého systému jsou definované následovně:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{bmatrix} \tag{1}$$

$$R_y(\beta) = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{bmatrix} \tag{2}$$

$$R_z(\gamma) = \begin{bmatrix} cos(\gamma) & -sin(\gamma) & 0 \\ sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

Výsledkem postupné rotace XYZ ze s.s. F_1 do s.s. F_2 je matice rotace, která je dána vztahem

$$R_1^2 = R_x(\alpha) \cdot R_y(\beta) \cdot R_z(\gamma) \tag{4}$$

a její výsledná matice po pronásobení vypadá následovně

$$R_1^2 = \begin{bmatrix} c(\beta)c(\gamma) & -c(\beta)s(\gamma) & s(\beta) \\ s(\alpha)s(\beta)c(\gamma) + c(\alpha)s(\gamma) & -s(\alpha)s(\beta)s(\gamma) + c(\alpha)c(\gamma) & -s(\alpha)c(\beta) \\ -c(\alpha)s(\beta)c(\gamma) + s(\alpha)s(\gamma) & c(\alpha)s(\beta)s(\gamma) + s(\alpha)c(\gamma) & c(\alpha)c(\beta) \end{bmatrix}, \tag{5}$$

kde písmena c a s přestavují zkrácený zápis *sin* a *cos*. Jednotlivé sloupce této matice pak reprezentují souřadnice jednotkových směrových vektorů našeho původního s.s. F_1 v s.s. F_2. Tento zápis můžeme napsat následovně:

$$R_1^2 = \begin{bmatrix} x_1^2 & y_1^2 & z_1^2 \end{bmatrix} \tag{6}$$

V naší úloze potřebujeme, abychom zarovnali osu z koncového efektoru se zjištěným směrovým vektorem u. Ten získáme rozdílem bodů ConeTop a ConeBottom, které jsou pomocí jedné komponenty cone umístěny na ose dané díry a jejichž hodnoty x, y, z máme exportované pomocí makra.

$$u = ConeTop - ConeBottom \tag{7}$$

$$u = \begin{bmatrix} u_x & u_y & u_z \end{bmatrix}^T \tag{8}$$

Nyní když známe vektor u, tak si můžeme vytvořit soustavu rovnic

$$\begin{bmatrix} u \end{bmatrix} = \begin{bmatrix} z_1^2 \end{bmatrix} \tag{9}$$

$$\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} s(\beta) \\ -s(\alpha)c(\beta) \\ c(\alpha)c(\beta) \end{bmatrix} \tag{10}$$

Máme tedy soustavu tří rovnic, ze kterých postupnými úpravami vyjádříme vztahy pro jednotlivé goniometrické funkce.

$$\text{I:} \quad u_x = \sin(\beta) \tag{11}$$
$$\text{II:} \quad u_y = -\sin(\alpha)\cos(\beta) \tag{12}$$
$$\text{III:} \quad u_z = \cos(\alpha)\cos(\beta) \tag{13}$$

$\text{II}^2 + \text{III}^2$:

$$\begin{aligned}(-\cos(\beta)\cdot\sin(\alpha))^2 + (\cos(\beta)\cdot\cos(\alpha))^2 &= u_y^2 + u_z^2 \\ \cos(\beta)^2\cdot\sin(\alpha)^2 + \cos(\beta)^2\cdot\cos(\alpha)^2 &= u_y^2 + u_z^2 \\ \cos(\beta)^2\cdot(\sin(\alpha)^2 + \cos(\alpha)^2) &= u_y^2 + u_z^2 \\ \cos(\beta)^2 &= u_y^2 + u_z^2 \\ \cos(\beta) &= \pm\sqrt{u_y^2 + u_z^2}\end{aligned} \tag{14}$$

II :

$$\begin{aligned}-\cos(\beta)\cdot\sin(\alpha) &= u_y \\ \sin(\alpha) &= \frac{u_y}{-\cos(\beta)}\end{aligned} \tag{15}$$

III :

$$\begin{aligned}\cos(\beta)\cdot\cos(\alpha) &= u_z \\ \cos(\alpha) &= \frac{u_z}{\cos(\beta)}\end{aligned} \tag{16}$$

Zjištěné vztahy goniometrických funkcí použijeme pro určení výsledných úhlů α a β.

$$\alpha = \text{atan2}(\sin(\alpha), \cos(\alpha)) \tag{17}$$
$$\beta = \text{atan2}(\sin(\beta), \cos(\beta)) \tag{18}$$

Z vyjádření $\cos(\beta)$ z rovnice 14 si můžeme všimnout dvou řešení, jedno kladné a druhé záporné. Výsledek odpovídá praktickým znalostem, že do jednoho bodu se můžeme dostat vždy dvěma způsoby a pro řešení výpočtů v naší bakalářské práci jsme použili záporné řešení $\cos(\beta)$.

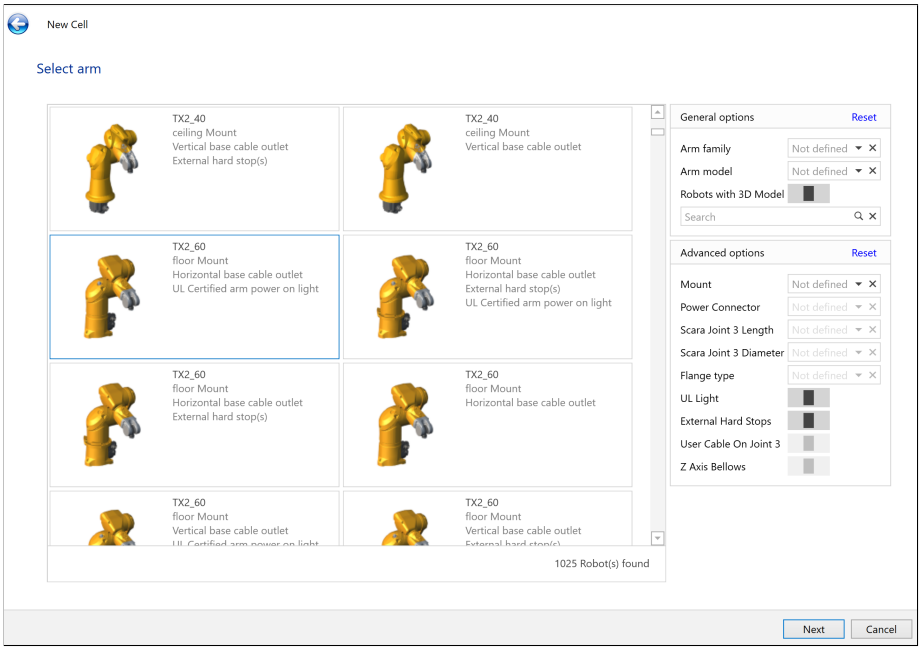
Co se týče hodnoty úhlu γ jakožto úhel natočení kolem osy z koncového efektoru, tak ten nastavujeme pevně na hodnotu 0, jelikož pro naše řešení není tento úhel potřebný.

5 Staubli Robotics Suite

Staubli Robotics Suite (SRS) je komplexní softwarová platforma, která umožňuje programování, provoz a monitorování robotických systémů Staubli. Nabízí snadno použitelné rozhraní pro tvorbu a úpravu programů robotů a také výkonné simulační prostředí pro testování programů před jejich nasazením do ostrého provozu. V rámci naší bakalářské práce budeme pracovat s verzí Staubli Robotics Suite 2019.3.4.

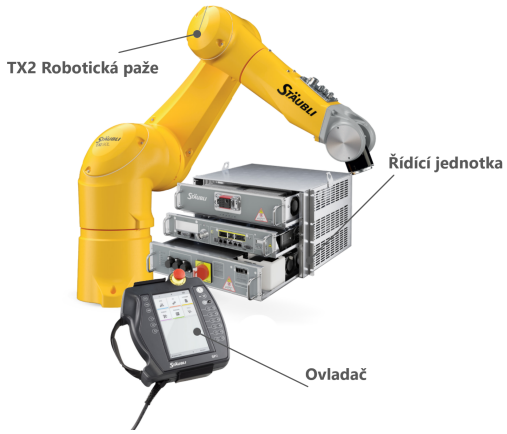
5.1 První spuštění

Po nainstalování a otevření programu si musíme založit nový projekt *New → New cell wizard* čímž se nám spustí průvodce vytvoření nového projektu. Při volbě typu kontroleru zvolíme *Add a local controller* a v následujícím okně (obrázek 33) si zvolíme z nabídky požadovaného robota. My jsme v bakalářské práci pracovali s robotem typu TX2_60 s podlahovým ukotvením. Volbu potvrdíme a otevře se nám další okno kde nalezneme volbu varianty kontroleru → máme nastavenou variantu *s8.9-Cs9_BS1789*. V tomto okně najdeme ještě nastavení ventilů a případně volíme i druh těchto ventilů. V poslední nabídce si zvolíme balíček VAL3, nastavení potvrdíme a máme vytvořený projekt s vybraným robotem a kontrolerem. Při prvním spouštění si také musíme zkontrolovat funkčnost licenčního USB dongle, abychom mohli program spustit. Narazili jsme na problém, že licenční USB dongle nevypíše chybovou hlášku při nutnosti doinstalování jednotlivých ovladačů, takže jsme problém hledali poměrně dlouho a po doinstalování ovladačů již licence začala fungovat. Funkční dongle poznáme podle aktivní červení LED diody na jeho konci.

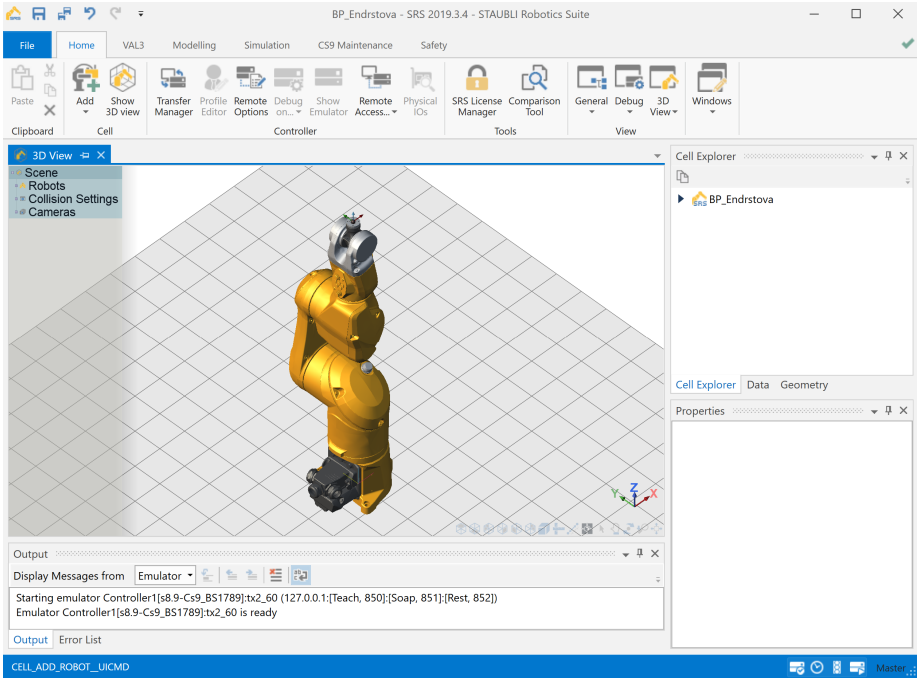


Obrázek 33: Ukázka nabídky robotů

V textu výše se bavíme o nastavení a typu kontroleru, tak bychom si měli říci co to je. Kontroler je řídící jednotka robotu, která se stará o všechny pohyby, limity, nastavení atd. a programujeme ji pomocí programovacího jazyka VAL3. Tato řídící jednotka má obecný rackový design s vyměnitelnými zásuvky a je instalovatelná do rack skříně. Z dostupných rozhraních pak na ni najdeme například USB porty, rychlé vstupy a výstupy, ethernetové porty, digitální a analogové vstupy a výstupy. K této řídící jednotce máme ještě metalickým spojení připojený ovladač (manuální řídící jednotku) s kterou můžeme nastavovat, programovat a ovládat robota bez nutnosti spouštět SRS.



Obrázek 34:



Obrázek 35: Prostředí SRS

5.2 Funkce prostředí SRS

V tomto bodě si popíšeme jednotlivé karty programu, co v nich najdeme za důležité bloky a jaká je jejich funkčnost či význam.

1. **Home**

   Uživatelská nastavení výběru jednotlivých oken, přenos dat a přidávání robotů.

   - Add - přidání dalšího robota včetně dalšího kontroleru
   - Show 3D view - zobrazení 3D náhledu robotu
   - Transfer Manager - slouží pro přenos existujícího projektu z/do kontroleru
   - Remote Access - propojení simulačního ovladače s reálným ovladačem, ale z hlediska bezpečnosti nelze následně simulačním ovladačem například zapnout napájení ramene
   - General - zobrazení jednotlivých oken s daty (Cell Explorer, Data, Geometry atd)

2. **VAL3**

   Zabývající se čistě programové části, přesněji programování v jazyce VAL3. Můžeme zde vytvořit novou aplikaci(celek) a program(spustitelné části např. start, stop), zkontrolovat syntaxi a následně otestovat funkčnost nebo nalézt případný problém (debug).

3. **Modelling**

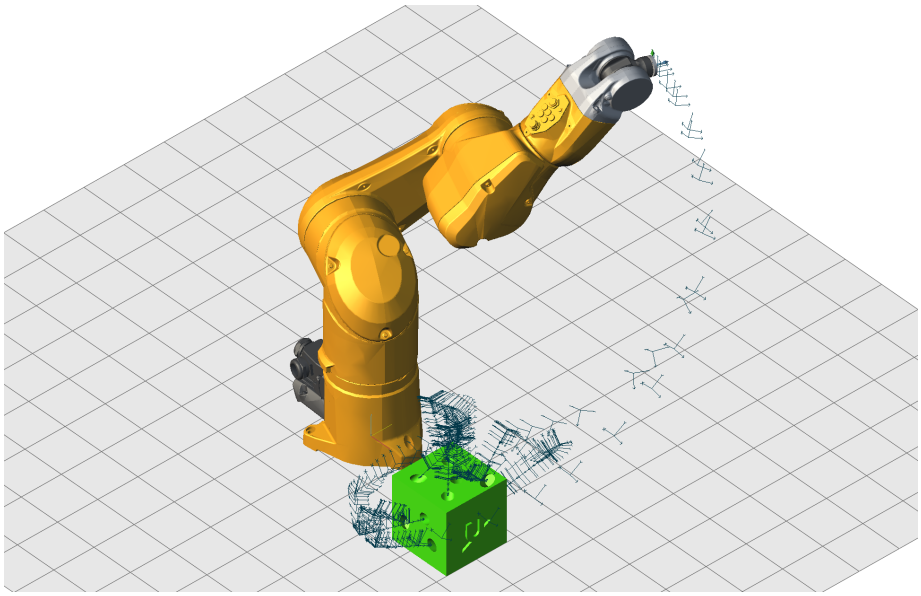
   Zabývající se 3D zobrazením a nastavením nástrojů a 3D komponentů.

   - Insert CAD - vložení 3D modelu vytvořeného například v Inventoru
   - Add - přidání základních 3D těles - krychle, kvádr, koule, válec
   - New - vybrané těleso nastaví jako nástroj, součást
   - Attach to - přichycení nástroje k efektoru
   - Detach - odebrání nástroje z efektoru

4. **Simulation**

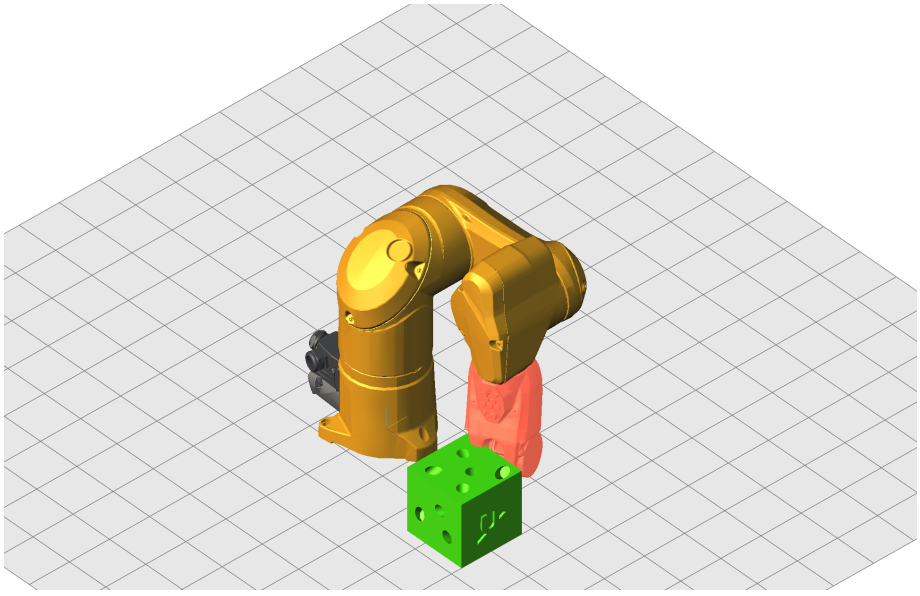
   Vše co se týče simulace pohybů robota ať v ručním režimu nebo při simulaci VAL3 aplikace.

   - Joint Mover - po vybrání můžeme v 3D zobrazení pohybovat robotem pomocí změny jednotlivých kloubových souřadnic
   - Cartesian Mover - po vybrání můžeme v 3D zobrazení pohybovat koncovým efektorem v kartézských souřadnicích
   - Move To - najetí pozice vybrané proměnné
   - Here - uložení aktuální polohy robota do vybrané proměnné



Obrázek 36: Náhled trajektorie při zapnutém Show traces

- Show traces - při simulaci pohybu se bude v 3D zobrazení zobrazovat projetá trajektorie (obrázek 36)
- Collisions - po vybrání se při simulaci bude označovat červeně část robota, která bude v kolizi s jinými předměty v 3D zobrazení (obrázek 37)



Obrázek 37: Zobrazení kolize při zapnutém Collisions

- Start synchro - musí být zapnuto aby se při spuštěné VAL3 aplikaci robot v 3D zobrazování pohyboval
- Record movie - pomocí tohoto bloku můžeme nahrávat pohyb robota při simulaci

5. **CS9 Maintenance**

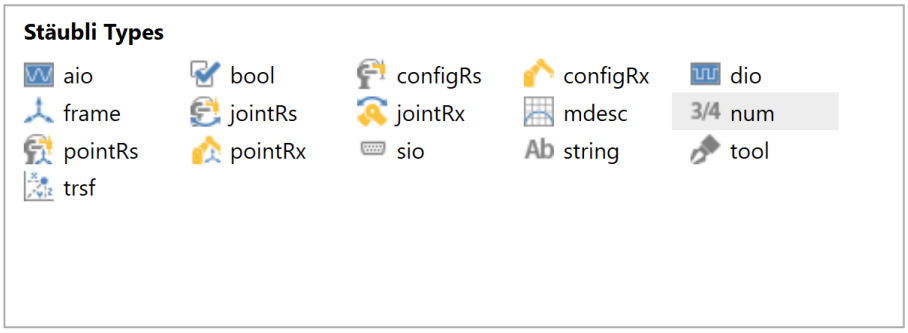
   Zobrazení logu simulace a nastavení jaké úrovně hlášení se nám budou při simulaci zobrazovat.

6. **Safety**

   Nastavení bezpečnosti, bezpečných zón a zobrazení těchto zón v 3D zobrazení. Nastavení bezpečných zón nejde zohlednit při simulaci jelikož přehrání bezpečnosti robota lze dělat pouze pomocí další aplikace a to pouze přímo nahráním do reálného robota. Ukázku nastavení bezpečné zóny pro náš obrobek si vysvětlíme později v bodě 5.7.4.

5.3 Základní datové typy

SRS disponuje základními datovými typy, které jsou vyznačeny níže na obrázku 38. Využívají se jak při programování ve VAL3, tak i například při ručním zadáváním bodů v 3D zobrazení. Jednoduše řečeno všechny body, nastavení atd. na které se chceme nějakým způsobem odkazovat a chceme je jednoduše používat opakovaně, tak je musíme mít uložené pod nějakou proměnnou daného datového typu.



Obrázek 38: Datové typy SRS

- **aio + dio** - analogové a digitální vstupy/výstupy (I/O), kdy proměnná se odkazuje (linkuje) na reálný I/O. V případě spouštění vzduchových ventilů s touto proměnnou pracujeme následovně: false → výstup rozepnut a true → výstup sepnut. `dOUT_BC_vystup = true`
- **bool** - proměnná typu boolean, která je reprezentována jednou ze dvou logických hodnot a to false a true
- **frame** - datový typ pro USS (uživatelský souřadnicový systém), pomocí kterého si můžeme vytvořit nový USS ke kterému můžeme vztahovat souřadnice bodů. Nový frame je vytvořen pomocí zadání posunu a natočení vůči UCS neboli pomocí zadání jeho absolutních souřadnic.
- **mdesc** - slouží pro nastavení parametrů pohybu (rychlost, zrychlení, prolínání)

- **num** - představuje číselnou hodnotu s přibližně 14 platnými číslicemi
- **sio** - používá se pro pojení proměnné se sériovým portem nebo připojením Ethernet Socket
- **string** - slouží pro ukládání textu
- **tool** - slouží pro nastavení velikosti používaného nástroje s kterou pak robot počítá při najíždění do určitých bodů
- **trsf** - vyjadřuje transformaci vůči nějakému jinému bodu pomocí translace a rotace. Využíváme například při najetí robotu 5cm nad určitý bod (aproximace). Nemusíme si vytvářet nový bod pouze s jednou změněnou souřadnicí, ale využijeme aproximaci bodu se zadanou transformací viz příkaz níže a právě `{ 0,0,-50,0,0,0}` je naše transformace, která se může pod proměnnou **trsf** zapsat.

  ```
  movel(appro(pPohyb[0],{ 0,0,-50,0,0,0}),tTool,mFast)
  ```

Zbylé datové typy jsou vždy dvojice, které mají stejnou funkčnost ale jsou pro různé roboty. Datové typy Rs jsou pro roboty se 4mi stupni volnosti (4DoF) tedy roboty typu Scara (obrázek 39) a datové typy Rx jsou pak pro klasické robotické paže s 6DoF. Pro tyto typy tedy vždy vysvětlíme název s Rx a pro Rs je vysvětlení stejné.

- **jointRx** - nastavení koncové polohy efektoru pomocí úhlových souřadnic jednotlivých motorů
- **pointRx** - nastavení koncové polohy efektoru pomocí souřadnic x, y, z a postupných rotací Rx, Ry, Rz
- **configRx** - konfigurace způsobu natáčení jednotlivých kloubů: rameno, loket, zápěstí (shoulder, elbow, wrist)

Obrázek 39: Staubli robot typu Scara

5.4 Konfigurace kloubů

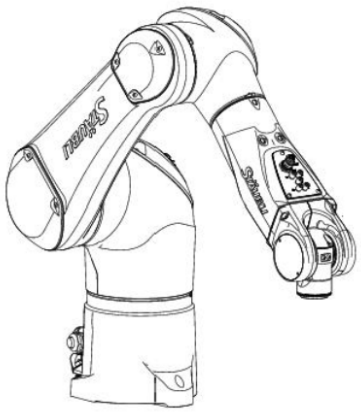
U datového typu configRx jsme narazili na možnost nastavovat funkčnost jednotlivých kloubů a zde si to více popíšeme a názorně ukážeme.

Každý kloub můžeme nezávisle na ostatních nastavit na jednu ze 4 přednastavených konfigurací, které změní způsob natočení jednotlivého kloubu. Do každého bodu se můžeme vždy dostat dvěma způsoby a touto konfigurací řekneme, jakým způsobem má robot k bodu najet. Můžeme ji nastavit pomocí již zmíněného datového typu configRx a nebo přímo v nastavení bodu datovým typem pointRx, kde prvních 6 pozic pole je pro nastavení hodnot x, y, z, Rx, Ry, Rz a následující 3 pozice jsou právě pro nastavení konfigurace kloubů. Posledním způsobem jak lze změnit konfigurace kloubů je přes okno *Jog*, do kterého se dostaneme po kliknutím pravým tlačítkem na tělo robota v 3D zobrazení a vybráním již zmíněné karty.

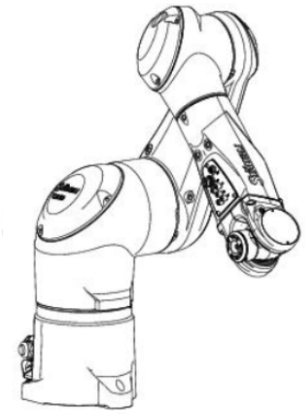
Nejčastějším nastavením je ssame, esame, wsame s využitím vhodného nastavení startovacího bodu. Robot si bude vždy brát nastavení z předchozího bodu a bude tedy vždy najíždět podobným způsobem a nebude se zbytečně kroutit a točit. V konfiguraci ramene prvního bodu je vhodné zvolit variantu lefty především v případě, že máme obrobek umístěný na středu před robotem. Robot pak má větší dosah bez nutnosti přetočení ramene na druhou stranu.

5.4.1 Konfigurace ramene

- righty - konfigurace viz obrázek 40
- lefty - konfigurace viz obrázek 41
- ssame - konfigurace přebírající nastavení z předchozího bodu
- sfree - volná konfigurace



Obrázek 40: Konfigurace righty

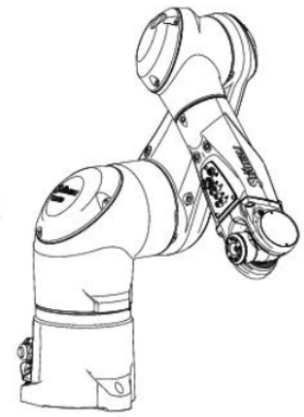


Obrázek 41: Konfigurace lefty

5.4.2 Konfigurace lokte

- enegative - konfigurace viz obrázek 42
- epositive - konfigurace viz obrázek 43
- esame - konfigurace přebírající nastavení z předchozího bodu
- efree - volná konfigurace

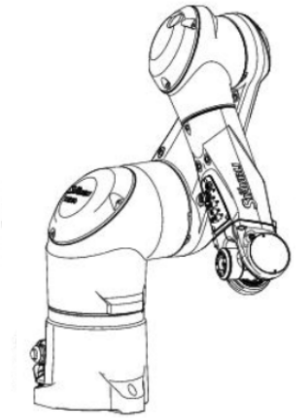
Obrázek 42: Konfigurace enegative



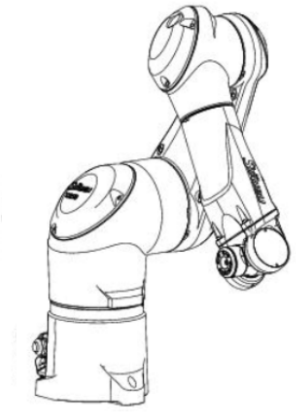
Obrázek 43: Konfigurace epositive

5.4.3 Konfigurace zápěstí

- wnegative - konfigurace viz obrázek 44
- wpositive - konfigurace viz obrázek 45
- wsame - konfigurace přebírající nastavení z předchozího bodu
- wfree - volná konfigurace



Obrázek 44: Konfigurace wnegative

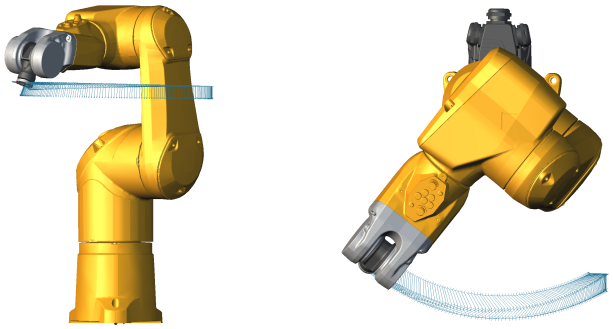


Obrázek 45: Konfigurace wpositive

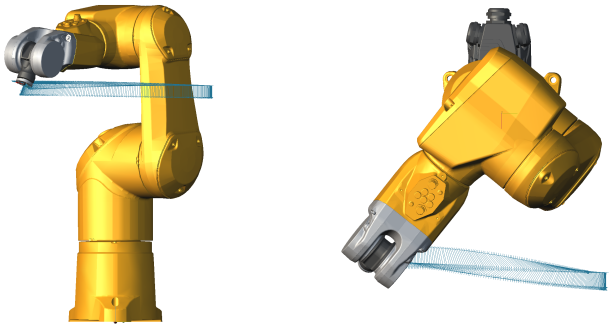
5.5 Pohybové příkazy

Mezi základní pohyby, které jsme v rámci naší bakalářské práce používali, patří příkaz **movej** a **movel**. Tyto dva příkazy slouží vždy pro pohyb mezi dvěma zadanými body a liší se především ve způsobu pohybu mezi nimi. Příkaz movej používáme v případě, kdy naším cílem je především pouze přesun do cílového bodu a nezáleží nám na tom, kudy a jak robot mezi těmito dvěma body pohyb vykoná. Při použití tohoto příkazu si robot určí sám takovou trajektorii, aby moc nezatěžoval své klouby a optimalizovat svou rychlost pohybu. Jak můžeme vidět na obrázku 46, robot přejíždí mezi dvěma body obloukem, který si sám vypočítal jako nejlepší cestu. Příkaz movel pak používáme především v případě, kdy potřebujeme aby se robot mezi dvěma body pohyboval se středem koncového efektoru po přímce (zobrazeno na obrázku 47). Tento pohyb je většinou pomalejší a pro robota náročný, protože má přesně danou trajektorii které se musí pomocí natočení jednotlivých kloubů přizpůsobit.

Pokud se tedy potřebujeme dostat z jednoho do druhého bodu co nejrychleji, nejjednodušeji a nezáleží nám na tom kudy robot trajektorii provede, zvolíme příkaz movej. Na druhou stranu pokud máme třeba zadanou nějakou aproximaci bodu a potřebujeme se například pohybovat na ose díry jako v rámci naší bakalářské práce, použijeme druhý příkaz movel.



Obrázek 46: Pohyb mezi 2 body - příkaz movej



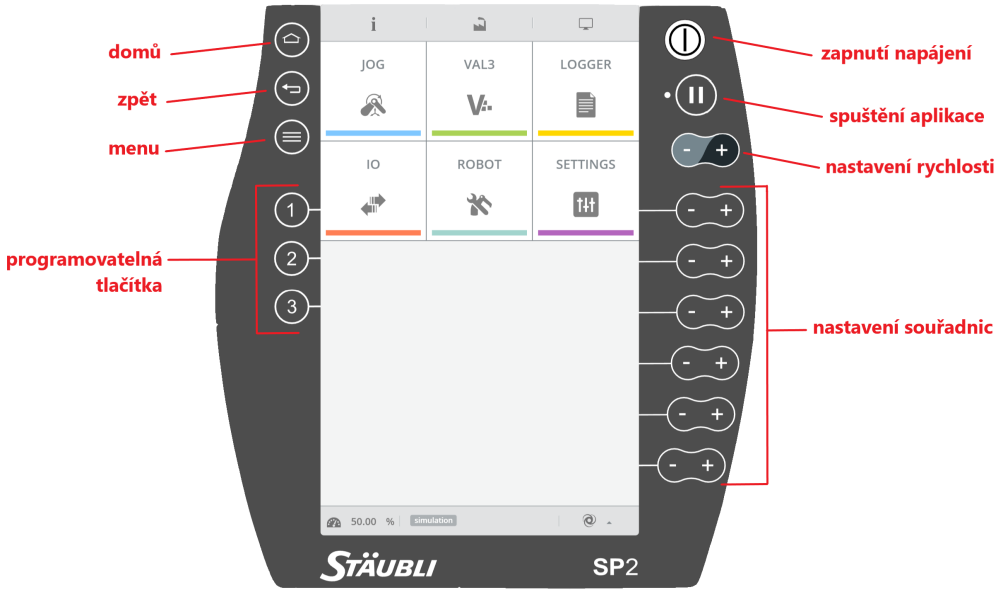
Obrázek 47: Pohyb mezi 2 body - příkaz movel

5.6 Seznámení se simulačním ovladačem

Stejně jako máme ovladač u reálného robota, tak i v simulačním prostředí používáme simulační ovladač. Tento ovladač slouží pro spouštění jednotlivých aplikací, najíždění s robotem do určité pozice a pro různá nastavení. Na obrázku 48 máme zobrazený náhled ovladače s popisky základních tlačítek. Nejdůležitějšími tlačítky pro automatický režim jsou určitě zapnutí napájení, spuštění aplikace a tlačítko z dotykové obrazovky VAL3. Pro manuální režim je to pak šestice tlačítek pro nastavení souřadnic ať úhlových či koncového efektoru.

Uprostřed ovladače najdeme dotykovou obrazovku s následujícími prvky:

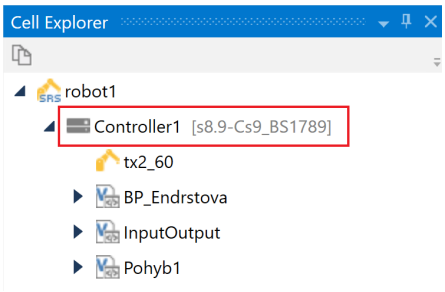
- JOG - v tomto okně můžeme s robotem pohybovat v ručním režimu pomocí zvoleného druhu pohybu (Joint, Frame, Tool) a můžeme zde najíždět i do existujících bodů z vybrané aplikace nebo do bodů vůči různým USS
- VAL3 - slouží pro nahrávání a spouštění naprogramované aplikace VAL3
- logger - protokol událostí (log)
- I/O - nastavení vstupů a výstupů
- Robot - nastavení limitů a brzd, provádění kalibrace atd.
- Settings - uživatelská nastavení a nastavení připojení k síti



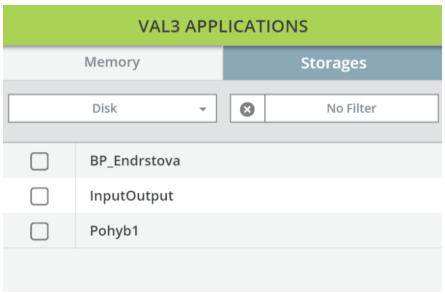
Obrázek 48: Náhled ovladače v simulaci

5.6.1 Spuštění aplikace VAL3

V této části si názorně ukážeme jak spustit vybranou aplikaci krok po kroku. Jako první si musíme v simulačním prostředí zobrazit ovladač. V okně *Cell Explorer* klikneme pravím tlačítkem na náš ovladač (obrázek 49) a v zobrazené nabídce vybereme *Show Emulator*. Následně klikneme na tlačítko *VAL3* na ovladači a zobrazí se nám nabídka viz obrázek 50.

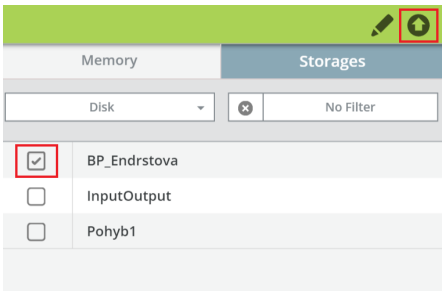


Obrázek 49: Vybrání ovladače

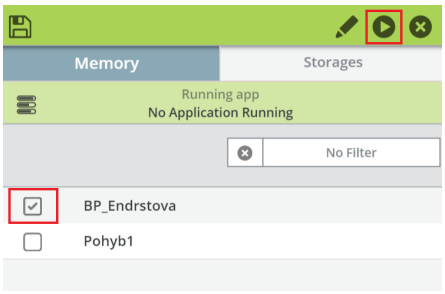


Obrázek 50: Základní zobrazení VAL3

Výběrem naší aplikace a kliknutím na ikonu šipky přehrajeme aplikaci z úložiště do operační paměti robotu (obrázek 51). Opětovným vybráním aplikace a kliknutím na ikonu spuštění naší aplikaci spustíme (obrázek 52). V seznamu existujících aplikací se nám nyní zobrazuje **Running** u naší právě spuštěné aplikace.



Obrázek 51: Nahrání do paměti



Obrázek 52: Spuštění aplikace

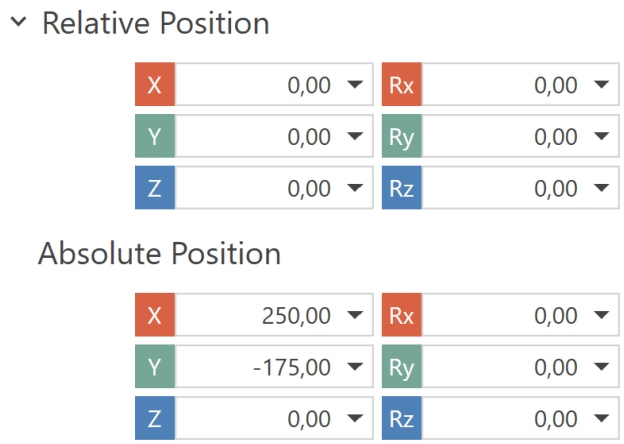
Již stačí pouze zapnout napájení robotické paže, spustit běh programu tlačítkem níže a mít zapnutou simulaci, abychom viděli naprogramovaný pohyb robota v 3D zobrazení.

5.7 Konkrétní konfigurace SRS

Zde si ukážeme všechna důležitá i podpůrná nastavení programu z hlediska funkčnosti našeho řešení bakalářské práce. Stejně tak si zde uvedeme všechny proměnné, které musí být v programu definované. Prvotní konfigurace souboru zůstává stejně, tak jak je uvedeno v bodě 5.1.

5.7.1 Umístění obrobku

Začneme přidáním naší zkušební krychle ve formátu .stl do 3D zobrazení pomocí příkazu *Insert CAD*. Po vložení musíme krychli umístit vhodně před robota a aby se nám s krychlí lépe přesouvalo, změníme si umístění její vztažné soustavy (reference frame). Krychli označíme a na kartě *Modelling* zvolíme *Edit Reference Frame*. Následným vybráním *Point to point* na stejné kartě a kliknutím na roh krychle kam chceme vztažnou soustavu přesunout, máme přesunuto. Nyní máme dvě možnosti jak krychli přesunout na požadované místo. První možností je manuální posunutí v potřebném směru pomocí potažením dané osy vztažné soustavy krychle. Druhou možností je pak zadání přesných absolutních souřadnic a krychle se nám přesune sama. Do nastavení absolutních souřadnic se dostaneme po kliknutí pravým tlačítkem myši na naši krychli a vybráním *Edit Position*. Poté se otevře nové okno kam souřadnice zadáme. Nastavení absolutních souřadnic v našem projektu je zobrazeno na obrázku 53.



Obrázek 53: Nastavení absolutních souřadnic krychle

5.7.2 Vytvoření nové soustavy (frame)

Aby robot mohl správně projet souřadnice bodů vyexportovaných z Inventoru, musíme si nastavit nový frame vůči kterému pak robot bude jednotlivé body projíždět. Než ale budeme moci vytvořit a nastavit nový frame, musíme si vytvořit naši VAL3 aplikaci a to tak, že na kartě *VAL3* vybereme *New Application*, kterou jsme pojmenovali jako `BP_Endrstova`.

Když tuto aplikaci budeme mít vytvořenou, zvolíme na stejné kartě *New Data* → *frame* a pojmenování ponecháme jako výchozí `fFrame`. Přes okno *Data* pak na tuto proměnnou klikneme pravým tlačítkem myši a vybereme *Select in 3D view*, čímž se nám náš frame označí v 3D prostoru. Stejným způsobem jako jsme měnili umístění referenční soustavy krychle, tak změníme i umístění tohoto frame. Tento frame musí být umístěn na stejném vrcholu krychle jako jsme nastavili UCS v Inventoru. Pokud jsme tedy v Inventoru nastavili počátek souřadnicového systému na spodní, leví, zadní vrchol, musíme na stejný vrchol zde umístit i náš frame. Tím zajistíme, že souřadnice děr v Inventoru a v SRS vzhledem k nově vytvořenému `fFrame` budou stejné.

5.7.3 Definování proměnných

Pro funkčnost vytvořeného kódu naší aplikací AItoSRS si musíme nastavit a definovat proměnné, které nalezneme ve vygenerovaném kódu.

Typ	**Název**	**Význam**	**Nastavení**
num	nTransit	uchování informace pro průjezdné body	NE
	nConeFor	(počet děr) - 1	NE
	nTransitFor	(počet průjezdných bodů) - 1	NE
mdesc	mFast	nastavení parametrů základního pohybu	NE
	mDesc	nastavení parametrů pohybu pro průjezdné body	NE
pointRx	pData	souřadnice všech bodů (díry i průjezdné body)	COPY
jointRx	jStart	startovní pozice (Home)	ANO
tool	tTool	nastavení nástroje	NE

Tabulka 2: Přehled proměnných nutných definovat

V posledním sloupci naší tabulky 2 můžeme vidět nastavení (NE, ANO, COPY). Proměnné s **NE** stačí definovat a můžeme ponechat výchozí nastavení, **ANO** musí uživatel nastavit sám a jedná se pouze o proměnnou `jStart`, kde si uživatel sám nastaví pozici odkud bude robot startovat a kde bude končit. Poslední typ je proměnná **COPY** a jedná se o `pData`. Do této proměnné musíme zkopírovat data z .csv souboru vyexportovaného z aplikace AItoSRS. Tuto proměnnou jako jedinou musíme mít vytvořenou pod naším vytvořeným `fFrame` nikoliv pod `world`. Toho docílíme tak, že nebudeme přidávat novou proměnnou přes kliknutí na hlavní aplikaci, ale klikneme pravým tlačítkem myši přímo na náš `fFrame` a zvolíme *New Data*.

Posledním důležitým nastavením je alokovat velikost pole neboli nastavit velikost jednotlivých proměnných. U všech proměnných typu *num* musíme nastavit velikost podle počtu použitých komponentů cone v Inventoru a u `pData` nastavíme tuto velikost dvojnásobnou. Pokud tedy v Inventoru použijeme 10 komponent cone, velikost proměnných typu *num* bude 10 a velikost `pData` bude 20. Tuto velikost nastavíme při vytváření proměnné zadáním hodnoty do okna *Size*.

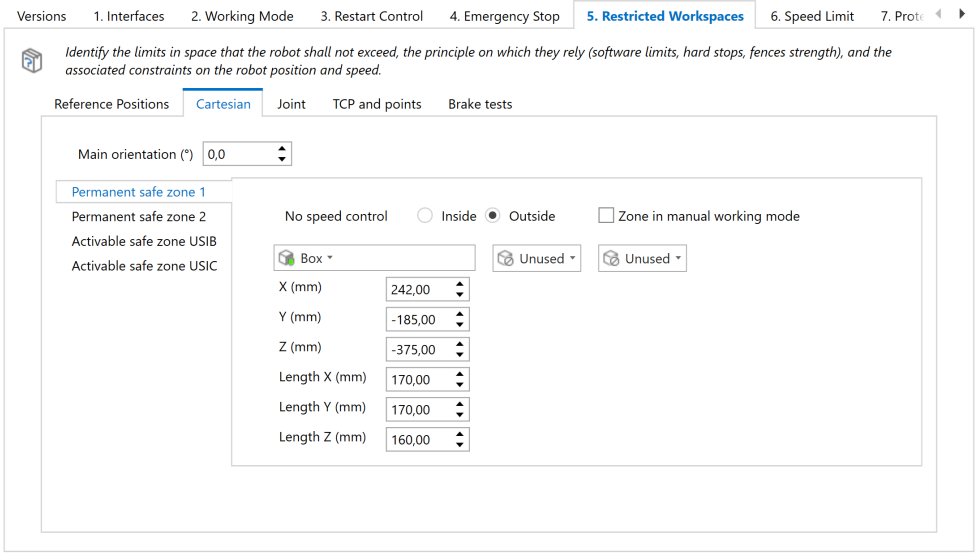
5.7.4 Nastavení bezpečné zóny

Jak již bylo zmíněno v bodě 5.2, tak můžeme nastavit bezpečnou zónu pro náš obrobek. Tedy zónu, do které když robot najede jakoukoliv svou částí, tak se ihned zastaví. Toto nastavení by bylo dobré například v případě, že by naši aplikaci mohli používat lidé, kteří nemají s programováním robotů žádnou zkušenost. Byli by pouze seznámeni s tím, jaká data kam zkopírovat a jak aplikaci do robota nahrát. Možná bezpečností pojistka, která zamezí kolizi s obrobkem v případě, že pověřený člověk nezkontroluje trajektorii předem pomocí simulace a program rovnou nahraje, může být právě nastavení této bezpečné zóny.

Bezpečná zóna se nastavuje v nastavení, na které se dostaneme přes kartu *Safety* → *Config* a zvolením daného ovladače. V nově otevřeném okně pak najdeme záložky 1-8, ale nás pro ukázku základního nastavení budou zajímat pouze záložky *5. Restricted Workspaces* a *6. Speed Limit*.

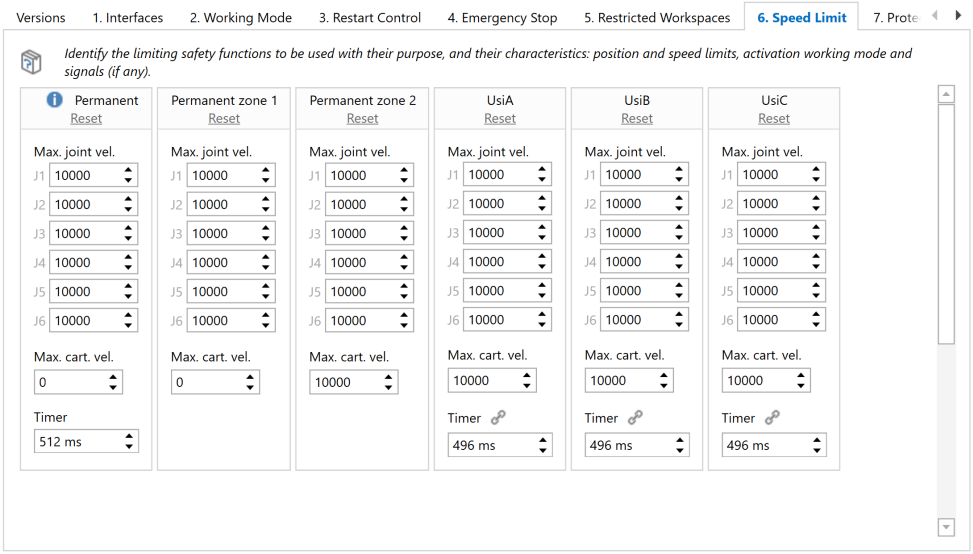
V záložce *Restricted Workspaces* → *Cartesian* si nastavíme tvar (box, cylinder, vertikální „zeď“), umístění a velikost. Při volbě bezpečné zóny pro daný obrobek je vhodné volit zónu o trochu větší, aby měl robot čas na zareagování. Kdybychom totiž nastavili zónu stejně velkou jako obrobek, robot by se sice zastavil ihned jak by do zóny najel, ale už by stihl i nabourat. Pro náš obrobek jsme tedy zvolili o 1cm větší krychli z každé strany než je naše krychle a výsledné nastavení můžeme vidět na obrázku 54.

Zobrazit bezpečnou zónu můžeme přes okno *Geometry*, rozbalením *Safety*→ *Permanent safe zone 1*, kliknutím pravým tlačítkem myši na naši zónu *Volume 1: Box* → *Show safe zones*. Bezpečná zóna kolem našeho obrobku je zobrazena na obrázku 56.

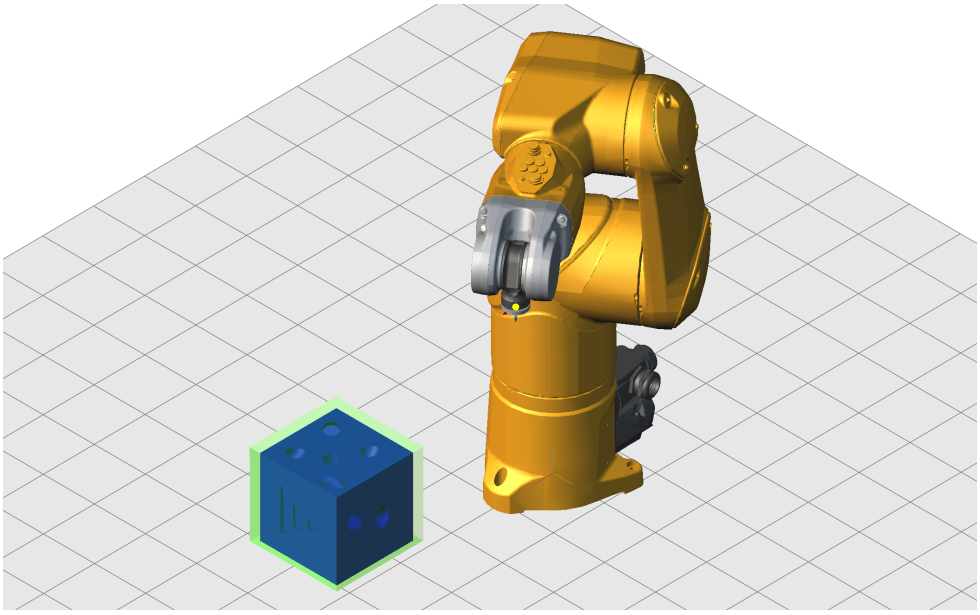


Obrázek 54: Nastavení umístění a velikosti bezpečné zóny

V druhé záložce *Speed Limit* pak nastavujeme co se má stát, když robot do bezpečné zóny najede. My chceme, aby se robot ihned zastavil, takže musíme nastavit pro naši zónu maximální kartézskou rychlost (maximum cartesian speed) na 0 mm/s. Závěrem bychom chtěli pouze říct, že nastavování takovéto bezpečné zóny lze pouze na reálném robotu a to přehráním jeho bezpečnosti, což při špatném nastavení může být problematické a i nebezpečné. Jelikož přehrání bezpečnosti robotu je poměrně složitý proces a je zde spousta návazností, není vhodné pro procesy, kde se tvar obrobku bude často měnit. Ideálním případem je vždy jeden tvar a velikost obrobku a například pouze jiné rozložení děr pro čištění.



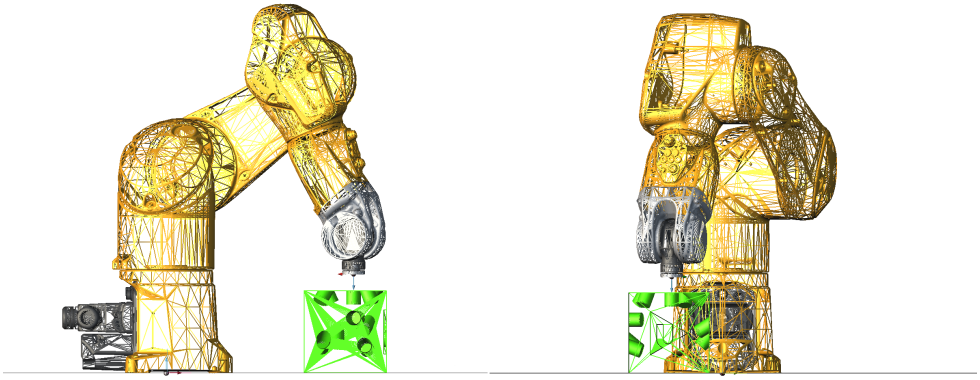
Obrázek 55: Nastavení rychlosti při najetí do bezpečné zóny



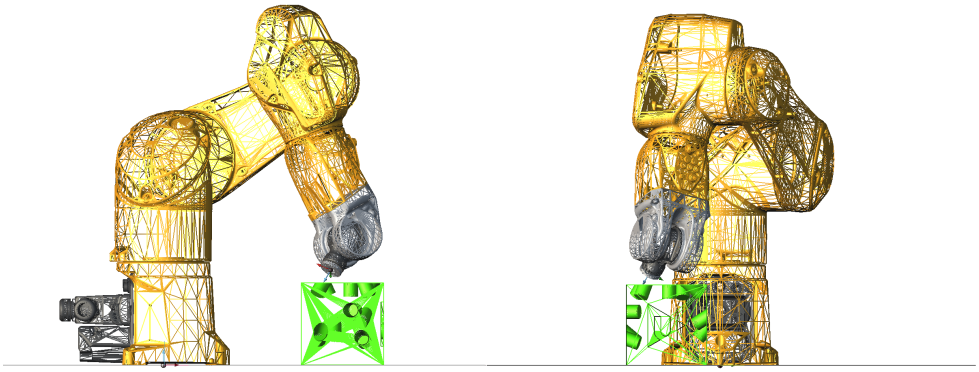
Obrázek 56: Bezpečná zóna kolem obrobku

5.8 Validace výpočtu

Důležitou částí naší bakalářské práce byla také simulace navrhnuté trajektorie s využitím vygenerovaného kódu a přepočtených dat získané z naší aplikace AItoSRS. Na základě této simulace můžeme tvrdit, že přepočtená data odpovídají přesnému úhlu a souřadnicím našich zkušebních děr. Tedy že máme koncový efektor natočený správným směrem a to ve směru osy dané díry. Zda máme při pohybu koncový efektor opravdu správně natočený jsme sledovali při simulaci navrhnuté trajektorie pomocí zobrazeného drátěného modelu naší krychle a robota. Správnost natočení a umístění pak dokládáme následujícími obrázky 57, 58, na kterých můžeme vidět koncový efektor najetý na pozice dvou děr a to vždy ve dvou směrech.



Obrázek 57: Dosažení správné pozice koncového efektoru - díra:1



Obrázek 58: Dosažení správné pozice koncového efektoru - díra:2

6 Závěr

Cílem této práce bylo navrhnout způsob využití aplikace Inventor pro naplánování trajektorie robota a tuto trajektorii následně simulovat pomocí prostředí SRS. V úvodu naší práce jsme si vysvětlili k čemu je dobré navrhnutí takového datového postprocesu pro programování robotů pomocí zadané trajektorie v CAD nástroji a to také na reálném příkladu pro lepší představu. V této části jsme se také seznámili s historií průmyslových robotů a automatizace, a to především z toho důvodu, abychom měli základní přehled o postupném vývoji těchto odvětví a co jednotlivé pojmy znamenají a popisují. Na základě toho můžeme o našem řešení říci, že se jedná také o jistý druh automatizace a to v tom smyslu, že ulehčujeme práci a snižujeme potřebné množství lidí, kteří se na této úloze musí podílet.

V druhé a třetí části jsme se seznámili s prvním programem, který k práci využíváme a jedná se o aplikaci Inventor. V této aplikaci jsme si navrhli naši zkušební krychli pro plánování trajektorie, způsob zakreslení této trajektorie pomocí námi definovaných komponent a také jsme navrhli makro pro export zájmových bodů z této aplikace.

Následně jsme se seznámili s námi vytvořenou aplikací AItoSRS, která slouží pro přepočet dat zvolené trajektorie v Inventoru a generuje nám kód pro robota využívající znalosti dat získaných z Inventoru. S aplikací jsme se seznámili jak z pohledu uživatelského, tak jsme si popsali i algoritmus kódu této aplikace, vysvětlili jsme si funkčnost generovaného kódu a popsali jsme si způsob jak využíváme exportovaná data z Inventoru k zjištění směrového vektoru jednotlivých děr krychle a následnému přepočítání tohoto směrového vektoru na úhly postupné rotace, které potřebujeme znát pro plánování pohybu robotu.

V poslední části jsme se seznámili se simulačním prostředím SRS, s jeho funkcemi a možnostmi zobrazení jednotlivých věcí potřebných pro simulování pohybu. Vysvětlili jsme si také prvotní nastavení projektu, datové typy, možnosti konfigurace jednotlivých kloubů a také jaký rozdíl je mezi pohybovými příkazy movej a movel. Stejně tak jsme popsali i používání simulačního ovladače a jak pomocí něho spustíme vytvořenou aplikaci. Na konci tohoto bodu jsme se zabývali konkrétní konfigurací projektu naší práce a popsali jsme například podmínky správně umístěného obrobku, teoretické nastavení bezpečné zóny a vytvořili seznam nutně definovaných proměnných v aplikaci SRS, aby generovaný kód aplikací AItoSRS byl v aplikaci SRS spustitelný.

Funkčnost přepočtu směrového vektoru na úhly postupných rotací jsme dokázali v bodě 5.8, kde správné natočení koncového efektoru demonstrujeme pomocí vizualizace pozice robotu nad danou dírou zkušební krychle.

Další možností vylepšení této práce by mohla být například možnost nastavení všech důležitých parametrů pohybu robota přímo v aplikaci AItoSRS v možné sekci pokročilé nastavení, kde bychom mohli například měnit hodnoty aproximací bodů, velikost nástroje, rychlost pohybu atd.. Druhou možností by pak bylo navrhnutí automatického projetí zadaných zájmových bodů podle jejich nejbližší vzdálenosti, protože momentálně musí být pořadí projetí těchto zájmových bodů zadáno uživatelem.

Reference

[1] SPŠ a VOŠ Chomutov. *Automatizace 1*, chapter 1.1, page 4. SPŠ a VOŠ Chomutov.

[2] Artefacts. Heron of alexandria – automated temple doors. https://www.artefacts-berlin.de/portfolio-item/heron-of-alexandira-automated-temple-doors/, 2021.

[3] AUTODESK. Hlavní funkce aplikace inventor. https://www.autodesk.cz/products/inventor/features.

[4] Technický deník. Hérón alexandrijský. https://www.technickytydenik.cz/rubriky/serialy/inspirujici-inovatori/heron-alexandrijsky-10-70-n-l_39639.html, 2017.

[5] Mikell P. Groover. Automation. https://www.britannica.com/technology/automation.

[6] Ing. Dobromila Lebrová. James watt, britský vynálezce. https://www.pozitivni-noviny.cz/cz/clanek-2009080048, 2009.

[7] Jeremy M. Norman. George devol invents unimate, the first industrial robot. https://www.historyofinformation.com/detail.php?id=3616, 2017.

[8] Robotnik. History of robots and robotics. https://robotnik.eu/history-of-robots-and-robotics/, 2021.

[9] Karlík Tomáš. Před sto lety svět poprvé uslyšel slovo „robot“. https://ct24.ceskatelevize.cz/veda/3260066, 2021.

[10] Bantam Tools. History of cnc machining. https://medium.com/cnc-life/history-of-cnc-machining-part-1-2a4b290d994d, 2019.

[11] S.G. Tzafestas. *Introduction to Mobile Robot Control*. Elsevier, 2014.