VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

NÁVRH TRAJEKTORIE KONCOVÉHO BODU ROBOTICKÉHO RAMENE METODOU VIRTUÁLNÍCH BODŮ

DESIGN OF THE ENDPOINT TRAJECTORY OF THE ROBOTIC ARM USING THE VIRTUAL POINT METHOD

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Ľubomír Bubeník

VEDOUCÍ PRÁCE
SUPERVISOR

prof. Ing. Petr Pivoňka, CSc.

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Kybernetika, automatizace a měření**
Ústav automatizace a měřicí techniky

Student: Bc. Ľubomír Bubeník ***ID:*** 164840
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Návrh trajektorie koncového bodu robotického ramene metodou virtuálních bodů

POKYNY PRO VYPRACOVÁNÍ:

1) Zpracujte rešerši týkající se možností vytváření trajektorií pohybu ramen robotů.
2) Navrhněte webovou aplikaci schopnou dynamicky vytvářet trajektorii koncového bodu robota.
3) Navrhněte a realizujte virtuální model robotického ramene ve webové aplikaci
4) Doplňte webovou aplikaci robotického ramene požadovanými virtuálními body
5) Implementujte komunikační protokoly pro přenos dat.
6) Webovou aplikaci implementujte do prostředí pro widget v Automation studiu.

DOPORUČENÁ LITERATURA:

[1] Firemní literatura B&R: Automation Studio V4

[2] Firemní literatura B&R: Produkty pro řízení pohybu a vizualizace společnosti B&R

Termín zadání: 4.2.2019 ***Termín odevzdání:*** 13.5.2019

Vedoucí práce: prof. Ing. Petr Pivoňka, CSc.
Konzultant: Ing. Roman Mužík B+R

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

ABSTRAKT

Diplomová práca sa zaoberá novým prístupom k vytvoreniu trajektórie robotického ramena. Prvá časť popisuje existujúce metódy plánovania trajektórie. Druhá časť sa zaoberá metódou virtuálnych bodov, implementáciou plánovania trajektórie do webového prostredia tvorbou trajektórie. Tretia časť popisuje virtuálny model robota a navrhnuté komunikačné protokoly. Štvrtá časť obsahuje grafické užívateľské rozhranie a popisuje jeho možnosti. V poslednej časti je prezentovaná implementácia webovej aplikácie do priemyselnej vizualizácie.

KĽÚČOVÉ SLOVÁ

Robotické rameno, plánovanie trajektórie, HTML5, CSS3, HMI, REST, OPC UA

ABSTRACT

The diploma thesis deals with a new approach to create a robotic arm trajectory. The first part descibes existing methods of trajectory planning. The second part shows virtual point method, implementation of the trajectory planning into the web environment and trajectory generation. Third part descibes the virtual robot model and designed comunnication protocols. The fourth part shows grafical user interface and his posibilities. The last part presents implemention of web aplication into industrial visualization.

KEYWORDS

Robotic arm, trajectory planning, HTML5, CSS3, HMI, REST, OPC UA

BUBENÍK, Ľubomír. *Návrh trajektorie koncového bodu robotického ramene metodou virtuálních bodů*. Brno, 2019, 72 s. Diplomová práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedúci práce: prof. Ing. Petr Pivoňka, CSc.

VYHLÁSENIE

Vyhlasujem, že som svoju diplomovú prácu na tému „Návrh trajektorie koncového bodu robotického ramene metodou virtuálních bodů“ vypracoval samostatne pod vedením vedúceho diplomovej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákoníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora

POĎAKOVANIE

V prvom by som chcel poďakovať vedúcemu diplomovej práce prof. Ing. Petrovi Pivoňkovi, CSc. za odborné vedenie a konzultácie k práci. Ďalej by som rád poďakoval Ing. Tomášovi Matuchovi, Ph.D. z firmy B&R a Tomášovi Brojírovi z firmy ABB za odborné rady. Nakoniec by som chcel poďakovať Ing. Romanovi Parákovi z ústavu automatizace a informatiky na FSI za ochotu a odborné rady pri testovaní.

Brno ....

podpis autora

Zoznam symbolov, veličín a skratiek

3D	Trojdimenzionálny priestor
ABB	Asea Brown Boveri
ANSI	American National Standards Institute
API	Application Programming Interface
B&R	Bernecker&Rainer
CAD	computer-aided design
CFC	Continous Function Chart
cm	centimeter
CSS3	Cascading Style Sheets 3
DOM	Document Object Model
FBD	Function Block Diagram
fov	Field of view
GNU	General Public License
GUI	Graphical User Interface
HTML5	Hypertext Markup Language 5
IE	Internet Explorer
IP	Internet Protocol
JS	JavaScript
JSON	JavaScript Object Notation
m	meter
MIT	Massachusettský technologický institut
ms	milisekunda
NURBS	non-uniform rational B-spline function
OPC	Ole for Process Control
OPC UA	Ole for Process Contro Unified Architecturel
PLC	Programmable Logic Controller
REST	Representational state transfer
rw	RobotWare
SFC	Sequential Function Chart
ST	Structured Text
stl	stereolithography
TCP	Transmission Control Protocol
UI	User Interface
UUID	Universally unique identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language

Zoznam obrázkov

Obsah

Úvod

Takmer v každom odvetví je snaha preniesť obraz skutočnej reality do počítačovej grafiky. Ale dôvody pre tento fenomén sa líšia. Napríklad, v stavebníctve sú modelované celé budovy, zatiaľ čo v zdravotníctve najmenšie detaily ľudského tela. Tento trend sa nevyhýba ani priemyselnej automatizácii, kde je hlavnou požiadavkou simulácia reálnej scény. Dôvody sú hlavne z ekonomického a časového hľadiska.

V praxi často dochádza k situácii, kedy vývoj a testovanie softvéru je možný až po dokončení mechanickej časti stroja. Preto existujú spoľahlivé nástroje rôznych výrobcov, pomocou ktorých sa dá tvoriť trajektória robotického ramena aj bez fyzického zariadenia. Tieto prostredia sú perfektne optimalizované a dokonca ponúkajú aj možnosti virtuálnej reality alebo digitálnych dvojíc. Nevýhodou je, že všetky vývojové prostredia je nutné inštalovať. To znamená, že software je často nekompatibilný s operačným systémom. To znamená, že užívateľ je obmedzený na úzke spektrum zariadení, na ktorých môže pracovať. Tento problém dokáže vyriešiť prostredie, ktoré využíva webové rozhranie. Veľká väčšina počítačov, ale aj prenosných zariadení ako sú tablety a telefóny, túto technológiu hravo zvláda. Ale žiadny z komerčných poskytovateľov takúto možnosť neponúka.

Samozrejmosťou dnešného moderného vývojového prostredia pre plánovanie trajektórie, je prepracovaná 3D grafika. Táto práca sa teda bude zaoberať najmä počítačovou grafikou v 3D priestore a vývojom užívateľského prostredia pre tvorbu trajektórie koncového bodu robotického ramena.

1 Prehľad dostupných riešení pre tvorbu trajektórie koncového bodu robota

Vo svete robotiky existuje veľké množstvo programovacích prostredí a editorov, pomocou ktorých je možné vytvoriť trajektóriu pre robotické rameno. Spravidla každý väčší výrobca robotických ramien má vo svojej ponuke zahrnutý aj software pre tvorbu trajektórie pre vlastné roboty. Jednotlivé prostredia sú veľmi odlišné, ale spája ich jedna vlastnosť. Každý komerčne poskytovaný software je nutné inštalovať na konkrétnom operačnom systéme, čo je nežiaduca vlastnosť. A preto bolo pre popis zvolené iba jedno komerčne používané programovacie prostredie. Týmto prostredím je *RobotStudio* od firmy ABB. Toto programovacie prostredie bolo zvolené z dôvodu dostupnosti robotických ramien od firmy ABB. Ďalej bude prieskum zameraný najmä na experimentálne aplikácie, pomocou ktorých je možné tvoriť trajektóriu z webového prostredia.

1.1 RobotStudio

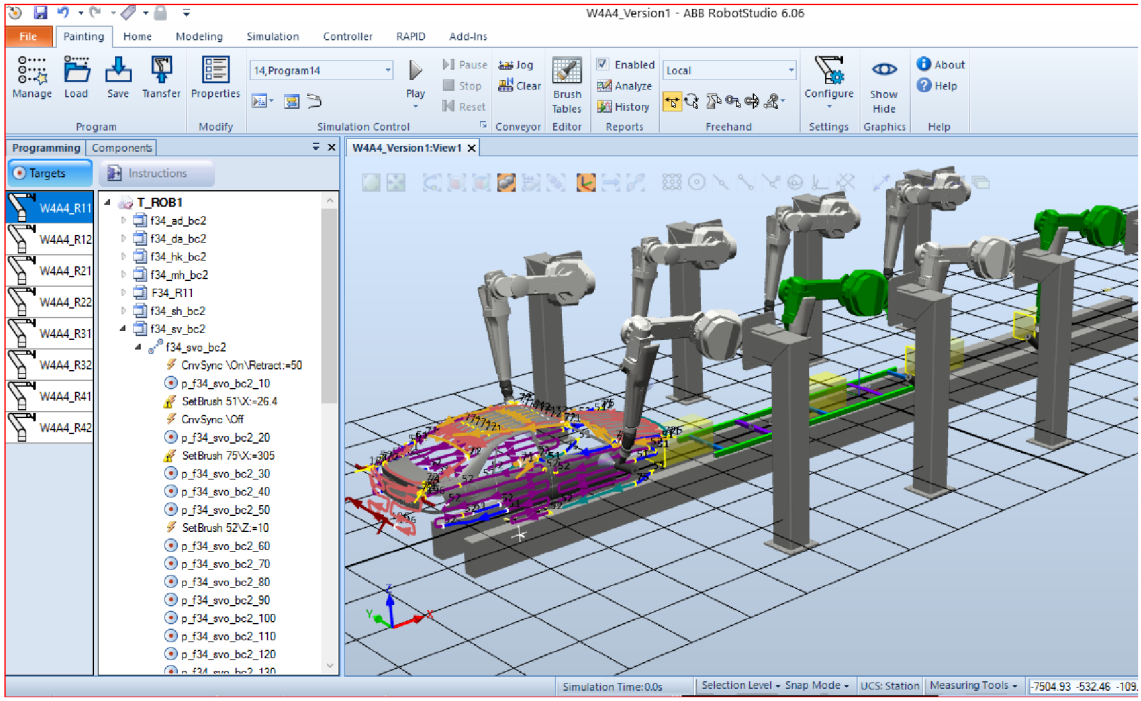
RobotStudio je komplexný nástroj, ktorý slúži primárne na tvorbu programov pre priemyselné roboty. Pomocou RobotStudia je možné modelovať, programovať bez fyzického kontroléru, simulovať rôzne procesy a mnoho ďalších možností. Pri programovaní bez fyzického kontroléru je použitý virtuálny kontrolér. Vizuálna stránka RobotStudia vychádza z prostredia MS Office. Hlavná ponuka je rozdelená do nasledujúcich častí:

- Home - Pomocou tejto ponuky je možné vytvoriť nový projekt, vytvárať trajektórie, upravovať nastavenia, ovládanie kamery a scény.
- Modeling - Ponuka je zameraná na jednoduchú prácu s CAD modelmi, ktoré je možné do prostredia importovať a ďalej s nimi pracovať.
- Simulation - V tejto časti je možné ovládať simulácie, získavať signály, nahrať priebehy a podobne.
- Controller - Slúži pre ovládanie kontroléru, jeho nastavenie, správu užívateľov, safety, zálohovanie a tak ďalej.
- RAPID - Ponúka možnosť programovania robotického ramena textovou formou pomocou pohybových inštrukcií. Ďalej je možné programy načítať, editovať alebo organizovať. Programy je následne možné testovať na simulácii alebo na reálnom zariadení.
- Add-ins - V karte je možné nájsť ďalšie dodatkové programy.

Kontrolér robotického ramena ponúka nasledujúce možnosti komunikácie:

- OPC
- TCP/IP
- REST
- WebSocket

Na obrázku 1.1 je zobrazený príklad programovacieho prostredia RobotStudio. [1][2]



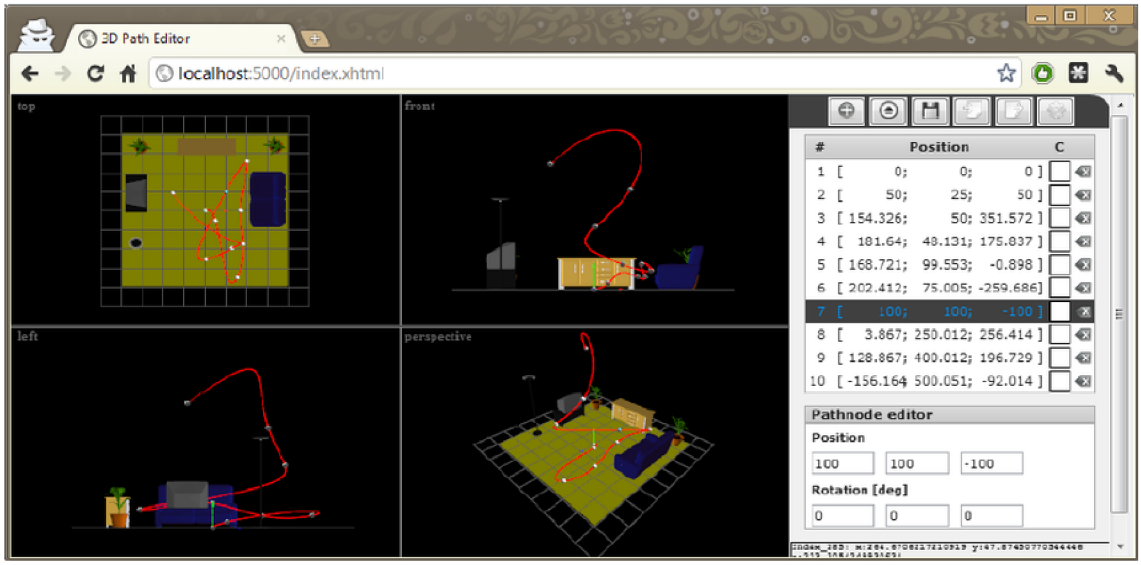
Obr. 1.1: Grafické prostredie RobotStudio od firmy ABB [3]

1.2 Experimentálne aplikácie

1.2.1 Webová aplikácia s možnosťou plánovania trajektórie

Prvá webová aplikácia, ktorá bola zaradená do časti experimentálnych, pochádza z roku 2012. Pochádza z publikácie, ktorá bola rozdelená do troch hlavných celkov. Prvá časť je zameraná na celé riadenie motorov. K riadeniu boli použité PLCopen funkčné bloky, pomocou ktorých je riadená rýchlosť a zrýchlenie jednotlivých ôs. Druhá časť popisuje 3D editor pre plánovanie trajektórie, ktorý je založený na webových technológiách. Pre tvorbu 3D priestoru bolo použité nízko-úrovňové API WebGL a vysoko-úrovňové API Three.js. Ako je vidieť na obrázku 1.2 3D zobrazenie je rozdelené do štyroch častí, kde je podľa publikácie možné každý pohľad ovládať

nezávisle. Vo webovej aplikácii je ďalej možné pridávať, editovať alebo odstrániť časť, ktorá tvorí trajektóriu. Na pravej strane aplikácie sú v zozname zobrazené súradnice jednotlivých častí, pomocou ktorých je vytvorená trajektória. Podľa publikácie je možné do scény načítať objekty generované CAD systémom, poprípade 3D editorom. Ďalej bola vytvorená aplikácia v jazyku C++, ktorá vytvára webový server a zároveň vykonáva všetky potrebné výpočty. Webová aplikácia prenáša všetky potrebné dáta vo formáte JSON pomocou HTTP protokolu. Dáta sú prenášané pomocou HTTP požiadaviek alebo pomocou WebSocketu. Súčasťou 2. časti je aj popis preloženia časti trajektórie pomocou krivky NURBS. Posledná časť je zameraná na popis implementácie na reálneho lanového robota. [4]



Obr. 1.2: Ukážka vizualizácie plánovania trajektórie robota [4]

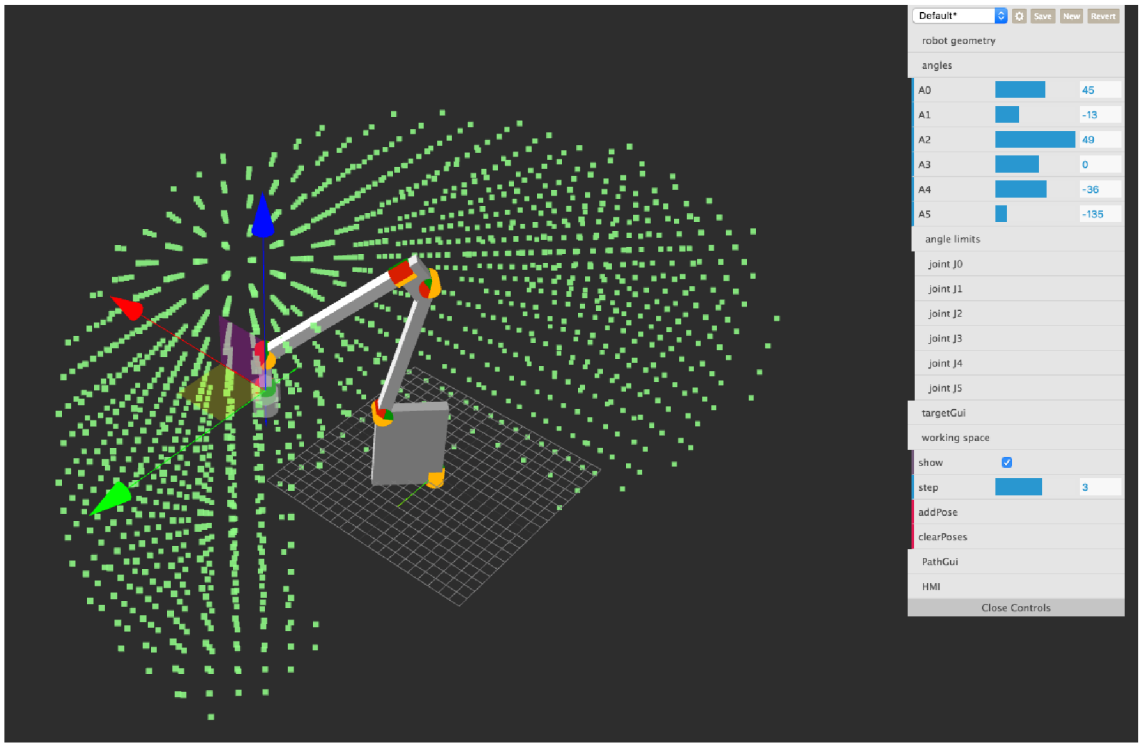
1.2.2 Webová aplikácia s implementovanou vlastnou kinematikou robotického ramena

Ďalšia webová aplikácia je zameraná iba na samotné robotické rameno. Úlohou webovej aplikácie je prezentovať vlastnú inverznú kinematiku. Aplikácia využíva nízkoúrovňové API WebGL a vysoko-úrovňové API Three.js. Ako je vidieť na obrázku 1.3, webová aplikácia podporuje uchopenie koncového bodu robotického ramena a jeho pohybom sa mení uhol ôs. Pomocou panelu, ktorý sa nachádza vpravo hore na obrázku 1.3, je možné meniť takmer každý podstatný parameter konfigurácie robota. Ponuka zahŕňa:

- Nastavenie geometrie robota
- Voliteľný uhol každej osi

- Nastavenie maximálneho a minimálneho limitu každej osi
- Predvolené konfigurácie robotického ramena
- Presun a rotáciu koncového bodu robotického ramena

Na obrázku sú ďalej zobrazené body, ktoré reprezentujú maximálny rozsah robota. Pokiaľ je tento limit prekročený, časť robota sa rozpojí. [5]



Obr. 1.3: Projekt s funkčnou vlastnou kinematikou robotického ramena [5]

Ponuka robotického ramena je pripravená aj na implementáciu plánovania trajektórie, ktorá je zatiaľ nefunkčná.

2 Prehľad dostupných aplikačných rozhraní pre prácu s 3D animáciami vo webovom prostredí

Aplikačné rozhranie, častejšie nazývané *API*, je zbierkou procedúr, funkcií, tried alebo protokolov, ktoré môže programátor využívať [6]. Prvé *API* bolo použité už v roku 1977 firmou ACM v súvislosti s herným priemyslom [7]. V dnešnej dobe nájdeme API s rôznym špecifickým zameraním. Aplikačné rozhrania sa delia na dve hlavné skupiny:

- Nízko-úrovňové API
- Vysoko-úrovňové API

Ďalšia časť práce bude venovaná rozboru a výberu rozhrania.

2.1 Nízko-úrovňové API

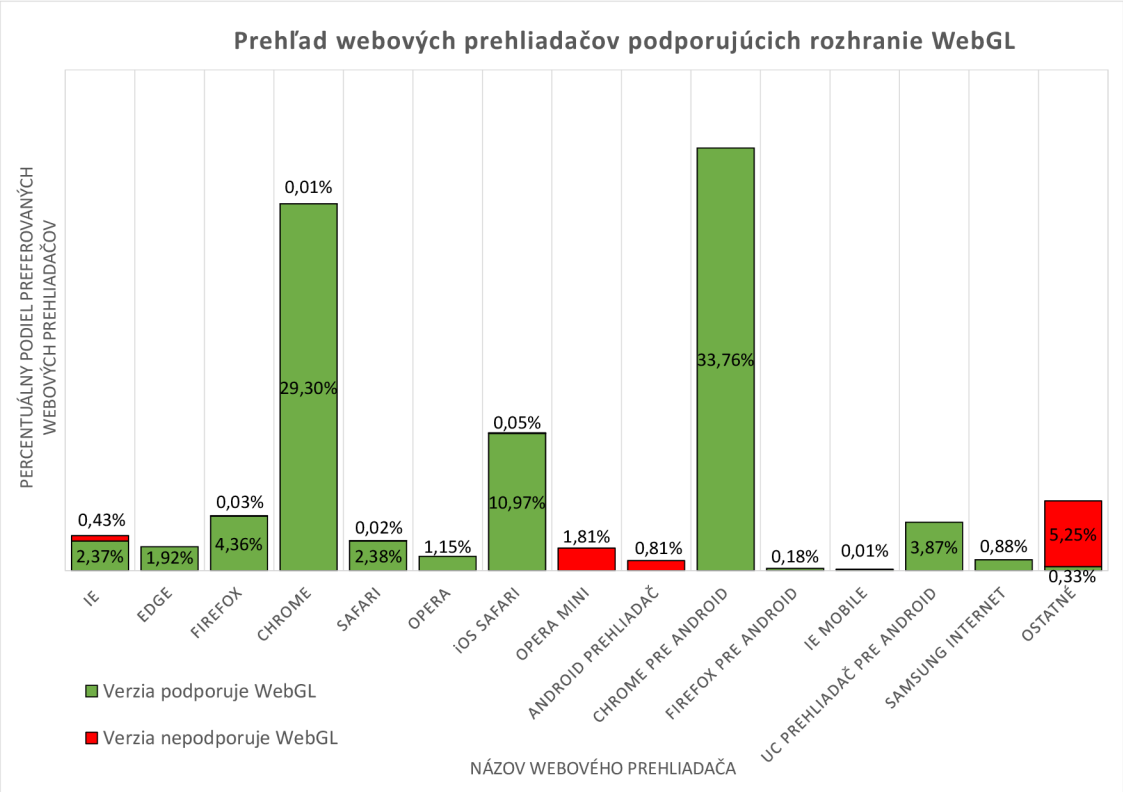
Jedná sa o základ každého vysoko-úrovňového API. Nízko-úrovňové API umožňuje programátorom prístup priamo k hardwardovým zdrojom za cenu zdĺhavého vývoja. Platí, že nízko-úrovňové API sa využíva v prípade efektívnejšieho využitia zdrojov. Na druhú stranu API s vyššou úrovňou bývajú v drvivej väčšine optimalizované vzhľadom na danú oblasť. [8]

Požiadavkou na užívateľskú aplikáciu je, aby bola dostupná vo webovom rozhraní a aby podporovala renderovanie 3D grafiky. Túto podmienku spĺňa iba API s názvom *WebGL*.

2.1.1 Aplikačné rozhranie WebGL

Názov WebGL je pojmom pri tvorbe počítačovej grafiky vo webovom prostredí. WebGL si získal svoju popularitu vďaka podpore 3D grafiky vo webovom prostredí bez nutnosti inštalovať prídavné moduly. WebGL, ktorý je dnes už vo verzii 2.0, využíva prvok zo značkovacieho jazyka HTML5 s názvom *Canvas*. Canvas využíva DOM objektový model, vďaka ktorému je možné k prvku pristupovať ako k súčasti funkčného kódu. API je plne podporovaná takmer všetkými webovými prehliadačmi, ktoré sú preferované užívateľmi. Graf zobrazený na obrázku 2.1 zobrazuje v percentách celosvetovú popularitu jednotlivých webových prehliadačov. Z dôvodu veľkého množstva rôznych typov prehliadačov sú zobrazené iba tie, ktoré sú využívané viac ako 0,01 % užívateľmi z celkového počtu, zvyšné sú v stĺpci "Ostatné". V grafe sú zelenou farbou zvýraznené verzie s podporou WebGL a červenou farbou verzie,

ktoré rozhranie nepodporujú. Z grafu vyplýva, že viac ako 90 % preferovaných verzií webových prehliadačov podporuje API WebGL.[16]



Obr. 2.1: Podpora rozhrania WebGL rôznymi webovými prehliadačmi [16]

Veľkou výhodou je, že časť kódu WebGL využíva *shader*, to znamená, že API dokáže využívať grafickú kartu zariadenia. O správu pamäti sa stará časť písaná v jazyku *JavaScript*. Ďalšou výhodou je, že WebGL je poskytovaná pod voľnou licenciou. [9]

2.2 Vysoko-úrovňové API

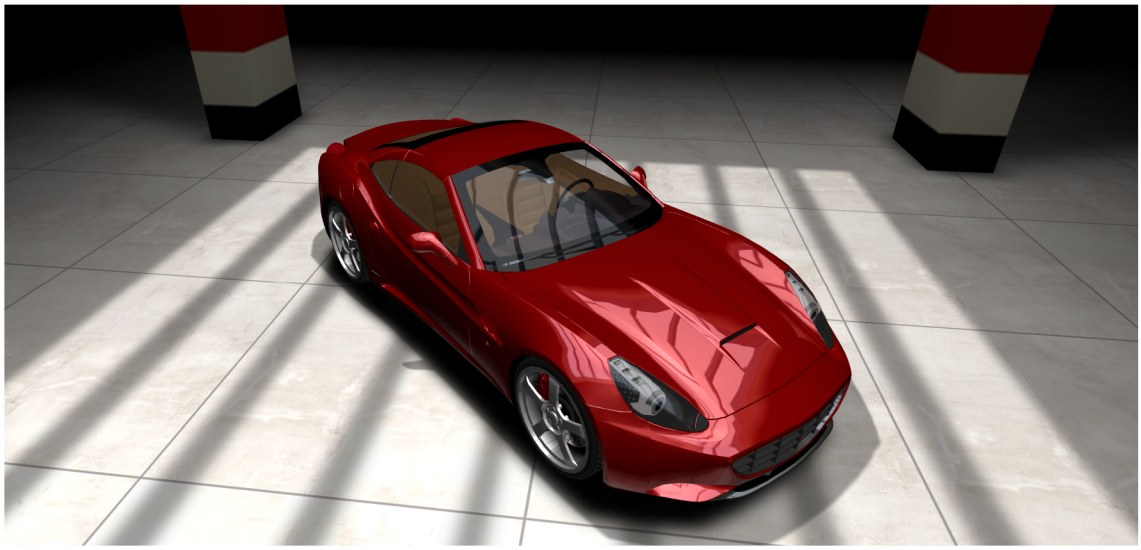
Vyššia úroveň API ponúka balík rozšírených možností, ktoré urýchľujú vývoj aplikácie. Výber aplikačného rozhrania, na ktorom bude vyvíjaná webová aplikácia, je ovplyvnený požiadavkou na podporu práce v 3D priestore. API musí ďalej podporovať sadu nástrojov, ktoré sa svojou funkciou blížia k vlastnostiam CAD systémov a zároveň API musí efektívne využívať výpočtové zdroje zariadenia, aby bola práca s prostredím pre užívateľa plynulá a nároky na hardware čo najnižšie.

2.2.1 Aplikačné rozhranie Three.js

API Three.js využíva skriptovací jazyk *JavaScript*. Rozhranie bolo prvýkrát zverejnené v roku 2010. Toto vysoko-úrovňové aplikačné rozhranie patrí medzi najväčšie a najrozšírenejšie v oblasti 3D grafiky na úrovni webovej aplikácie. Rozhranie ponúka možnosti ako napríklad:

- Práca s kamerou
- Tvorba scén
- Dynamické pridávanie a odstraňovanie objektov
- Dopredaná kinematika, inverzná kinematika
- Pomôcky pre orientovanie sa v priestore
- Rôzne materiály a textúry
- Tvorba kriviek a splajnov
- Načítanie rôznych druhov objektov
- Podpora reálneho času
- Podpora virtuálnej reality

Výhodou Three.js je veľká podpora zo strany komunity. Na domovskej stránke sa nachádza prepracovaná dokumentácia a veľké množstvo príkladov ako API použiť. [11] Zdrojový kód je voľne dostupný a pod licenciou MIT. [10]



Obr. 2.2: Príklad použitia API Three.js [12]

2.2.2 Aplikačné rozhranie Babylon.js

Babylon je ďalšie rozhranie, ktoré využíva nízko-úrovňové API WebGL. Babylon bol prezentovaný v roku 2015, teda dá sa označiť za novinku. Svojim princípom

pripomína API Three.js. Babylon taktiež využíva prvok HTML5 *Canvas*. Ďalšou výhodou je modularita, vďaka ktorej je možné načítať pluginy tretích strán. Jedným z takýchto pluginov je napríklad *Canon.js*, vďaka ktorému je možné simulovať reálnu fyziku a v konečnom dôsledku je možné vytvoriť digitálne dvojča. Babylon ponúka možnosti:

- Práca s kamerou
- Tvorba scén
- Dynamické pridávanie a odstraňovanie objektov
- Rôzne materiály a textúry
- Tvorba kriviek a splajnov
- Podpora reálneho času
- Podpora pre reálnu fyziku
- Podpora realistickej grafiky
- Podpora virtuálnej reality

Domovská stránka ponúka širokú škálu príkladov použitia. Zdrojový kód je voľne dostupný a je voľne šíriteľný pod *Apache* licenciou [13].

Obr. 2.3: Príklad použitia realistickej grafiky pomocou Babylon.js API [14]

2.3 Výber aplikačného rozhrania

Ako nízko-úrovňové API bude použité jediné rozhranie, ktoré spĺňa kritériá spomenuté v kapitole 2.1 s názvom WebGL. Pre potreby tvorby používateľského prostredia je výhodnejšie použiť API vyššej úrovne, poskytne to rýchlejší vývoj aplikácie. Nespornou výhodou je, že nie je nutné znovu vyvíjať funkcie podporujúce prácu s

grafikou a 3D priestorom. Ako vysoko-úrovňové API bolo zvolené rozhranie Three.js. Toto API poskytuje širokú škálu funkcií zameraných na kinematiku. Ďalšou užitočnou výhodou sú pomôcky, vďaka ktorým sa dá orientovať v priestore, čo môže byť pre užívateľa výhodné. Rozhranie je od svojho vydania preverené užívateľmi a dá sa považovať za odladené.

3 Návrh webovej aplikácie pre tvorbu trajektórie koncového bodu robotického ramena

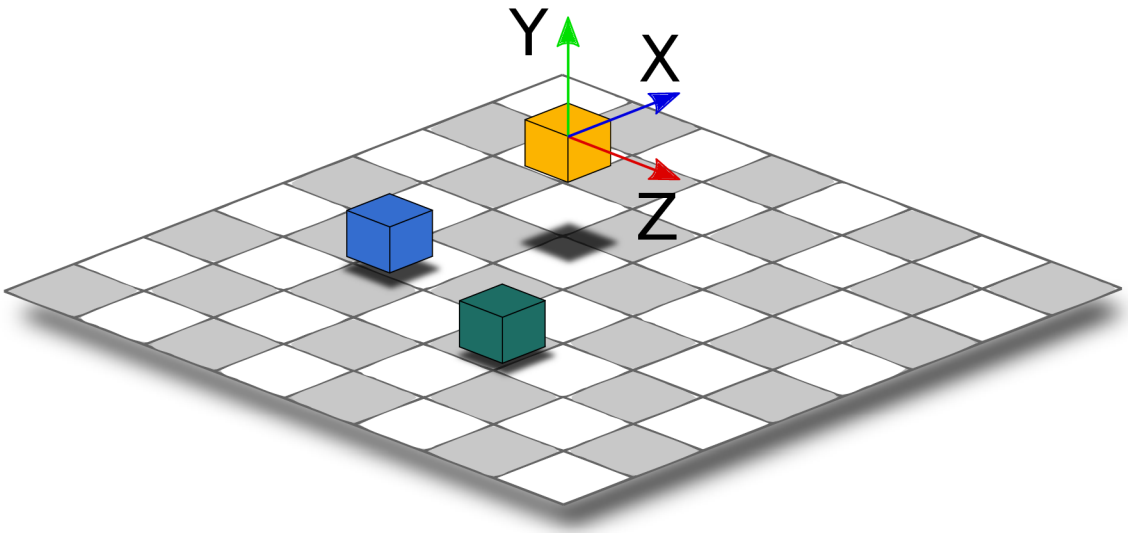
Webové prostredie vyžaduje, aby bolo užívateľsky intuitívne a zároveň škálovateľné pre čo najširšie možnosti použitia. Z programového hľadiska musí byť zdrojový kód prostredia navrhnutý s ohľadom na modularitu, aby bolo možné jednoducho dopĺňať alebo meniť funkcionality. Podstatnou požiadavkou je, aby bola webová aplikácia optimalizovaná na výkon a kompatibilná s rôznymi zariadeniami a prehliadačmi.

3.1 Základné vlastnosti objektov rozhrania THREE.js

Pred samotným popisom častí, ktoré tvoria grafickú časť webovej aplikácie, je dôležité uviesť základné vlastnosti objektov. Objekty sú stavebnými prvkami celej vizualizácie webovej aplikácie. Príklad objektov je zobrazený na obrázku 3.1. Na obrázku sa nachádzajú štyri objekty, z toho jedným je základná plocha a zvyšné sú kocky s rozličnou pozíciou a rôznymi farbami. Aplikačné rozhranie THREE.js ponúka veľké množstvo objektov s rozličnými vlastnosťami a metódami, ktorým sa budú venovať ďalšie kapitoly.

3.1.1 Súradnicový systém

Priestor v scéne je popísaný v kartézskom súradnicovom systéme. Kartézsky súradnicový systém tvoria 3 navzájom kolmé osy [X ;Y ;Z]. Pozícia objektov je potom daná relatívnou vzdialenosťou od stredu, ktorý má súradnice [0; 0; 0]. Pozícia každej súradnice môže nadobúdať kladné aj záporné hodnoty. K štandardnému popisu pozície v priestore sa v rozhraní THREE.js používa trieda *Vector3*. Triedou sa ďalej často popisuje vektor v 3D priestore a vzdialenosť dvoch objektov. Trieda tiež obsahuje veľké množstvo metód určených pre prácu s vektormi v 3D priestore. Na obrázku 3.1 je názorne zobrazený smer ôs v kartézskom súradnicovom systéme, ich názov a farebné označenie, ktoré bude použité v scéne. [11]

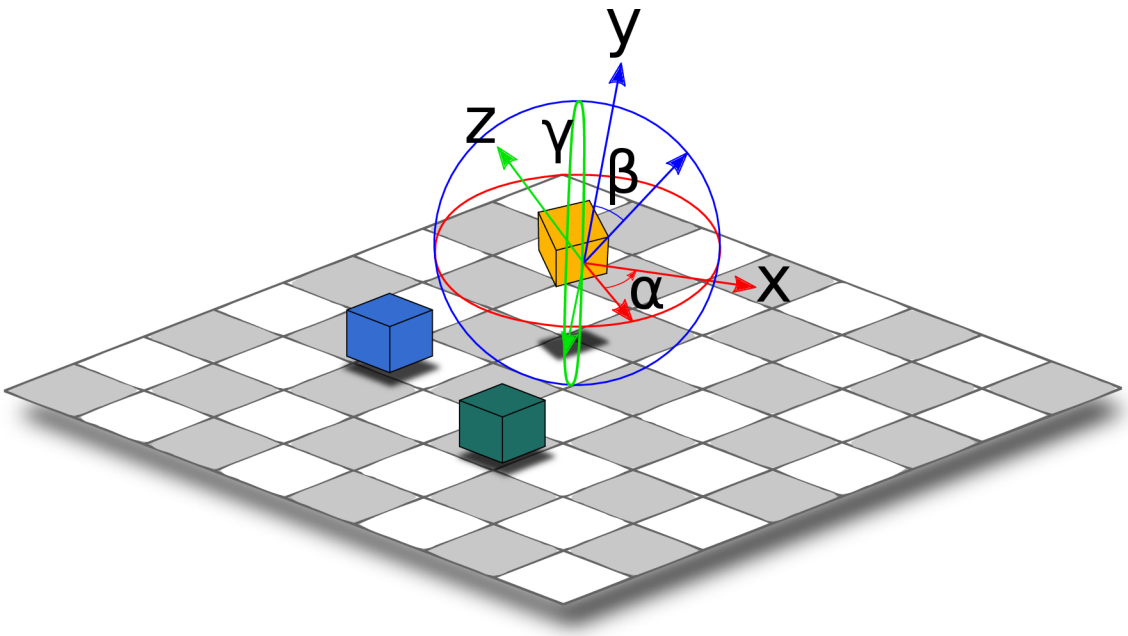


Obr. 3.1: Smer ôs v kartézskom súradnicovom systéme

Pozíciu objektu v priestore je možné rozdeliť na lokálnu a globálnu. Lokálna poloha je relatívna, vždy k objektu, ktorý mu je predkom. To znamená, že ak sú dva objekty previazané spôsobom rodič-potomok, tak súradnice počiatku [0; 0; 0] sa pre potomka nachádzajú v počiatku rodiča. Na rozdiel od lokálnej pozície globálna pozícia je absolútna s počiatkom v strede scény. Príkladom objektu, ktorý pracuje s globálnou pozíciou je napríklad scéna, ktorá je popísaná v kapitole 3.2.1. [11]

3.1.2 Rotácia v priestore

V práci je používaná primárne reprezentácia v eulerových uhloch. Rozklad rotácie je možné rozložiť do troch vzájomne kolmých ôs s označením X, Y, Z. Potom môžeme ako príklad zvoliť os Y, okolo ktorej chceme, aby zvyšné osy rotovali. Rotáciou vzniknú osi X', Y', Z', ktorých rozdiel od pôvodných ôs predstavuje uhol rotácie s názvom α, β a γ. Uhol rotácie sa pohybuje v rozsahu <0;2π> teda <0;360> stupňov. Na obrázku 3.2 je zobrazený smer jednotlivý ôs, ich názov a farebné označenie, ktoré je použité v scéne.



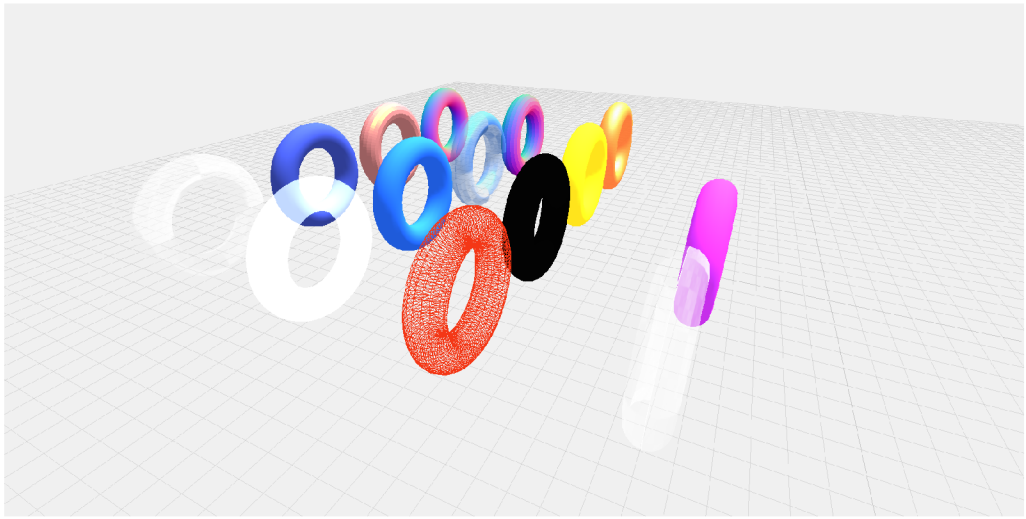
Obr. 3.2: Spôsob rotácie v priestore

3.1.3 Vlastnosti materiálov

Takmer každý objekt musí mať pri vytváraní svojej inštancie triedy definovaný materiál. Materiál ovplyvňuje vizuálne vlastnosti objektov. Na fyzikálne vlastnosti objektov nemá typ materiálu žiadny vplyv. Rozhranie THREE.js obsahuje 16 základných typov materiálov. Na obrázku 3.3 je príklad niektorých materiálov použitých na objekte v tvare torusu. Vzhľad materiálov ovplyvňujú najmä nasledujúce vlastnosti:

- Priehľadnosť materiálu
- Farba materiálu
- Lesklosť/matnosť materiálu
- Spôsob, akým materiál vypĺňa priestor mimo geometrie objektu

Pri potrebe iných materiálov ako ponúka základný balík rozhrania THREE.js je možné importovať ďalšie materiály, ktoré sú voľne dostupné na internete, poprípade je možné vytvoriť vlastné materiály.



Obr. 3.3: Rôzne druhy dostupných materiálov, ktoré ponúka rozhranie THREE.js

3.2 Tvorba scény a jej ovládanie

3.2.1 Scéna

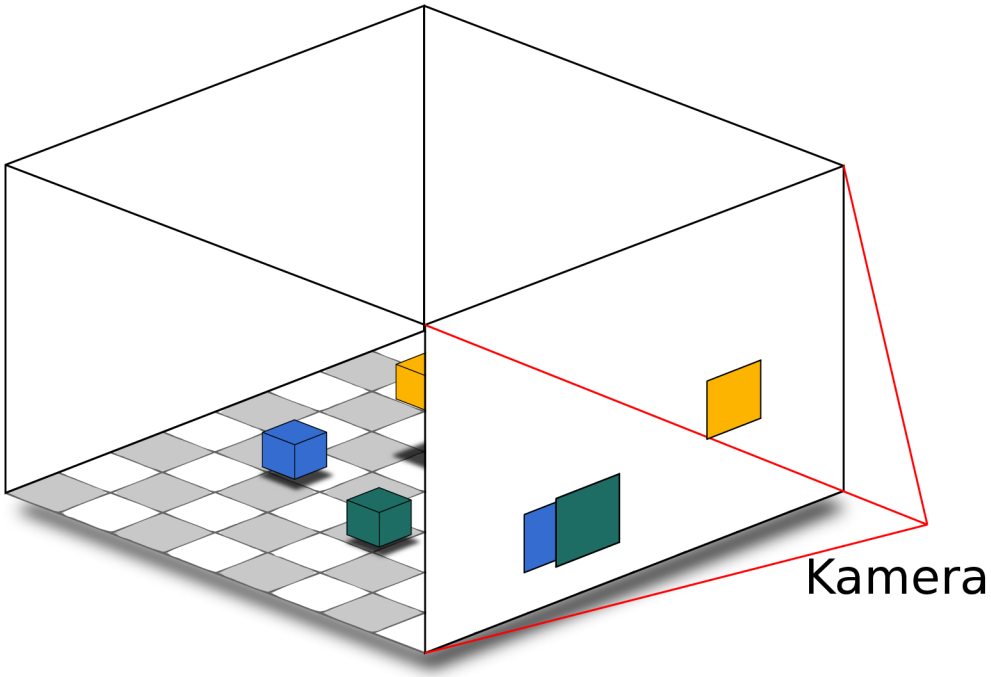
V našom prípade je scéna tvorená trojrozmerným virtuálnym priestorom. Virtuálny priestor obsahuje všetky objekty rozhrania THREE.js, ktoré či už priamo alebo nepriamo zasahujú do plánovania trajektórie koncového bodu robotického ramena. Inštanciou triedy s názvom `THREE.Scene()` vznikne trojrozmerný euklidovský virtuálny priestor. Z princípu vyplýva, že pozícia a rotácia scény je nulová. Objekty sú do scény pridávané štandardnou metódou `Add(Object)`, kde vstupným parametrom je celý objekt. Následne je možné pristúpiť k objektu ako ku potomkovi scény. Týmto spôsobom je možné zobraziť alebo skryť každý objekt zvlášť cez rodičovskú triedu. Aby boli objekty a ich zmeny skutočne viditeľné, scéna musí byť cyklicky renderovaná pomocou triedy `THREE.WebGLRenderer()`. Klasické zariadenia, na ktorých je zobrazená scéna, vykresľujú obraz 60-krát za sekundu. Obnovovaciu frekvenciu je možné znížiť pre menej výkonné zariadenia. Odporúčaná minimálna obnovovacia frekvencia je 24-krát za sekundu. Nižšia hodnota by pre užívateľa pôsobila viditeľným sekaním obrazu. Práve vďaka triede `THREE.WebGLRenderer()` je možné celú scénu zobraziť na akomkoľvek zariadení s ľubovoľným pomerom strán a rozlíšením. Po pridaní rozšírenia *WebVR.js* podporuje trieda aj mód virtuálnej reality. [11]

3.2.2 Kamera

Hlavnou úlohou kamery je premietnuť 3D obraz na 2D plochu, ktorú predstavuje obrazovka alebo displej. Podľa spôsobu reprezentácie obrazu sa projekcia obrazu delí

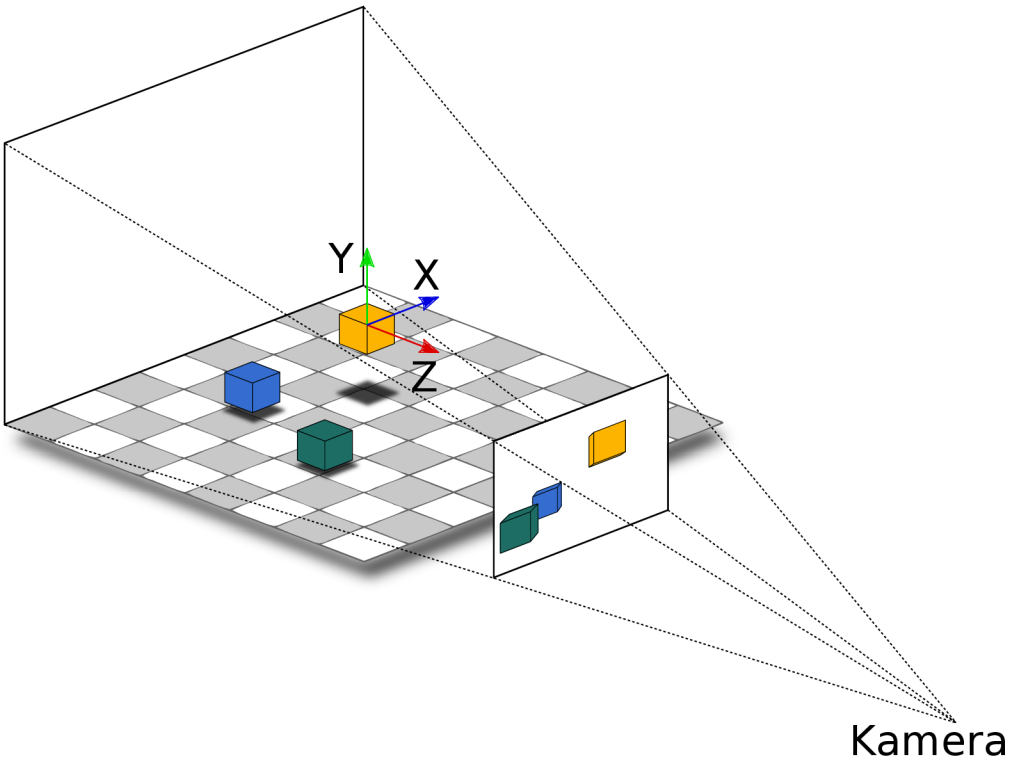
na:

- Ortografická projekcia - Scéna je prezentovaná ako pravouhlý hranol. Vlastnosťou zobrazenia je, že s meniacou sa vzdialenosťou nemenia objekty svoju veľkosť. Z toho dôvodu sa táto projekcia využíva najmä pri CAD systémoch.



Obr. 3.4: Princíp ortografickej projekcie

- Perspektívna projekcia - Scéna je prezentovaná ako skosený symetrický kolmý ihlan. Projekcia svojím výsledkom najviac pripomína pohľad ľudského oka. Je to najčastejšie používaný mód v 3D scénach. [15] Pre projekciu sú podstatné parametre:
  - fov - uhol pohľadu, typicky 50 stupňov.
  - aspect - pomer šírky a výšky skosenia ihlanu, typicky súčin šírky a dĺžky zobrazovaného okna.
  - near, far - minimálny a maximálny viditeľný rozsah kolmo od kamery, typicky 0.1 až 2000 .[11]



Obr. 3.5: Princíp perspektívnej projekcie

Webová aplikácia bude pracovať s perspektívnou projekciou, pre užívateľa je vhodnejšia z hľadiska prirodzeného pohľadu. Použitá trieda pre perspektívnu kameru sa nazýva `PerspectiveCamera()`. Keďže parameter *aspect* je zadávaný iba pri inicializácii a trieda neprepočítava automaticky veľkosť okna, musí byť prepočítavaná externe. K tomuto je využitá systémová funkcia `onWindowResize()`, ktorá je automaticky volaná pri zmene veľkosti okna. Funkcia znova prepočítava veľkosť okna a obnoví parametre kamery.

O pohyb kamery sa scéne sa stará trieda `THREE.OrbitControl()`. Pomocou tejto triedy je možné pristúpiť ku kamere ako k objektu. To znamená, že je možné meniť pozíciu a smer natočenia kamery. OrbitControl podporuje tri rôzne spôsoby ako ovládať kameru:

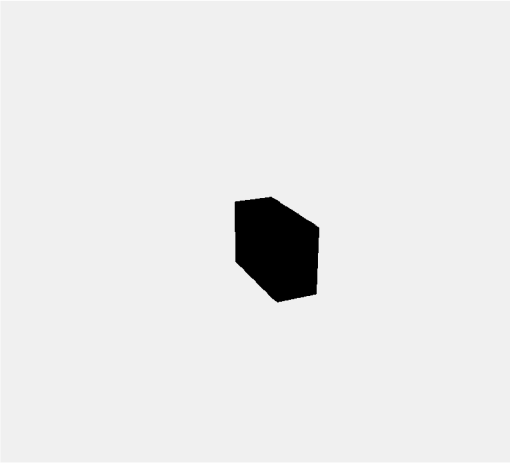
- Ovládanie pomocou klávesnice
  - pohyb v priestore - šípky na klávesnici odpovedajú smeru pohybu v priestore
- Ovládanie pomocou myši
  - pohyb v priestore - stlačené pravé tlačidlo na myši a ľubovoľný pohyb
  - rotácia okolo objektu - stlačené ľavé tlačidlo na myši a ľubovoľný pohyb
  - priblíženie/oddialenie od objektu - koliesko na myši alebo stlačené koliesko a ľubovoľný pohyb myši

- Ovládanie pomocou dotykovej obrazovky
  - pohyb v priestore - rovnaký pohyb dvoch prstov na obrazovke v požadovanom smere
  - rotácia okolo objektu - pohyb jedného prsta na obrazovke v požadovanom smere rotácie
  - priblíženie/oddialenie od objektu - pohyb dvoch prstov na obrazovke tak, aby sa od seba vzďaľovali alebo približovali.

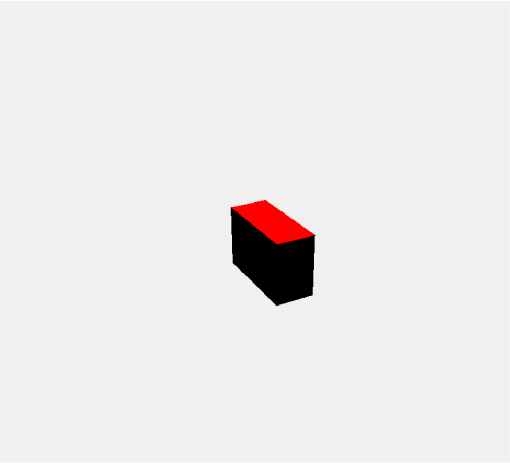
Pre všetky spôsoby ovládania je možné zmeniť gesto pre vstupnú akciu a taktiež aká bude výstupná reakcia. Vo webovej aplikácii sú dostupné všetky tri spôsoby ovládania, ale ovládanie kamery je optimalizované najmä pre vstup myši a na ovládanie pomocou dotykových gest pre priemyselné obrazovky a mobilné displeje.

3.2.3 Osvetlenie scény a tiene

Ako klasická scéna, tak aj virtuálna scéna, musí byť osvetlená. Bez osvetlenia by virtuálna scéna síce bola viditeľná, ale farba a detaily objektov by boli čierne. V scéne môže pôsobiť viacero druhov osvetlenia zároveň. Z dostupnej ponuky knižnice bolo vybrané osvetlenie typu reflektor s názvom triedy `SpotLight(color, intensity)`. Smerové svietidlo zvýrazní farby podľa dopadu svetla na objekt. V závislosti na tvare objektu sa zmenou odtieňa zvýraznia aj hrany. Farba osvetlenia je nastavená na hodnotu 0xFFFFFF, čo je biela farba s intenzitou 1.5. Porovnanie scény pred a po použití reflektora je na obrázkoch 3.6 a 3.7.[11]



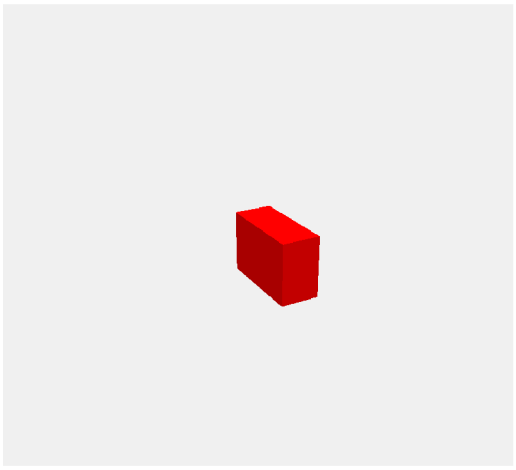
Obr. 3.6: Scéna bez použitia osvetlenia



Obr. 3.7: Scéna s použitím reflektoru

Ako je na obrázku 3.7 vidieť, pri použití reflektora sú hrany dobre viditeľné len pod určitým uhlom. Strany objektu, na ktoré nesvieti reflektor, ostávajú tmavé. Tento problém dokáže vyriešiť osvetlenie, ktoré osvetľuje všetky objekty globálne. To znamená, že všetky objekty sú osvetlené rovnako nezávisle na uhle alebo tvare

objektu. Táto trieda sa nazýva `AmbientLight(color)`. Rozdiel je možné pozorovať medzi obrázkami 3.7 a 3.8. [11]



Obr. 3.8: Scéna s použitím reflektoru a globálnym osvetlením

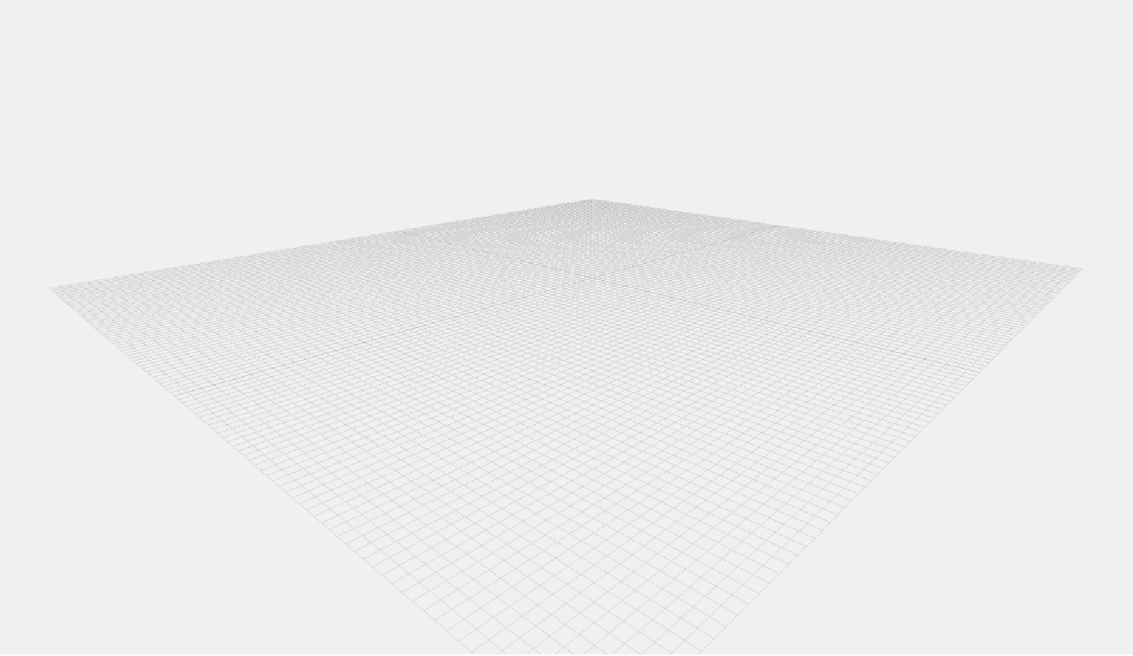
Keďže v scéne je použité osvetlenie typu *SpotLight*, ktoré má konkrétny smer, dá sa uvažovať aj o vzniknutých tieňoch objektov. Vyššie spomenutá funkcia má už v sebe implementovanú túto možnosť a stačí ju aktivovať. [11]

3.2.4 Základná rovina

Orientáciu v 3D priestore je možné určiť na základe absolútnej polohy alebo relatívne na základe vzdialenosti od objektov v priestore. Pri orientácii v priestore často nie je pre užívateľa potrebná presná poloha v súradniciach, ale relatívna pozícia, vďaka ktorej sa dokáže okamžite zorientovať.

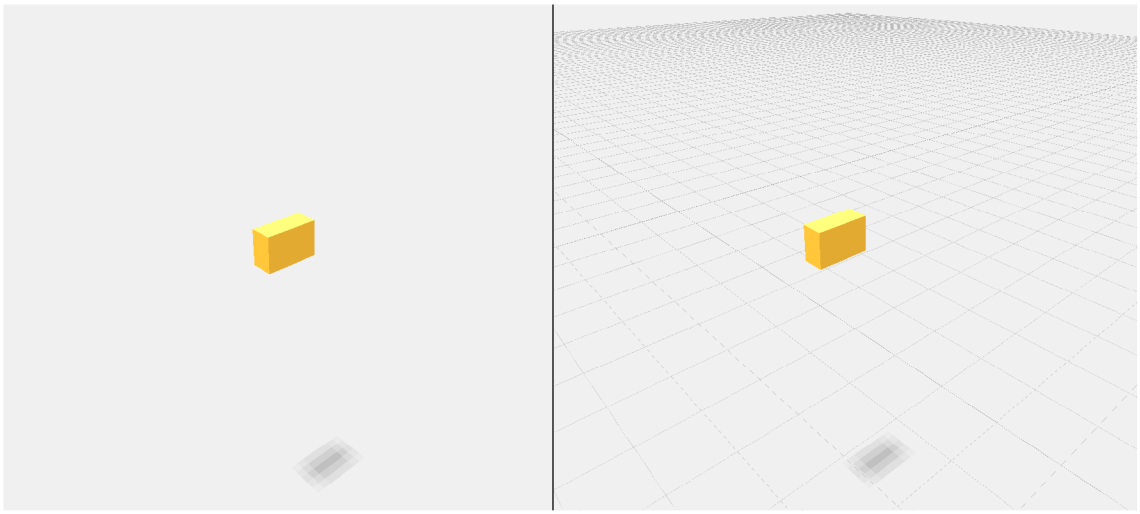
Základom každého viacrozmerného priestoru je rovina, ktorá určuje základňu umiestnenú v nulovom bode. V našom prípade bude rovina tvorená v osiach X, Z s nulovou výškou Y. API THREE.js ponúka niekoľko tried určených pre tvorbu roviny. Najvhodnejšou z ponuky je trieda s názvom `THREE.PlaneBufferGeometry()`. Prednosťou voľby je najlepšia optimalizácia z hľadiska nárokov na pamäť a výkon. Inštanciou triedy vznikne 2D rovina umiestnená v 3D priestore. To znamená, že hrúbka roviny v osi Y je nulová. Pomocou triedy je možné vytvoriť rovinu s ľubovoľnou dĺžkou a šírkou. [11][17] Jedným z najpodstatnejších parametrov roviny je voľba materiálu z akého bude vytvorená. Ako materiál bol zvolený `THREE.ShadowMaterial()`, ktorý ponúka unikátne vlastnosti a transparentnú plochu. Prvým parametrom pre voľbu transparentnej plochy bola požiadavka, že objekty musia byť viditeľné aj pod základnou rovinou, čo by s nepriehľadným materiálom nebolo možné. Ďalšou výhodnou vlastnosťou je, že materiál sa správa ako pevný objekt, teda aj napriek svojej priehľadnosti objekty môžu naň vrhať svoj tieň. S transparentnou plochou ale vzniká

problém, že rovina nie je viditeľná. Tento problém rieši trieda `THREE.GridHelper()`, ktorá sa používa ako súčasť základnej plochy. Inštanciou triedy je vytvorená mriežkovaná plocha, ktorá je zobrazená na obrázku 3.9.



Obr. 3.9: Zobrazenie Euklidovskej základne

Ďalej na obrázku 3.10 je zobrazená situácia, kde na ľavej strane sa nachádza základná plocha bez pomocnej mriežky a na pravej strane s pomocnou mriežkou. Ako je vidieť na obrázku, v oboch prípadoch skutočne tieň objektu dopadá na transparentnú plochu. Rozdiel ale nastáva pri odhade vzdialenosti objektu od základnej plochy. Zatiaľ, čo pri ploche bez mriežky je takmer nemožné odhadnúť vzdialenosť od základnej plochy, s mriežkou je tento odhad niekoľkonásobne jednoduchší.



Obr. 3.10: Zobrazenie Euklidovskej základne bez orientačnej mriežky a s orientačnou mriežkou

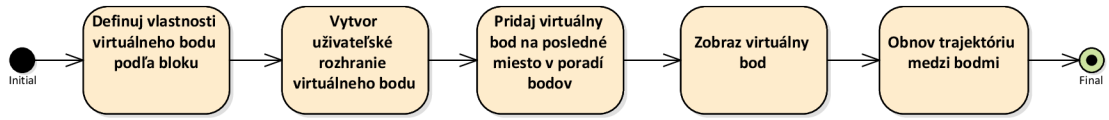
3.3 Plánovanie trajektórie koncového bodu robotického ramena

Spôsob, akým je tvorená trajektória koncového bodu robotického ramena, je jedna z najpodstatnejších vlastností každého prostredia zaoberajúceho sa plánovaním trajektórie. V tejto práci bol zvolený spôsob vytvárania pomocou virtuálnych bodov, ktorým sa zaoberá kapitola 3.3.1. Priestor medzi bodmi musí byť taktiež presne definovaný. Trajektóriou medzi virtuálnymi bodmi sa zaoberá kapitola 3.3.4. V rámci práce bola vytvorená knižnica *VPLib.js*, ktorá sa spolu s podpornými knižnicami stará o celú správu virtuálnych bodov.

3.3.1 Virtuálne body v priestore

Virtuálny bod je súčasťou akéhokoľvek objektu, ktorý sa nachádza vo virtuálnom priestore a priamo ovplyvňuje výslednú trajektóriu robotického ramena. Z logického hľadiska nemôže byť celý objekt so svojou plochou súčasťou trajektórie, preto výslednú trajektóriu určuje nekonečne malý bod, ktorý sa nachádza najčastejšie v strede objektu. Aby bol bod pre užívateľa viditeľný, v priestore je reprezentovaný ako trojrozmerný objekt s určitými rozmermi. O generovanie a správu virtuálnych bodov vo webovej aplikácií sa stará knižnica *VPLib.js*, ktorá bola vytvorená v rámci diplomovej práce. Pre generovanie virtuálnych bodov obsahuje knižnica *VPLib.js* metódu

s názvom `addPoint(block,customPoint)`. Prvý parameter metódy je povinný a určuje, v akom bloku sa bude virtuálny bod nachádzať. Význam blokov je vysvetlený v kapitole 3.3.4. Druhý parameter metódy je nepovinný a používa sa pri importovaní virtuálnych bodov generovaných CAD systémom. Pokiaľ nie je parameter vyplnený, je použitý predvolený tvar hranolu. K vytvoreniu objektu v tvare hranolu bola využitá metóda z rozhrania *THREE.js* s názvom `BoxBufferGeometry()`. Na obrázku 3.11 je zobrazený stavový diagram, ktorý zobrazuje proces pri vytváraní nového virtuálneho bodu. Tvorba užívateľského rozhrania je popísaná v kapitole 5.



Obr. 3.11: Stavový diagram pri vytváraní nového virtuálneho bodu pomocou metódy addPoint

Vygenerovaný virtuálny bod je vložený na náhodne vygenerované miesto v rozsahoch:

v súradnici X leží v uzavretom intervale $\langle -50, 50 \rangle$
v súradnici Y leží v uzavretom intervale $\langle 0, 60 \rangle$
v súradnici Z leží v uzavretom intervale $\langle -40, 40 \rangle$

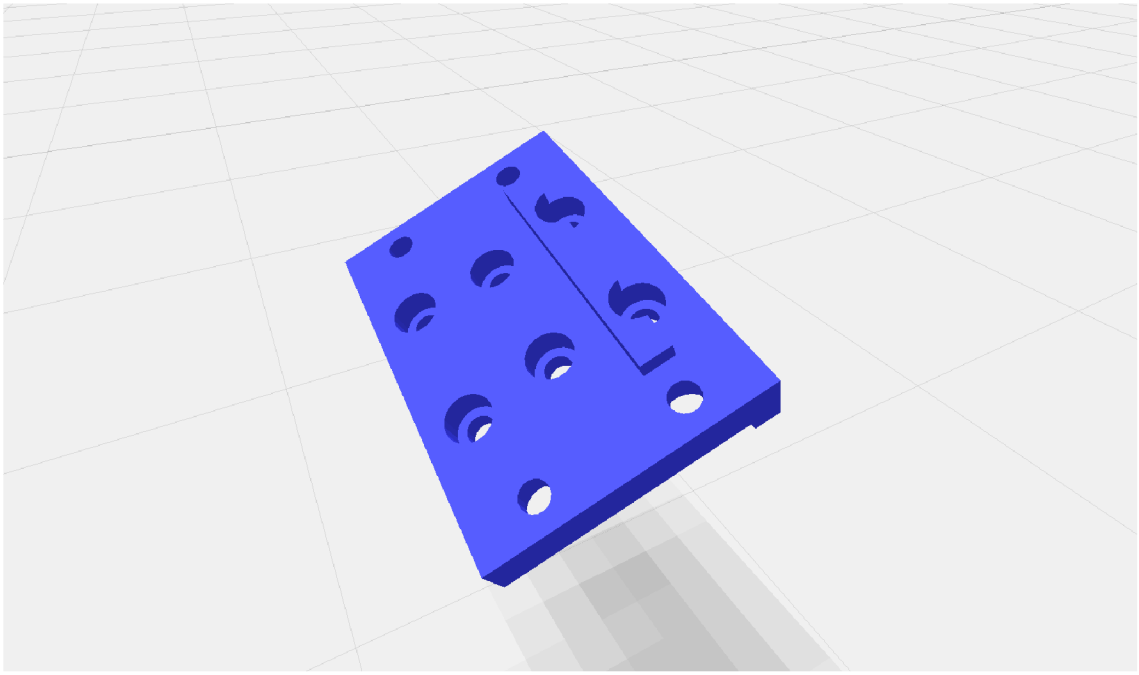
Počiatočná rotácia virtuálneho bodu je nastavená vo všetkých uhloch na nulu. Materiál a farba virtuálnych bodov je určovaná každým blokom zvlášť, nezávisle na užívateľovi. Každému virtuálnemu bodu je automaticky pridaný, náhodne vygenerovaný identifikátor UUID v tvare napríklad: *22B5C3F2-2899-4AF6-BADF-F0B9DC05E7A1*. Identifikátor sa využíva pre jednoduchšie a rýchlejšie vyhľadávanie objektov v scéne. Po pridaní akéhokoľvek objektu sa kamera automaticky premiestní pred novovytvorenú pozíciu. Spôsob presunu kamery je popísaný v kapitole 5.3. Každému virtuálnemu bodu je pri inicializácii pridaný aj ukazovateľ v podobe šípky, ktorá ukazuje smerom do stredu objektu. Rotácia šípky určuje, pod akým uhlom pristúpi koncový bod robotického ramena k virtuálnemu bodu. Problematikou sa viac zaoberá kapitola 3.3.5.

Pri plánovaní trajektórie sa často stáva, že vznikne prebytočný virtuálny bod, ktorý treba zo scény zmazať. Pre tento účel bola v rámci diplomovej práce vytvorená metóda s názvom `RemovePoint(pointUUID)`, ktorá je súčasťou knižnice *VPLib.js*. Vstupným parametrom metódy je jedinečný identifikátor UUID, ktorý bol pri vytvorení objektu pridelený.

3.3.2 Virtuálne body generované CAD systémom

Pri plánovaní presunu objektu z miesta A na miesto B pomocou robotického ramena je potrebné, aby boli známe rozmery a váha objektu kvôli uchopeniu do chápadla. Pokiaľ má prenášaný objekt špecifickú geometriu značne komplexnejšiu ako základné tvary, je zvykom daný objekt najprv vymodelovať v CAD systéme. Z výkresu je následne veľmi jednoduché exportovať objekt do 3D súborového formátu.

Časť aplikačného rozhrania THREE.js je zameraná aj na načítanie a zobrazovanie 3D objektov vygenerovaných pomocou CAD systému. Z veľkého množstva typov súborov, ktoré rozhranie dokáže načítať, bol zvolený **.stl* formát. Do zvoleného formátu dokáže model exportovať takmer každý 3D návrhový systém a taktiež tento súborový formát patrí medzi jeden z najpopulárnejších. Stl súbory popisujú iba povrchovú geometriu, to znamená, že farba musí byť doplnená zvlášť, čo je v našom prípade žiadanou vlastnosťou. Pre načítanie stl súborov sa používa metóda `STLLoader()` aplikačného rozhrania THREE.js. Aby boli načítané objekty kompatibilné s vytvorenou scénou a jej objektami, bola v rámci diplomovej práce vytvorená knižnica *customPointLoader.js*. Všetky 3D modely, ktoré majú byť použité ako virtuálne body v scéne, musia byť uložené v špeciálnom priečinku s názvom *pointModels*. Tento priečinok sa nachádza v koreňovej štruktúre webovej aplikácie. Dnešné prehliadače ale z bezpečnostných dôvodov nepovoľujú priamy prístup do priečinkov. Preto musí byť webová aplikácia spustená pomocou webového servera alebo musia byť nefunkčné bezpečnostné prvky prehliadača. Po spustení knižnica customPointLoader.js, ktorá využíva metódu STLLoader, načíta všetky modely do špeciálneho formátu vhodného na zobrazenie v scéne. Na obrázku 3.12 je zobrazená komplexná geometria virtuálneho bodu, ktorý bol generovaný CAD systémom do súborového formátu stl.



Obr. 3.12: Virtuálny bod exportovaný pomocou CAD systému

Stred rotácie každého objektu je zvlášť centrovaný podľa svojich rozmerov. V prípade, že model nie je možné načítať alebo neexistuje, je použitý predvolený tvar virtuálnych bodov.

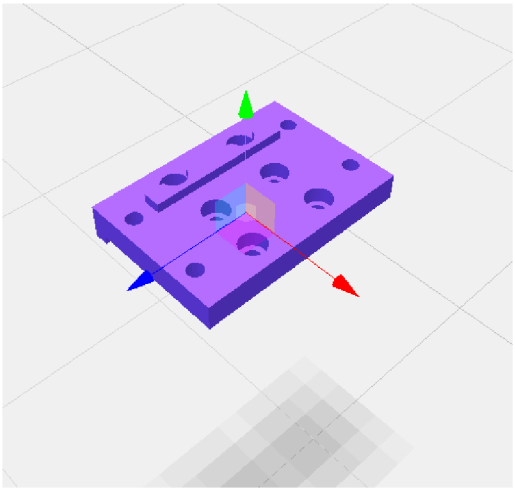
3.3.3 Interakcia virtuálnych bodov s užívateľom

Spôsob, akým užívateľ vytvára alebo mení trajektóriu, patrí k najpodstatnejším vlastnostiam webovej aplikácie. Pre užívateľa musí byť spôsob manipulácie s virtuálnymi bodmi jednoduchý, ale zároveň efektívny a intuitívny. Najvhodnejším spôsobom je priama interakcia s každým bodom. Pokiaľ užívateľ klikne na akýkoľvek virtuálny bod v priestore, zobrazí sa ponuka, kde je na výber presun objektu v priestore, rotácia objektu alebo centrovanie stredu rotácie kamery na zvolený bod. Ponuka ďalej podľa výberu obsahuje vstupno-výstupné polia pre polohu alebo rotáciu. Tvorbe ponuky sa venuje kapitola 5.2. O transformáciu objektov v priestore sa stará rozšírenie rozhrania THREE.js s názvom `TransformControls(Camera, renderer)`. Pomocou tejto metódy je možné zvolený objekt transformovať. Spôsob transformácie je volený pomocou metódy `setMode(mode)` s parametrom dátového typu string. Aktuálne podporované spôsoby transformácie sú:

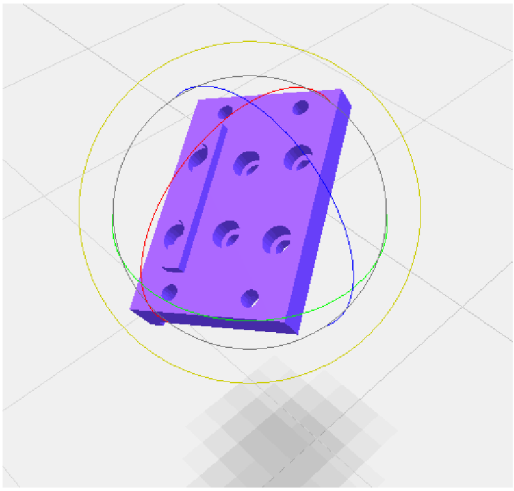
- translate - v tomto móde je možný ľubovoľný pohyb s objektom v priestore. Od stredu virtuálneho bodu sa zobrazia tri na seba kolmé, farebne rozlíšené šípky,

kde každá predstavuje jednu os. Pri kliknutí na jednu zo šípok je možné pohybovať objektom výhradne vo zvolenom smere. Ďalej sú zobrazené tri roviny, ktoré sú vždy umiestnené medzi dvomi protiľahlými osami. Po stlačení roviny je možne pohybovať objektom v dvoch zvolených osiach. Poslednou časťou je kocka, ktorá pribudne v strede objektu. Jej uchopením je možné pohybovať objektom vo všetkých osiach zároveň. Na obrázku 3.13 sú zobrazené jednotlivé interaktívne osi, kde červená os je pre zmenu pozície v smere X, modrá os pre zmenu v smere Y a zelená os pre zmenu v smere Z.

- rotate - v tomto móde je možné ľubovoľne rotovať objekt okolo svojho stredu. V okolí objektu sú vytvorené tri farebne rozlíšené kruhy, ktoré sú svojou plochou na seba kolmé. Uchopením a pohybom zvoleným kruhom je objekt rotovaný v danom smere. Na obrázku 3.14 je zobrazený spôsob rotácie objektu, kde červený kruh dominantne mení uhol α, zelený kruh mení dominantne uhol β a modrý kruh mení dominantne uhol natočenia objektu v smere γ.
- scale - v tomto móde je možné meniť rozmery zvoleného objektu. Po kliknutí na zvolený objekt sa zobrazia tri farebne rozlíšené osi zakončené objektom v tvare kocky. Ich uchopením sa vo zvolenom smere mení rozmer objektu.



Obr. 3.13: Grafický vstup pre transláciu objektu



Obr. 3.14: Grafický vstup pre rotáciu objektu

Aby mohol byť objekt transformovaný, musí byť pridelený metóde `attach(object)` objektu transfromControl. TransformControl ale nepodporuje voľbu objektov v 3D priestore, preto musí byť použitá ďalšia metóda s názvom `DragControls(objects, scene, renderer)`, ktorá je opäť súčasťou THREE.js. Prvý parameter pri vytváraní nového objektu je pole objektov, na ktoré bude DragControls reagovať. To znamená, že po kliknutí na objekt, ktorý bol pridelený DragControls, je formou udalosti vrátená celá štruktúra objektu, ktorá je pridelená metóde pre transformáciu objektu.

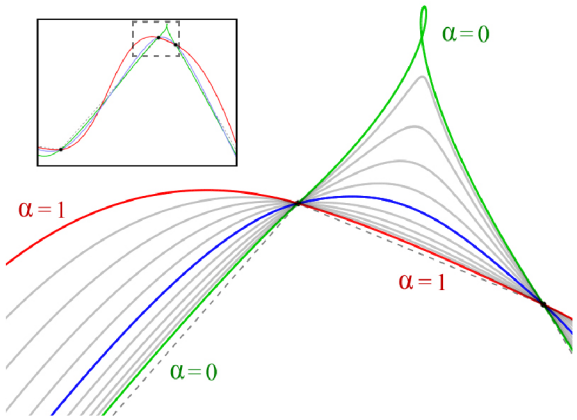
Na opačnú stranu, aby bola vlastnosť transformácie odňatá, musí byť použitá metóda `dettach()` objektu TransformControls. Po kliknutí na zvolený virtuálny bod je volaný časovač na 10 sekúnd. Po uplynutí tejto doby je automaticky volaná funkcia `dettach()`.

3.3.4 Trajektória medzi virtuálnymi bodmi

Každý virtuálny bod, ktorý je použitý v priestore, určuje konkrétne umiestnenie a rotáciu koncového bodu robotického ramena, s akou by mal toto miesto dosiahnuť v určitom čase. Trajektória, ktorú musí koncový bod robotického ramena uraziť medzi virtuálnymi bodmi, nie je presne definovaná. Existuje veľké množstvo hľadísk, podľa ktorých by mohla byť táto trajektória definovaná. Medzi najpodstatnejšie ale patrí:

- trajektória s časovo optimálnym prechodom
- energicky optimálna trajektória
- trajektória s ohľadom na životnosť mechaniky robotického ramena

Vzhľadom na rozsah témy boli zvolené dva rôzne typy trajektórie, ktoré si môže užívateľ zvoliť. Prvým typom je trajektória, ktorá medzi jednotlivými virtuálnymi bodmi vytvára splajn. Trajektória je tvorená splajnom typu Catmull-Rom. Tento typ splajnu je často používaný v počítačovej grafike vďaka svojej vlastnosti vyhladzovať pohyb medzi zadanými bodmi a zároveň krivka stále prejde zadaným bodom. Ku každej krivke medzi dvoma bodmi sú vyžadované ďalšie dva body z ich okolia. Následne zmena pozície jedného z bodov vytvorí hladký prechod a zároveň ovplyvní len malé okolie. Najčastejšie používané parametrizácie Catmull-Rom krivky sú: uniform, chordal a centripetal. Na obrázku 3.15 sú zobrazené jednotlivé parametrizácie. Uhlom α je určené pnutie, ktorým sú udávané veľkosti oblúkov. [18]

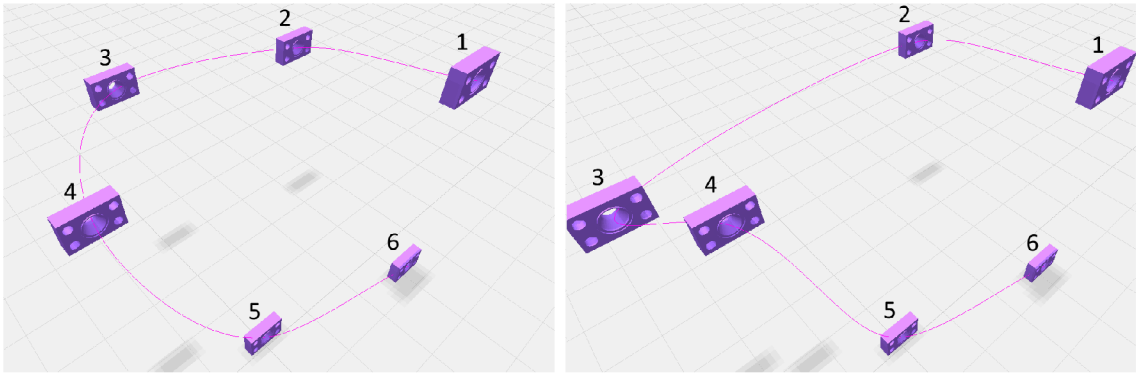


Obr. 3.15: Krivky Catmull-Rom s parametrizáciami: $\alpha = 0$ - uniform(zelená), $\alpha = 0.5$ - Centripetal(modrá), $\alpha = 1$ - Chordal(červená). Krivky šedou farbou sú hodnoty α z rozsahu 0 až 1 s krokom 0.1[18]

Rozhranie THREE.js podporuje prácu so splajnami a jedným z typov je práve Catmull-Rom. Metóda `CatmullRomCurve3(points, closed, curveType, tension)` vytvára určitý počet bodov v priestore medzi dvoma virtuálnymi bodmi. Hustota vytvorených bodov následne určuje, aká hladká krivka bude. Prvým parametrom metódy je pole bodov, cez kotré ma plajn prechádzať. Jednotlivé body poľa musia byť v dátovom type *Vector3*, ktorý je popísaný v kapitole 3.1.1. Druhý parameter určuje, či bude výsledná krivka uzavretá. Tretím parametrom je, o akú parametrizáciu krivky sa bude jednať. Na výber sú vyššie spomenuté parametrizácie: uniform, chordal a centripetal. Posledný parameter určuje pnutie krivky, teda čím vyššia hodnota, tým je krivka menej vyhladená. V prípade, že je metóde predaný iba jeden bod, krivka nie je vytvorená. V prípade, že sú metóde predané iba dva body, krivka nie je vyhlaďovaná. V prípade, že sú metóde predané 3 body, je vyhlaďovaná iba jedna strana krivky. O tvorbu splajnov sa stará knižnica s názvom *SplineControls.js*, ktorá bola vytvorená v rámci diplomovej práce. Prvou funkciou je `getSplinePoints(actPoints, prevPoints, type)`. Prvým vstupom sú všetky body, ktoré majú byť prepojené splajnom. Druhý parameter je nepovinný, používa sa pri práci s virtuálnymi bodmi tak, aby bol posledný bod predchádzajúceho bloku spojený s prvým bodom aktuálneho bloku. Posledný parameter je taktiež voliteľný a používa sa výhradne pri práci s blokmi. Výstupom funkcie je pole vektorov, ktoré tvoria vytvorený splajn medzi bodmi.

Ďalšia funkcia, ktorá sa nachádza v knižnici SplineControls.js, sa nazýva `getSpline (actPoints, prevPoints, type)`. Všetky tri parametre sú zhodné ako v predchádzajúcej funkcii. Časť funkcie využíva vyššie spomenutú funkciu getSplinePoints. Návratovým typom je objekt z knižnice THREE.js s názvom *Line*. Jedná sa čiaru, ktorá je poskladaná z určitého množstva segmentov. Hrúbka čiary je vždy 1 pixel. Funkciou je teda vrátená spojitá trajektória medzi zvolenou skupinou bodov. Každej trajektórii tvorenou čiarou, ktorá bola vytvorená vrámci blokov, je pridelené meno v tvare *blockXspline*, kde X označuje poradie bloku.

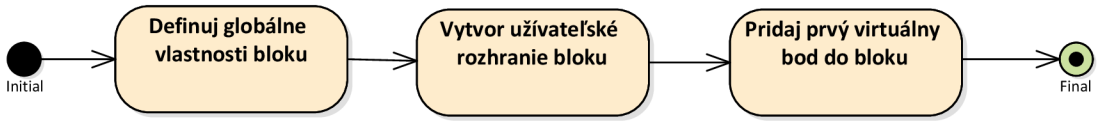
Posledná funkcia z knižnice SplineControls.js má názov `updateSpline (blocks, scene)`. Úlohou funkcie je vygenerovať, poprípade pri zmene polohy niektorého z virtuálnych bodov, obnoviť trajektóriu v rámci jedného bloku. Funkcia v prvom rade odstráni existujúcu trajektóriu v rámci bloku, následne je vygenerovaná nová trajektória vďaka funkcii getSpline. Na obrázku 3.16 je splajn reprezentovaný fialovou čiarou, ktorý bol vygenerovaný vyššie spomenutým postupom. V zobrazenej scéne je ďalej vidieť, ako splajn vytvoril hladkú trajektóriu medzi jednotlivými virtuálnymi bodmi a zároveň prechádza ich stredom. Pri porovnaní obrázkov vľavo a vpravo je vidieť, že najväčšia zmena trajektórie prebehla medzi virtuálnymi bodmi číslo 2 - 3 - 4. Malá zmena trajektórie ale nastala aj medzi virtuálnymi bodmi 1 - 2 a 4 - 5 aj napriek tomu, že ich poloha sa nezmenila.



Obr. 3.16: Trajektória tvorená splajnom pred a po zmene pozície virtuálneho bodu

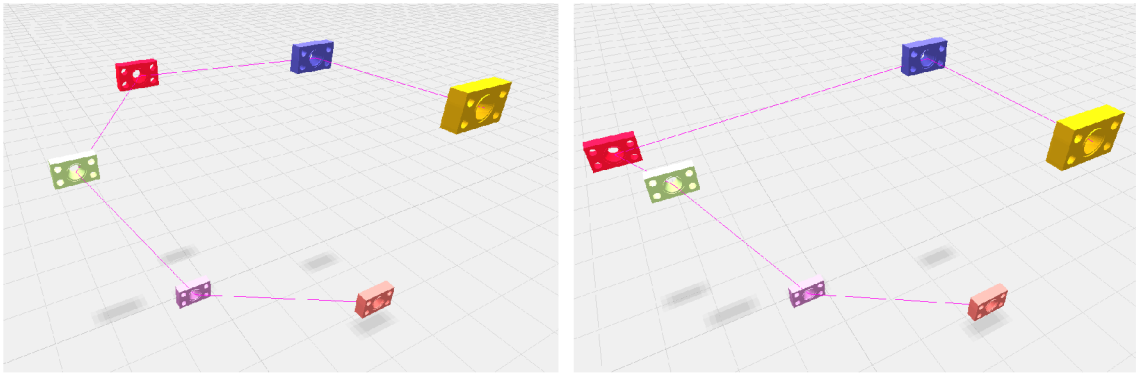
Druhý spôsob, ktorým môže užívateľ definovať trajektóriu, je pomocou blokov. Blok slúži na združenie virtuálnych bodov, medzi ktorými má vzniknúť splajn do jednej skupiny. Je pravidlom, že každý blok musí obsahovať aspoň jeden bod. V prípade, že máme vytvorený blok 1 a blok 2 platí, že posledný virtuálny bod bloku 1 je priamo spojený s prvým virtuálnym bodom bloku 2. Medzi virtuálnymi bodmi, ktoré spájajú dva bloky, nie je vytvorený žiadny splajn. To znamená, že iba zmena pozície posledného virtuálneho bodu bloku 1 alebo prvého virtuálneho bodu bloku 2, môže zmeniť trajektóriu medzi blokmi.

Knižnica *VPLib.js*, ktorá bola vytvorená v rámci diplomovej práce, obsahuje metódu s názvom `addBlock(customPoint)`. Pomocou tejto metódy je vytvorený nový blok. Ako je vidieť v stavovom diagrame na obrázku 3.17, po vytvorení objektu bloku je automaticky volaná metóda, ktorá vytvorí prvý virtuálny bod bloku. Jediný parameter metódy addBlock dáva užívateľovi možnosť vytvoriť blok tak, aby každý ďalší virtuálny bod, ktorý bude pridaný v rámci tohto bloku mal tvar objektu generovaného CAD systémom. Pri vytváraní nového bloku je náhodne vygenerovaná farba, ktorou budú pokryté všetky virtuálne body. Medzi globálne vlastnosti bloku patrí názov bloku, ktorý je automaticky generovaný v tvare *blockX*, kde X je číslo poradia vytvoreného bloku. Ďalej je definovaná referencia na užívateľom zvolený model virtuálneho bodu, materiál a farba budúcich virtuálnych bodov.



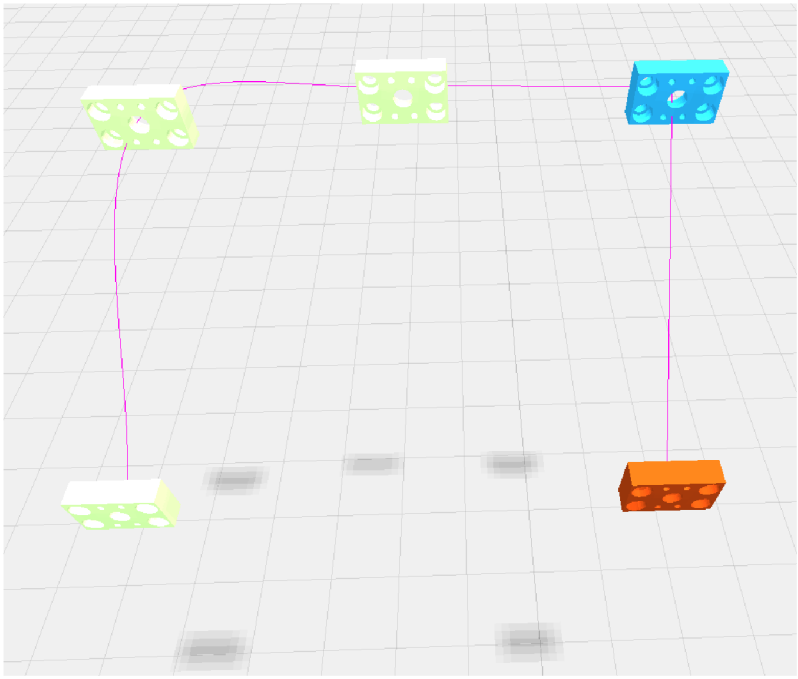
Obr. 3.17: Stavový diagram pri vytváraní nového bloku pomocou metódy addBlock

Na obrázku 3.18 je ukážka trajektórie tvorenej výhradne blokmi, ktoré obsahujú iba jeden virtuálny bod. Keďže každý blok generuje odlišnú, náhodnú farbu, každý virtuálny bod je inak sfarbený. V zobrazenej scéne sa nenachádza žiadny splajn, a preto je trajektória medzi virtuálnymi bodmi priama. Tento typ trajektórie poskytuje najkratšiu možnú trajektóriu medzi virtuálnymi bodmi, ale trajektória obsahuje ostré hrany pri prechode na ďalší bod. Pri porovnaní obrázkov vľavo a vpravo je vidieť, že trajektória sa zmenila len medzi modrým, červeným a zeleným virtuálnym bodom.



Obr. 3.18: Trajektória tvorená blokmi pred a po zmene pozície virtuálneho bodu

Vo webovej aplikácií je možné použiť aj kombináciu splajnov a blokov zároveň. Na obrázku 3.19 je zobrazený spôsob, akým je splajn prepojený s novými blokmi.



Obr. 3.19: Trajektória tvorená kombináciou splajnu a blokov

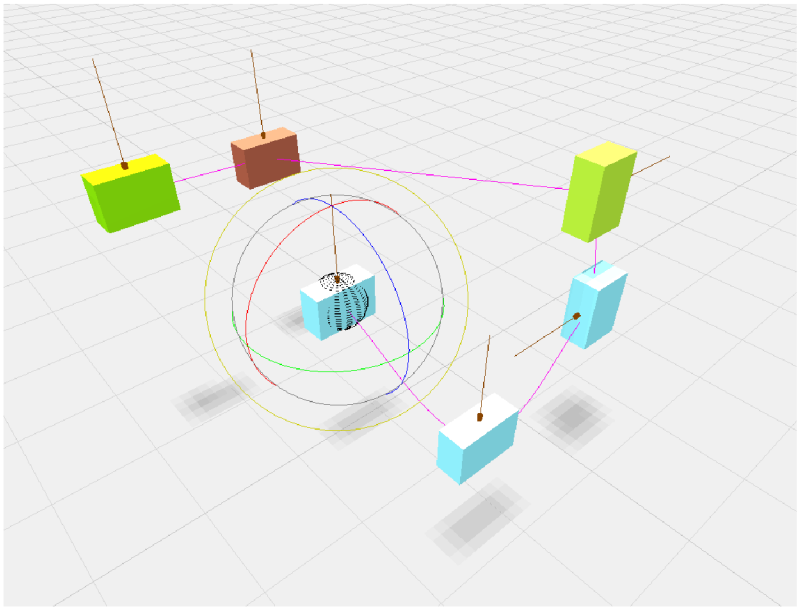
Pokiaľ je z bloku odstránený posledný virtuálny bod, je automaticky volaná metóda s názvom `removeBlock(blockID)`, ktorá zmaže prázdnu štruktúru z pamäte a aj z užívateľského rozhrania. V prípade, že je vytvorený jediný blok, nie je možné ho odstrániť.

3.3.5 Prístup koncového bodu robotického ramena k virtuálnym bodom

V prípade, že máme v priestore definovaný virtuálny bod, existuje veľké množstvo rôznych možností, ako môže koncový bod robotického ramena pristúpiť k virtuálnemu bodu. Preto každý virtuálny bod má potomka v podobe ukazovateľa s tvarom šípky, ktorá ukazuje do jeho stredu. Uhol, pod ktorým ukazovateľ smeruje do virtuálneho bodu naznačuje požadovaný smer, pod ktorým má koncový bod robotického ramena pristúpiť k virtuálnemu bodu. Užívateľ musí mať možnosť ovplyvniť smer prístupu koncového bodu robotického ramena nezávisle na aktuálnej rotácii virtuálneho bodu. Ako vhodný spôsob rotácie ukazovateľa bol zvolený rovnaký princíp ako pri rotácií virtuálneho bodu. Pri tomto spôsobe rotácie ale vznikal konflikt, kedy na miesto ukazovateľa bol rotovaný virtuálny bod a naopak. Preto bola tvorba trajektórie rozdelená na dva módy:

- Mód virtuálnych bodov
- Mód robotického ramena

Mód virtuálnych bodov je popísaný v predchádzajúcej kapitole 3.3.3. Po prepnutí do robotického módu ostane interaktívny iba ukazovateľ na prvý virtuálny bod. Po uchopení reálneho objektu do chápadla robotickým ramenom, sa jeho uhol vzhľadom k robotovi už nemení. Z tohto dôvodu je interaktívny iba prvý virtuálny bod. Po zmene rotácie ukazovateľa sa mení ukazovateľ aj na všetkých zvyšných virtuálnych bodoch vrátane všetkých blokov. Na obrázku 3.20 sú vidieť ukazovatele v tvare hnedej šípky, ktorá smeruje do stredu virtuálnych bodov pod určitým uhlom. Prvý virtuálny bod pozostáva z ukazovateľa s priehľadnou guľou uprostred, ktorá uľahčuje rotáciu šípky. Ako je ďalej na obrázku vidieť, niektoré virtuálne body majú rozličnú rotáciu, s ktorou sa zmenil aj smer prístupu koncového bodu robotického ramena.



Obr. 3.20: Náhľad na určenie smeru prístupu koncového bodu robotického ramena k virtuálnemu bodu

Módy sú prepínané z grafického užívateľského prostredia, ktoré je popísané v kapitole 5.4.

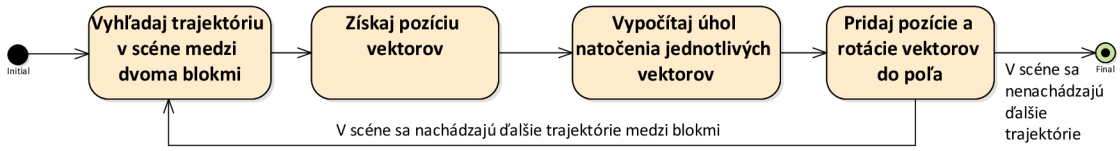
3.3.6 Generovanie trajektórie

V rámci modularity je generovaná trajektória do univerzálnej štruktúry, z ktorej je ďalej možné vytvoriť trajektóriu špecifickú pre každé robotické rameno. O generovanie trajektórie sa stará funkcia s názvom `exportLines(scene, blocks)`, ktorá sa nachádza v knižnici export.js a bola vytvorená v rámci diplomovej práce. Ako vyplýva z druhého parametru definície funkcie, vstupom je celý objekt s blokmi a ich virtuálnymi bodmi. Na obrázku 3.21 je zobrazený stavový diagram, ktorý popisuje proces vytvárania štruktúry popisujúcej celú trajektóriu vo vektoroch. Ako je vidieť

na obrázku, v prvej časti sú v scéne vyhľadávané trajektórie v rámci celého bloku. Trajektória je vyhľadávaná na základe mena, ktoré jej bolo pridelené. Spôsob prideľovania mena je popísaný v kapitole 3.3.4. Funkcia z knižnice export.js s názvom `getVertices(line)` následne z vyhľadanej trajektórie extrahuje počiatky vektorov. Keďže pri plánovaní trajektórie nie je nutné poznať smer vektorov, je počítaný až pri generovaní trajektórie. O výpočet smeru vektorov sa ďalej stará funkcia s názvom `getAvgAngle(block, blockIndex, line, vertices, extraPoint)` z knižnice export.js. Funkcia vyžaduje nasledovné parametre:

- block - vyžadovaný je iba konkrétny blok, v rámci ktorého je tvorený splajn.
- blockIndex - súvisí s parametrom extraPoint. Využíva sa na ošetrenie stavu v prípade, keď je blok prvý v poradí.
- vertices - z extrahovaných počiatkov vektorov bude počítaný smer vektoru.
- extraPoint - jedná sa o posledný bod predchádzajúceho bloku, s ktorým je spojený aktuálny blok. V prípade prvého bloku parameter nie je zadávaný.

Smer každého vektoru trajektórie priamo vplýva na plynulý pohyb robotického ramena, a preto musí byť aj prechod medzi jednotlivými vektormi plynulý. Zmena smeru vektoru je počítaná priemerne vzhľadom na počet vektorov trajektórie medzi virtuálnymi bodmi, rozdiel počiatkov vektorov a rotácie vždy dvoch ukazovateľov na virtuálny bod.



Obr. 3.21: Stavový diagram generovania celej trajektórie do univerzálnej štruktúry pri použití funkcie exportLines

Nasledujúce rovnice popisujú výpočet smeru každého vektoru trajektórie. Rovnica 3.1 popisuje výpočet vzdialenosti jednotlivých vektorov pomocou Pytagorovej vety v 3D priestore. Keďže trajektória medzi virtuálnymi bodmi často mení svoj smer, aj jednotlivé vektory sú od seba rôzne vzdialené v závislosti na vytvorenom oblúku.

$$\Delta v_{dist}(i) = \sqrt{[x_i \ - x_{i+1}\ ]^2 + [y_i \ - y_{i+1}\ ]^2 + [z_i \ - z_{i+1}\ ]^2} \tag{3.1}$$

,kde Δv_{dist} vzdialenosť dvoch vektorov [cm]
x hodnota súradnice x vektoru trajektórie [cm]
y hodnota súradnice y vektoru trajektórie [cm]
z hodnota súradnice z vektoru trajektórie [cm]
i poradie vektoru v trajektórií [-]

Pokiaľ sú známe všetky vzdialenosti medzi susednými vektormi, je možné spočítať celkovú dĺžku trajektórie medzi dvoma virtuálnymi bodmi. Zo získaných vzdialeností vektorov a dĺžky trajektórie je následne možné určiť, aký podiel na zmene uhla bude mať každý vektor. Vzťah pre výpočet úmernej zmeny je použitý v rovnici 3.4.

$$v_{distTotal} = \sum_{i=M}^{N-1} v_{distTotal} + \Delta v_{dist}(i) \tag{3.2}$$

Kde $v_{distTotal}$ celková dĺžka trajektórie medzi dvomi virtuálnymi bodmi [cm]
$\Delta v_{dist}(i)$ funkcia vzdialenosti dvoch vektorov [cm]
M index vektoru s polohou predchádzajúceho virtuálneho bodu [-]
N index vektoru s polohou aktuálneho virtuálneho bodu [-]

Ďalšia časť výpočtu sa zameriava na určenie rozdielu rotácie medzi dvoma ukazovateľmi na virtuálne body. Rotácie ôs X, Y, Z sú počítané zvlášť. V práci bude uvedený vzorový výpočet pre rotáciu osi X v rovnici 3.3, zvyšné osi sú počítané rovnakým spôsobom.

$$v_{rotTotalX} = X_M - X_N \tag{3.3}$$

Kde $v_{rotTotalX}$ rozdiel dvoch uhlov ukazovateľov na virtuálny bod [rad]
X_M uhol x vektora s polohou virtuálneho bodu 1 [rad]
X_N uhol x vektora s polohou virtuálneho bodu 2 [rad]

Následne prvá časť rovnice 3.4 dáva do rovnosti neznámu úmernú zmenu uhla X a úmernú zmenu pozície vektora. Z rovnice je následne vyjadrená neznáma zmena uhlu X.

$$\frac{\Delta v_{rotX}(i)}{v_{rotTotalX}} = \frac{\Delta v_{dist}(i)}{v_{distTotal}} \Rightarrow \Delta v_{rotX}(i) = \frac{\Delta v_{dist}(i) \cdot v_{rotTotalX}}{v_{distTotal}} \tag{3.4}$$

Kde $\Delta v_{rotX}(i)$ zmena uhla X úmerná zmena pozície vektora [rad]

Poslednou rovnicou 3.5 je vyjadrený konečný uhol X aký vektor nadobudne.

$$v_{rotX}(i) \;=\; v_{rotX}(i-1) \;-\Delta v_{rotX}(i) \tag{3.5}$$

kde $v_{rotX}(i)$ výsledný uhol X aktuálny s indexom i [rad]

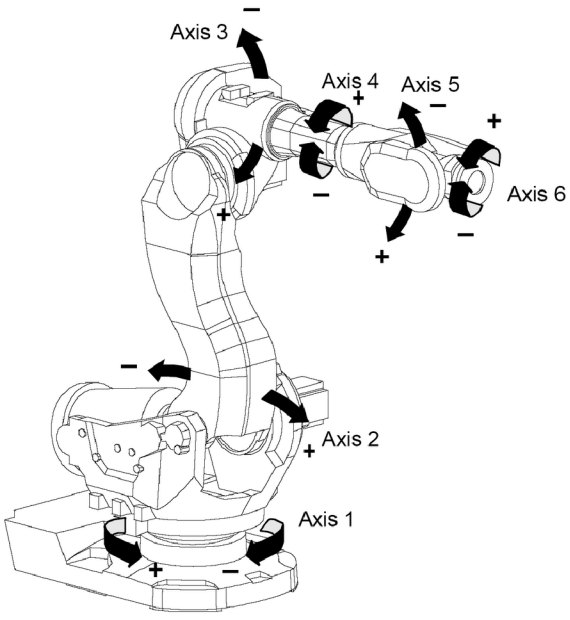
Rovnakým princípom sú počítané všetky trajektórie medzi virtuálnymi bodmi. Po výpočte smeru a pozície je každý vektor uložený do poľa štruktúr v tvare: `x, y, z, _x,_y,_z`, kde prvé tri hodnoty určujú pozíciu v priestore a zvyšné určujú smer vektoru.

4 Virtuálny model robotického ramena

Aby mal užívateľ najlepšiu možnú predstavu o pripravovanej trajektórií je nutné, aby webová aplikácia obsahovala model robotického ramena. Požiadavkou na webovú aplikáciu je, aby bola vytvorená dostatočne dynamicky pre načítanie a prácu s robotmi rôznych typov a výrobcov.

4.1 Implementácia grafických prvkov do webového prostredia

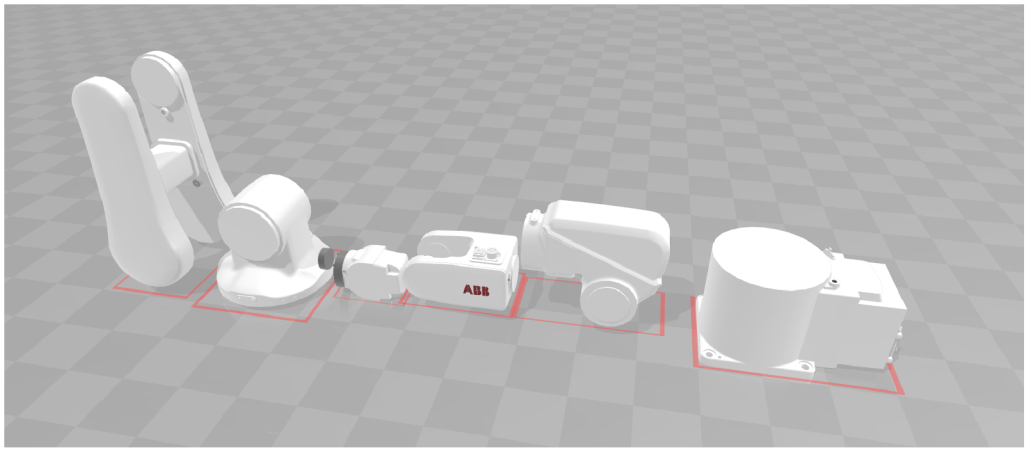
Najvhodnejším spôsobom ako implementovať robotické rameno do scény, je pomocou modelu generovaného CAD systémom. Takmer každý výrobca poskytuje na svojich webových stránkach 3D modely rôznych typov a formátov. Model je často dostupný ako celý, nedeliteľný objekt alebo niekoľko objektov, z ktorých sa dá poskladať celé robotické rameno. Pre webovú aplikáciu je vhodná druhá možnosť - robotické rameno rozložené do niekoľkých častí, keďže robotické rameno má byť pohyblivé vo všetkých osiach. Na obrázku 4.1 je principiálne zobrazená pozícia ôs a smer ich rotácie.



Obr. 4.1: Ukážka pozície pohyblivých ôs a ich smeru rotácie [30]

Výrobcovia ďalej poskytujú jednotlivé časti robotického ramena v rôznych súborových typoch. V rámci webovej aplikácie bol opäť zvolený typ súboru vo formáte

*.stl kvôli možnosti znovu použitia funkcií využitých pri načítaní užívateľom definovaného virtuálneho bodu.



Obr. 4.2: Časti robotického ramena IRB120_5 pred zostavením v prostredí 3D Builder

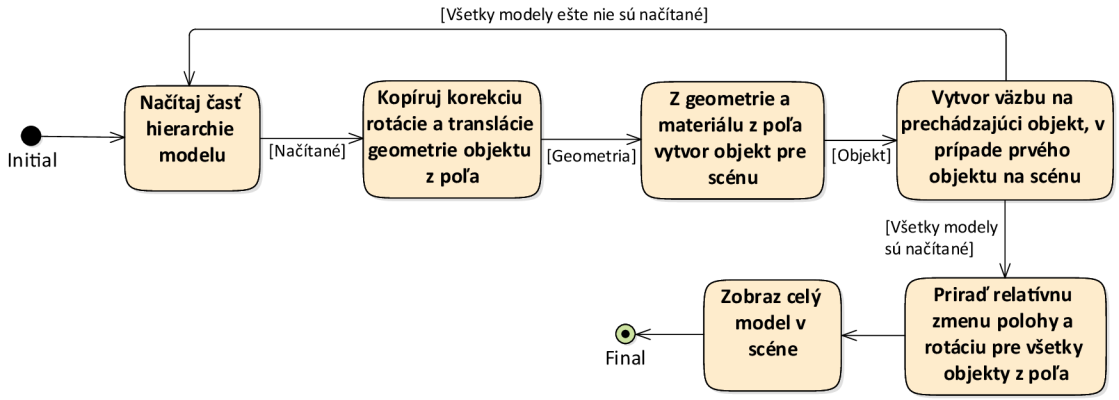
V rámci diplomovej práce bola vytvorená trieda s názvom `createModelsHierarchy (scene, loader, hierarchyType)`. Trieda je vytvorená natoľko dynamicky, že môže byť použitá na akúkoľvek hierarchiu modelov, ktoré majú vykonávať pohyb. Prvým vstupom pre konštruktor triedy je požadovaná scéna na zobrazenie hierarchie objektov. Ďalším vstupom je objekt, ktorý sa stará o načítavanie modelov generovaných CAD systémom, v našom prípade pre súborový typ *.stl. Posledným parametrom je špeciálna štruktúra, v ktorej sú definované podstatné parametre pre vytvorenie hierarchie modelov. Inštanciou triedy vzniká objekt, ktorý je možné zobraziť alebo skryť zo scény. Objekt taktiež obsahuje priamy prístup k ovládaniu rotácie a pohybu jednotlivých dielov. Vyššie spomenutá vstupná štruktúra, v našom prípade pomenovaná *hierarchyTypes*, obsahuje pole štruktúr, v ktorom každá štruktúra musí definovať nasledujúce položky:

- Cesta k priečinku s jednotlivými modelmi vo formáte string.
- Pole názvov modelov umiestnených v priečinku vo formáte string.
- Pole materiálov, ktoré definujú farbu každej časti vo formáte materiálu z rozhrania THREE.js.
- Pole korekcií rotácie geometrie načítanej časti modelu vo formáte: x, y, z. Korekcia uhlu rotácie je zadávaná v radiánoch.
- Pole korekcií translácie geometrie načítanej časti modelu vo formáte: x, y, z. Tento typ korekcie sa využíva na posun stredu rotácie. Korekcia zmeny pozície je zadávaná v centimetroch.
- Pole relatívnych posuvov častí modelu k predchádzajúcej časti modelu. V prípade prvej časti je pozícia relatívna k scéne. Vstupný formát je v dátovom

type THREE.Vector3.

- Pole relatívnych rotácií častí modelu k predchádzajúcej časti modelu. V prípade prvej časti je rotácia relatívna k scéne. Vstupný formát je v dátovom type THREE.Euler3.
- Pole indexov, ktorých hodnota určuje, vzhľadom k akému má byť časť hierarchie relatívna.

Na obrázku 4.3 je zobrazený stavový diagram popisujúci všeobecný proces načítania častí generovaných CAD systémom. V prvej časti diagramu sú načítané časti hierarchie. V našom prípade sú načítané jednotlivé časti robotického ramena. Po načítaní časti robota sa kopíruje z definovaných špecifikácií korekcia geometrie objektu. Z programu je stred rotácie automaticky centrovaný na stred objektu, v špecifických prípadoch, napríklad pri určení stredu rotácie osi robota, je treba stred rotácie zmeniť pomocou translácie geometrie. Po korekcii je pripravená geometria časti robota. Ďalej je z poľa špecifikácii dielu získaný materiál. Následne je pomocou triedy `Mesh` z rozhrania THREE.js vytvorený objekt kombináciou geometrie a materiálu. Po vytvorení objektu môže byť objekt pridelený do hierarchie už vytvorených objektov. Hierarchia je tvorená na základe čísla dielu, ktorý je uvedený v poli so špecifikáciami. Pokiaľ sú všetky časti úspešne načítané do hierarchie, ktorá tvorí celý model robota, je kopírované celé pole relatívnych zmien pozície a rotácie pre každý diel. V poslednom kroku sa zobrazí celý robot v scéne. Pokiaľ je načítanie niektorého z dielov neúspešné, hierarchia modelov nie je zobrazená v scéne.



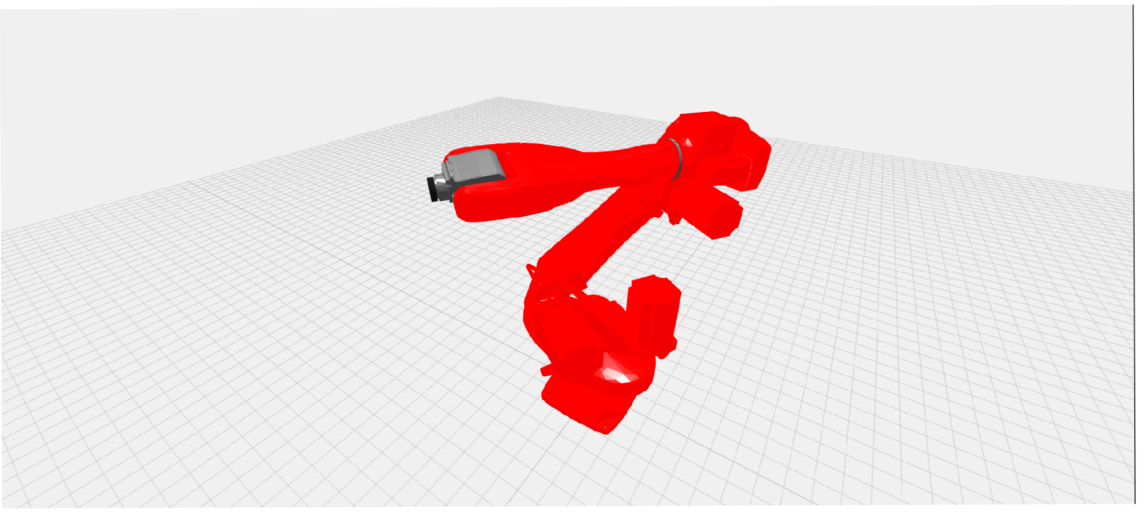
Obr. 4.3: Stavový diagram pri načítaní hierarchie celého modelu pomocou triedy createModelsHierarchy

Do scény boli implementované špecifikácie pre 3 typy robotických ramien. Jedná sa o nasledujúce typy:

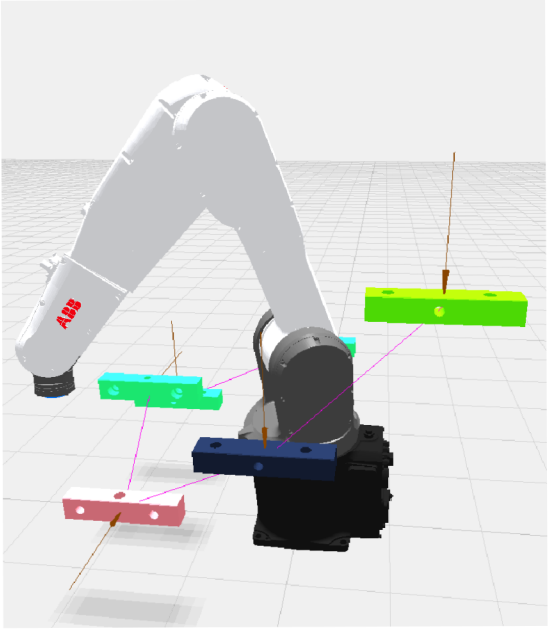
- IRB_120_3kg_0.58m od firmy ABB. [19]

- IRB_1200_5kg_0.9m od firmy ABB. [20]
- Racer-7-1.4 od firmy Comau. [21]

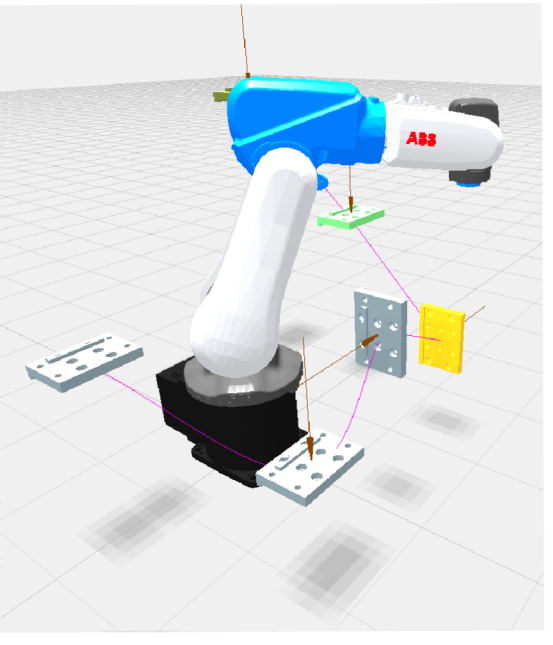
Vyššie spomenuté robotické ramená boli zvolené pre implementáciu do webovej aplikácie z dôvodu dostupnosti reálneho robotického ramena.



Obr. 4.4: Racer-7-1.4 od firmy Comau



Obr. 4.5: IRB_1200_5kg_0.9m od firmy ABB



Obr. 4.6: IRB_120_3kg_0.58m od firmy ABB

4.2 Komunikácia

Ako prvý komunikačný protokol bude uvedený OPC UA. V túto chvíľu protokol nie je priamo podporovaný webovými technológiami aj napriek predispozíciam, na druhú stranu má silné zastúpenie v priemyselnej automatizácií, v ktorej majú veľké zastúpenie aj robotické ramená a ich riadenie. Preto bude uvedené univerzálne riešenie, pomocou ktorého sa bude možné pripojiť na ľubovoľné zariadenie podporujúce komunikácie cez OPC UA.

V ďalšej kapitole 4.2.2 bude uvedené riešenie komunikácie pre robotický kontrolér s robotickým ramenom of firmy ABB. Z dostupnej ponuky možností komunikácie, ktorá sa nachádza v kapitole 1, bol vybraný protokol Rest API, ktorý priamo podporuje webovú komunikáciu medzi robotickým kontrolérom a webovou aplikáciou.

4.2.1 OPC UA

Ako už bolo spomenuté, OPC UA je komunikačný protokol, navrhnutý na komunikáciu medzi strojmi v priemyselnej automatizácii. Táto komunikácia je populárna vďaka vlastnostiam ako:

- Voľne implementovateľný protokol pod licenciou GPL.2.0.
- Protokol nie je závislý na konkrétnom programovacom jazyku alebo operačnom systéme
- Robustná bezpečnosť

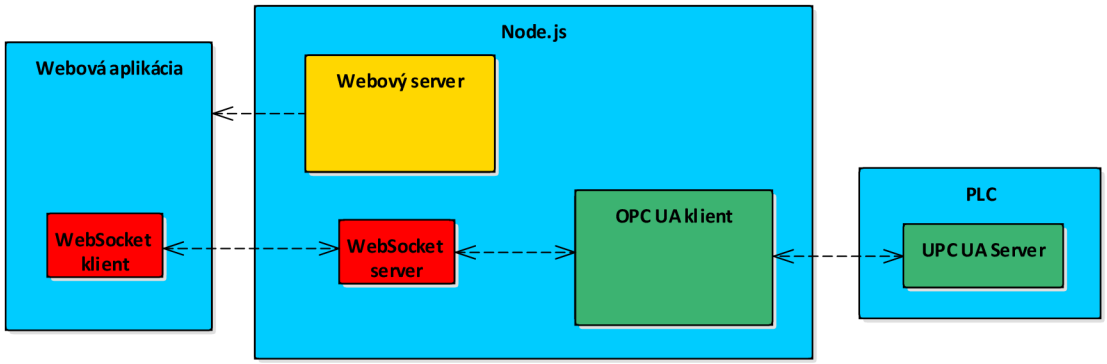
Komunikácia je rozdelená do troch relačných vrstiev, ktoré sú:

- Transportná vrstva - Táto vrstva je využívaná pre odosielanie správ. Transfer je zabezpečený tak, aby ju nemohla tretia strana čítať alebo modifikovať.
- Komunikačná vrstva - Táto vrstva je určená ako bezpečnostný kanál, ktorý je vytvorený ihneď po nadviazaní komunikácie.
- Aplikačná vrstva - Vrstva slúži na volanie alebo spracovanie služieb. Po pripojení je vytvorená relácia, ktorej musí klient predať prihlasovacie údaje a na základe toho mu budú pridelené práva.

Medzi služby patrí napríklad monitorovanie hodnoty, neustále odoberanie hodnoty, historické dáta, prenos súborov a podobne. [25][26]

V našom prípade bude využívaná implementácia v podaní softwarového systému s názvom *Node.js*. Systém je využívaný predovšetkým ako webový server. Node.js ako server je podporovaný na rôznych operačných systémom ako Windows, Mac alebo Linux. [24] Ako OPC UA server je použité PLC od firmy B&R. Ako je vidieť na obrázku 4.7, node.js slúži ako klient, ktorý sa pripojí k OPC UA serveru na PLC. Zároveň s pripájaním k PLC slúži node.js ako webový server pre webovú aplikáciu. V momente vytvorenia webového servera je vytvorený server pre komunikáciu pomocou

protokolu WebSocket. WebSocket je implementovaný pomocou API *Socket.IO*, ktorá je používaná pre jednoduchú komunikáciu v reálnom čase. V momente pripojenia užívateľa na webovú aplikáciu prebehne pripojenie na WebSocket server. V našom prípade sú následne z PLC vyčítané hodnoty ôs robotického ramena do webovej aplikácie.



Obr. 4.7: Topológia pri komunikácií PLC - webová aplikácia

Takýmto spôsobom dokáže webová aplikácia vyčítať údaje priemerne každých 25 až 40 milisekúnd, vďaka čomu je pohyb robotického ramena plynulý.

4.2.2 Rest API

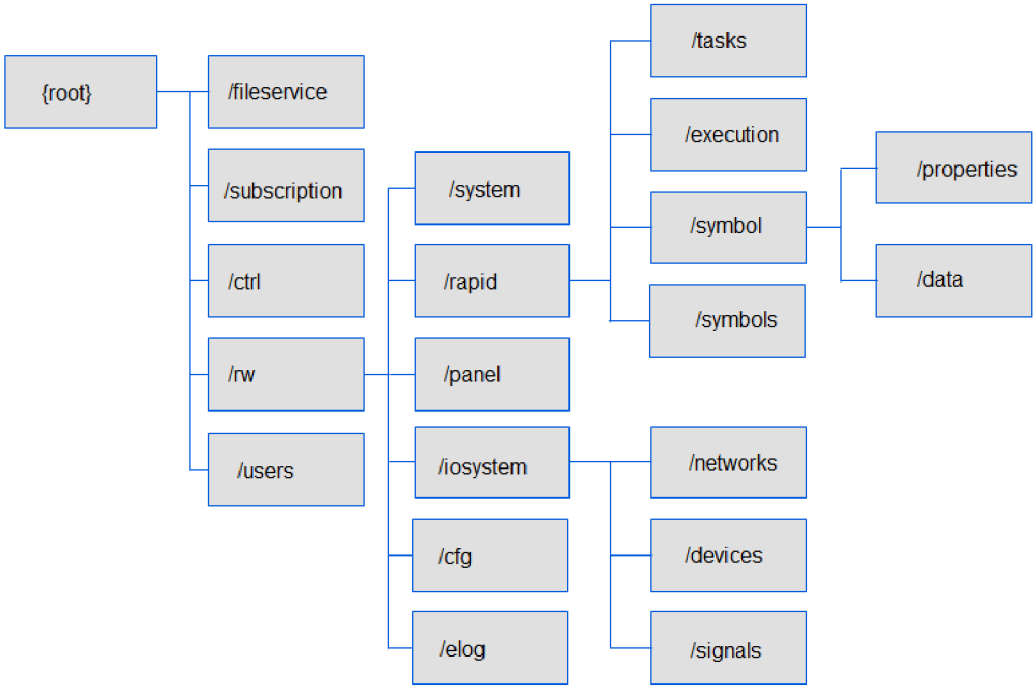
Jedná o rozhranie, ktoré je navrhnuté tak, aby dokázalo prepojiť časti programov na fyzicky oddelených zariadeniach. REST komunikácia prenáša doménovo špecifikované údaje pomocou HTTP protokolu. Stav aplikácie je možné získať pomocou unikátnej adresy zdroja (URL). REST komunikácia je vždy rozdelená na klient-server. V nasledujúcom rozdelení sú popísané základne HTTP metódy, ktoré sa využívajú pri REST komunikácii.

- GET - metóda používaná pre vyžiadanie určitých dát od servera/klienta.
- PUT - metóda používaná pre požiadavku zápisu dát. Všeobecne používané na obnovenie existujúcich dát.
- POST - metóda používaná pre požiadavku zápisu dát. Všeobecne používané na vytvorenie nových operácií alebo dát.
- DELETE - metóda používaná na požiadavku pre odstránenia dát.

Pri výmene dát sú často údaje v textovom formáte JSON. [27] [28]

Na obrázku 4.8 je možné vidieť všetky sekcie, ku ktorým je možné pristupovať cez REST API v robotickom kontroléri od firmy ABB. Z rozdelenia vyplýva, že takmer každá časť nastavení robotického kontroléra je dostupná z webového prostredia.

Komunikácia je dostupná aj v simulácii pomocou virtuálneho kontroléra. Z dôvodu veľkého množstva oblastí, ktoré ponúka REST server od ABB, bude popísaná iba konkrétna oblasť záujmu.



Obr. 4.8: Rozdelenie sekcií, ku ktorým je možné pristupovať cez komunikáciu REST [29]

Pre reálny pohyb virtuálneho robotického ramena vo webovej aplikácií je nutné získať aktuálne polohy ôs. Z dostupnej dokumentácie, ktorej časť je zobrazená na obrázku 4.8,bolo získané URL pre vyčítanie polohy ôs. Príkladom takejto adresy je *http://127.0.0.1/rw/rapid/tasks/T_ROB1/motion?resource=jointtarget&json=1*. [29] V tejto URL sú podstatné nasledujúce časti:

- 127.0.0.1 - IP adresa robotického kontroléru, v našom prípade robotického virtuálneho kontroléru, ktorý je dostupný v simulácii.
- rw - obsluhuje služby, napríklad ako vstupy, výstupy, RAPID programy a podobne.
- rapid - oblasť pre prácu s textovým programom robotického ramena. Viac o tejto časti je napísane v kapitole 1.
- tasks/T_ROB1 - program robota, ktorý obsahuje systémové moduly, v ktorých sú definované užívateľské dáta, systémové dáta a podobne.
- motion - informácia pochádza z pohybových dát.

- ?resource=jointtarget&json=1 - za otáznikom sa nachádzajú URL parametre. Prvým je požiadavka na polohy ôs. Pomocou znaku & je možné zadať ďalšie parametre, v našom prípade je požadované, aby boli hodnoty vrátené v textovom formáte JSON.

Pri získavaní dát bola použitá metóda GET. Pri každom vyčítaní alebo zapisovaní údajov je potrebné prejsť autentifikáciou užívateľa. V našom prípade je použité predvolené meno *DefaultUser* a heslo *robotics*. Na získaných dátach musela byť použitá nasledujúca korekcia z dôvodu rozdielnosti orientácie osí. Pre príklad bude použitý robot rady IRB1200_5 od firmy ABB. Počet ôs, ich umiestnenie, smer rotácie a poradie je zhodné s obrázkom 4.1

- os 1: $\pi/180$
- os 2: $-\pi/180$
- os 3: $-\pi/180$
- os 4: $\pi/180$
- os 5: $-\pi/180$
- os 6: $\pi/180$

V rámci diplomovej práce bola vytvorená funkcia s názvom `httpRequest(method, url, paramters)`, ktorá sa stará o správu REST komunikácie. Prvým vstupným parametrom je požadovaná HTTP metóda. Druhým parametrom je URL adresa serveru, v našom prípade robotického kontroléru. Posledným parametrom sú voliteľné parametre. Funkcia ma asynchrónny výstup, ktorý je generovaný pri odpovedi zo servera. Z dôvodu modularity bola webová aplikácia vybavená voliteľným parametrom URL, ktorý sa zadáva na koniec jej adresy. Ako voliteľný parameter môže byť použitý napríklad *"?JointSource=127.0.0.1"*. *JointSource* je nemenné, ale IP adresa môže byť nahradená za adresu robotického kontroléru. Následne je táto IP adresa implementovaná aj do REST komunikácie. Za IP adresu môže byť použité aj kľúčové slovo *Simulation*. V tomto prípade bude zdrojom pohybu ôs pole hodnôt získaných z reálneho pohybu robotického ramena po definovanej trajektórii. Pri zadaní IP adresy robotického kontroléru je vyššie spomenutá procedúra získavania polohy ôs vykonávaná cyklicky každých 20 milisekúnd. Pri tomto časovom intervale je získavaná poloha 50-krát za sekundu, čím vzniká plynulý pohyb robotického ramena s minimálnym oneskorím oproti reálnemu, poprípade simulovanému robotickému ramenu.

4.3 Export trajektórie do robotického kontroléru ABB

Ešte pred samotným exportom trajektórie do kontroléru robotického ramena je nutné pripraviť kód, ktorý sa bude vykonávať. Kód je uložený v moduloch, ktoré sú súčasťou programu robota v časti RAPID. Časti kódu, ktoré v module určujú

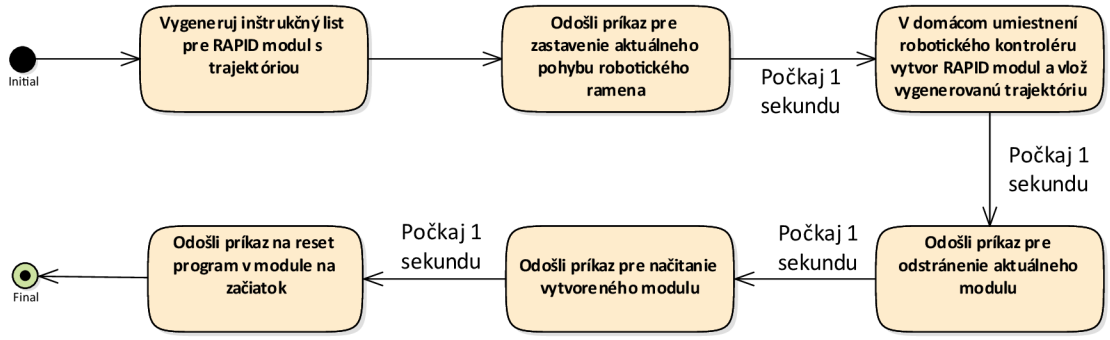
trajektóriu, sa nazývajú pohybové inštrukcie. V našom prípade bude platiť, že počtu vygenerovaných vektorov z trajektórie webovej aplikácie sa bude rovnať počet pohybových inštrukcií RAPID modulu. Ako je vysvetlené v kapitole 3.3.6, trajektória je pripravená v univerzálnom tvare pre ďalšie použitie a obsahuje pole všetkých vektorov ktoré tvoria trajektóriu. Z ponuky pohybových inštrukcií bola vybraná inštrukcia s názvom *MoveJ*. Inštrukcia sa používa pri rýchlom presune robota z jedného bodu do druhého. Zároveň je rýchlosť každej osy interpolovaná tak, aby všetky osy dosiahli cieľovú polohu zároveň. Tento pohyb môže spôsobovať nelinearity, ktoré sú na krátke vzdialenosti zanedbateľné. [1]

V rámci diplomovej práce bola vytvorená knižnica *exportABB.js*, ktorá obsahuje funkcie pre prípravu inštrukčného listu a exportu dát do robotického kontroléru. O prípravu inštrukčného listu sa stará 27 funkcií, ktoré skladajú jednotlivé časti inštrukcie do jedného celku. Hlavná funkcia, ktorá využíva zvyšných 26 funkcií, sa nazýva `getRapidFileStr(exportedData)`. Vstupným parametrom funkcie je celá univerzálna štruktúra generovaných vektorov, ktorá je popísaná v kapitole 3.3.6. Výstupom funkcie je text, ktorý obsahuje celý inštrukčný list a všetky náležitosti, ktoré vyžaduje RAPID kompilátor. V prílohe C je priložený príklad výstupného textu funkcie getRapidFileStr. V nasledujúcej časti budú vysvetlené časti pohybovej inštrukcie MoveJ, ktoré sú definované v rámci funkcie.

- ToPoint: Jedná sa o cieľovú destináciu a rotáciu koncového bodu robotického ramena v priestore. Informácia je reprezentovaná vo štvorici quaternionov pre pozíciu a v ďalších štyroch quaternionoch pre rotáciu. Informácia je z Eulerových uhlov konvertovaná do quaternionov pomocou funkcie z rozhrania THREE.js.
- Speed: rýchlosť na presun je pri každej pohybovej inštrukcií nastavená na 300 milimetrov za sekundu. Rotácia je nastavená na 150 stupňov za sekundu. Jedná sa o relatívne pomaly pohyb, ktorý je možné zmeniť.
- Zone: Pre plynulý pohyb cez všetky pohybové inštrukcie bola zvolená systémová premenná *z10*.

Druhá časť knižnice je zameraná na export vygenerovaného inštrukčného listu do robotického kontroléru. O export trajektórie sa stará funkcia `sentRapidStr()`, ktorá využíva komunikáciu REST z kapitoly 4.2.2. Na obrázku 4.9 je zobrazený stavový diagram pri procese exportovania trajektórie do robotického kontroléru. V prvom stave je generovaná trajektórie v tvare RAPID modulu, ktorý je popísaný v prvej časti tejto kapitoly. Následne je odoslaná požiadavka na zastavenie aktuálneho programu `/rw/rapid/execution?action=stop` pomocou POST metódy. V ďalšej časti je vytvorený súbor v domovskom umiestnení robotického kontroléru. Príkaz `/fileservice/$home/DP.mod` je odosielaný PUT metódou. V odosielanej

URL, časť s názvom */fileservice*, umožňuje prístup priamo do súborového priestoru robotického kontroléru cez REST komunikáciu. Ďalšia časť *$home* popisuje priečinok a na koniec DP.mod určuje názov vytvoreného súboru. V prípade, že súbor existuje, je nahradený. S týmto príkazom je odosielaný aj parameter, ktorý obsahuje celý inštrukčný list, ktorý je následne vložený do súboru DP.mod. Následne je odoslaný príkaz `/rw/rapid/tasks/T_ROB1?action=unloadmod` pomocou POST metódy, ktorý zmaže aktuálne používaný modul. Po úspešnom odstránení je odoslaný príkaz `/rw/rapid/tasks/T_ROB1?action=loadmod` pomocou POST metódy pre načítanie novovytvoreného modulu s názvom DP.mod. Nakoniec je pomocou príkazu `/rw/rapid/execution?action=resetpp` resetovaný program tak, aby sa vykonával od začiatku. [29]



Obr. 4.9: Stavový diagram pri generovaní a exportovaní trajektórie do robotického kontroléru pomocou funkcie getRapidFileStr

Na obrázku 4.10 sa nachádza výstrižok z vytvoreného videa pri pohybe reálneho a virtuálneho robotického ramena. Toto video je priložené v prílohe B.

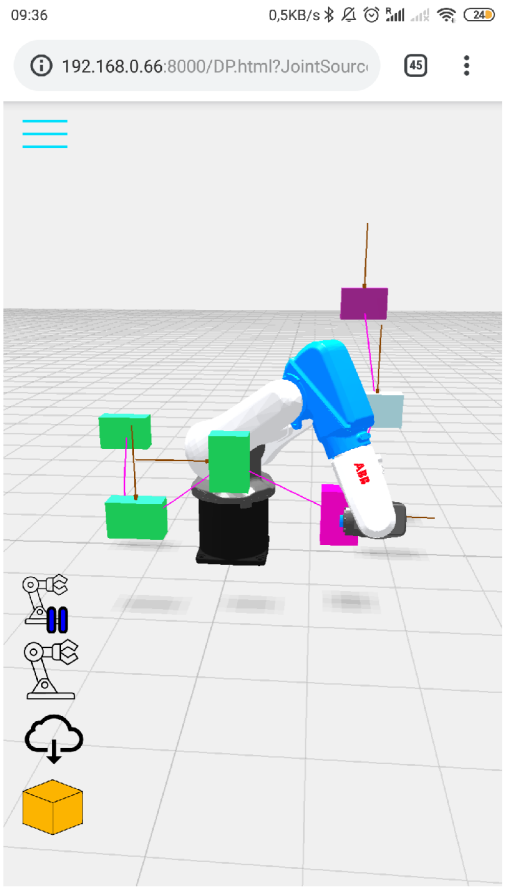
Obr. 4.10: Porovnanie reálnej a virtuálnej scény

5 Grafické užívateľské rozhranie webovej aplikácie

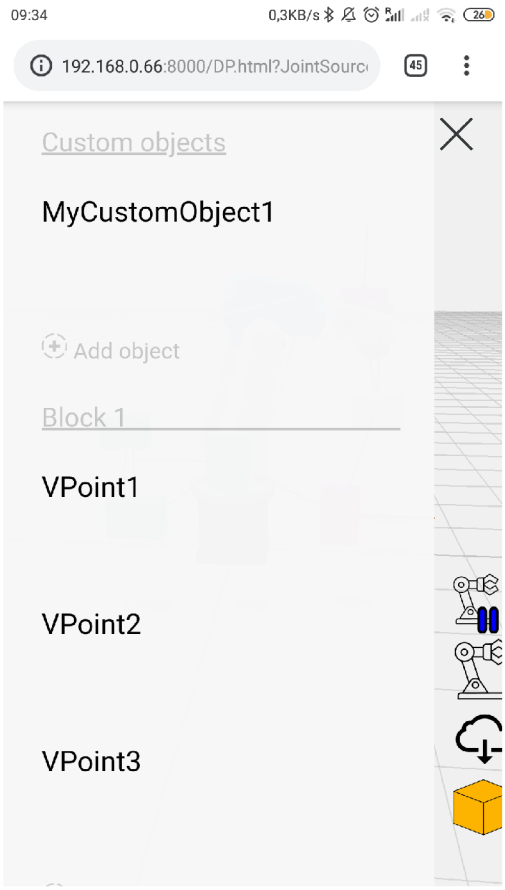
Grafické užívateľské rozhranie, inak nazývané GUI, je z pohľadu užívateľa jedna z najpodstatnejších vlastností aplikácie. Spôsob, akým je možné interagovať s prostredím určuje užívateľský zážitok z webovej aplikácie a taktiež dokáže výrazným spôsobom urýchliť vykonávanú prácu užívateľa. Nasledujúce kapitoly sa budú zaoberať najmä grafickými vstupmi a výstupmi, pomocou ktorých môže užívateľ ovládať celú aplikáciu.

5.1 Prehľad a správa virtuálnych bodov

Ako už bolo spomenuté v predchádzajúcich kapitolách, webová aplikácia musí poskytovať vhodný spôsob pre užívateľa ako pridávať alebo odstraňovať virtuálne body. Ďalej je predpoklad, že užívateľ pri svojej práci vo webovej aplikácií vytvorí väčšie množstvo virtuálnych bodov, ktoré budú definovať trajektóriu. S komplexnosťou trajektórie sa ale znižuje prehľadnosť v scéne. Prvky na sprehľadnenie scény, ako napríklad farebné rozlíšenie blokov, síce napomáhajú užívateľovi, ale v určitom momente už nemusia byť dostatočne účinné. Preto bolo v rámci diplomovej práci vyvinuté grafické užívateľské rozhranie podobné zoznamu, v ktorom je možné spravovať virtuálne body alebo celé bloky. Na obrázku 5.3 v ľavej časti je zobrazený výsuvný panel, ktorý slúži pre správu virtuálnych bodov. Veľká časť výsuvného panelu využíva vlastnosti a animácie tretej verzie kaskádových štýlov (CSS3). Výsuvný panel je možné zobraziť kliknutím na modrý znak ”≡”, ktorý sa nachádza v ľavej vrchnej časti webovej aplikácie ako je vidieť na obrázku 5.1. Po kliknutí sa vysunie do podoby, ktorá je na obrázku 5.3 a 5.2 a modrý znak sa transformuje na krížik. Šírka výsuvného panelu je dynamická a mení sa relatívne so šírkou okna prehliadača. Vďaka tejto vlastnosti je možné pohodlne pracovať s výsuvným panelom aj na mobilnom zariadení.



Obr. 5.1: Zasunutý bočný panel pre správu virtuálnych bodov a objektov na mobilnom zariadení



Obr. 5.2: Vysunutý bočný panel pre správu virtuálnych bodov a objektov na mobilnom zariadení

Každý prvok výsuvného panelu je štrukturovaný do niekoľkých úrovní, čím je vytvorená modularita. V nasledujúcom rozdelení je zobrazená štruktúra každého bloku alebo vlastných objektov v scéne.

```
Bloky
├── Hlavička bloku/vlastných objektov v scéne
├── Blok/vlastné objekty
│   ├── Virtuálne body/objekty
│   │   ├── Text virtuálneho bodu/objektu
│   │   ├── Tlačidlo pre odstránenie bodu/objektu
│   │   └── Tlačidlo pre premenovanie bodu/objektu
│   └── Tlačidlo pre pridanie nového virtuálneho bodu/objektu
├── Hlavička bloku/vlastných objektov v scéne
└── Blok/vlastné objekty
    └── ...
```

V rámci optimalizácie je užívateľské rozhranie pre správu virtuálnych bodov tvorené dynamicky. To znamená, že text, ktorým je tvorený výsuvný panel, je vytvorený alebo zmenený až v momente udalosti. O správu výsuvného panelu sa stará knižnica s názvom *UIControls.js*. Knižnica obsahuje triedu s názvom `VPointSidepanel()`, ktorá definuje sadu metód, ktoré sa starajú o každú úroveň výsuvného panelu.

Prvou je metóda `addPointUI(blockID, objectUUID, pointName)`, ktorá vytvorí celú štruktúru pre správu virtuálneho bodu vo zvolenom bloku. Prvým vstupom metódy je pridelené meno bloku, ktoré bolo popísané v kapitole 3.3.4. Ďalším vstupom je meno novovytvoreného virtuálneho bodu, ktorého proces prideľovania mena je popísaný v kapitole 3.3.1. Posledným parametrom je menovka, ktorú dostane virtuálny bod na úrovni výsuvného panelu. Menovka je teda čisto vizuálna záležitosť. Z užívateľského hľadiska sú virtuálnemu bodu pridelené nasledujúce vlastnosti:

- Pri prechode myšou alebo kliknutí je celá časť virtuálneho bodu zvýraznená, ako je vidieť na obrázku 5.3 pod blokom, s názvom *Block1* a virtuálnym bodom *VPoint2*. Po kliknutí je volaná funkcia `showPoint(camera, pointRef)` z kapitoly 5.3, pomocou ktorej je kamera premiestnená k zvolenému virtuálnemu bodu. V prípade, že užívateľ klikne na virtuálny bod priamo v scéne, je zvolený bod zvýraznený vo výsuvnom paneli.
- Je pridané tlačidlo, pomocou ktorého je možné zmeniť názov virtuálneho bodu. Po vyplnení nového mena je volaná metóda `renamePointUI(objectUUID, newName)`.
- Je pridané tlačidlo, pomocou ktorého je možné zmazať virtuálny bod zo scény aj z výsuvného panelu. Po stlačení tlačidla je volaná metóda `RemovePoint (pointUUID)`, ktorá je vysvetlená v kapitole 3.3.1.

Pre prípad zmeny prideleného UUID virtuálneho bodu bola vytvorená metóda `changePointUUID(oldPointUUID, newPointUUID)`, ktorá túto zmenu vykoná aj na úrovni užívateľského rozhrania, v opačnom prípade by tlačidlo bolo nefunkčné. Vyššie popísané vlastnosti sa uplatňujú aj pri pridávaní vlastného objektu do scény.

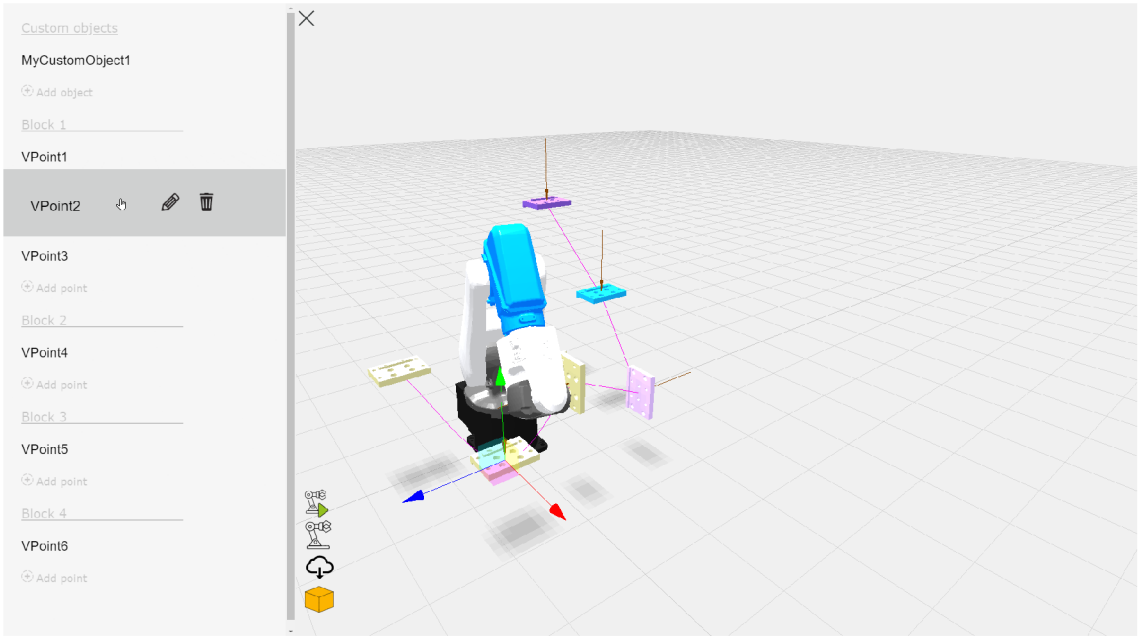
Ďalšia metóda, ktorá sa zaoberá tvorbou užívateľského rozhrania, má názov `addBlockUI(blockID, blockName)`. Vstupnými parametrami je UUID nového bloku a názov, ktorý sa zobrazí užívateľovi. Pri volaní metódy sú vytvorené nasledujúce vlastnosti:

- Hlavička celého bloku, ktorá obsahuje názov bloku z parametru *blockName*. Po kliknutí na hlavičku bloku je volaná metóda `addBlock()` z knižnice *VPLib*, ktorá vytvorí ďalší blok.
- Štruktúru, do ktorej je možné vkladať virtuálne body.
- Tlačidlo pre pridanie nového virtuálneho bodu do bloku. Tlačidlo sa nachádza na úrovni virtuálnych bodov a taktiež zdieľa ich grafické vlastnosti. Toto

tlačidlo sa ale vždy nachádza na konci zoznamu virtuálnych bodov. Po jeho kliknutí je volaná metóda `addPoint( blockID , customPoint)`, ktorá je popísaná v kapitole 3.3.1. Každý blok obsahuje vlastné tlačidlo pre pridávanie virtuálnych bodov.

Pri odstraňovaní bloku z grafického užívateľského prostredia je volaná metóda s názvom `removeBlockUI(blockID)`. V prípade, že sa vo výsuvnom paneli nachádza väčšie množstvo blokov alebo virtuálnych bodov ako je možné zobraziť, je automaticky pridaný posuvník, ktorý je zobrazený na obrázku 5.3.

Výsuvný panel obsahuje aj možnosť načítania vlastných objektov do scény s hlavičkou *Custom Objects*. Táto štruktúra je ale statická, to znamená, že nie je tvorená dynamicky a taktiež nie je odstránená v prípade, že neobsahuje žiadne objekty. Štruktúra sa nachádza vždy na prvom mieste v zozname. Z grafického hľadiska je podobná štruktúre blokov a ako potomok obsahuje jednotlivé načítané objekty. K načítaným objektom je možné pristupovať ako k virtuálnym bodom, teda je možné ich premenovať alebo odstrániť. Po kliknutí na zvolený objekt je kamera premiestnená pomocou funkcie `showPoint(camera, pointRef)`. Po kliknutí na tlačidlo pridania nového objektu je zobrazené okno, ktorým si môže užívateľ vybrať voliteľný model v súborom type .*stl. Po zvolení je volaná funkcia `readSingleFile()`, ktorá načíta a zobrazí objekt.



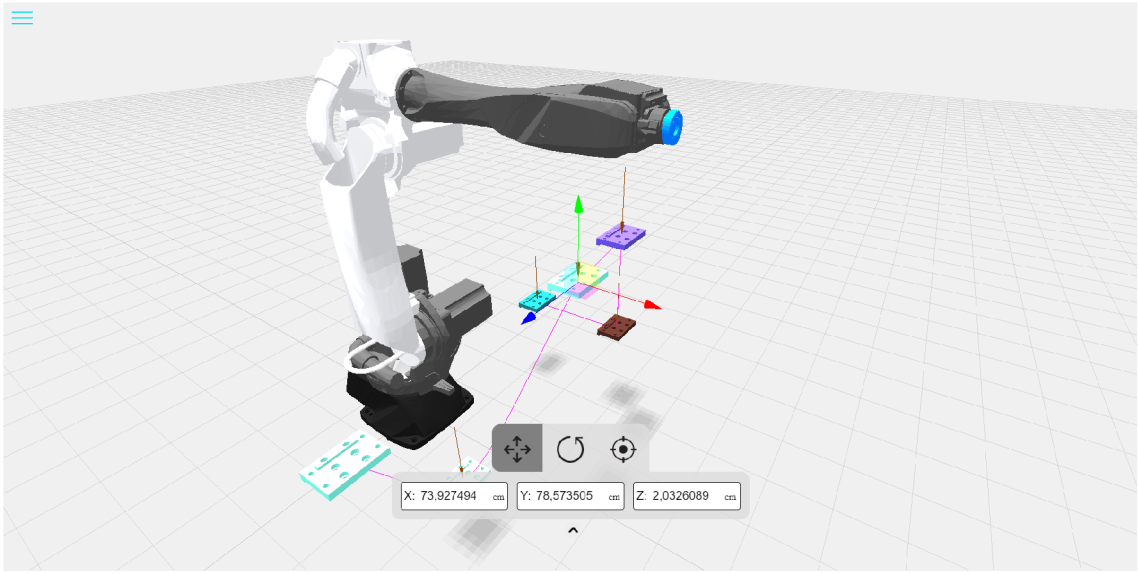
Obr. 5.3: Vysunutý bočný panel pre správu virtuálnych bodov a objektov

5.2 Zobrazenie polohy virtuálnych bodov

Pri tvorbe trajektórie je užívateľ často v situácii, kedy nepotrebuje poznať presnú pozíciu virtuálnych bodov a postačuje mu približná pozícia od stredu scény, kde sa nachádza robotické rameno. Ale v situácii plánovania trajektórie podľa presných súradníc, poprípade konkrétnej rotácie virtuálneho bodu, potrebuje užívateľ rozhranie, pomocou ktorého môže kontrolovať alebo zapisovať údaje. Pre užívateľa bol vytvorený panel, ktorý je zobrazený na obrázku 5.4 v spodnej časti. Panel ponúka nasledujúce možnosti:

- Po kliknutí na logo protichodných šípok je možné zobraziť aktuálnu polohu virtuálneho bodu v súradniciach X, Y, Z. Po kliknutí na konkrétnu súradnicu je možné meniť polohu v rozsahu ± 1000 centimetrov.
- Po kliknutí na logo rotujúcej šípky je možné zobraziť aktuálnu rotáciu virtuálneho bodu v súradniciach X, Y, Z. Po kliknutí na konkrétnu súradnicu je možné meniť rotáciu v rozsahu 360°
- Po kliknutí na logo centrovania je stred rotácie kamery premiestnený do stredu zvoleného virtuálneho bodu.

Panel pre zobrazenie polohy sa zobrazí vždy po kliknutí na virtuálny bod. Po zobrazení sa spustí časovač. Pokiaľ užívateľ znova klikne na virtuálny bod alebo panel, časovač sa resetuje. V opačnom prípade je panel skrytý. Časť panelu, ktorá užívateľovi zobrazuje informácie o polohe alebo rotácii virtuálneho bodu, je možné schovať po kliknutí na spodnú časť panelu.



Obr. 5.4: Vstupno-výstupný panel pre zmenu pozície, rotácie a centrovanie virtuálneho bodu

5.3 Náhľad kamery na zvolený bod

V prípade udalosti virtuálneho bodu alebo potreby užívateľa vyhľadať v scéne určitý objekt, je najvhodnejšie plynule premiestniť kameru priamo ku konkrétnemu miestu udalosti. Pri plynulom pohybe kamery užívateľ nestratí orientáciu v scéne a zároveň zaznamená cestu k zvolenému umiestneniu. Z vyššie spomenutých špecifikácií bola vytvorená funkcia s názvom `animateCamera(camera, routePoints, pointOfInterest)` v knižnici s názvom *animateCamera.js*. Prvým vstupom je kamera, ktorá ma byť premiestnená. Ďalším vstupom funkcie je pole vektorov, ktoré budú udávať trajektóriu kamery. Posledný parameter je voliteľný a určuje, či sa má v priebehu premiestňovania kamera automaticky otáčať na zvolené miesto. Kamera potom každých 20 milisekúnd inkrementuje svoju polohu a pre užívateľa sa pohyb javí ako plynulý.

O vytvorenie trajektórie pre kameru sa stará funkcia s názvom function `showPoint (camera, pointRef)`, ktorá je súčasťou knižnice *animateCamera.js*. Prvým vstupným parametrom je opäť zvolená kamera. Druhým parametrom je celý objekt, ktorým v našom prípade môže byť virtuálny bod alebo vlastný model. Na obrázku 5.5 je zobrazená situácia, kedy sa kamera nachádzala na mieste so súradnicami [175;189;268]. Z výsuvného panelu bol zvolený 3. bod a kamera sa premiestnila po trajektórii zvýraznenej zelenou čiarou až k vybranému virtuálnemu bodu. Vo webovej aplikácii táto čiara ani s objektmi v tvare modrej gule nie je viditeľná. Trajektória je vždy vytvorená z troch bodov. Jediným známym bodom je pozícia kamery, zvyšné body trajektórie boli vypočítané predĺžením priesečníku, ktorý vznikol medzi počiatkom scény so súradnicami [0;0;0] a stredom zvoleného virtuálneho bodu. Priesečník je na obrázku 5.5 zvýraznený červenou šípkou smerujúcou do počiatku scény. V nasledujúcej časti sa nachádzajú rovnice, pomocou ktorých je počítaná neznáma poloha bodov, ktoré tvoria trajektóriu pre presun kamery. Prvá rovnica 5.1 popisuje výpočet vzdialenosti medzi počiatkom so súradnicami [0;0;0] a stredom virtuálneho bodu. V rovnici nie je uvedený rozdiel dvoch Pytagorových viet z dôvodu nulového výsledku počiatku.

$$s_1 = \sqrt{x^2 + y^2 + z^2} \tag{5.1}$$

kde
- s_1 vzdialenosť medzi počiatkom so súradnicami [0;0;0] a stredom virtuálneho bodu [cm]
- x súradnica x pozície virtuálneho bodu [cm]
- y súradnica y pozície virtuálneho bodu [cm]
- z súradnica z pozície virtuálneho bodu [cm]

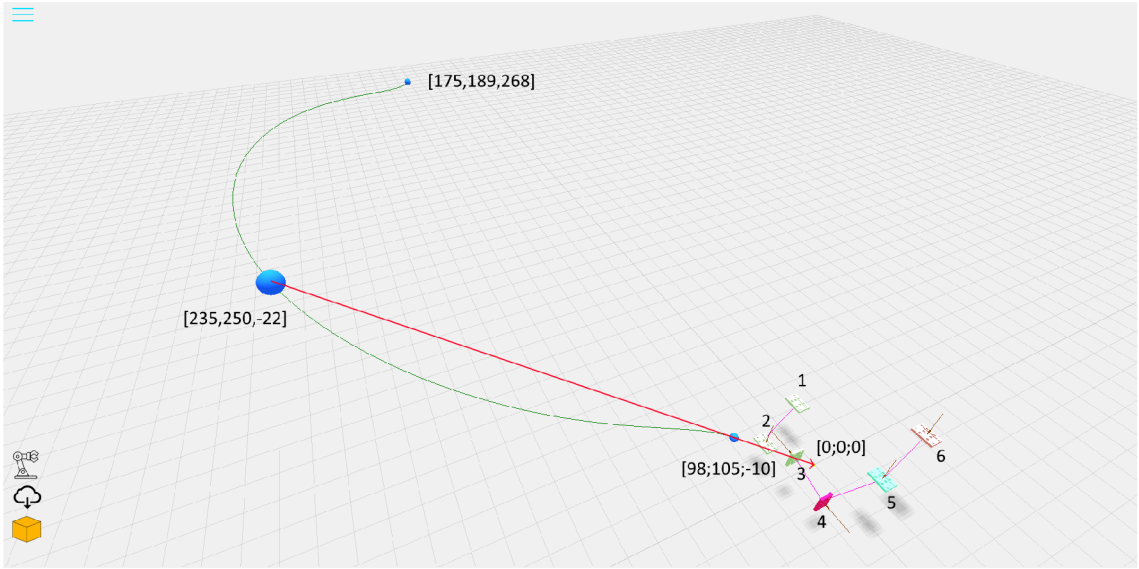
Následne rovnice 5.2 a 5.3 popisujú výpočet neznámych bodov trajektórie s pevne zadanými konštantami.

$$s_2 = \left| \frac{s_1 + 100}{s_1} \right| \tag{5.2}$$

kde s_2 vzdialenosť medzi počiatkom so súradnicami [0;0;0] a posledným bodom trajektórie kamery [cm]

$$s_3 = \left| \frac{s_1 + 300}{s_1} \right| \tag{5.3}$$

kde s_3 vzdialenosť medzi počiatkom so súradnicami [0;0;0] a prostredným bodom trajektórie kamery [cm]

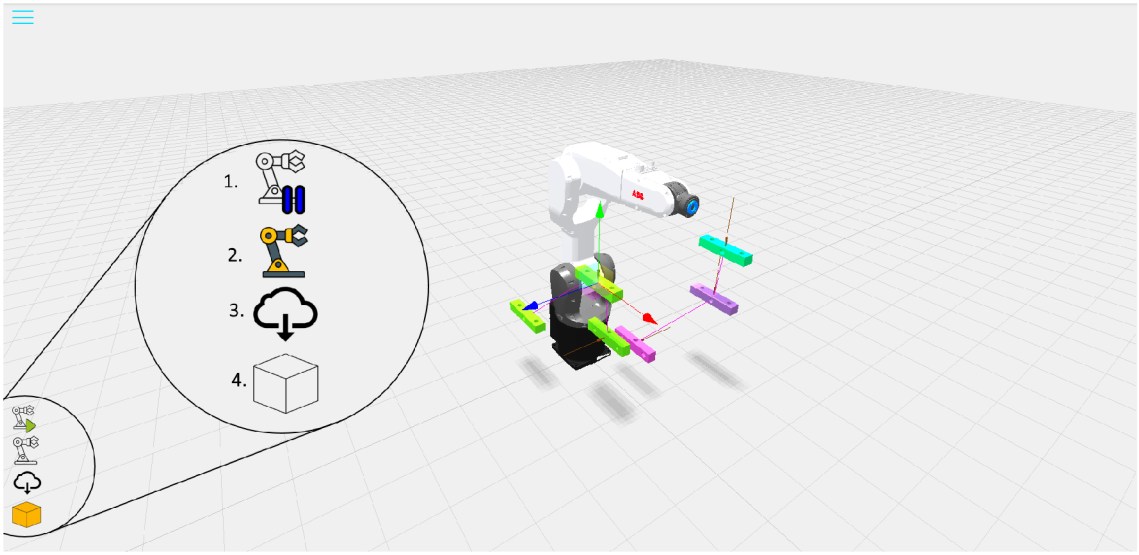


Obr. 5.5: Vizualizácia trajektórie pri pohybe kamery k objektu

Z počiatočnej pozície kamery, ktorá predstavuje jeden bod trajektórie a zo zvyšných dvoch bodov, ktoré boli vypočítané, bol vytvorený splajn typu CatmullRom. Výsledkom je hladká trajektória, ktorá sa vo svojom strede začne meniť tak, aby bol koniec trajektórie kamery súčasťou priesečníku smerom k počiatku scény. Týmto spôsob sa aj po zmene pozície kamery dokáže užívateľ orientovať v priestore a zároveň je kamera rotovaná smerom do stredu scény, kde bude pravdepodobne najväčšia koncentrácia virtuálnych bodov.

5.4 Globálne nastavenia aplikácie

Webová aplikácia obsahuje niekoľko ovládacích prvkov, pomocou ktorých je možné ovládať externý hardware alebo celú funkcionalitu aplikácie. Na obrázku 5.6 je zobrazená kompletná ponuka tlačidiel, ktoré môže mať užívateľ k dispozícii. V ich zobrazenom detaile je každé tlačidlo zobrazené s indexom a v invertovanom stave. Tlačidlá sú pomocou kaskádových štýlov synchronizované tak, aby sa ich relatívna vzdialenosť od hrany výsuvného panela nemenila. To znamená, že po vysunutí výsuvného panelu sú tlačidlá stále užívateľovi stále k dispozícii.



Obr. 5.6: Prehľad globálnych nastavení webovej aplikácie

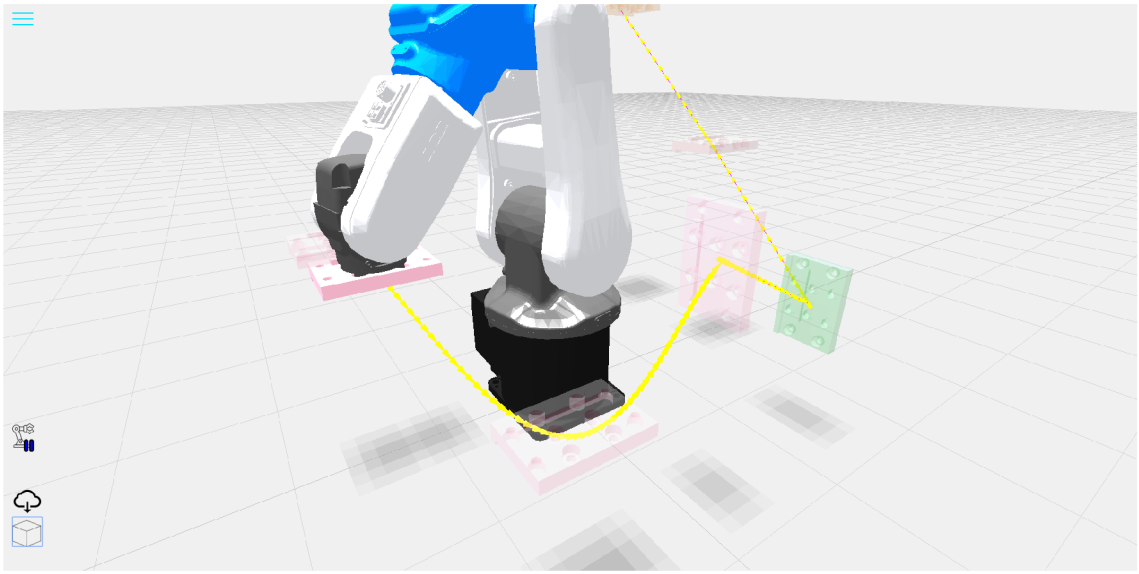
Prvé tlačidlo z ponuky označené indexom 1. slúži pre manuálne spúšťanie pohybu robotického ramena.

Ďalšie tlačidlo s indexom 2. slúži na zmenu módu vytvárania trajektórie. Webová aplikácia obsahuje mód virtuálnych bodov a mód robotického ramena, ktoré sú popísane v kapitolách 3.3.3 a 3.3.5. Po spustení webovej aplikácie je počiatočne nastavený mód virtuálnych bodov. Po kliknutí sa formou udalosti zmení tvorba trajektórie na mód robotického ramena. Zároveň so zmenou módu sa invertuje aj ikona tlačidla, ktorá užívateľovi oznamuje aktuálny mód. Po ďalšom kliknutí na tlačidlo je opäť možné tvoriť trajektóriu pomocou virtuálnych bodov. V prípade invertovaného stavu tlačidla 4. nie je tlačidlo číslo 2. viditeľné.

Tretie tlačidlo v poradí mení svoju funkcionalitu v závislosti na tom, akým spôsobom aplikácia komunikuje alebo nekomunikuje s robotickým ramenom. V prípade, že webová aplikácia nie je pripojená k žiadnemu robotickému ramenu, slúži toto tlačidlo pre ukladanie textových súborov, ktoré obsahujú vygenerovanú trajektóriu,

do zariadenia s webovou aplikáciou. V závislosti na zvolenom robotovi v scéne je exportovaný program pre konkrétne robotické rameno. Momentálne je export implementovaný len na robotické ramená od firmy ABB. K ukladaniu súborov slúži funkcia `saveAs(blob, filename)`, ktorá bola pod MIT licenciou prevzatá z internetu. [22] Pokiaľ webová komunikácia aktívne komunikuje s robotickým ramenom, slúži tlačidlo pre export a následné odoslanie programu priamo do robotického ramena. Po kliknutí tlačidlo zmizne, čo indikuje, že program sa odosiela do robotického ramena, pokiaľ sa tlačidlo zobrazí, je možné program znova exportovať.

Pomocou posledného, 4. tlačidla je možné voliť medzi módom plánovania trajektórie a módom pozorovateľa. Módom plánovania trajektórie sa zaoberá celá kapitola 3.3.4. Mód pozorovateľa slúži výhradne na sledovanie pohybu robotického ramena, na konci ktorého sa nachádza virtuálny bod. Ako je vidieť na obrázku 5.7, v tomto móde sa všetky virtuálne body stanú z časti priehľadné a nie je možné meniť ich pozíciu alebo rotáciu. Ďalej je každý vektor, ktorý tvorí trajektóriu, obohatený o žltú šípku, ktorá znázorňuje smer a orientáciu trajektórie. Kamera je automaticky centrovaná na stred scény. Pokiaľ užívateľ znova klikne na tlačidlo, mód sa zmení do tvorby trajektórie.



Obr. 5.7: Mód pozorovateľa

6 Implementácia do prostredia Automation studio

Automation studio je software navrhnutý pre programovanie PLC systémov od firmy B&R. Pomocou jedného programovacieho prostredia je možné programovať PLC systémy, vytvoriť vlastnú safety aplikáciu, riadiť pohony, vytvárať vizualizáciu, pracovať so strojovým videním a tak ďalej. Programovacie prostredie ponúka možnosť simulácie, pomocou ktorej je možné simulovať celý systém PLC. Hlavná ponuka programovacieho prostredia Automation Studio je rozdelená na tri časti:

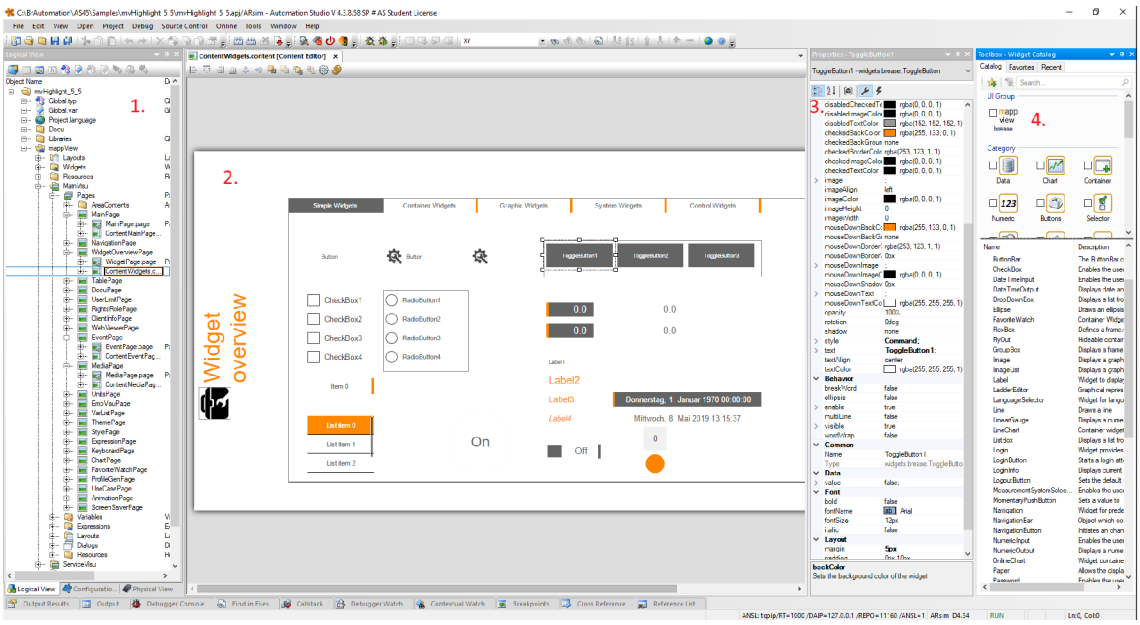
- Logical View - táto časť je zameraná najmä na programovaciu oblasť. PLC od firmy B&R je možné programovať hlavne v textových jazykoch ako ANSI C/C++, ST, ale podporované sú aj grafické jazyky ako napríklad Ladder, FBD, SFC, CFC. V tejto časti je ďalej možné vytvárať vizualizáciu, ktorou sa bude zaoberať podkapitola 6.1. Ponuku logical view je možné vidieť na obrázku 6.1 vľavo označenú indexom 1.
- Configuration View - v tejto časti je možné vytvárať konfigurácie pre rôzne hardwarové zostavy, medzi ktorými je možné prepínať.
- Physical View - obsah tejto časti sa mení v závislosti na zvolenej konfigurácií. Pomocou prehľadného editoru je možné vytvárať hardwarové zostavy.

Práca v programovacom prostredí Automation Studio bude orientovaná najmä na priemyselnú vizualizáciu, do ktorej bude implementovaná webová aplikácia vytvorená v rámci diplomovej práce. Výhodou je použité koncepčné riešenie webovej aplikácie, aby sa mohla stala súčasťou inej vizualizácie. Tým bude možné používať webovú aplikáciu aj v priemyselnom prostredí, kde bývajú najčastejšie umiestnené robotické ramená. [23]

6.1 Mapp View

Mapp View predstavuje nástroj pre návrh a realizáciu priemyselnej vizualizácie s podporou webových technológii. Riešenie Mapp View podporuje dva módy pre tvorbu vizualizácie. Prvý mód predstavuje grafické okno, ktoré je viditeľné na obrázku 6.1 označené indexom 2. Panel označený indexom 4. ponúka užívateľovi funkčné grafické prvky, ktoré sú nazývané *widgety*. Pre príklad existujú widgety pre prácu s textom, tlačidlami, obrázkami, grafmi a podobne. Užívateľ môže v grafickom okne pracovať s widgetmi pomocou dvoj-kliku na vybraný prvok alebo pretiahnutím priamo do okna. Na obrázku 6.1 je ďalej indexom číslo 3 označený panel, pomocou ktorého je možné meniť vlastnosti každého zvoleného widgetu. V tomto paneli je taktiež možné konkrétny widget prepojiť s užívateľom definovanou premennou

z programu PLC. Druhý podporovaný mód tvorby vizualizácie je pomocou textového editoru. Všetky prvky, ktoré sú zobrazené v grafickom okne, sú definované v textovej forme vo formáte XML. Pri kompilácií projektu je následne tento text automaticky prekladaný do značkovacieho jazyka HTML5, kaskádových štýlov CSS3 a programovacieho jazyka Javascript. Vďaka tomuto procesu stačia užívateľom základné znalosti z oblasti webových technológií. Priemyselná vizualizácia je následne podporovaná akýmkoľvek zariadením, ktoré podporuje aktualizovaný webový prehliadač. Komunikácia medzi PLC a zobrazovacím zariadením je realizovaná cez rozhranie OPC-UA. [23]



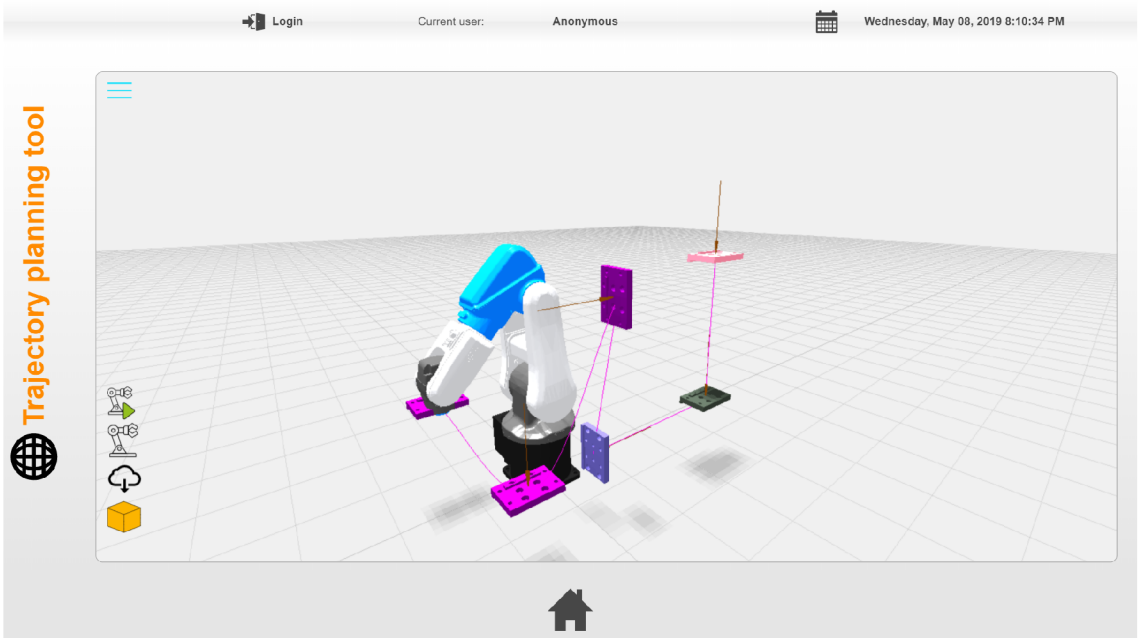
Obr. 6.1: Vývojové prostredie Automation Studio s nástrojom pre tvorbu vizualizácie Mapp View

6.2 Implementácia webovej aplikácie pre tvorbu trajektórie do priemyselnej vizualizácie

Z predchádzajúcej kapitoly vyplýva, že priemyselná vizualizácia je tvorená pomocou rôznych widgetov. Z hľadiska implementácie vlastného obsahu do priemyselnej vizualizácie ponúka Mapp View niekoľko možností, z ktorých je najvhodnejšou voľbou widget s názvom *WebViewer*. Pomocou tohoto widgetu je možné do vizualizácie načítať ľubovolnú webovú stránku, teda v našom prípade webovú aplikáciu. Widget bol nastavený na nasledujúce hodnoty:

- useURL - hodnota je nastavená na *false*, nastavenie rozhoduje o tom, či adresa webovej stránky bude poskladaná z rôznych častí alebo ako jeden text.
- src - text, ktorý určuje aká webová stránka sa má načítať. V našom prípade bol použitý text */FileDevice:CTEMP/DP/DP.html?JointSource=127.0.0.1;*, kde časť text "/FileDevice:CTEMP" určuje umiestnenie, z ktorého dokáže PLC a teda aj vizualizácia vyčítať súbory. Ďalšia časť "DP/DP.html" určuje relatívne umiestnenie od CTEMP, kde sa nachádza webová stránka. Posledná časť textu určuje zdroj pre polohy kĺbov robotického ramena. V našom prípade sú polohy vyčítané zo simulácie pomocou RobotStudia. Text pre načítanie je možné voliť dynamicky, čo znamená, že pre zmenu napríklad IP adresy robotického ramena je nutné kompilovať celý projekt.

Na obrázku 6.2 je zobrazený príklad priemyselnej vizualizácie, do ktorej je implementovaná webová aplikácia. Tento spôsob zobrazenia webovej aplikácie umožňuje zanechať koncept priemyselnej vizualizácie tak, aby mohol byť prehliadač neustále zobrazený na celej obrazovke a operátor sa nemohol dostať mimo vizualizácie.



Obr. 6.2: Webová aplikácia pre tvorbu trajektórie implementovaná do priemyselnej vizualizácie

7 Záver

Pri spracovaní rešerši týkajúcej sa možnosti vytvárania trajektórie pohybu ramien robotov som sa zameral na popis konkrétneho komerčného riešenia. Z dostupných možností som vybral nástroj pre tvorbu programu robotických ramien od firmy ABB s názvom RobotStudio. Nástroj ponúka veľké množstvo možností, z ktorých bol využitý najmä robotický virtuálny kontrolér, pomocou ktorého je možné simulovať reálne robotické rameno. Rešerš možností tvorby trajektórie bol ďalej zameraný na experimentálne aplikácie využívajúce webové prostredie. Z porovnania vyplýva, že takmer všetky dostupné webové aplikácie využívajú aplikačné rozhranie Three.js.

Výsledkom mojej práce je funkčná webová aplikácia, ktorá je určená pre širokú škálu zariadení bez ohľadu na operačný systém. Webová aplikácia ponúka nový prístup k vytváraniu trajektórie z webového prostredia. Užívateľ ma k dispozícii virtuálny 3D priestor, v ktorom má možnosť ľubovoľného pohybu pomocou klávesnice, myši alebo dotykových gest. Aplikácia bola vybavená virtuálnymi bodmi a blokmi, pomocou ktorých je tvorená trajektória. Pri návrhu spôsobu tvorby trajektórie som sa zameral hlavne na to, aby zo strany užívateľa bola práca s webovým prostredím intuitívna a jednoduchá. Z pohľadu programátora je aplikácia vytvorená modulárne pre prípadné zmeny alebo nový obsah.

V ďalšej časti diplomovej práce som sa venoval implementácií robotického ramena do webovej aplikácie. Do webového prostredia boli implementované 3 typy virtuálnych robotov s možnosťou pridania ďalších. Pre reálne polohy ôs robotických ramien boli navrhnuté a realizované dva komunikačné protokoly: REST API a OPC UA. Z meraných výsledkov vyplýva, že komunikácia REST dokáže vyčítať hodnoty ôs každý 20 ms a komunikácia cez OPC UA umožňuje získavať dáta každých 25 - 40 ms. Pri protokole REST je navyše implementovaná možnosť exportovať vytvorenú trajektóriu do robotického kontroléru a následne zastaviť alebo spustiť pohyb robotického ramena.

Pre webovú aplikáciu som zhotovil grafické užívateľské prostredie, ktoré s užívateľom spolupracuje pri vytváraní trajektórie. GUI ponúka možnosti ako importovanie CAD modelov priamo do scény, mód pozorovateľa a podobne. Pri testovaní na potrebný výkon zariadenia som zistil, že klasický mobilný telefón alebo počítač dosahuje pri vykresľovaní 50 až 60 snímok za sekundu, čo značí, že chod webovej aplikácie je úplne plynulý.

Na záver som implementoval webovú aplikáciu do priemyselnej vizualizácie Mapp View od firmy B&R. Pre implementáciu bol použitý widget s názvom WebViewer z prostredia Mapp View.

Literatúra

[1] *RobotStudio, RobotStudio Help 6.08* [cit. 7. 5. 2019].

[2] *Operating manual RobotStudio* [online]. [cit. 7. 5. 2019]. Dostupné z URL: <https://library.e.abb.com/public/c5872e80fa697e76c1257b440052854c/3HAC032104-001_revC_en.pdf>.

[3] *ABB Robotics to bring Electric Vehicle Factory of the Future* [online]. [cit. 7. 5. 2019]. Dostupné z URL: <https://new.abb.com/>.

[4] O.SEVERA, R.PIŠL, M.ČECH, M.GOUBEJ, M.ŠTETINA, M.SCHLEGEL *New 3D HMI tool for robot path planning based on latest W3C standards* Proceedings of the 13th International Carpathian Control Conference (ICCC), High Tatras, 2012, pp. 631-636, ISBN: 978-1-4577-1868-7, [cit. 23 . 3. 2019].

[5] *A three.js based 3D robot interface* [online]. [cit. 7. 5. 2019]. Dostupné z URL: <https://github.com/glumb/robot-gui>.

[6] *API* [online]. [cit. 21. 12. 2018]. Dostupné z URL: <https://cs.wikipedia.org/wiki/API>.

[7] *List of 3D graphics libraries* [online]. [cit. 21. 12. 2018]. Dostupné z URL: <https://en.wikipedia.org/wiki/List_of_3D_graphics_libraries>.

[8] *Low Level API vs High Level API* [online]. Brno: 2017, poslední aktualizace 2017 [cit. 21. 12. 2018]. Dostupné z URL: <https://www.componentspace.com/Forums/8245/Low-Level-API-vs-High-Level-API>.

[9] *Getting Started* [online]. [cit. 21. 12. 2018]. Dostupné z URL: <https://www.khronos.org/webgl/wiki/Getting_Started>.

[10] *An introduction to Three.js* [online]. [cit. 21. 12. 2018]. Dostupné z URL: <https://humaan.com/blog/web-3d-graphics-using-three-js/>.

[11] *three.js / examples* [online]. [cit. 22. 12. 2018]. Dostupné z URL: <https://threejs.org/>.

[12] *carvisualizer* [online]. [cit. 22. 12. 2018]. Dostupné z URL: <http://carvisualizer.plus360degrees.com/threejs/>.

[13] *Babylon.js* [online]. [cit. 22. 12. 2018]. Dostupné z URL: <https://en.wikipedia.org/wiki/Babylon.js>.

[14] *Flight Helmet by Patrick Ryan* [online]. [cit. 22. 12. 2018]. Dostupné z URL: <https://www.babylonjs.com/demos/flighthelmet/>.

[15] *Nastavení projekce* [online]. [cit. 25. 12. 2018]. Dostupné z URL: <https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=22411>.

[16] *WebGL - 3D Canvas graphics* [online]. [cit. 7. 2. 2019]. Dostupné z URL: <https://caniuse.com/#feat=webgl>.

[17] *Three.js - What is PlaneBufferGeometry* [online]. [cit. 14. 2. 2019]. Dostupné z URL: <https://stackoverflow.com/questions/27198525/three-js-what-is-planebuffergeometry>.

[18] C.YUKSEL, S.SCHAEFER, J.KEYSER, *On the Parameterization of Catmull-Rom Curves* Proceeding SPM '09 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling, Pages 47-53, ISBN: 978-1-60558-711-0, [cit. 24. 4. 2019]. Dostupné z URL: <https://doi.org/10.1145/1629255.1629262>.

[19] *IRB 120* [online]. [cit. 28. 4. 2019]. Dostupné z URL: <https://new.abb.com/products/robotics/cs/prumyslove-roboty/irb-120>.

[20] *IRB 120* [online]. [cit. 28. 4. 2019]. Dostupné z URL: <https://new.abb.com/products/robotics/cs/prumyslove-roboty/irb-1200>.

[21] *IRB 120* [online]. [cit. 28. 4. 2019]. Dostupné z URL: <https://www.comau.com/EN/our-competences/robotics/robot-team/Racer7-14>.

[22] *saveAs.js* [online]. [cit. 7. 5. 2019]. Dostupné z URL: <https://gist.github.com/ypnos-web/951977d545e5840eb6e8594829b714bb>.

[23] *B&R, Automation Studio Help 4.5.10* [cit. 8. 5. 2019].

[24] *Build OPC UA applications in JavaScript and NodeJS* [online]. [cit. 9. 5. 2019]. Dostupné z URL: <https://opcfoundation.org/about/opc-technologies/opc-ua/>.

[25] *Unified Architecture* [online]. [cit. 9. 5. 2019]. Dostupné z URL: <https://node-opcua.github.io/>.

[26] T. Ausberger, *OPC UA servery a jejich použití pro přenos dat řídicích systémů* [online]. Plzeň,2015 [cit. 24. 4. 2019]. Dostupné z URL: <https://dspace5.zcu.cz/bitstream/11025/17921/1/prace.pdf>. Bakalárska práca. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra kybernetiky.

[27] *REST – PUT vs POST* [online]. [cit. 8. 5. 2019]. Dostupné z URL: <https://restfulapi.net/rest-put-vs-post/>.

[28] *Representational state transfer* [online]. [cit. 8. 5. 2019]. Dostupné z URL: <https://en.wikipedia.org/wiki/Representational_state_transfer>.

[29] *Robot Web Services* [online]. [cit. 8. 5. 2019]. Dostupné z URL: <http://developercenter.robotstudio.com/webservice>.

[30] E.Wenholt *On multivariable and nonlinear identification of industrial robots* Linköping Studies in Science and Technology, ISBN 91-85295-89-2, [cit. 24. 4. 2019]. Dostupné z URL: <http://www.control.isy.liu.se/research/reports/LicentiateThesis/Lic1131.pdf>.

Zoznam príloh