VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2020 Milan Doležal

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

VIZUALIZACE STACIONÁRNÍHO ROBOTU V C#/WPF

VISUALISATION OF STATIONERY ROBOT IN C#/WPF

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Milan Doležal

AUTHOR

VEDOUCÍ PRÁCE — Ing. František Burian, Ph.D.

SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ | FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Milan Doležal ***ID:*** 195562

Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Vizualizace stacionárního robotu v C#/WPF

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je vytvořit jednoduchý simulační nástroj pro vizualizaci stacionárního robotu v C#/WPF s Helix Toolkit. Práce by měla probíhat s využitím verzovacího systému
Git/GitLab.

1. Prozkoumejte vlastnosti knihovny Helix Toolkit pro vizualizaci 3D modelů.
2. Vytvořte WPF komponentu (DLL), která umožní parametricky vizualizovat pohyby dodaného modelu robotu (Kuka KRC6 sixx).
3. Vytvořte testovací aplikaci, ve které použijete tuto komponentu a umožníte uživateli číselně polohovat klouby.
4. Vytvořte protokolový most, který bude sbírat reálná data z živého robotu

DOPORUČENÁ LITERATURA:

SPONG, Mark W., Seth HUTCHINSON a M. VIDYASAGAR. Robot modeling and control. Hoboken, NJ: John Wiley, c2006. ISBN 978-0471649908.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 8.6.2020

Vedoucí práce: Ing. František Burian, Ph.D.

doc. Ing. Václav Jirsík, CSc.
předseda rady studijního programu

UPOZORNĚNÍ:

Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRAKT
Práce se zabývá vizualizací simulace stacionárního robotu v jazyce C# a platformě WPF. Byla vytvořena knihovna pro polohování robotu. K realizaci byla použita knihovna Helix-Toolkit. Pro testování byla vytvořena testovací aplika s editováním polohy kloubů robotu Kuka KRC6 sixx. Protokolový most byl vytvořen jako TCP/IP klient, který sbírá XML data z robotu Kuka KRC6 sixx.

KLÍČOVÁ SLOVA

Kuka KRC6 sixx, graf scény, Helix Toolkit, WPF, C#, TCP/IP

ABSTRACT
This bachelor thesis deals with visualisation and simulation of stationery robot in programming language C# and WPF platform. In this thesis a DLL file was made with use of Helix Toolkit library for parametric manipulation of stationery robot. Then for testing purposes was made a application which can parametrically change values of robot joints. Protocol bridge is TCP/IP client which gets XML data from Kuka KRC6 sixx robot.

KEYWORDS

Kuka KRC6 sixx, scene graph, Helix Toolkit, WPF, C#, TCP/IP

DOLEŽAL, Milan. *Vizualizace stacionárního robotu v C#/WPF*. Brno, 2020, 63 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce: Ing. František Burian, Ph.D.

Vysázeno pomocí balíčku thesis verze 3.05; http://latex.feec.vutbr.cz

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Vizualizace stacionárního robotu v C#/WPF“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Františku Burianovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k této bakalářské práci.

Obsah

1 Úvod **11**

2 Knihovna Helix Toolkit **12**

2.1 Knihovna Media3D . . . 12

2.1.1 Ukládání 3D obsahu . . . 12

2.1.2 3D model . . . 13

2.1.3 Kamera . . . 13

2.1.4 Materiál . . . 15

2.1.5 Osvětlení scény . . . 15

2.2 HelixToolkit.WPF . . . 17

2.2.1 Visual3D objekty . . . 17

2.2.2 Importování 3D modelů . . . 17

2.3 Helixtoolkit.SharpDX.Assimp . . . 18

2.4 HelixToolkit.SharpDX.WPF . . . 18

3 Vytvoření knihovny pro vizualizaci stacionárního robota **20**

3.1 Grafické transformace . . . 20

3.1.1 Homogenní souřadnice . . . 21

3.2 Skládání transformačních matic . . . 22

3.3 Graf scény . . . 23

3.3.1 Typy uzlů . . . 23

3.3.2 Implementace grafu scény . . . 25

3.3.3 Průchod grafem scény . . . 27

3.3.4 Výpočet celkové transformace . . . 28

4 Vytvoření testovací aplikace **30**

4.1 Návrhový vzor Model-View-ViewModel . . . 30

4.1.1 Model . . . 30

4.1.2 ViewModel . . . 30

4.1.3 View . . . 30

4.2 Data binding . . . 31

4.2.1 Závislé vlastnosti . . . 32

4.3 Události a příkazy . . . 32

4.3.1 Návrhový vzor příkaz . . . 33

4.3.2 Nahrazení událostí příkazem . . . 34

4.4 Implementace . . . 34

4.4.1 Zobrazení scény . . . 34

4.4.2 Ovládací panel pro parametrizaci scény 36
4.4.3 Třídy pro ViewModel 37
4.4.4 Testování polohování robotu 41

5 Protokolový most 44
5.1 Komunikační protokol TCP/IP 44
5.2 Ethernet KRL 45
5.3 Implementace protokolového mostu 45

6 Závěr 50

Literatura 51

Seznam symbolů, veličin a zkratek 53

Seznam příloh 54

A Geometrie robotu Kuka KRC6 sixx 55

B Výpisy z programu 58
B.1 Konfigurace Ethernet KRL 58

C Elektronická příloha 63
C.1 Knihovny 63

Seznam obrázků

2.1 Souřadnicové systémy pro 2D a 3D prvky(vychází z [2]) 12
2.2 Třídy pro 3D obsah knihovny Media3D(vychází z [3]) 13
2.3 Virtuální kamera, parametry kamery a transformační řetězec (vychází z [12]) 14
2.4 Perspektivní a ortogonální kamera (vychází z [3]) 15
2.5 Ukázky různých zdrojů světel použitelných ve WPF (vychází z [3]) . . 16
3.1 Class diagram vytvořené knihovny 25
3.2 Průchod grafem scény 29
4.1 Návrhový vzor Model-View-ViewModel(vychází z [2]) 31
4.2 UML diagram návrhového vzoru příkaz(vychází z [10]) 34
4.3 Zobrazení scény 35
4.4 Ovládací panel pro scénu 36
4.5 Zobrazení scény 37
4.6 Class diagram ViewModelů 38
4.7 Scéna s robotem 41
4.8 Ovládací panel 42
4.9 Scéna s robotem 2 42
4.10 Ovládací panel 2 43
5.1 TCP/IP síť(vychází z [11]) 44
5.2 Simulace TCP/IP serveru robotu 49
A.1 Směr otáčení kloubů 55
A.2 Výchozí pozice [9] 56
A.3 Směr natáčení kloubů [9] 57

Seznam tabulek

2.1 Rozdíly mezi HelixToolkit.WPF a ShardDX.WPF [1] 19
3.1 Limitní hodnoty kloubů manipulátoru KRC6 sixx 26

Seznam výpisů

3.1 Vkládání matic do zásobníku při průchodu grafu scény 28
4.1 Nastavení závislé proměnné Content pomocí databindingu. 32
4.2 Vytvoření události pro stisknutí tlačítka 33
4.3 Automaticky vytvořený event handler v kódu na pozadí okna 33
4.4 Implementace BaseViewModel.cs 38
4.5 Implementace BaseViewModel.cs 38
4.6 Implementace KRC6ViewModel.cs 39
4.7 Implementace TreeViewViewModel.cs 39
5.1 Třída KukaComm.cs 46
5.2 Implementace TcpIPViewModel.cs 47
B.1 XML Ethernet KRL 58
B.2 XML data z TCP/IP serveru 59
B.3 XAML TreeView 59
B.4 Závislá vlastnost pro příkaz 60
B.5 Jezdec a editační políčko 61
B.6 Relay command 61

1 Úvod

Stacionárních robotů přibývá, a jelikož je finančně nákladné pořizovat ke každému novému typu simulační software, je cílem této práce vytvořit simulační nástroj pro vizualizaci dodaného modelu stacionárního robotu Kuka KRC6 sixx.

Knihovna bude napsána v jazyce C# s pomocí otevřené knihovny Helix Toolkit rozšiřující 3D funkčnost technologie WPF, s níž lze tvořit aplikace grafického uživatelského rozhraní. Knihovna Helix Toolkit je rozdělená do několika komponent. V bakalářské práci se mají prozkoumat vlastnosti těchto jednotlivých komponent a vytvořit se má pomocí těchto knihoven vlastní knihovna pro platformu WPF, jež umožní správnou simulaci a vizualizaci zadaných hodnot kloubů dodaného modelu stacionárního robotu.

Funkčnost této knihovny se ověří vytvořením testovací aplikace, která umožní pohyb kloubů robotu.

Jako poslední bod této závěrečné práce bude vytvoření protokolového mostu pro sběr dat z živého robotu komunikací EthernetKRL.

2 Knihovna Helix Toolkit

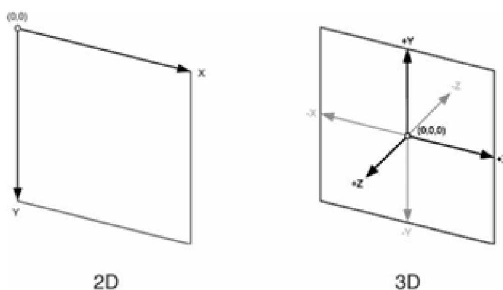
Helix Toolkit je kolekce 3D grafických komponent pro .NET platformu. Knihovna HelixToolkit.WPF rozšiřuje vlastnosti grafického rozhraní platformy WPF, které zajišťuje knihovna Media3D. Aplikace se může tvořit jak v jazyku C#, tak i pomocí jazyka XAML. Jazyk XAML je značkovací jazyk pro popis grafického uživatelského rozhraní a je založen na obecné struktuře jazyka XML. Pro seznámení s knihovnou Helix Toolkit je potřeba nejdříve prozkoumat vlastnosti knihovny Media3D.

2.1 Knihovna Media3D

Pro vytvoření plnohodnotné virtuální trojrozměrné grafické scény je potřeba provést několik kroků. Trojrozměrná počítačová grafika využívá 3D reprezentaci geometrického modelu k výpočtu a vykreslení 2D obrázku, který je promítnut na obrazovku. Jedná se o prostorový objekt ve scéně, která obsahuje další informace jak dané objekty ve scéně zobrazit. Ve scéně musí být tvar modelu pokrytý materiálem pro odraz světla, osvětlení scény a kamera pro promítání na zobrazovací plochu v grafickém uživatelském rozhraní. Kapitoly popisující knihovnu Media3D a její podkapitoly vychází z [2, 3].

2.1.1 Ukládání 3D obsahu

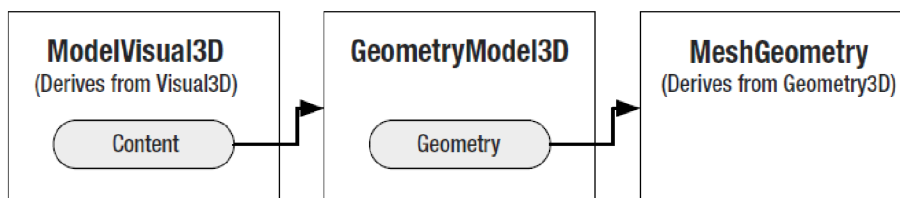
WPF využívá k uchování 3D obsahu prvek ViewPort3D. Jedná se o 2D prvek uživatelského rozhraní, který má funkčnost okna do virtuální grafické scény. Tento prvek obsahuje kolekci tříd Visual3D. V této kolekci se nachází všechny modely, které chceme zobrazit v grafické scéně. Třída Visual3D je abstraktní základní třída pro vykreslování 3D modelů. Konkrétní třída pro vykreslení 3D modelu je třída ModelVisual3D.



Obr. 2.1: Souřadnicové systémy pro 2D a 3D prvky(vychází z [2])

2.1.2 3D model

Model3D je abstraktní základní třída, která reprezentuje obecný 3D model. K vytvoření 3D modelu potřebujeme síťové těleso určující tvar modelu a materiál pro interakci se světelnými zdroji. Těleso je soubor vrcholů, které jsou spojeny pomocí hran. Tímto vznikne síťový model z polygonů. Nejvíce se využívá nejjednodušší polygon, jímž je trojúhelník. Definici tohoto tělesa umožňuje třída MeshGeometry3D, kde každý vrchol je prezentován pomocí třídy Point3D. Třída Point3D obsahuje informaci o poloze bodu v 3D prostoru. Jestli chceme vytvořit ve WPF vlastní těleso, musíme určit, které vrcholy patří konkrétnímu trojúhelníku pomocí vlastnosti TriangleIndices. Pro přidání materiálu na vzniklé těleso musíme použít třídu GeometryModel3D, který dědí z abstraktní třídy Model3D. Modely reprezentované pomocí třídy Model3D můžeme seskupit pomocí třídy Model3DGroup a tím získat složitější 3D model.

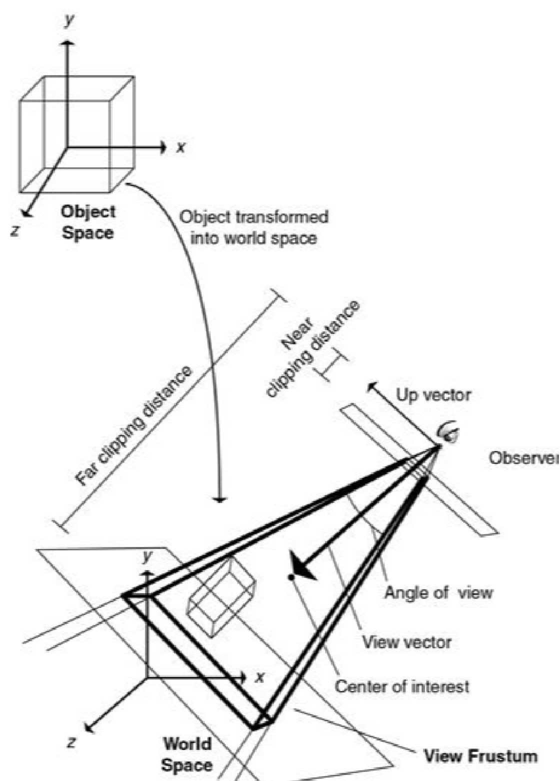


Obr. 2.2: Třídy pro 3D obsah knihovny Media3D(vychází z [3])

2.1.3 Kamera

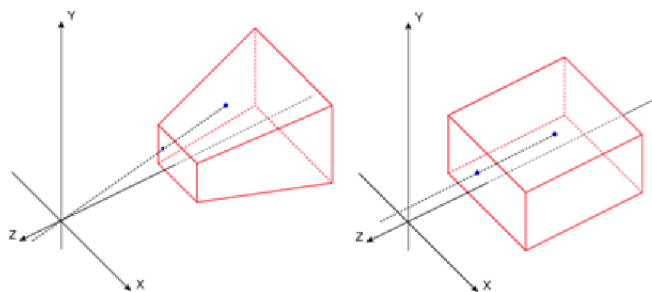
Pro zobrazení trojrozměrných modelů musíme zajistit převod modelů do dvojrozměrného prostoru. K vytvoření obrázku modelů musíme do grafické scény umístit virtuální kameru. Transformace převodu modelu na obrázek se nazývá promítání. Pohled na vytvořenou scénu se mění se změnou pozice a orientace kamery. Promítnutou scénu získáme dopadem promítacích paprsků na plochu v prostoru tzv. průmětnu, kterou lze poté zobrazit ovládacím prvkem Viewport3D. Abstraktní základní třída ProjectionCamera nám dokáže specifikovat různé projekce a jejich vlastnosti. Třída PerspectiveCamera je kamera, která promítá modely tak, že obsahují optický jev perspektivu. Ten způsobuje, že dvě rovnoběžné čáry se se vzdáleností opticky zužují a tedy dochází ke zmenšování modelů se vzrůstající vzdáleností od kamery. Jevu perspektivy dosáhneme vyzařováním paprsků z jednoho bodu prostoru. Můžeme upřesnit polohu kamery, zorný úhel a vektor, který definuje pohled směrem vzhůru, minimum a maximum zobrazovací vzdálenosti.

Projekce kamery je poslední transformace v transformačním řetězci, v němž definujeme transformace z originálně definovaného prostoru přes sérii transformací až k celkové transformaci modelu na zobrazovací plátno. Prostor, ve kterém je model originálně definován(pozice vrcholů), je souřadný systém modelu. Pozice vrcholů jsou obvykle relativně k počátku souřadného systému. Tento model je poté transformován do souřadného systému grafické scény, kde je osvětlení a kamera. Poslední transformace jsou transformace relativně ke kameře.[12]



Obr. 2.3: Virtuální kamera, parametry kamery a transformační řetězec (vychází z [12])

Třída OrthographicCamera vytváří rovnoběžné promítání. To znamená, že kamera neobsahuje perspektivu. Rovnoběžné promítání dosáhneme vyzařováním paprsků, které jsou rovnoběžné. Jak u předchozí perspektivní kamery, i zde můžeme upřesnit vlastnosti kamery jako polohu kamery, zorný úhel a vektor, který definuje pohled směrem vzhůru, minimum a maximum zobrazovací vzdálenosti.



Obr. 2.4: Perspektivní a ortogonální kamera (vychází z [3])

2.1.4 Materiál

Aby těleso vypadalo jako 3D objekt, musí být na jeho povrch aplikována textura. Jeden ze způsobů, jak můžeme ovlivnit vzhled tělesa, jsou materiály. Vlastnosti různých materiálů určují, jak jsou paprsky světla odraženy k pozorovateli pro vytvoření obrazu, který vidí. Objekty v reálném světě odráží světlo na základě jejich povrchů, například některé objekty světlo pohlcují, jiné naopak vyzařují. Ve WPF se definuje materiál pomocí abstraktní třídy Material. Odvozené konkrétní třídy určují charakteristický vzhled objektu. Typ vzhledu lze určit pomocí vlastnosti Brush (štětec), např.: SolidColorBrush. Přidat texturu, např. obrázek na 3D model, můžeme uskutečnit třídou ImageBrush.

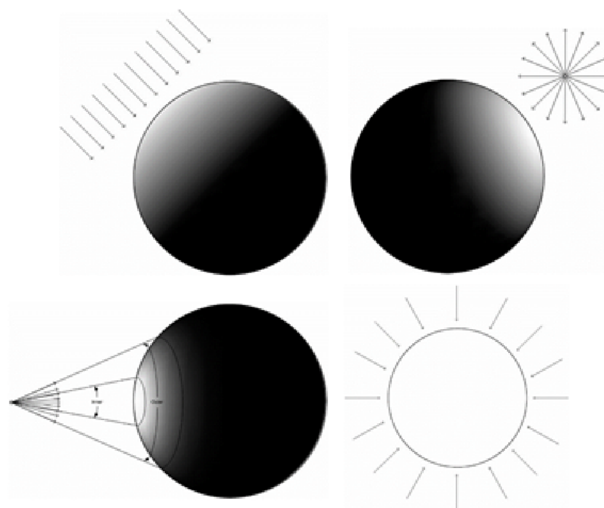
- DiffuseMaterial
  - Vytváří rovný matný povrch, který rozptyluje světlo do všech směrů.
- SpecularMaterial
  - Vytváří lesklý zvýrazněný vzhled. Světlo odráží přímo jako zrcadlo. Odražený paprsek od povrchu má stejný úhel jako při jeho dopadu na objekt. Pozorovatel může vidět jenom část, nebo žádné paprsky.
- EmissiveMaterial
  - Vytváří svítivý vzhled. Tento materiál generuje vlastní zdroj světla, avšak toto světlo se neodráží do ostatních objektů na scéně.
- MaterialGroup
  - Tato třída dovoluje kombinovat více materiálů. Tyto materiály jsou poté naskládány na sebe v pořadí, ve kterém byly přidány do této třídy.

2.1.5 Osvětlení scény

V počítačové grafice se simuluje dopad světla na povrch modelu s různými typy materiálu. Nejvíce se používá geometrická optika modelující světlo jako nezávislé paprsky putující prostorem. Vyhodnocování osvětlovacího modelu, které určí barevné

odstíny zobrazovaných bodů se nazývá stínování.[7] Světlo v počítačové grafice zajistí, že budou grafické objekty scény viditelné. Cílem světel je simulovat světelné zdroje a dopad emitovaných paprsků na objekty ve scéně. Světelný zdroj je objekt, který světlo nejen vyzařuje, ale i odráží. Světelný zdroj je charakterizován emisním spektrem, který udává barvu světelného zdroje. WPF obsahuje několik světelných objektů vytvářejících různé stínové efekty. Tyto světelné objekty jsou vytvořeny podle různých zdrojů světla z reálného světa. Jestli nevložíme alespoň jeden světelný objekt do scény, žádný 3D model se nezobrazí. Základní třída, z níž jsou odvozeny všechny světelné objekty, je abstraktní třída Light.

- AmbientLight - Poskytuje světelný objekt, který osvítí všechny objekty stejně, nemá směr. Velký jas u tohoto světla vytváří ploché obrázky neobsahující stín.
- DirectionalLight - Vyzařuje rovnoběžné paprsky do scény z počátku v nekonečnu. Aproximuje zdroj světla podobný slunečnému svitu.
- PointLight - Jedná se o bodový světelný zdroj. Tento světelný zdroj vyzařuje paprsky o konstantní barvě a intenzitě rovnoměrně do všech směrů ve scéně. Intenzita světla se snižuje se vzdáleností od polohy světla. S tímto světelným zdrojem lze například aproximovat světelná žárovka.
- SpotLight - Emituje kuželovitý zdroj světla, který je určen polohou a směrem, kterým vyzařuje paprsky. Intenzita světla se snižuje se vzdáleností od polohy světla. Na tento zdroj světla se dá dívat jako na paprsek světla ze svítilny.



Obr. 2.5: Ukázky různých zdrojů světel použitelných ve WPF (vychází z [3])

2.2 HelixToolkit.WPF

Tato sada rozšiřuje 3D funkčnost technologie WPF pomocí rozšířené kolekce ovládacích prvků a pomocných tříd pro rozšíření funkčností tříd Media3D[1]. Podkapitoly se věnují konkrétnímu rozšíření.

2.2.1 Visual3D objekty

HelixToolkit. WPF přidává předdefinované 3D modely, které jsou konkrétními implementacemi základní abstraktní třídy Visual3D. Jedná se například o tyto modely: kvádr, elipsoid, koule, mřížka, souřadný systém se šipkami v osách x, y, z. Knihovna obsahuje třídu MeshBuilder, se kterou můžeme tyto tvary kombinovat v jeden. Ohraničení 3D modelů lze vykreslit pomocí třídy BoundingBoxWireFrameVisual3D.

2.2.2 Importování 3D modelů

Pro importování vlastních 3D modelů HelixToolkit poskytuje třídu ModelImporter. Tato třída importuje 3D model, který je uložen na námi specifikované cestě k souboru a přiřadí mu zvolený materiál. ModelImporter vrací načtený model jako třídu Model3DGroup. V současnosti jsou podporované tyto typy souborů:

- .3ds
- .lwo
- .obj
- .stl

Ovládací prvky

HelixToolkit přidává platformě WPF obecně rozšířené ovládací prvky v 3D počítačové grafice.

- HelixViewport3D - Jedno rozšíření ovládacích prvků WPF je implementace vlastního prvku Viewport3D, který navíc rozšiřuje vlastnosti tohoto prvku. Obsahuje manipulaci a přídavné funkce kamery jako setrvačnost kamery a změnu pohledu pomocí prvku ViewCube. V HelixViewport3D lze měnit efekty, jako je anaglyf a prokládání obrazu.
- Manipulator - Tento ovládací prvek slouží k manipulování s konkrétním 3D modelem, jako je posuv a rotace.

Světelné zdroje

Světelné zdroje z knihovny HelixToolkit jsou odvozené z abstraktní třídy LightSetup, která patří pod skupinu Visual3D. Tímto se liší od světel z knihovny Media3D, kde jsou zdroje světla odvozené z třídy Model3D.

2.3 Helixtoolkit.SharpDX.Assimp

Tato knihovna umožňuje import a export 3D modelů pro HelixToolkit.SharpDX kolekci, která využívá knihovnu Assimp.NET. Assimp.NET je wrapper knihovny Open Asset Import Library(Assimp). Assimp je knihovna napsaná v jazyce C++, která načítá a zpracovává grafickou scénu různých datových formátů. Scéna je uložena jako datová struktura strom, která obsahuje nejvyšší uzel, kořen stromu, z něhož získáme přístup ke všem zbývajícím uzlům stromu jako 3D objekty, materiály, animace nebo textury, které byly přečteny z importovaného souboru.[4] Kořen tohoto stromu se ukládá jako objekt třídy HelixToolkitScene. Grafickou scénu vytvořenou s knihovnou HelixToolkit.SharpDX lze přeměnit na scénu z knihovny Assimp a obráceně.

2.4 HelixToolkit.SharpDX.WPF

Vlastní vykreslovací jádro a XAML kompatibilní graf scény s návrhovým vzorem MVVM založené na knihovně SharpDX. SharpDX je otevřená knihovna sloužící jako wrapper knihovny DirectX pro platformu .NET. Hlavní částí knihovny je implementace grafu scény (viz strana 23), který obsahuje všechny typy uzlů potřebných ke zpracování grafické scény z importovaného souboru knihovnou SharpDX.Assimp. Tato knihovna je vytvořena i pro technologii .NET Core a UWP.[1] Rozdíly mezi HelixToolkit.WPF a HelixToolkit.SharpDX.WPF jsou shrnuty v tabulce **??**

Tab. 2.1: Rozdíly mezi HelixToolkit.WPF a ShardDX.WPF [1]

\	HelixToolkit.WPF	HelixToolkit.WPF.SharpDX
Vykreslovací jádro	WPF DirectX 9	Vlastní vykreslovací jádro DirectX 11
Material	Phong	Phong, PBR, Vertex Color atd.
Shader	Pixel stínování	Kompletní stínovací řetězec
Point,Line	CPU	GPU
Funkce	Žádné	Bone Skinning, Instancing, WireFrame, Tessellation
Hit Test	WPF Framework	Octree
Vlastní pořadí renderu	Žádné	Podpora
Měření výkonnosti	FPS	FPW, Draw Calls, Draw Time

3 Vytvoření knihovny pro vizualizaci stacionárního robota

Pro vytvoření simulačního nástroje pro polohování kloubů nás budou zajímat geometrické vlastnosti manipulátoru. Robotické manipulátory jsou složeny z rámů spojených pomocí kloubů do kinematického řetězce. Dodaný model robota je KUKA KRC6 sixx. Jedná se o stacionární manipulátor, který má šest stupňů volnosti. Počet kloubů určuje stupeň volnosti. Manipulátor obsahuje typicky alespoň 6 stupňů volnosti: tři pro pozici a tři pro orientaci. Rameno robota KUKA KRC6 sixx obsahuje 3 rotační klouby (RRR), jedná se tedy o kloubové rameno. Zbytek robota je tvořen eulerovým zápěstím, třemi rotačními klouby(RRR).[5]

Rotace kloubu ovlivňuje všechny části robotického manipulátoru za otočeným kloubem. Jestli tedy otočíme kloubem A3, výpočet jednoho vrcholu 3D modelu p_3 bude záviset také na předcházejících rotacích kloubů A1 a A2:

$$\mathbf{p}_3 = \mathbf{R}_{A1} \cdot \mathbf{R}_{A2} \cdot \mathbf{R}_{A3} \cdot \mathbf{p}_0, \tag{3.1}$$

kde $\mathbf{R}_{\mathrm{A1}}$, $\mathbf{R}_{\mathrm{A2}}$ a $\mathbf{R}_{\mathrm{A3}}$ jsou homogenní matice rotace. Simulace modelu manipulátoru KUKA KRC6 sixx potom bude obsahovat alespoň 6 rotačních transformací, když budou po importování tohoto modelu všechny rámy na správném místě. Číslo se může zvýšit, jestliže potřebujeme část manipulátoru, nebo celý manipulátor posunout. Rozhodl jsem se vytvořit datovou strukturu graf scény, která se používá v 3D počítačové grafice, kde může definovat hierarchii transformací pro grafické objekty. Jelikož v knihovně HelixToolkit.SharpDX je už implementovaný graf scény, použiji k vypracování knihovnu HelixToolkit.WPF.

3.1 Grafické transformace

Grafické transformace mění pozici vrcholů modelu v 3D souřadném systému. Vrcholy objektu jsou transformovány ze souřadného systému objektu do definované pozice relativně k počátku souřadného systému grafické scény. Souřadný systém může být definovaný jako levotočivý, nebo pravotočivý. Typ souřadného systému můžeme určit pomocí pravidla pravé ruky. Grafické transformace jsou reprezentovány transformačními maticemi. Jejich skládáním dosáhneme složitějších transformací. Transformační matice jsou typu rotace, translace a změna měřítka.[12] Máme-li vrchol 3D modelu v kartézském souřadném systému:

$$\mathbf{p} = [x, y, z]^T \tag{3.2}$$

po aplikaci transformace vznikne vrchol

$$\mathbf{p}' = [x', y', z']^T \tag{3.3}$$

3.1.1 Homogenní souřadnice

Homogenní transformace nám umožňuje reprezentovat lineární transformace pomocí jediné matice. V této matici je také transformace promítání kamery. Skládání matic se realizuje násobením homogenních transformačních matic. Tímto vznikne jedna výsledná matice, která bude mít rozměr 4 × 4. Jelikož se jedná o čtvercovou matici, můžeme najít její inverzi. Bod v prostoru vyjádřený homogenními souřadnicemi má potom tvar

$$\mathbf{p} = [x, y, z, \omega]^T, \tag{3.4}$$

kde ω je koeficient, který určuje významnost daného bodu. Často se volí ω=1.[7]

Translační matice

Posuv bodu v prostoru o vektor $\vec{v}(v_x,v_y,v_z)$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.5}$$

Rotační matice

Rotace ve dvojrozměrném prostoru je rotace aplikována kolem bodu, pro trojrozměrný prostor platí rotace kolem osy.

Rotace kolem osy x

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.6}$$

Rotace kolem osy y

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.7}$$

Rotace kolem osy z

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.8}$$

Změna měřítka

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.9}$$

Všechny transformační matice vychází z: [7]

3.2 Skládání transformačních matic

Pro dosažení celkového pohybu musíme násobit matice ve správném pořadí. Transformace modelu ze svého souřadného systému do souřadného systému scény je dosažena správným pořadím skládání afinních transformací. Tímto vznikne jedna výsledná matice pro transformaci modelu. Změna transformace se musí vypočítat znovu ze všech transformačních matic, aby nevznikala chyba zaokrouhlování. Modely si nesmí pamatovat předchozí transformace. U násobení matic neplatí komutativní zákon ($\mathbf{M}_1 \cdot \mathbf{M}_2 \neq \mathbf{M}_2 \cdot \mathbf{M}_1$). Matice s rotací, translací a změnou měřítka se vypočítá tak, že nejdříve zvětšíme měřítko modelu, poté změníme rotací orientaci a nakonec jej posuneme. Pořadí bude proto

$$\mathbf{M} = \mathbf{T}(\vec{v}) \cdot \mathbf{R}(\theta) \cdot \mathbf{S}(\vec{s}), \tag{3.10}$$

kde $\mathbf{S}(\vec{v})$ je matice pro změnu měřítka, $\mathbf{R}(\theta)$ je rotační matice a $\mathbf{T}(\vec{v})$ je matice translace. Nicméně struktura Matrix3D z knihovny Media3D je ve tvaru[2]:

$$\begin{bmatrix} M11 & M12 & M13 & M14 \\ M21 & M22 & M23 & M24 \\ M31 & M32 & M33 & M34 \\ OffsetX & OffsetY & OffsetZ & M44 \end{bmatrix} \tag{3.11}$$

Matice reprezentována strukturou Matrix3D je tedy transponovaná. Výpočet transformovaného bodu bode poté vynásobením bodu transponovanou maticí $\mathbf{M}^T$

$$\mathbf{p}' = \mathbf{M}^T \cdot \mathbf{p}, \tag{3.12}$$

dosazením transformačních matic

$$\mathbf{p}' = (\mathbf{T}(\vec{v}) \cdot \mathbf{R}(\theta) \cdot \mathbf{S}(\vec{s}))^T \cdot \mathbf{p} \quad (3.13)$$

Výsledné pořadí bude po transpozici matic obrácené.

$$\mathbf{p}' = \mathbf{S}(\vec{s})^T \cdot \ \mathbf{R}(\theta)^T \cdot \mathbf{T}(\vec{v})^T \cdot \mathbf{p} \quad (3.14)$$

3.3 Graf scény

Graf scény je datová struktura používaná v počítačové grafice pro reprezentaci grafické scény. Je to kolekce uzlů v datových strukturách strom nebo graf, které vyjadřují vztahy mezi jednotlivými uzly. Platnost vlastnosti, která ovlivňuje ostatní uzly je dána polohou uzlu, ve kterém se tato vlastnost nachází. Tato hierarchická struktura je zakořeněná, tedy obsahuje jeden uzel na nejvyšší úrovni. Dále obsahuje skupinové uzly, které mají libovolný počet potomků, jedná se tedy o n-ární strom. Nejspodnější úroveň hierarchického stromu tvoří uzly nazývané listy, které nemají žádné potomky. Typický graf scény neumožňuje orientovaný cyklus (kde některé uzly jsou propojeny do uzavřeného řetězce) nebo oddělené uzly, které nemají ani předka, ani potomka. Některé grafy scén umožňují vytvořit instance, které obsahují referenci na uzel reprezentující objekt ve scéně. Tímto bude uzel instancovaný a hierarchická struktura se může definovat jako orientovaný acyklický graf a dosáhneme snížení náročnosti na paměť sdílením informací konkrétního uzlu.

Graf scény může určovat, které objekty se budou vykreslovat, jaké budou obsahovat textury nebo jejich osvětlení. Jedním ze vztahů předek-potomek je hierarchické skládání transformací, kde potomek zdědí matici pro vynásobení s jeho lokální maticí. Při sestupu do nižší úrovně se transformační matice uloží do zásobníku.[7, 8]

3.3.1 Typy uzlů

Struktura grafu scény obsahuje tři typy uzlů, které se liší vztahem předek-potomek, funkčností a uchovávaným obsahem. Vzhledem k zadání práce se vytvoří jen uzly potřebné ke správné transformaci 3D modelů.

Kořen stromu

Je to nejvyšší uzel stromu, nemá tedy předka a má libovolný počet potomků. Je to nejdůležitější část scény. Tento uzel stromu zastupuje třída RootNode. Z tohoto uzlu se většinou aktualizují transformace modelů a aktualizuje vykreslení scény.

Vnitřní uzly

Obsahují libovolný počet potomků a předků. Tyto objekty jsou kořeny podstromů v hlavní stromové struktuře.

Listy stromu

Tvoří nejspodnější část stromové struktury, a proto nemohou obsahovat potomky. Tyto uzly jako jediné uchovávají obsah vykreslovaný do virtuální scény.

3.3.2 Implementace grafu scény

Obr. 3.1: Class diagram vytvořené knihovny

Třída SceneNode

Abstraktní třída určující obecný typ uzlu scény. Tato třída obsahuje vlastnosti a metody, které jsou společné pro všechny uzly stromu, jako jsou například jméno

objektu a abstraktní metoda AccumulateTransform() pro výpočet transformací uzlů.

Třída RootNode

Uzel reprezentující kořen stromu, z něhož můžeme vstupovat do všech částí grafu scény. Všechny 3D modely, které chceme ve scéně vykreslit, musí být uloženy v kolekci typu Visual3D vykreslovacího prvku ViewPort. Všechny modely v grafu scény získáme metodou UpdateSceneGraph, která prochází celý graf scény rekurzivně a při nalezení uzlu typu list, tedy objektu třídy VisualNode následně přidá model do kolekce iRenderableVisual.

Třída TransformNode

Tato abstraktní třída poskytuje vlastnosti a funkce pro transformování modelů. Jsou zde metody pro nastavení transformačních matic a jejich násobení. Transformace předků jsou uloženy v zásobníku iMatrixStack. Lokální matice se vypočítá vynásobením matic rotace, translace a změny měřítka dle rovnice 3.14, kde transformační matice jsou matice z knihovny Media3D ve tvaru 3.11. Po výpočtu se lokální matice uloží na vrchol zásobníku iMatrixStack metodou PushTransform. Pozice modelu ve scéně je uložena ve vlastnosti iPosition.

Třída GroupNode

Reprezentace vnitřních uzlů, které mají předka a libovolný počet potomků. Tento uzel obsahuje funkčnost pro změnu předcházejícího uzlu a manipulaci s potomky.

Třída RevoluteJoint

Třída reprezentující rotační kloub robota. V grafu scény tato třída zastává pozici vnitřního uzlu. U kloubu robota nás zajímají hodnoty limitních úhlů, úhel natočení a stupeň volnosti. Rotace se dá invertovat metodou SetInvertion. Rotace se provede kolem bodu, který je definován vlastností iPosition metodou RotateJointAt().

Tab. 3.1: Limitní hodnoty kloubů manipulátoru KRC6 sixx

Kloub	Limitní hodnoty
A1	+/-170°
A2	+45° až -170°
A3	+156° až -120°
A4	+/-185°
A5	+/-120°
A6	+/-350°

Třída PrismaticJoint

Třída reprezentující posuvný kloub robota. V grafu scény tato třída zastává pozici vnitřního uzlu. U kloubu robota nás zajímají limitní hodnoty posuvu, aktuální hodnota posuvu a stupeň volnosti. Posuv se dá invertovat metodou SetInvertion.

Třída RobotManipulator

Třída reprezentující průmyslový manipulátor. V grafu scény zastává tato třída pozici vnitřního uzlu. Třída obsahuje vlastnost iJointCount pro počet kloubů manipulátoru a dvě pole typu double pro limitní hodnoty těchto kloubů.

Třída LeafNode

Abstraktní třída pro uzly stromu, které nemají potomky (listí stromu), ale mají předka. Tyto objekty jsou také uchovávány v kolekci uzlu RootNode, kde konkrétní 3D model získáme pomocí indexu, který je uložen ve vlastnosti iIndex.

Třída VisualNode

List stromu, který obsahuje objekt ModelVisual3D, který dědí vlastnosti z třídy Visual3D, proto se dá vykreslit v prvku ViewPort3D. Je to jediný objekt, který reprezentuje 3D model v grafu scény. Třída ModelVisual3D obsahuje vlastnost Content typu GeometryModel3D, do které je uložen tvar a barva 3D modelu. S 3D modelem se pojí transformace. Tento uzel dále může reprezentovat jak světla z knihovny Media3D, která jsou odvozená jak z třídy Model3D, tak z třídy světel z Helixtoolkit. WPF, které jsou odvozené z třídy Visual3D. Jako jediný z uzlů aplikuje transformační matice ze zásobníku iMatrixStack pro transformaci modelu.

3.3.3 Průchod grafem scény

Depth-first search je algoritmus pro průchod nebo vyhledávání datových struktur strom nebo graf. Způsob průchodu je postup co nejhlouběji do grafu scény, tedy až do nalezení listu stromu. Po nalezení nejhlubšího uzu se algoritmus vrací k předchozímu uzlu a opět postupuje hlouběji k nalezení listu stromu.[14]

Algoritmus začíná v libovolném uzlu (nejčastěji kořen stromu) a pokračuje levým podstromem, dokud nenarazí na uzel bez potomků (preorder). Průchod stromem realizuji rekurzivně metodou AccumulateTransform.

Výpis 3.1: Vkládání matic do zásobníku při průchodu grafu scény

```
public virtual void AccumulateTransform()
{
  PushTransform()
  int i = 0;
    while (iChildren.Count != 0)
    {
      if (iChildren.Count <= i)
        return;
      iChildren[i].AccumulateTransform();
      ++i;
    }
}
```

3.3.4 Výpočet celkové transformace

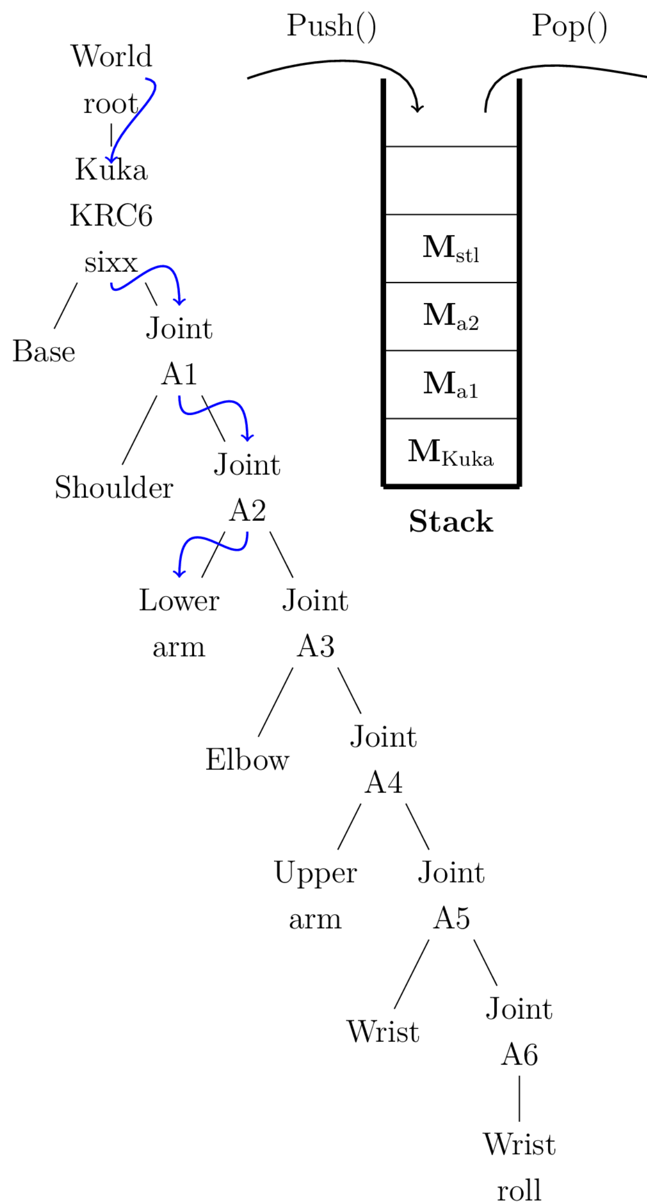
Výpočet transformace pro model s názvem Lower Arm (modrá šipka v obrázku 3.2) se realizuje průchodem stromu z kořenového uzlu až po uzel reprezentující model Lower arm. Každý uzel obsahující lokální transformační matici, který je navštíven tímto průchodem, vloží lokální transformační matici na vrchol zásobníku. Jakmile při průchodu narazíme na uzel typu list, opět vložíme na vrchol zásobníku lokální matici a poté vynásobíme celkovou transformaci postupným násobením odebíraných matic ze zásobníku. Výsledná matice by byla poté

$$\mathbf{M} = \mathbf{M}_{\mathrm{Kuka}} \cdot \mathbf{M}_{\mathrm{a1}} \cdot \mathbf{M}_{\mathrm{a2}} \cdot \mathbf{M}_{\mathrm{stl}}, \tag{3.15}$$

ale matice 3.11 je transponovaná a pořadí matic se při násobení změní na

$$\mathbf{M} = \mathbf{M}_{\mathrm{stl}} \cdot \mathbf{M}_{\mathrm{a2}} \cdot \mathbf{M}_{\mathrm{a1}} \cdot \mathbf{M}_{\mathrm{Kuka}} \tag{3.16}$$

Po této operaci vypadá pořadí matic obráceně k pořadí v zásobníku a proto tedy by byla vhodná spíše datová struktura fronta, ale rozhodl jsem se ponechat zásobník.



Obr. 3.2: Průchod grafem scény

4 Vytvoření testovací aplikace

Testovací aplikace bude vytvořena jako dvě okna. První okno bude obsahovat pohled na grafickou scénu, kde je zobrazen stacionární robot. Druhé okno bude obsahovat posuvníky s editačními políčky pro nastavení úhlu otočení libovolného kloubu stacionárního robota. Dále zde budou editační políčka a tlačítka pro obsluhu protokolového mostu. Pro vytvoření aplikace jsem zvolil návrhový vzor Model-View-ViewModel (MVVM).

4.1 Návrhový vzor Model-View-ViewModel

Jedná se o návrhový vzor pro vytváření aplikací na platformě WPF. Tento návrhový vzor rozděluje logiku aplikace (Model) od uživatelského rozhraní (View). Rozdělení je vykonáno vytvořením pomocné třídy ViewModel. V těchto vrstvách aplikace jsou poté implementovány další návrhové vzory jako Command a Observer. MVVM definuje oddělení zodpovědností (Separation of Concerns) a nezávislost jednotlivých vrstev [10].

4.1.1 Model

Jedná se o vrstvu, která reprezentuje třídy obsahující data a aplikační logiku používané ve třídě ViewModel. Data v těchto třídách by měla podporovat rozhraní INotifyPropertyChanged a INotifyCollectionChanged. V našem případě se jedná o knihovnu pro vizualizaci stacionárního robota a protokolový most. Uživatelské rozhraní tuto vrstvu nezná.[10]

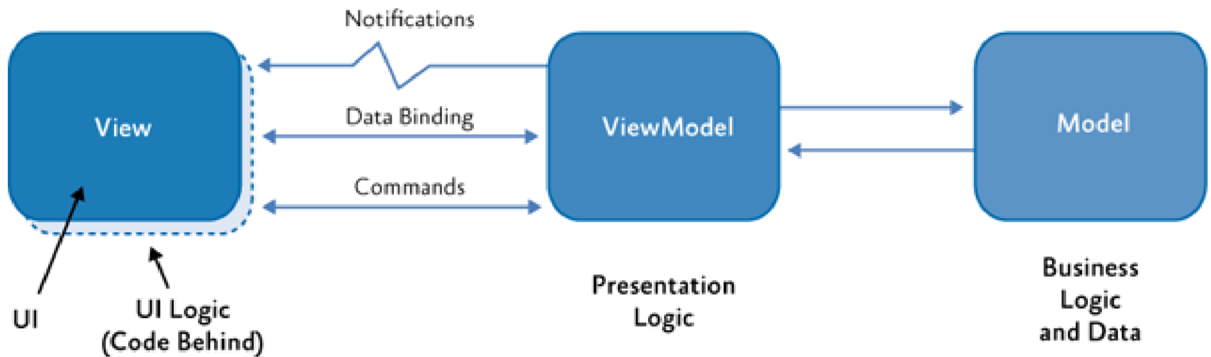
4.1.2 ViewModel

Tato vrstva je zasazena mezi uživatelské rozhraní a Model a je v ní uchováván stav aplikace. Prvky uživatelského rozhraní jsou data bindingem spojená s vlastnostmi definovanými v konkrétním ViewModelu. Všechny třídy reprezentující ViewModel implementují rozhraní INotifyPorpertyChanged, jež zaručí upozornění uživatelského rozhraní při změně vlastnosti, která se objeví v konkrétním ovládacím prvku.[10]

4.1.3 View

Jedná se o uživatelské rozhraní, které je popsáno pomocí jazyka XAML. View jako celek obsahuje vlastnost DataContext, s níž určíme výchozí binding element. Jednotlivé ovládací prvky jsou propojeny s vlastnostmi ViewModelu pomocí Data bindingu.[10]

MVVM zajistí přenositelnost mezi platformami, protože uživatelské rozhraní nebude mít žádný nebo minimální code behind. Dále nám bude umožněno měnit technologie uživatelského rozhraní nebo ovládacích prvků bez změny logiky ve ViewModelu, protože je na uživatelském rozhraní nezávislý. Události jsou implementovány ve ViewModelu pomocí rozhraní ICommand, což umožní jednodušší testování aplikace.[10]



Obr. 4.1: Návrhový vzor Model-View-ViewModel(vychází z [2])

4.2 Data binding

Komunikace mezi View a ViewModelem je realizována pomocí data bindingu. Data binding je funkce, která propojuje dvě datové vlastnosti a udržuje mezi nimi komunikační kanál. Data binding poskytuje upozornění na změnu vlastnosti při jejím zápisu a změna je provedena i v druhé datové vlastnosti. V naší práci se bude jednat o propojení vlastností definovaných ve třídách ViewModelu a ovládacími prvky v uživatelském rozhraní. Jestliže dva ovládací prvky potřebují stejnou vlastnost, dá se data bindignem spojit i dva ovládací prvky. Platforma WPF obsahuje mnoho možností data bindingu. Data binding podporuje několik způsobů:

- Jednosměrný - s tímto módem jsou ovládací prvky svázány s ViewModelem, tedy pracují s určitými daty, která nemohou ovlivnit (readonly).
- Obousměrný - obousměrný mód má možnost na rozdíl od jednosměrného módu změnit datové vlastnosti ve ViewModelu, když jej uživatel upraví v uživatelském rozhraní.

Data binding obsahuje vlastnost Path, se kterou můžeme přistupovat do zanořených dat. Vlastnosti propojené data bindingem se rozdělují na Binding Target a Binding Source, kde Binding Target musí být speciální vlastnost typu závislá vlastnost (dendencency property) a Binding Target může být obyčejná CLR vlastnost

typu string,int,double atd. Většina vlastností ovládacích prvků jsou typu závislá vlastnost.[2, 10]

Výpis 4.1: Nastavení závislé proměnné Content pomocí databindingu.

```
1 <Button Content="{Binding Source=BindingSource , Path=Property}" />
```

4.2.1 Závislé vlastnosti

Platforma WPF zajišťuje soubor služeb pro rozšíření funkcionality CLR vlastností. Vlastnost podporovaná tímto systémem se nazývá závislá vlastnost (dependency property). CLR vlastnost zapisuje nebo čte hodnotu přímo z privátní proměnné v dané třídě, závislá proměnná má uchovanou hodnotu ve slovníku hodnot v základní třídě DependencyObject. Účelem závislé vlastnosti je poskytovat možnost nastavit hodnotu vlastnosti na základě hodnoty ostatních vstupů. Tyto ostatní vstupy mohou být například systémové vlastnosti jako styling, trigger a animace. Nás bude nejvíce zajímat nastavení hodnoty pomocí data bindingu pro neporušení zásad MVVM. Další přidaná funkčnost je poté notifikace změny, validace. Výhodou tohoto typu vlastností jsou malé nároky na paměť. Většina vlastností ovládacích prvků zůstává ve výchozích hodnotách. Závislé vlastnosti na rozdíl od CLR vlastností uchovávají jenom hodnoty změněných vlastností konkrétních ovládacích prvků a výchozí hodnotu vlastností jenom jednou (je statická). Závislá vlastnost je tedy definovaná jako statická a neumožňuje zápis (readonly), tedy nelze změnit po inicializaci. Závislou vlastnost vytvoříme metodou DependencyProperty.Register(). Tato metoda vyžaduje jméno a typ CLR vlastnosti, typ třídy, ve které je tato vlastnost definována. Jako poslední musíme určit hodnotu Metadata, která určuje výchozí hodnotu vlastnosti. Pro možnost čtení a zápis závislé vlastnosti jako k CLR vlastnosti musíme definovat wrapper. Wrapper čte hodnoty metodou GetValue() a zapisuje metodou SetValue().[10]

4.3 Události a příkazy

Událost (event) je programová struktura, která reaguje na změnu určitého stavu jako například stisknutí tlačítka uživatelem v grafickém rozhraní. Události se implementují tam, kde potřebujeme provést blok kódu jako automatickou odezvu na změnu stavu určitého prvku aplikace přicházející náhodně. Z hlediska událostí se objekty aplikace dělí na vydavatele (publisher) a pozorovatele (subscriber), kdy vydavatel může mít libovolný počet pozorovatelů. Při vzniku události vydavatel upozorní všechny své pozorovatele. Pozorovatel poté vykoná akci (event handler) pro

zpracování této události. Event handler obdrží jako argument vydavatele a informaci o stavu vzniklé události.[10]

Výpis 4.2: Vytvoření události pro stisknutí tlačítka

```
1 <Button Click="Button_Click"/>
```

Výpis 4.3: Automaticky vytvořený event handler v kódu na pozadí okna

```
1 private void Button_Click(object sender, RoutedEventArgs e)
2 {
3
4 }
```

Takto vytvořená událost ale porušuje zásady návrhového vzoru MVVM, protože jsme vytvořili metodu v kódu na pozadí, kde nemá být žádná logika pro nezávislost uživatelského rozhraní. Proto se místo událostí používá návrhový vzor příkaz (Command pattern).

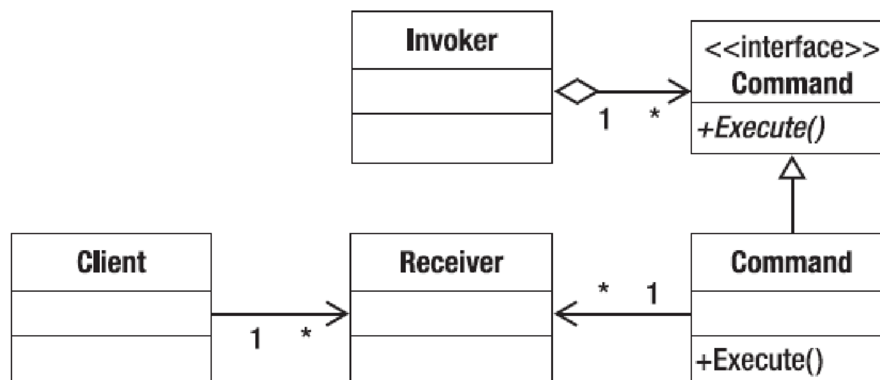
4.3.1 Návrhový vzor příkaz

Příkaz zapouzdřuje požadavek jako objekt, který můžeme parametrizovat. Výhoda je ta, že příkaz nemusí znát prvek, který ho vyvolal. Příkaz se využívá ve WPF aplikacích pro zapouzdření událostí vyvolaných ovládacími prvky uživatelského rozhraní. Na rozdíl od událostí můžeme příkazy nastavit pomocí data bindingu. Tento způsob nám umožní se vyhnout vytvoření metody pro zpracování události v kódu na pozadí okna, ale obslužná metoda se definuje v konkrétním ViewModelu. Na obrázku 4.2 je vidět UML diagram příkazu. Nachází se zde třída Invoker, která umí vyvolat metodu Execute() v abstraktním příkazu ICommand, ale nezná již podrobnosti o konkrétním příkazu. Ve WPF půjde o ovládací prvek uživatelského rozhraní. Třída Reciever poté uchovává metodu, kterou chceme vyvolat metodou Execute() v konkrétním příkazu Command. Command tedy slouží jako prostředník mezi těmito dvěma třídami.[10]

Systémové rozhraní *ICommand* obsahuje navíc metodu *CanExecute()*

Relay command

Tato třída slouží jako šablona pro tvorbu příkazů. Při vytváření příkazu konstruktorem se pomocí delegátů vloží metody, které se mají vykonat při volání metod Execute() a CanExecute(). Toto je důvod, proč můžeme vytvořit příkaz ve ViewModelu. Událost CanExecuteChanged zaručí nevykonat příkaz, když metoda CanExecute() vrací hodnotu false. Tato událost se vykoná vždy při změně vrácené hodnoty této metody, protože CommandManager zjistí změny v uživatelském rozhraní.[10] Kód je v příloze B.6.



Obr. 4.2: UML diagram návrhového vzoru příkaz(vychází z [10])

4.3.2 Nahrazení událostí příkazem

Většina ovládacích prvků obsahuje události místo příkazů, proto je dle MVMM nutné tyto události nahradit příkazem. Jedním ze způsobů je speciální závislou vlastnosti attached property. Ta se od obyčejné závislé vlastnosti liší registrováním, kde attached property používá metodu DependencyProperty. RegisterAttached(), definuje registraci vzniku události a obslužnou metodu události. Tuto vlastnost poté můžeme připojit k ovládacímu prvku a data bindingem přidělit obslužný příkaz.[10]

Nahradil jsem událost posuvu jezdce v ovládacím prvku typu slider pro rotaci kloubu robotu. Kód je ve výpisu B.4 Dalším způsobem nahrazení události je pomocí knihovny Microsoft.Xaml.Behaviours.Wpf. Tato knihovna nám umožní vytvořit v uživatelském rozhraní EventTriggery, která zabalí událost a při vzniku události vyvolá příkaz, který můžeme přiřadit data bindingem. Výhodou je jednoduchá tvorba a možnost předání argumentů události.

4.4 Implementace

Aplikace obsahuje dvě okna, z nichž první okno pouze zobrazuje grafickou scénu s modelem robota. Druhé okno obsahuje ovládací prvky pro změnu struktury scény jako například funkcionalitu přidání pojezdu nebo koncového nástroje robota, parametrické polohování stacionárního robota a obsluha protokolového mostu.

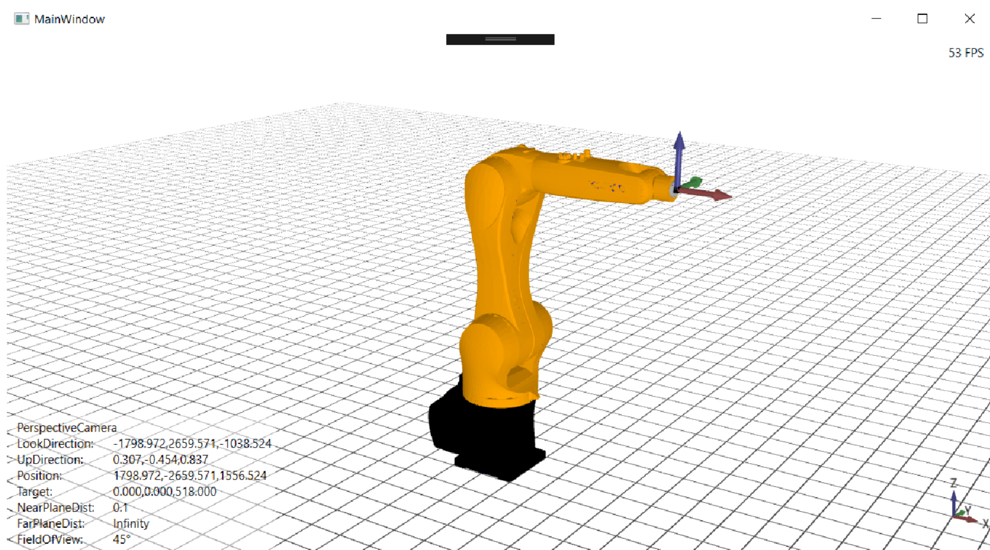
4.4.1 Zobrazení scény

Při importování modelu robota do grafické scény je výhodné mít umístěné modely tak, aby počátek modelu byl na místě osy rotace a tento počátek ležel při

importu modelu také v počátku grafické scény. Výhodou při použití tohoto souřadného systému je, že získáme pozici modelu z translační části matice 3.11, která je jako vlastnosti v objektu třídy ModelVisual3D. Knihovna Media3D obsahuje rotaci kolem bodu v prostoru narozdíl od matice z knihovny SharpDX, proto nemusíme při rotaci posunout model do počátku souřadného systému. U importováných modelů mimo počátek, například modely definované již ve výchozí poloze stacionárního robota není možné získat pozici z translační matice. Z dokumentace robota jsem získal potřebné pozice rotačních kloubů a rozměry modelů, poté jsem vytvořil podstrom definující transformaci modelu robota. Trojrozměrnou scénu jsem zobrazil prvkem HelixViewport3D, který obsahuje kolekci typu Visual3DCollection, kam se musí uložit všechny modely, jež chceme ve scéně vykreslit. Patří sem i osvětlení scény. Prvky této kolekce jsou typu ModelVisual3D.

HelixViewport3D obsahuje další definované vlastnosti, které jsem v XAML souboru uživatelského rozhraní nastavil jako zobrazení souřadného rámu pro orientaci ve scéně, doplňkovou funkcionalitu interaktivity kamery, stav kamery a počet snímků za sekundu.

Do scény jsem přidal dva světelné zdroje z knihovny HelixToolkit.WPF typu Sunlight a model mřížky pro zobrazení podlahy. Umístění a existenci těchto prvků v hierarchii scény lze vidět v obrázku 4.4.



Obr. 4.3: Zobrazení scény

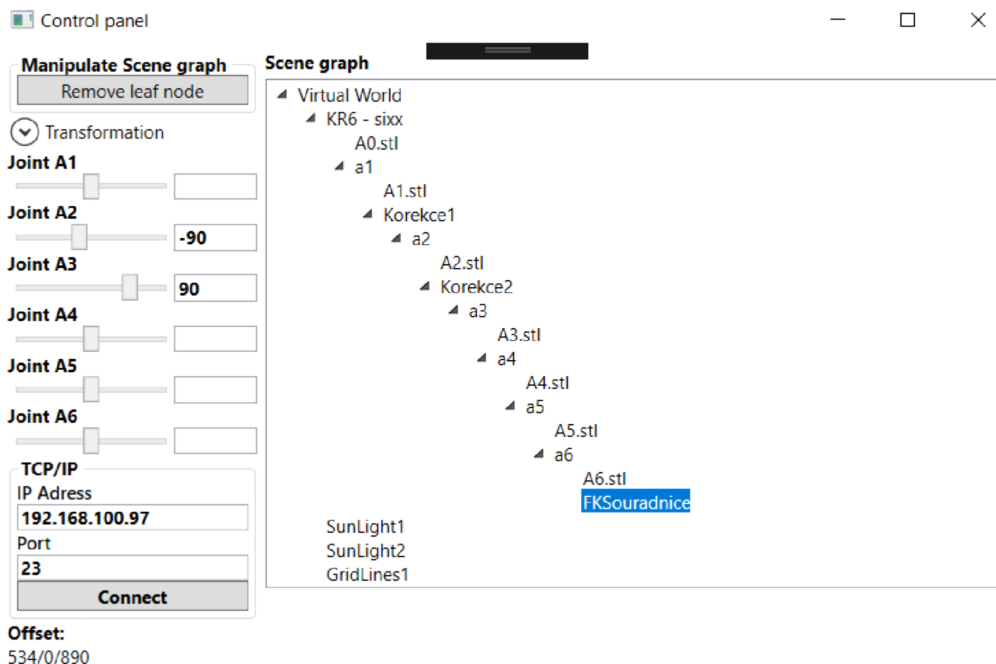
Pro lepší manipulaci s kamerou jsem vytvořil nový příkaz, který se provede při zmáčknutí tlačítka na klávesnici. Dle zmáčknuté klávesy se kamera v prostoru posune v kladném, nebo záporném směru libovolné osy. Pravým tlačítkem myši můžeme měnit orientaci kamery.

- W - Kladný směr osy x.

- A - Záporný směr osy x.
- S - Kladný směr osy y.
- D - Záporný směr osy y.
- Mezerník - Kladný směr osy z.
- Shift - Záporný směr osy z.

4.4.2 Ovládací panel pro parametrizaci scény

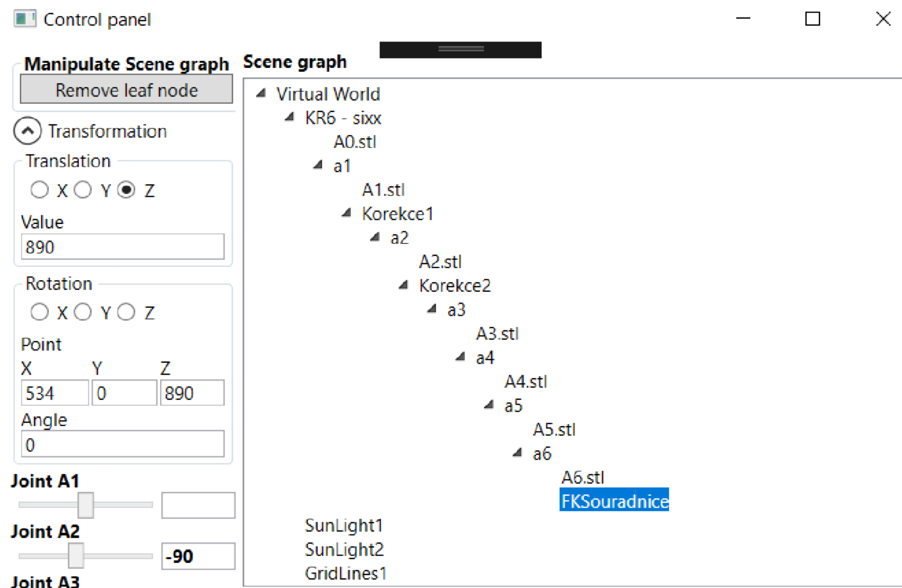
V tomto okně se nachází ovládací prvek TreeView zobrazující hierarchickou strukturu scény. Vnitřní uzly typu SceneNode hierarchie jsou vykresleny datovou šablonou HierarchicalDataTemplate, kde je upřesněn typ uzlu a cesta ke kolekci uzlů. Uzly reprezentující listy stromu musí mít pro správné vykreslení vlastní datovou šablonu DataTemplate, kde jsem upřesnil typ těchto uzlů. Výsledný kód je v příloze B.3. Pro manipulaci se scénou je implementován příkaz, který se vykoná při vybrání uzlu v tomto prvku. Tento příkaz umožní použití tlačítek umístěných vlevo nahoře v okně. K číselnému polohování kloubů jsem vytvořil posuvné jezdce a editační políčka. Editační políčka jsou data bindingem spojena s hodnotou jezdce, viz výpis B.5. Pro připojení k TCP/IP serveru slouží tlačítko Connect. Pro výslednou hodnotu přímé úlohy kinematiky označíme v grafu scény koncový rám robota a výsledek se zobrazí pod textem Offset.



Obr. 4.4: Ovládací panel pro scénu

Ovládací panel dále obsahuje sekci pro transformování modelů, která lze rozložit

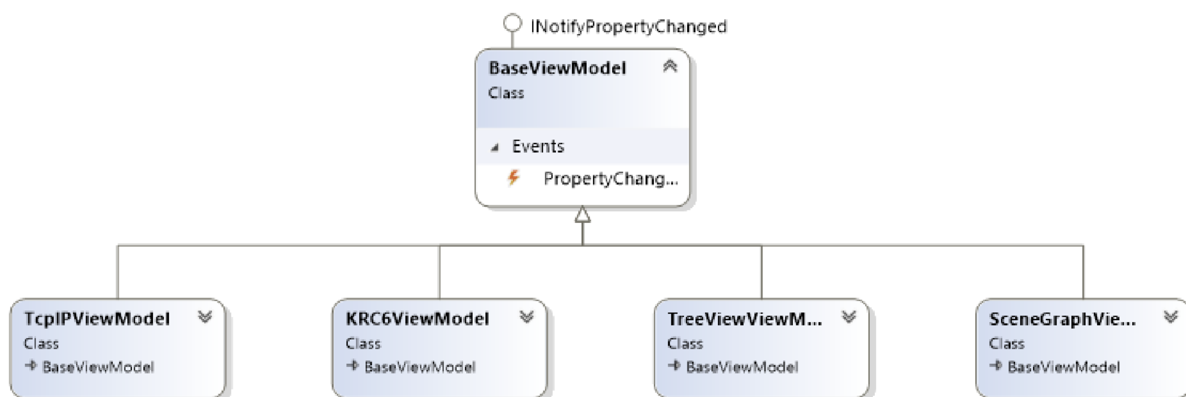
kliknutím na ovládací prvek Expander s názvem Transformation. Zaškrtnutím políček X, Y nebo Z lze vybrat osa, podle které se bude provádět posuv nebo rotace. Editační políčko na textem Value je pro hodnotu posuvu a editační políčko nad textem Angle pro hodnotu úhlu rotace. V sekci Rotation jsou také souřadnice bodu, kolem kterého se má rotace vykonat.



Obr. 4.5: Zobrazení scény

4.4.3 Třídy pro ViewModel

V této sekci jsou popsány třídy zastávající funkci ViewModelů. V nich jsou definovány vlastnosti uchovávající stav aplikace. Tyto třídy rozdělují data potřebná v uživatelském rozhraní do logických bloků a zpřehledňují tak výslednou aplikaci. Všechny kolekce jsou typu ObservableCollection, která implementuje rozhraní INotifyCollectionChanged, které aktualizuje uživatelské rozhraní při přidání či odebrání prvků z kolekce. V těchto třídách je uchována logika všech příkazů.



Obr. 4.6: Class diagram ViewModelů

Třída BaseViewModel

Základní třída, ze které dědí všechny ViewModely. Tento ViewModel implementuje rozhraní INotifyPropertyChanged, které oznamuje změnu vlastností svázaných data bindingem k uživatelskému rozhraní. Aby se nemuselo nastavovat vykonávání události při zápisu do každé vlastnosti, použil jsem knihovnu Propertychanged.Fody.

Výpis 4.4: Implementace BaseViewModel.cs

```
1 public class BaseViewModel : INotifyPropertyChanged
2     {public event PropertyChangedEventHandler PropertyChanged =
          (sender, e) => { };}
```

Třída SceneGraphViewModel

ViewModel, který udržuje stav grafu scény a slouží jako hlavní ViewModel, kde jsou definovány ostatní ViewModely.

Výpis 4.5: Implementace BaseViewModel.cs

```
1  public class SceneGraphViewModel : BaseViewModel
2      {
3          public KRC6ViewModel iKRC6ViewModel { get; set; }
4          public ObservableCollection<RootNode> iRoot { get; set; }
5          public TreeViewViewModel iTreeViewVM { get; set; }
6          public ICommand iCameraChange { get; set; }
7          public SceneGraphViewModel(){}
8          public bool CanChangeCamera()
9          public void OnCameraChange()
10     }
```

- iRoot - Nejvyšší uzel grafu scény. Jelikož zobrazují graf scény v prvku treeview, musí být tento uzel umístěn v kolekci.
- iKRC6ViewModel - ViewModel pro definici robota Kuka KRC6.
- iTreeViewVM - ViewModel pro úpravu grafu scény.
- iCameraChange - Příkaz pro manipulaci s kamerou a metody CanChangeCamera() a OnCameraChange() jako logika tohoto příkazu.

Třída KRC6ViewModel

Definice robota Kuka KRC6 sixx. Nachází se zde pomocné vektory pro určení rotace kloubů, materiál použit pro modely, název složky, kde se nachází tyto modely. Výsledný podstrom reprezentující robot je vytvořen v konstruktoru.

Výpis 4.6: Implementace KRC6ViewModel.cs

```
1 public class KRC6ViewModel : BaseViewModel
2     {
3         public CoordinateSystemVisual3D FKsouradnice;
4         public LoadObject KUKAparts
5         public RobotManipulator Kuka
6         public TcpIPViewModel KukaCommunication
7         public KRC6ViewModel()
8     }
```

- FKSouradnice - Pomocný 3D objekt pro reprezentaci nástroje robota. Translační matice tohoto prvku obsahuje výsledek přímé úlohy kinematiky.
- KUKAparts - třída obsahující importované modely a jejich názvy.
- Kuka-Uzel reprezentující podstrom robota Kuka KRC6 sixx.

Třída TreeViewViewModel

Logika pro ovládací prvek TreeView. Příkazy pro přidání a odebrání uzlů. Nedařila se mi aktualizace uživatelského rozhraní při změne kolekce obsahující modely pro vykreslení v třídě RootNode, proto je zde i kolekce iVisuals a všechny příkazy tuto kolekci aktualizují.

Výpis 4.7: Implementace TreeViewViewModel.cs

```
1 public class TreeViewViewModel : BaseViewModel
2     {
3         public ICommand iSelectedItem
4         public ICommand iAddGroupNode
5         public ICommand iAddLeafNode
6         public ICommand iRemoveGroupNode
7         public ICommand iRemoveLeafNode
8         public Visibility iGroupnode
```

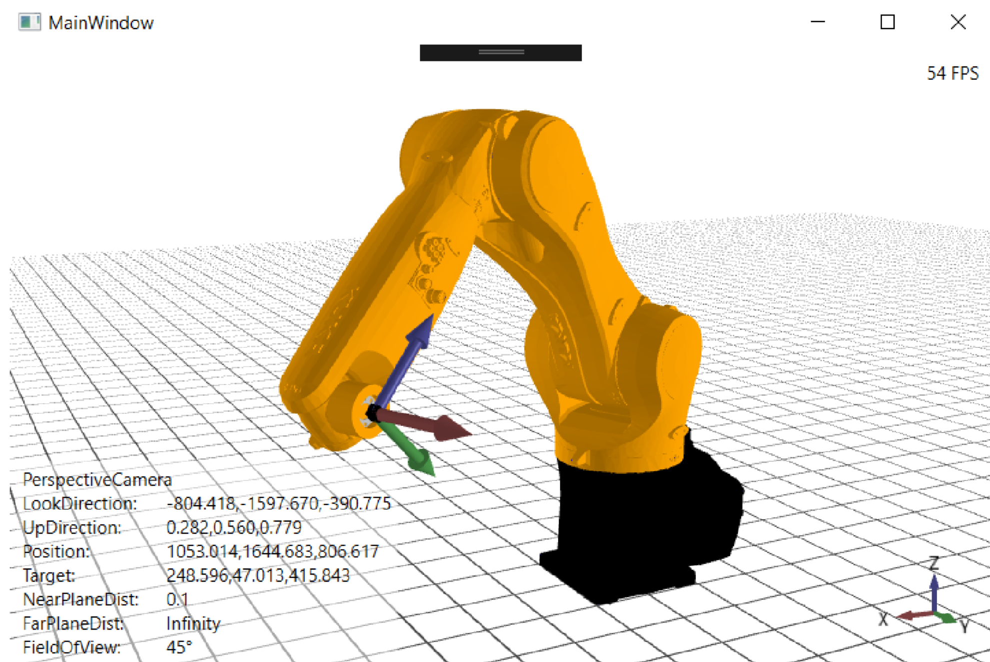
```
9           public Visibility iLeafnode
10          public bool iIsSelectedItem
11          public ObservableCollection<ModelVisual3D> iVisuals
12          public string iForwardKinematics
13          public double iTranslation
14          public int iAxisToTranslate
15          public double iAngle
16          public Point3D iRotatePoint
17          public int iAxisToRotate
18          public TreeViewViewModel()
19          public bool IsSelected()
20          public bool CanExecute()
21          public void OnSelectedItem()
22          public void OnAddGroupNode()
23          public void OnAddLeafNode()
24          public void OnRemoveGroupNode()
25          public void OnRemoveLeafNode()
26          public void OnChecked()
27          public void OnCheckedRotate()
28          public void OnTextChanged()
29          public void OnTextRotateChanged()
30          public void OnSetPosition()
31      }
```

- iSelectedItem - Tento příkaz rozhoduje o zobrazení tlačítek v uživatelském rozhraní podle vybraného typu uzlu. Nastavuje hodnotu vlastností iGroupnode a iLeafnode. Pro vykonání tohoto příkazu musí vracet metoda CanExecute() boolean hodnotu true.
- iAddGroupNode - Přidá skupinový uzel do grafu scény. Logika příkazu je metoda OnAddGroupNode().
- iRemoveGroupNode - Odebere skupinový uzel z grafu scény. Logika příkazu je metoda OnRemoveGroupNode().
- iAddLeafNode - Přidá uzel reprezentující grafický objekt do grafu scény. Při vykonávání tohoto příkazu se zobrazí dialogové okno pro výběr .stl souboru. Logika příkazu je metoda OnAddLeafNode().
- iRemoveLeafNode - Odebere uzel typu list z grafu scény. Logika příkazu je metoda OnRemoveLeafNode().
- Metoda isSelected - Rozhoduje o možnosti provedení všech příkazů ovlivňující strukturu grafu scény.
- iForwardKinematics - Řetězec obsahující hodnotu úlohy přímé kinematiky pro vybraný model.
- Metoda OnChecked - Tato metoda obsluhuje výběr osy pro transformaci typu translace z ovládacích prvků typu RadioButton.

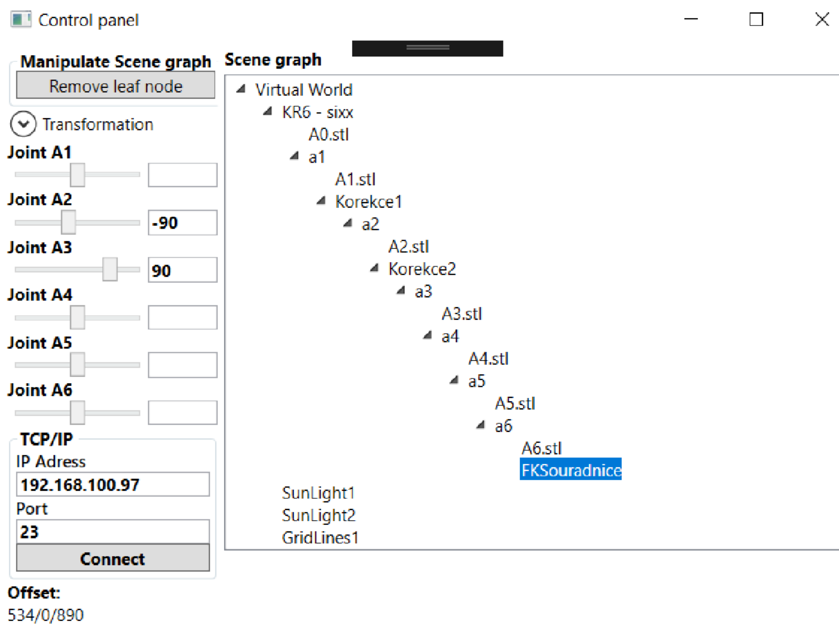
- Metoda OnCheckedRotate - Výběr osy pro transformaci typu rotace z ovládacích prvků typu RadioButton. Při výběru uzlu typu RevoluteJoint nebo PrismaticJoint se osa otáčení řídí dle vlastnosti iDoF.
- Metoda OnTextChanged - Při změně hodnoty editačního políčka pro nastavení hodnoty translace převede daný řetězec na hodnotu typu double a vytvoří novou translační matici.
- Metoda OnTextRotateChanged - Podobně jako metoda OnTectChanged i zde získáme hodnotu, ale pro rotační matici.
- Metoda OnSetPosition - Z editačních políček pro změnu bodu otáčení se získají hodnoty a vytvoří se nový bod v prostoru ve vybraném uzlu.

4.4.4 Testování polohování robotu

Z dokumentace robotu jsem získal kladné směry otáčení kloubů, viz. příloha A.3. Jezdcem nebo editačním políčkem jsem nastavil hodnotu kloubu a a jestli se robot ve vizualizaci otočil ve špatném směru, nastavil jsem invertování pohybu kloubu. Takto jsem nastavil všechny klouby. Na obrázku 4.7 lze vidět natočení kloubů a na obrázku 4.8 v editačním políčku hodnotu natočení. V tomto obrázku je také zobrazena pozice tohoto rámu pod textem Offset a zakrytí některých tlačítek pro manipulaci s hiearachií. U uzlu typu list má smysl jen tlačítko pro odstranění tohotu uzlu.

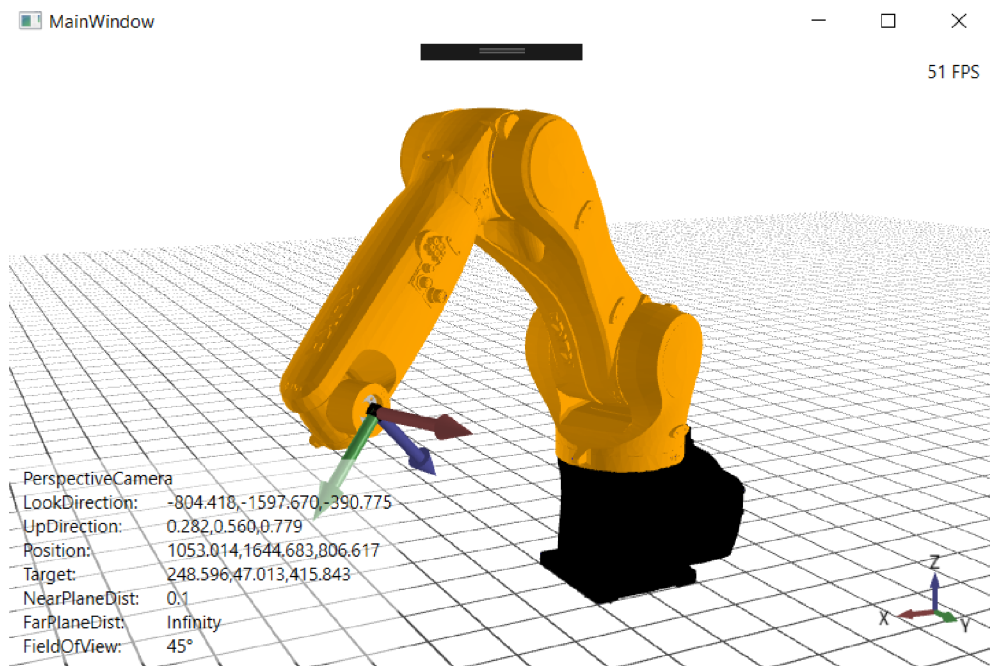


Obr. 4.7: Scéna s robotem



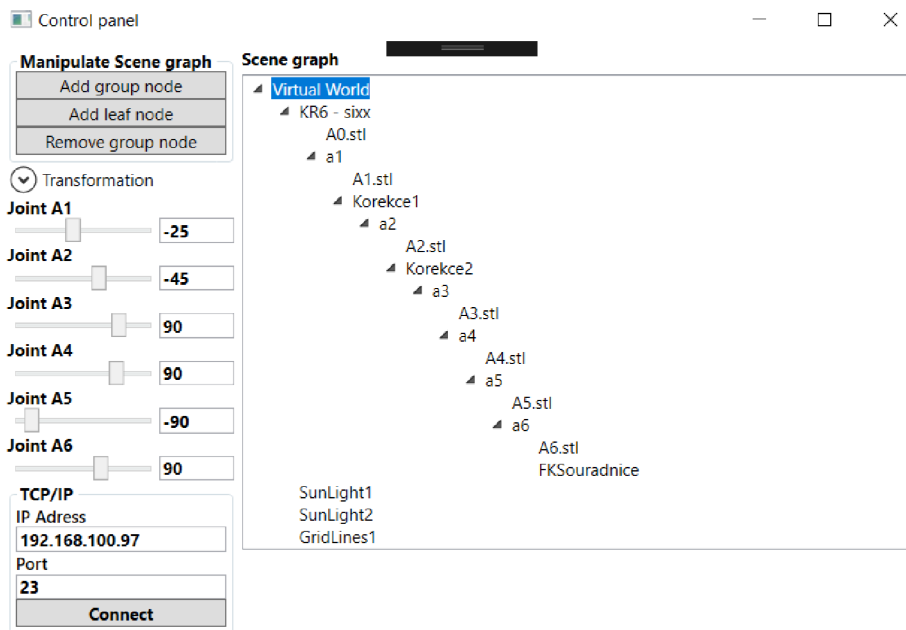
Obr. 4.8: Ovládací panel

V obrázku 4.9 je poslední kloub robotu otočem o 90° oproti předcházejícímu obrázku. Rozdílu si lze všimnout na orientaci souřadného rámu na konci robotu



Obr. 4.9: Scéna s robotem 2

Poslední obrázek ovládacího panelu, kde můžeme vidět označený kořen scény a další zobrazená tlačítka pro manipulaci se scénou v levém horním rohu.



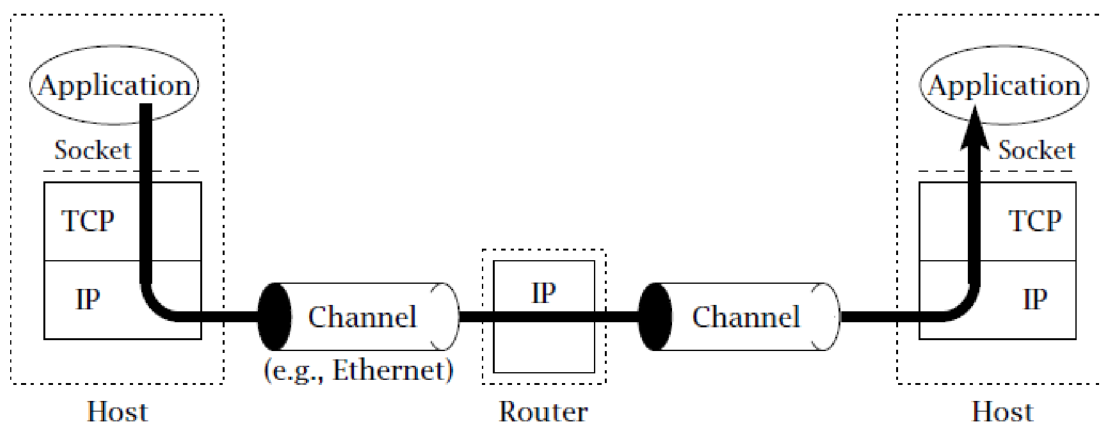
Obr. 4.10: Ovládací panel 2

5 Protokolový most

Robot Kuka KRC6 obsahuje několik aplikací běžících paralelně. Jedním z těchto aplikací je TCP/IP server, který odesílá data o aktuálním stavu robota. Tento server funguje jako program v jazyce KRL (Kuka Robot Language). Pro konfiguraci programu slouží XML soubor, který je uložen ve složce souboru robota. Konfigurace je nastavena při vzniku připojení. Pro přenos datových packetů je možné také protokolem UDP/IP.

5.1 Komunikační protokol TCP/IP

Komunikační protokol je soubor pravidel, který určuje, jak budou účastníci počítačové sítě mezi sebou komunikovat. Počítačová síť je propojená pomocí komunikačních kanálů, které propojují hosty mezi sebou. Mezi těmito hosty může být připojen router. Data posílaná po komunikačním kanálu jsou sekvence bytů, které jsou rozděleny do packetů. Jak budou packety strukturovány, určuje komunikační protokol. Hlavní části TCP/IP protokolu jsou Transmission Control Protocol (TCP), User Datagram Protocol (UDP) a Internet Protocol (IP). V této aplikaci budeme komunikovat jenom s pomocí transportní vrstvy typu TCP.[11]



Obr. 5.1: TCP/IP síť(vychází z [11])

TCP/IP je strukturován do čtyř vrstev:

- Aplikační vrstva
- Transportní vrstva
- Síťová vrstva
- Vrstva síťového rozhraní

Aplikační vrstva přistupuje k transportní vrstvě pomocí socketů. Síťová vrstva protokolem IP se snaží doručit data ke koncovému adresátovi. Má tedy na starosti adresování a doručování dat hostům v síťové vrstvě, směrování a propojení s transportní vrstvou. Doručení dat až do aplikační vrstvy zaručuje transportní vrstva TCP a UDP jako číslo portu, které identifikuje konkrétní cílovou aplikaci.

Komunikační protokol TCP/IP rozděluje účastníky na dva typy - klient a server. Klient zahájí komunikaci, zatímco server čeká na žádost od uživatele, na kterou následně odpoví. Typ účastníka určuje socket. Klient na rozdíl od serveru musí znát adresu a číslo portu. Socket je abstrakce, pomocí které aplikace přijímá a odesílá data. Socket umožní připojení aplikace do sítě a následnou komunikaci s ostatními aplikacemi. Transportní vrstva TCP využívá socket typu stream, který pracuje s daty jako byte-stream, tedy odesílatel zapisuje do streamu data v bytech.[11]

5.2 Ethernet KRL

Posílaná data jsou uložena v paměťové struktuře. Data mohou být uložena jako fronta a můžeme je číst způsobem FIFO (First In First Out) a nebo jako zásobník LIFO (Last In First Out). Při variantě FIFO je pořadí čtení stejné jako pořadí vkládání prvků. LIFO čte poslední prvek zásobníku jako první, tedy opačně než při vkládání. Veškerá konfigurace KRL je nastavována v XML souboru robota. Konkrétní nastavení robota je v příloze B.1. Nastavení je rozděleno do několika sekcí:

- Configuration - Konfigurace připojení mezi externím systémem a Ethernet KRL. Jsou zde parametry potřebné k uskutečnění komunikace pomocí TCP/IP protokolu jako upřesnění typu externího systému (klient nebo server), IP adresa, data port, velikost a způsob čtení zprávy, způsob oznámení při úspěšném připojení k robotu.
- Recieve - Data, která jsou posílána robotu z externího systému.
- Send - Data oznamující aktuální status robota. Nás bude zajímat jenom tagy pro úhly jednotlivých kloubů robota s tagem Status/CurrAxis/@A1-6.

5.3 Implementace protokolového mostu

Pro navázání komunikace přes TCP/IP klienta lze použít třídu TcpClient z knihovny System.Net.Sockets. Objektu této třídy přiřadíme IP adresu serveru a číslo portu, na kterém server naslouchá. Jestliže se klient úspěšně připojí k serveru, můžeme získat data ze streamu tohoto klienta. Odesílaná data ze serveru robota jsou ve tvaru XML souboru. Pro čtení XML dat ze streamu lze využít třídu XMLReader z knihovny System.Xml. Čtení dat ze streamu se provádí v přiřazeném vlákně.

Kdyby se proces komunikace vykonával na hlavním vlákně aplikace, způsobil by tento proces zamrznutí aplikace. Pro manipulaci s daty z více vláken využiji kolekci ConcurrentQueue obsahující prvky typu XElement, která zamezí změnu kolekce vláknem, když s ní pracuje jiné vlákno. Veškerá logika pro připojení a čtení dat je v třídě KukaComm.

Výpis 5.1: Třída KukaComm.cs

```
public class KukaComm : INotifyPropertyChanged
    {
        public ConcurrentQueue<XElement> iXMLelements
        public TcpClient iTcpSocket
        public NetworkStream iStream
        public Thread iThread
        public ObservableCollection<myDouble> iJointValues
        public KukaComm()
        public bool ConnectClient()
        public void ReadTcpIP()
        public void Parsuj()
        public void SetThread()
        //Vnořené třídy
        public class myDouble
    }
```

- Metoda ConnectClient - Pokusí se připojit klienta k serveru. Vrací výsledek pokusu o připojení v hodnotě bool.
- Metoda ReadTcpIP - Tato metoda definuje třídu XmlReaderSettings pro nastavení vlastností třídy XmlReader a čte ve while smyčce XML soubor ze streamu TcpIP klienta. Při úspěšném přečtení XML elementu jej zařadí do kolekce a zavolá metodu Parsuj().
- Metoda Parsuj - Tato metoda zkouší ve while smyčce odebírat XML elementy z kolekce iXMLelements. Při úspěšném získání elementu CurrAxis se prvek uloží do kolekce, která udržuje stav natočení kloubů.
- Metoda SetThread - Kontrola a vytvoření vlákna pro čtení XML dat.
- Metoda KukaComm - Konstruktor třídy
- Třída myDouble - Jelikož kolekce typu ObservableCollection oznamuje uživatelské rozhraní, jenom při odebrání a přidání prvků kolekce. Pro aktualizaci hodnoty v uživatelském prostředí musí prvky v této kolekci implementovat rozhraní INotifyPropertyChanged. Toto je tedy jenom pomocná třída obalující datový typ double.

Třída TcpIPViewModel

ViewModel pro komunikaci TCP/IP.

Výpis 5.2: Implementace TcpIPViewModel.cs

```
1  public class TcpIPViewModel : BaseViewModel
2      {
3          public string iIPAdress
4          public string iPort
5          public ObservableCollection<RevoluteJoint> iJoints
6          public string iConnectButtonContent
7          public KukaComm iBridge
8          public ICommand iSliderCommand
9          public ICommand iConnectClick
10         public TcpIPViewModel()
11         public bool CanConnect()
12         public void SliderChanged()
13         public void Connect()
14     }
```

- iSliderComman - Příkaz, který se vykoná při změně hodnoty jezdce v uživatelském rozhraní. Pomocí definovaných tagů jezdce zjistí kloub, který se má pootočit v kolekci iJoints a aktualizuje celkovou transformaci podstromu grafu scény, kde je kořenem pootočený kloub. Logika příkazu je metoda SliderChanged().
- iConnectClick - Příkaz pro možnost připojení a odpojení TCP/IP klienta. IP adresu serveru získá metoda z vlastnosti iIPAdress a port se parsuje z textu editačního políčka v uživatelském prostředí spojeného data bindigem s vlastností iPort. Logika příkazu je metoda Connect(). Tato metoda nastavuje vlastnosti iConnectButtonContent, která se využívá pro text tlačítka indikující připojení TCP/IP klienta.
- Metoda CanConnect - Určuje, zda můžeme vykonat příkazy v této třídě.
- TcpIPViewModel - Konstruktor, který definuje příkazy a vlastnosti modelu iBridge.

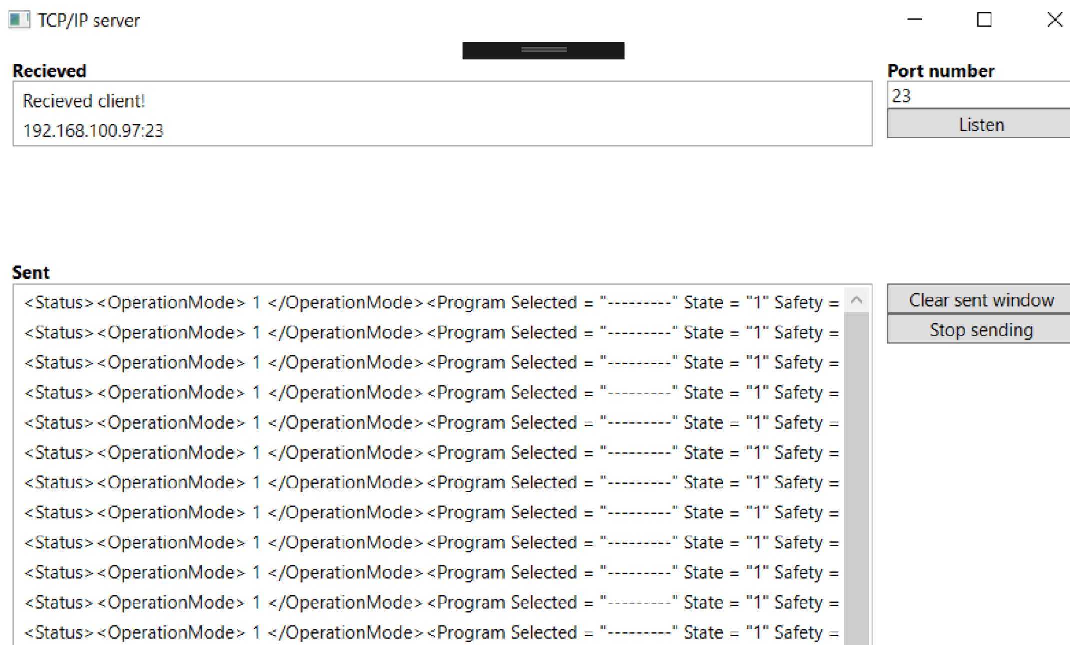
Testování protokolového mostu

Testování jsem provedl vytvořením programu, který simuluje TCP/IP server, který zapisuje XML data do byte-streamu TCP/IP klienta. Do editačního políčka se napíše číslo portu, na kterém bude server naslouchat TCP/IP klientovi. Tlačítkem Listen se spustí naslouchání a při připojení klienta server začne v periodickém intervalu generovat náhodné hodnoty natočení kloubů a posílat je připojenému klientovi. K periodické kontrole zaslání žádostí klientů jsem použil třídu Timer z knihovny System.Threading. Tato třída ve vlákně volá metodu GetTcpClient(), která zjistí, zda má server nějakou žádost od klienta, pokud ji má, klienta připojí, zapíše záznam do okna Recieved, vypne Timer pro zpracování žádosti klientů a vyvolá další metodu pomocí třídy Timer pro periodický zápis dat do streamu. Změna prvků uživatelského rozhraní ve vlákně se musí vykonávat metodou Invoke() objektu typu Dispatcher, který bezpečně přistoupí k zápisu a čtení dat ovládacího prvku. Data jsou vytvořena tak, aby hodnoty kloubů byly v rozmezí maximální a minimální hodnoty konkrétního kloubu.

$$U_{\mathrm{n}} = Random() \cdot (max_{\mathrm{n}} - min_{\mathrm{n}}) + min_{\mathrm{n}} \tag{5.1}$$

Funkce Random() generuje hodnotu $x \in <0;1>$. Tlačítkem Clear sent window se vymaže všechen text v okně Sent a pomocí tlačítka Stop sending se přestanou posílat data klientovi, jenž ale zůstane připojen k serveru. Data posílaná serverem jsou ve stejném tvaru jako data z živého robota, kromě upravených hodnot poloh kloubů viz výpis B.2. Pro správné čtení klieta ze streamu jsem hodnoty kloubů U_{n} typu double převedl do řetězce ve formátu CultureInfo.InvariantCulture. Tento formát je nezávislý na jazykové verzi a tedy nejsou ovlivněny konvencemi aktuální jazykové verze.

Zapneme naslouchání serveru tlačítkem Listen a v okně ovládacího panelu vyplníme IP adresu a port serveru. Poté zmáčknutím tlačítka Connect se přípojíme k serveru a simulovaný robot se začne polohovat dle dodaných dat ze serveru. Po zmáčknutí tlačítka Disconnect v ovládacím panelu se přeruší spojení se serverem a robot zůstane v poslední poloze, kterou klient přečetl ze svého streamu.



Obr. 5.2: Simulace TCP/IP serveru robotu

6 Závěr

Začátek práce se věnuje prozkoumání vlastností 3D grafické knihovny Helix Toolkit, s níž mám vytvořit simulaci stacionárního robota. Nejdříve se zde nachází popis vnitřní knihovny Media3D, která zavádí 3D funkčnost pro grafickou platformu WPF. Jsou zde popsány kroky k vytvoření grafické scény a 3D knihovny nacházející se v Helix Toolkitu, které rozšiřují 3D grafické vlastnosti .NET platformy.

K vytvoření knihovny pro simulaci stacionárního robota jsem využil knihovny Helix Toolkit.WPF, která představuje nadstavbu knihovny Media3D. V knihovně je vytvořená struktura graf scény pro popis grafické scény. Knihovna Helix Toolkit.SharpDX už má implementovaný graf scény podporovaný importovací a exportovací knihovnou SharpDX.Assimp. Vytvořil jsem tedy třídy reprezentující graf scény, které se dají rozdělit do tří skupin - kořen, větev a list stromu. Tímto jsem vytvořil hierarchii určující správnou transformaci dodaných modelů pro simulaci stacionárního robota.

Toto je řešení, které by se dalo jednoduše rozšířit bez nějaké změny již vypracovaných tříd. Nicméně pro výkonnější aplikaci by bylo lepší použít již vytvořený graf scény v knihovně Helix Toolkit.SharpDX, která má více funkcí než Helix Toolkit.WPF.

Testovací aplikace jsem vytvořil jako dvě okna. První zobrazuje grafickou scénu s robotem sestaveným z dodaných modelů. Druhé okno umožňuje parametricky polohovat klouby robota, měnit grafickou scénu a připojit se k TCP/IP serveru. Aplikaci jsem vytvořil podle návrhového vzoru MVVM.

Protokolový most přijímá z TCP/IP serveru robota XML data o stavu robota. Využil jsem systémové třídy pro vytvoření TCP/IP klienta a čtení XML dat ve vlákně, aby program nezamrzl. Z důvodu nemožnosti otestování protokolového mostu na živém robotu jsem vytvořil program, který vytvoří TCP/IP server a který poté může posílat data ve formátu XML připojenému klientovi.

Při vytváření práce byl využíván verzovací systém GitLab. Odkaz na online repozitář je v [15].

Literatura

[1] *HelixToolkit* [online]. [cit. 5. 10. 2019] Dostupné z URL: <https://github.com/helix-toolkit/helix-toolkit>.

[2] *Windows .NET WPF* [online]. [cit. 12. 10. 2019] Dostupné z URL: <https://docs.microsoft.com/en-us/dotnet/framework/wpf/>.

[3] NATHAN, Adam *WPF 4.5 unleashed* 2014 Indianapolis, Ind.: Sams, 2014 [cit. 15. 10. 2019]

[4] *The Open-Asset-Importer-Library*[online]. [cit. 12. 12. 2019]. Dostupné z URL: <http://sir-kimmi.de/assimp/lib_html/index.html>.

[5] MARK W. SPONG, SETH HUTCHINSON, and M. VIDYASAGAR *Robot Dynamics and Control Second Edition* Leden 28, 2004 [cit. 12. 10. 2019].

[6] MICHAEL ASHIKHMIN, STEVE MARSCHNER *Fundamentals of Computer Graphics 4rd Edition.* Červenec 21, 2009 [cit. 15. 11. 2019].

[7] JIŘÍ ŽÁRA, BEDŘICH BENEŠ, JIŘÍ SOCHOR, PETR FELKEL *Moderní počítačová grafika* Brno 2004 [cit. 9. 12. 2019].

[8] RUI WANG, XUELEI QIAN *OpenSceneGraph 3.0* Beginner's Guide 2010 [cit. 13. 12. 2019].

[9] *Kuka KRC6 sixx*[online]. [cit. 4. 1. 2019]. Dostupné z URL: <https://www.kuka.com/cs-cz/services/downloads?terms=Language:cs:1;Language:en:1Language:en:1&q=>.

[10] SHERIDAN YUEN *Mastering Windows Presentation Foundation* 02/2017 Birmingham, United Kingdom [cit. 25. 2. 2019].

[11] MICHAEL J. DONAHOO, KENNETH L. CALVERT *TCP/IP Sockets in C : Practical Guide for Programmers* 07/2009 San Francisco, United States [cit. 25. 5. 2019].

[12] RICK PARENT *Computer animation algorithms & techniques third edition* 09/2012 Ohio State University [cit. 25. 5. 2019].

[13] ERICH GAMMA, RICHARD HELM, RALPH JOHNSON, JOHN VLISSIDES *Design Patterns - Elements of Reusable Object-Oriented Software* [cit. 25. 5. 2019].

[14] THOMAS H. CORMEN, CHARLES E.LEISERSON, RONALD L. RIVEST, CLIFFORD STEIN *Introduction to algorithms* 2009 Cambridge, Massachusetts London, England [cit. 25. 5. 2019].

[15] MILAN DOLEŽAL *Vizualizace stacionárního robotu v C#/WPF* 2020, Brno. Bakalářská práce [cit. 7. 6. 2019] Dostupné z URL: <`https://student.robotika.ceitec.vutbr.cz/DPBP/2019_bp_dolezal_vizualizacewpf`>.

Seznam symbolů, veličin a zkratek

XAML	Značkovací jazyk pro tvorbu uživatelského rozhraní – eXtensible Application Markup Language
XML	Obecný značkovací jazyk – eXtensible Markup Language
WPF	Framework pro tvorbu formulářových aplikací – Windows Presentation Foundation
MVVM	Návrhový vzor pro WPF aplikace – Model-ViewModel-Model
CLR	Část virtuálního stroje, kompilace – Common Language Runtime
FIFO	První jde dovnitř a první také ven. Struktura fronty – First In First Out
LIFO	Poslední jde dovnitř, ale jako první jde ven – Last In First Out
TCP	Protokol transportní vrstvy – Transmission Control Protocol
UDP	Protokol transportní vrstvy – User Datagram Protocol
KRL	Jazyk pro roboty kuka –Kuka RObot Language
IP	Adresa v počítačové síti – Internet Protocol

Seznam příloh

A Geometrie robotu Kuka KRC6 sixx **55**

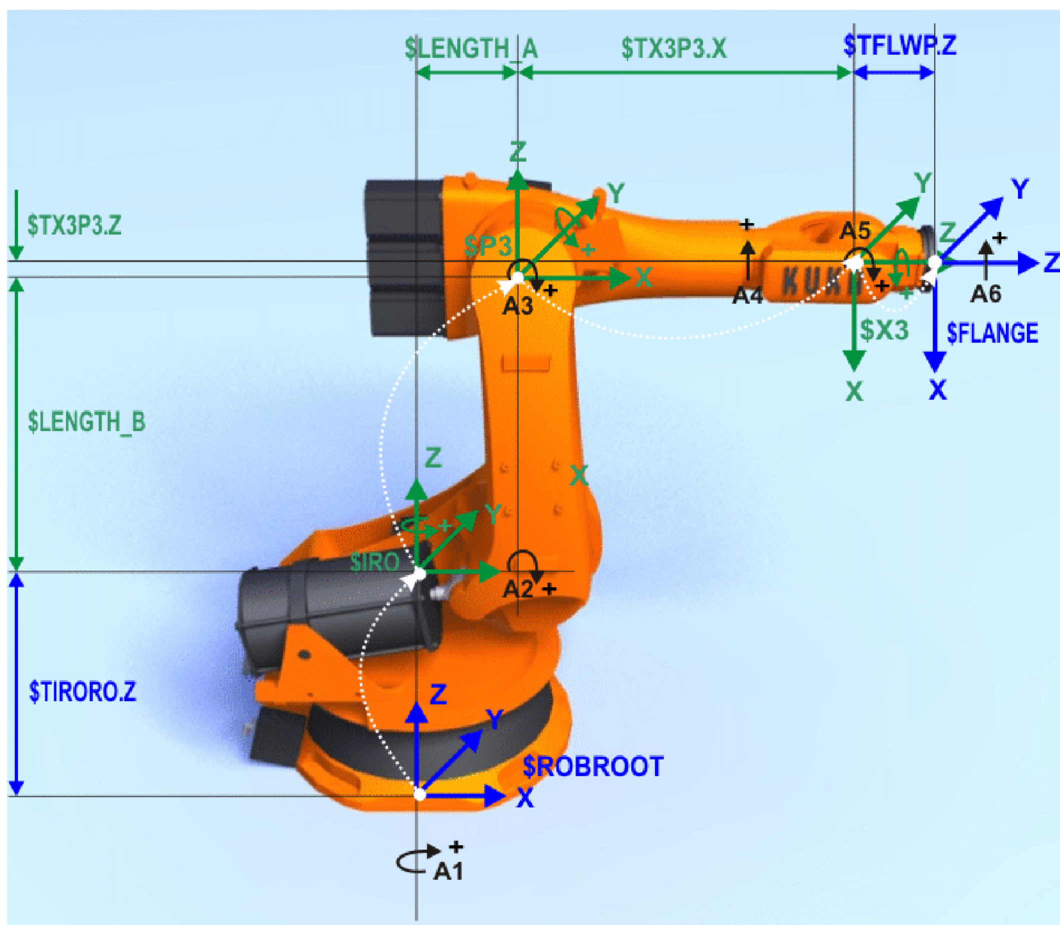
B Výpisy z programu **58**

B.1 Konfigurace Ethernet KRL 58

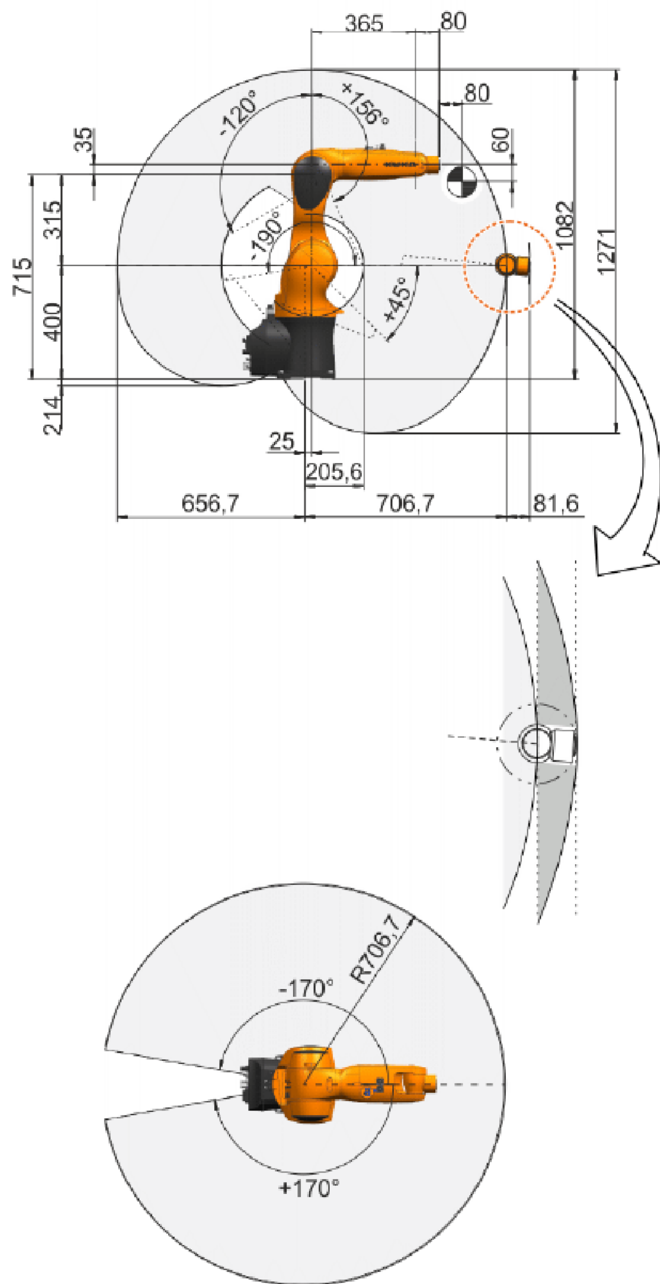
C Elektronická příloha **63**

C.1 Knihovny 63

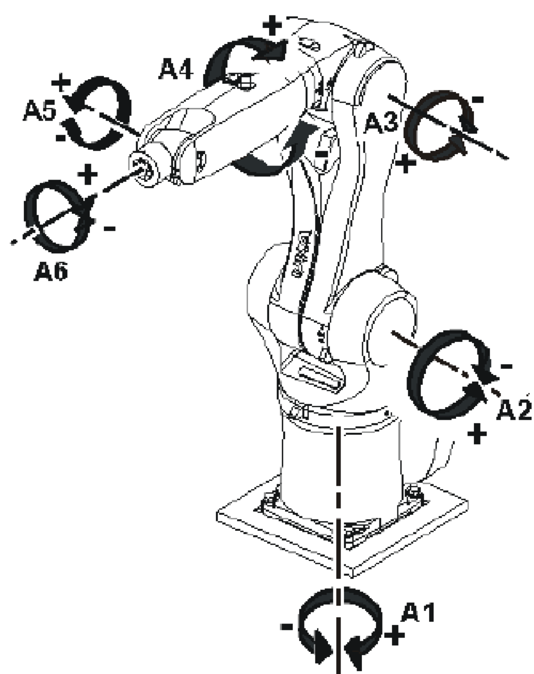
A Geometrie robotu Kuka KRC6 sixx



Obr. A.1: Směr otáčení kloubů



Obr. A.2: Výchozí pozice [9]



Obr. A.3: Směr natáčení kloubů [9]

B Výpisy z programu

B.1 Konfigurace Ethernet KRL

Výpis B.1: XML Ethernet KRL

```
<ETHERNETKRL>
  <CONFIGURATION>
    <EXTERNAL>
      <TYPE>Client</TYPE>
    </EXTERNAL>
    <INTERNAL>
      <PROTOCOL>TCP</PROTOCOL>
      <IP>10.3.0.12</IP>
      <PORT>54601</PORT>
      <ALIVE Set_Flag="10"/>
      <MESSAGES Logging="warning" Display="disabled"/>
      <BUFFERING Mode="FIFO" Limit="256" />
      <ENVIRONMENT>Submit</ENVIRONMENT>
    </INTERNAL>
  </CONFIGURATION>
  <RECEIVE>
    <XML>
      <ELEMENT Tag="Sigin/DesiredPos/@X" Type="REAL" />
      <ELEMENT Tag="Sigin/DesiredPos/@Y" Type="REAL" />
      <ELEMENT Tag="Sigin/DesiredPos/@RZ" Type="REAL" />
      <ELEMENT Tag="Sigin/Gripper" Type="BOOL" />
      <ELEMENT Tag="Sigin/Enable" Type="BOOL" />
      <ELEMENT Tag="Sigin/SelectApp" Type="INT"/>
      <ELEMENT Tag="Sigin/Control/@Id" Type="INT" />
      <ELEMENT Tag="Sigin/Control/@Version" Type="INT" />
      <ELEMENT Tag="Sigin" Set_Flag="11" />
    </XML>
  </RECEIVE>
  <SEND>
    <XML>
      <ELEMENT Tag="Status/OperationMode" Type="INT"/>
      <ELEMENT Tag="Status/Program/@Selected" Type="STRING"/>
      <ELEMENT Tag="Status/Program/@State" Type="INT"/>
      <ELEMENT Tag="Status/Program/@Safety" Type="INT"/>
      <ELEMENT Tag="Status/CurrPos/@X" Type="REAL"/>
      <ELEMENT Tag="Status/CurrPos/@Y" Type="REAL"/>
      <ELEMENT Tag="Status/CurrPos/@Z" Type="REAL"/>
      <ELEMENT Tag="Status/CurrPos/@RZ" Type="REAL"/>
      <ELEMENT Tag="Status/CurrPos/@RY" Type="REAL"/>
      <ELEMENT Tag="Status/CurrPos/@RX" Type="REAL"/>
```

```
      <ELEMENT Tag="Status/CurrPos/@S" Type="INT"/>
      <ELEMENT Tag="Status/CurrPos/@T" Type="INT"/>
      <ELEMENT Tag="Status/CurrPos/@TOOL" Type="INT"/>
      <ELEMENT Tag="Status/CurrPos/@BASE" Type="INT"/>
      <ELEMENT Tag="Status/CurrAxis/@A1" Type="REAL"/>
      <ELEMENT Tag="Status/CurrAxis/@A2" Type="REAL"/>
      <ELEMENT Tag="Status/CurrAxis/@A3" Type="REAL"/>
      <ELEMENT Tag="Status/CurrAxis/@A4" Type="REAL"/>
      <ELEMENT Tag="Status/CurrAxis/@A5" Type="REAL"/>
      <ELEMENT Tag="Status/CurrAxis/@A6" Type="REAL"/>
      <ELEMENT Tag="Sigout/DesiredPos/@X" Type="REAL" />
      <ELEMENT Tag="Sigout/DesiredPos/@Y" Type="REAL" />
      <ELEMENT Tag="Sigout/DesiredPos/@RZ" Type="REAL" />
      <ELEMENT Tag="Sigout/Gripper" Type="BOOL"/>
      <ELEMENT Tag="Sigout/Enable" Type="BOOL"/>
      <ELEMENT Tag="Sigout/SelectApp" Type="INT"/>
      <ELEMENT Tag="Sigout/Control/@Id" Type="INT" />
      <ELEMENT Tag="Sigout/Control/@Version" Type="INT" />
    </XML>
  </SEND>
</ETHERNETKRL>
```

Výpis B.2: XML data z TCP/IP serveru

```
<Status>
  <OperationMode>1</OperationMode>
  <Program Selected="----------" State="1" Safety="12">
  </Program>
  <CurrPos X="0.000000" Y="0.000000" Z="0.000000" RZ="0.000000"
     RY="0.000000" RX="0.000000" S="0" T="0" TOOL="-1" BASE="-1">
  </CurrPos>
  <CurrAxis A1="-6.620526" A2="-89.371811" A3="117.436874"
     A4="-0.428021" A5="61.908714" A6="-51.404312">
  </CurrAxis>
</Status>
```

Výpis B.3: XAML TreeView

```
<TreeView Margin="0 3 0 0" Name="SceneHierarchy"
   ItemsSource="{Binding iRoot}"   >
  <TreeView.Resources>
    <HierarchicalDataTemplate DataType="{x:Type hw:SceneNode}"
   ItemsSource="{Binding iChildren }" >
      <TextBlock Text="{Binding iName}" />
    </HierarchicalDataTemplate>
    <DataTemplate DataType="{x:Type hw:VisualNode}">
      <TextBlock Text="{Binding iName}" />
    </DataTemplate>
```

```
9      </TreeView.Resources>
10     <TreeView.ItemContainerStyle>
11       <Style TargetType="{x:Type TreeViewItem}">
12         <Setter Property="IsExpanded" Value="True"/>
13       </Style>
14     </TreeView.ItemContainerStyle>
15     <i:Interaction.Triggers>
16       <i:EventTrigger EventName="SelectedItemChanged">
17         <i:InvokeCommandAction Command="{Binding
              iTreeViewVM.iSelectedItem}" PassEventArgsToCommand="True">
18         </i:InvokeCommandAction>
19       </i:EventTrigger>
20     </i:Interaction.Triggers>
21   </TreeView>
```

Výpis B.4: Závislá vlastnost pro příkaz

```
1   public class SliderProperties
2   {
3       public static DependencyProperty SliderValueChangedProperty =
4          DependencyProperty.RegisterAttached("SliderValueChanged",
              typeof(ICommand), typeof(SliderProperties),
5          new PropertyMetadata(OnSliderValueChanged));
6       public static ICommand GetSliderValueChanged(DependencyObject
           aDepObject)
7       {
8           return
              (ICommand)aDepObject.GetValue(SliderValueChangedProperty);
9       }
10      public static void SetSliderValueChanged(DependencyObject
           aDepObject, ICommand aValue)
11      {
12          aDepObject.SetValue(SliderValueChangedProperty, aValue);
13      }
14      public static void OnSliderValueChanged(DependencyObject
           aDepObject, DependencyPropertyChangedEventArgs aE)
15      {
16          Slider tSlider = aDepObject as Slider;
17          if (aE.OldValue == null && aE.OldValue != null)
18              tSlider.ValueChanged += Slider_SliderValueChanged;
19          else if (aE.OldValue != null && aE.NewValue == null)
20              tSlider.ValueChanged -= Slider_SliderValueChanged;
21          if (aE.NewValue != aE.OldValue)
22              tSlider.ValueChanged += Slider_SliderValueChanged;
23      }
24      private static void Slider_SliderValueChanged(object sender,
           RoutedPropertyChangedEventArgs<double> eE)
```

```
25      {
26          Slider tSlider = sender as Slider;
27          ICommand Command = GetSliderValueChanged(tSlider);
28          if (Command != null && Command.CanExecute(tSlider))
29              Command.Execute(tSlider);
30      }
31  }
```

Výpis B.5: Jezdec a editační políčko

```
1   <DockPanel>
2       <TextBlock DockPanel.Dock="Top" Text="Joint A6"
3                  FontWeight="Bold"
4       </TextBlock>
5       <Grid>
6           <Grid.ColumnDefinitions>
7               <ColumnDefinition Width="2*"/>
8               <ColumnDefinition Width="*"/>
9           </Grid.ColumnDefinitions>
10      <Slider Tag="6" x:Name="SliderA6"
11       Value="{Binding Path=iKRC6ViewModel.KukaCommunication.iBridge.
12                  iInterJointValues[5].iValue, Mode=TwoWay"
13       attach: SliderProperties.SliderValueChanged = "{Binding
            iKRC6ViewModel.KukaCommunication.iSliderCommand}"
14       Grid.Column="0" Minimum="{Binding
            iKRC6ViewModel.KukaCommunication.iJoints[5].iMinAngle}"
15       Maximum="{Binding
            iKRC6ViewModel.KukaCommunication.iJoints[5].iMaxAngle}"
16      </Slider>
17      <TextBox Text="{Binding ElementName=SliderA6, Path=Value,
18                     UpdateSourceTrigger=PropertyChanged,
19                     StringFormat={}{0:#.###}}"
20              DockPanel.Dock="Right"
21              TextAlignment="Left" FontWeight="Bold" Grid.Column="1"
22      </TextBox>
23      </Grid>
24  </DockPanel>
```

Výpis B.6: Relay command

```
1   public class RelayCommand : ICommand
2       {
3           private readonly Action<object> iExecute;
4           private readonly Predicate<object> iCanExecute;
5           public RelayCommand(Predicate<object> aCanExecute,
                Action<object> aExecute)
6           {
7               iCanExecute = aCanExecute;
```

```
8               iExecute = aExecute;
9           }
10
11          public event EventHandler CanExecuteChanged
12          {
13              add { CommandManager.RequerySuggested += value; }
14              remove { CommandManager.RequerySuggested -= value; }
15          }
16
17          public bool CanExecute(object parameter)
18          {
19              return iCanExecute == null ? true :
                    iCanExecute(parameter);
20          }
21
22          public void Execute(object parameter)
23          {
24              iExecute.Invoke(parameter);
25          }
26      }
```

C Elektronická příloha

```
/ ............................................. kořenový adresář přiloženého CD
├── data
│   └── Kuka KRC6 sixx ............................. Složka s .stl soubory robota
├── doc .......................................................... Dokumentace
│   └── BP.pdf ..................... Výsledný soubor dokumentace bakalářské práce
├── HelloWorld3D ......................... Projekt knihovny pro polohování robota
├── src ............................................ Projekt testovací aplikace
│   ├── BP_Simulace .......................... Zdrojové soubory testovací aplikace
│   └── HelloWorld3D knihovna
│       └── HelloWorld3D.dll ..................... Knihovna pro testovací aplikaci
├── TCPIPserver ....................................... Projekt TCP/IP serveru
├── FodyWeavers ................. XML soubor pro knihovnu Propertychanged.Fody
└── GIT ........................................................ Odkaz na Gitlab
```

C.1 Knihovny

Jestliže nepujde zkompilovat projekt testovací aplikace díky knihovně Fody, musí se XML soubor FodyWeavers v adresáři BP_Simulace přepsat XML souborem FodyWeavers v adresáři s ostatními projekty.

V projektu testovací aplikace se musí v referencích vybrat cesta ke knihovně pro polohování robotu HelloWorld3D.dll.