Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

BAKALÁŘSKÁ PRÁCE

PLZEŇ, 2012 **MAREK FEHÉR**

Prohlášení

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

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

V Plzni dne 19.8.2012

.....
(vlastnoruční podpis)

Poděkování

Rád bych tímto způsobem poděkoval vedoucímu své bakalářské práce Ing. Romanovi Pišlovi za vzorné vedení, maximální ochotu a patřičnou dávku trpělivosti.

Dále bych chtěl poděkovat Ing. Ondřejovi Severovi, jenž má též na vzniku této práce nemalý podíl, za cenné rady a připomínky.

Slova díků z mé strany směřují také k všem těm, kteří navrhli, vytvořili nebo se všemožně jinak podíleli na vzniku technologií, použitých pro realizaci bakalářské práce, jejíž výslednou podobu si máte nyní možnost prohlédnout.

Abstrakt

Hlavním cílem této bakalářské práce je navržení a implementace aplikace založené na interaktivním rozhraní, které je realizováno mimo jiné pomocí nástrojů umožňujících sledování lidského těla prostřednictvím zařízení Kinect. Aplikace slouží jako rozšíření 3D editoru pro plánování trajektorie lanových a podobných robotů vyvíjeného členy Katedry kybernetiky Fakulty aplikovaných věd Západočeské univerzity. Práce by měla také být vnímána jako vstupní bod do zmíněných problematik.

Klíčová slova: 3D editor pro plánování trajektorie, Hand Tracking, interaktivní rozhraní, JSON, Kinect, Kinect for Windows SDK, OpenNI, Skeletal Tracking, WebSocket klient.

Abstract

The main goal of this bachelor thesis is to design and implement an application based on interactive interface that is realized by using, among other things, tools that allow tracking human body by Kinect. The application serves as an extension of 3D trajectory planning editor of cable and similar robots that is developing by members of the Department of Cybernetics of the Faculty of Applied Sciences at the University of West Bohemia. The thesis should be also seen as an entry point to mentioned problems.

Key words: 3D trajectory planning editor, Hand Tracking, interactive interface, JSON, Kinect, Kinect for Windows SDK, OpenNI, Skeletal Tracking, WebSocket client.

Obsah

1 Úvod

Text, který se Vám právě dostal do ruky, si klade za cíl seznámit svého čtenáře s problematikou tvorby interaktivního rozhraní využívajícího schopností herního zařízení *Kinect*.

Kromě základních faktů o tomto zařízení by měl čtenář přečtením textu získat i obecné povědomí o funkcích *Kinect*u a možnostech vývoje aplikací pracujících s *Kinect*em na PC.

Dalším stěžejním bodem práce je vysvětlení komunikace s *3D editorem pro plánování trajektorie* lanových a podobných robotů, jejímž výstupem by pak mělo být nejen základní porozumění formě této komunikace, ale i nabytí znalostí a informací o technologiích, které komunikaci s *editorem* umožňují.

Aplikační částí se stala implementace aplikace, která bude založena na interaktivním rozhraní využívajícím *Kinect*u a která dokáže toto rozhraní přenést i do prostředí *3D editoru pro plánování trajektorie*. Výsledná aplikace tak může být vnímána jako možné rozšíření pro uživatele *editoru*.

Z již zmíněných pojednání o obsahu práce plyne i její struktura. Práce je rozdělena do tří velkých kapitol – Kinect, 3D editor pro plánování trajektorie a Aplikační část. U každé kapitoly je uveden krátký úvod shrnující ve stručnosti to, co bude následovat, tj. obsah dané kapitoly.

Vzhledem k tomu, že většina technologického názvosloví použitého v textu pochází z anglického jazyka, nebude se práce pouštět do nějakého zavádění českých pojmů, ale pouze zde bude uveden český překlad v závorce a pojem se bude dále využívat jen v jeho původním znění. Bude-li však nutno pojem nějak skloňovat – bude použit následující zápis: *Hand Tracking – Hand Tracking*u, *Hand Tracking*em atd.

Tento postup by měl čtenáři usnadnit následné vyhledávání těchto pojmů v literatuře a na internetu, pokud si bude chtít dále rozšířit své znalosti v odpovídající problematice.

Závěrem úvodu nezbývá než doufat, že práce splní svůj cíl, tj. informovat o tom, jak byla navržena a realizována aplikace založená na vytvořeném interaktivním rozhraní využívajícím *Kinect* a rozšířit čtenářovy obzory v problematikách, kterých se práce svým obsahem dotýká.

2 Kinect

Následující kapitola se věnuje stručnému představení tohoto průlomového zařízení. Jejím úkolem je seznámit čtenáře s *Kinect*em, podívat se krátce na jeho historii i současnost, popsat důležité komponenty zařízení a stručně charakterizovat principy dodávající *Kinect*u onu jedinečnost. Jsou zde zavedeny stěžejní pojmy jako např. *hloubková kamera*, *hloubková mapa* atd. tolik nezbytné pro porozumění zařízení a s tím související aplikaci nabytých poznatků v praxi. Dalším podstatnou částí kapitoly je pak text věnující se možnostem vývoje softwaru využívajícího *Kinect* na PC zejména v prostředí .NET a programovacím jazyku C#.

Obrázek 1: Zařízení *Microsoft Kinect*.

2.1 Představení zařízení

Kinect (viz Obrázek 1) je ovládací zařízení k herní konzoli *XBOX 360*, vyvinuté pod záštitou společnosti *Microsoft*, která jeho uvedením na trh v listopadu roku 2010 vstoupila do konkurenční bitvy s ostatními výrobci podobných „revolučních“ ovladačů jako bylo a je například *Nintendo* s ovladačem *Wii Remote* ke své herní konzoli *Wii*.

Na rozdíl od *Wii Remote* však s sebou *Kinect* přináší možnost hrát hry bez nutnosti držení nějakého dalšího ovládacího zařízení, a tudíž lepší a živější pocit pro hráče, neboť ten je při ovládání hry odkázaný pouze na pohyby svého těla, popř. svůj hlas. Z této skutečnosti vychází i název *Kinect* skládající se ze dvou anglických slov *kinetic* (pohybující se) a *connect* (spojit), která naznačují, že hráč bude právě při hraní her určených pro *Kinect* v neustálém pohybu a ve stálém spojení se zábavou a svými přáteli, jelikož mezi další velké přednosti zařízení lze zařadit i schopnost rozpoznat více před ním stojících osob [19].

Bylo jen otázkou času, kdy o *Kinect* začne být zájem i v oblasti výzkumu, protože svými možnostmi toto zařízení překračuje herní průmysl, a tak se takřka vzápětí po jeho oficiálním uvedení do prodeje objevují první pokusy o tzv. *hacknutí Kinect*u[1], tj. získat z *Kinect*u data a pracovat s nimi dále na PC či dalších zařízení. Na *Microsoft* byl tak vyvíjen nátlak na vytvoření ovladačů a vývojových nástrojů, aby mohl být *Kinect* plně využit i na jiných zařízení než pouze na *XBOX*u. Ten na to odpověděl doslova šalamounsky zpřístupněním vývojářského balíku *Kinect for Windows SDK* [2], který, jak už název napovídá, má jedno velké omezení a to konkrétně takové, že jeho plné využití je možné pouze v rámci operačního systému *Windows 7*, popř. *Windows 8* (v době psaní práce ve verzi *Customer Preview*) a vývojového prostředí *Visual Studio 10*. Pro ostatní platformy však existuje řešení, jak získat a dále zpracovávat data z *Kinect*u – uchýlit se k jiným ovladačům a vývojovým nástrojům. Tato práce se z těchto „sekundárních“ možností zaměřuje především na framework *OpenNI* od stejnojmenné neziskové organizace [1], mezi jejíž zakládající členy patří i izraelská společnost *Prime Sense* [7], která mimo jiné dodává technologie *Microsoft*u právě pro *Kinect*.

Závěrem stručného představení *Kinect*u by bylo dobré zmínit pár zajímavých faktů, které dokumentují oblibu tohoto hardwaru. K lednu roku 2011 byl *Kinect* zapsán do *Guinessovy knihy rekordů* jako nejrychleji prodávané elektronické zařízení, když se jej prodalo v období od svého uvedení do začátku ledna 8 miliónů kusů, čímž překonal i taková, mezi lidmi toho času populární, zařízení jako chytrý telefon *iPhone* či tablet *iPad* společnosti *Apple*. Lákavou se stal především nízkou cenou – podobná zařízení v době před uvedením *Kinect*u byla většinou drahá a určená hlavně pro vědecký výzkum. Kromě tohoto rekordu získal *Kinect* i řadu dalších ocenění jako například cenu *Gadget of the Year* (*Zařízení roku*) pro rok 2011 vyhlašovanou každoročně populárním magazínem *T3* zaměřujícím se na nové technologie ve světě [21].

2.2 Popis a princip zařízení

Hned při prvním pohledu na *Kinect* si jistě každý všimne tří kruhových otvorů, pod jejichž průhledným plastovým krytem se skrývá dvojice senzorů a jeden vysílač.

Prvním ze senzorů je RGB kamera umístěná uprostřed. Ta dokáže při snímací frekvenci 30Hz pořizovat snímky s rozlišením 640x512 pixelů, které se následně redukují na 640x480 pixelů tak, aby odpovídaly rozlišení *hloubkové kamery*, o níž bude zmínka níže. *Kinect* má však ještě možnost přepnout RGB kameru do režimu vyššího rozlišení, což s sebou přináší bohužel i snížení hodnoty počtu pořízených snímků za sekundu – pří zvýšení rozlišení na

[1] *hacknutí Kinectu* – spojení používané fanouškovskou komunitou *Kinect*u k signalizaci toho, že se podařilo získat data ze zařízení. Jako tzv. *Kinect Hacks* jsou občas označovány i aplikace využívající *Kinect*, které jsou určeny pro PC a dalších zařízení. Dále viz `<http://www.kinecthacks.com/>`.

1280x1024 pixelů se uvádí snížení *fps*[2] na patnáct (v praxi se však toto číslo pohybuje zhruba okolo deseti). Dále má kamera i několik dodatečných funkcí jako např. automatické vyvážení bílé apod.

V blízkosti kamery je umístěna infračervená kamera snímající pouze infračervené paprsky o vlnové délce 830 nm. Tyto paprsky jsou nepřetržitě vysílány vysílačem známého infračerveného záření, jenž je umístěn pod kruhovým otvorem vpravo.

Infračervená kamera spolu s vysílačem infračerveného záření tvoří celek, který je často označován jako *hloubková kamera* (*depth camera*) [6].

Umístění senzorů je pro lepší představu znázorněno na Obrázku 2.

Obrázek 2: Umístění senzorů na *Kinect*u [6].

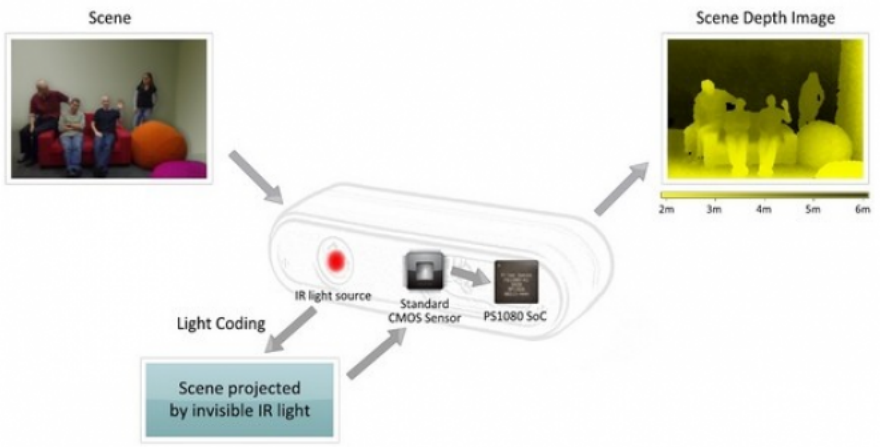
Celý systém hloubkového snímání pracuje, podobně jako některé druhy dalších 3D scannerů, na tzv. *principu strukturovaného světla* (*structured light principle*), tj. principu, kdy je vysílána do snímaného prostoru předem známá struktura světla jako např. pruhy nebo, jak tomu je u *Kinect*u, uspořádané body. Známá struktura se vlivem odrazu od překážek v prostoru deformuje a tato deformace je snímána právě infračervenou kamerou. Následně se provede analýza odlišností, z nichž lze určit, které body jsou blíže ke kameře a které dále od kamery (pozn.: způsob aplikace *principu strukturovaného světla* a jeho další vylepšení a rozšíření vedoucí k způsobu práce hloubkové kamery v *Kinect*u je patentem již zmíněné společnosti *Prime Sense*) [6][7][22].

Samotná infračervená kamera pak pracuje s frekvencí snímání 30Hz a rozlišením snímků 1200x960 pixelů, které jsou následně redukovány na 640x480 pixelů tak, aby bylo možné posílat skrze USB takové množství dat, jakým data z RGB a *hloubkové kamery* jsou.

[2] *fps* - frames per second – číslo užívané nejen ve video technice udávající počet snímaných snímků za sekundu

Produktem snímání pomocí *hloubkové kamery* je tzv. *hloubková mapa* (*depth map*). Každý snímek *hloubkové mapy* si s sebou nese informaci o hloubce jednotlivých pixelů. Obvykle se vykresluje jako barevná mozaika jedné či více barev, kde světlejší odstíny nebo barvy reprezentují objekty umístěné v prostoru blíže ke *Kinect*u, tj. čím světlejší barva, tím je snímaný objekt v prostoru blíž. Tato logika ale není pravidlem. Záleží čistě na fantazii programátora, ke kterému se dostávají informace z *hloubkové kamery* zabalené v určité struktuře závislé na volbě vývojových nástrojů, jak se rozhodne konkrétně on *hloubkovou mapu* zobrazit.

Obrázek 3 názorně dokumentuje, jak probíhá snímání scény před *Kinect*em, způsob převodu na *hloubkovou mapu* a její možnou reprezentaci v odstínech žluté barvy. Komponenta *PS1080 SoC* je čip od *Prime Sense* zajišťující logiku zpracování dat v *Kinect*u a dnes již v dalších podobných zařízení. Bližší informace o *PS1080 SoC* viz [7].



Obrázek 3: Způsob převodu na hloubkovou mapu [7].

Pootočením *Kinect*u si je možné všimnout dvou řad malých otvorů v podobě zkosených čar, pod nimiž je umístěno pole čtyř mikrofonů značící, že *Kinect* není ovladačem založeným pouze na obraze, ale jde o zařízení, umožňující zpracovávat i zvuk. Díky většímu počtu mikrofonů dokáže být potlačen nežádoucí šum, což umožňuje *Kinect*u lépe se „koncentrovat“ na mluvící osobu.

Celé zařízení je umístěno na masivním podstavci. V něm se skrývá motor umožňující vertikálně natočit hlavovou část *Kinect*u v rozmezí $\pm 27°$. Horizontální natočení je možné pouze ručně.

Výše v textu byla zmíněná sběrnice *USB*, a tak je nutné věnovat zde pozornost i konektivitě *Kinect*u. *Kinect* je primárně určen pro přímé připojení k *XBOX*u *360 S*, tj. k aktuálně nejnovější verzi konzole *XBOX*, vybavené tzv. *AUX port*em. *AUX port* je případ *proprietárního USB portu*[3]. Umožňuje napájet zařízení napětím 12V na rozdíl od 5V klasického *USB portu*. Aby bylo možné připojit *Kinect* k zařízení vybaveným pouze klasickým *USB port*em, je nutné použít speciální napájecí kabel sloužící také jako *AUX/USB převodník* [20].

Často kladené dotazy ohledně *Kinect*u se týkají dosahu zařízení, a proto by se závěrem popisu mělo pozastavit i u tohoto údaje. Pozorovací úhly zařízení jsou výrobcem udávány jako 58° horizontálně a 45° vertikálně. Dosah hloubkového snímání se pak pohybuje od 0,8m do 3,5m [7].

2.3 Interaktivní rozhraní využívající Kinect

Jak již bylo uvedeno při představení zařízení - možnosti využít schopnosti *Kinect*u nejen ke hraní či vývoji her na konzoli, ale i při návrhu zajímavých interaktivních rozhraní, která by využívala *Kinect* pro počítače a notebooky, zde jsou. To, jak by vlastně mělo takové rozhraní vypadat, upřesní následující řádky. Z tohoto náhledu navíc vyplynou i některé funkce, které by měla vhodná knihovna použitá pro aplikační část práce obsahovat.

Uživatelské rozhraní lze stručně definovat jako způsob komunikace mezi člověkem a strojem. V případě komunikace uživatele prostřednictvím *Kinect*u s počítačem se stane formou komunikace převážně pohyb určitých částí uživatelova těla. Často bývá takové „pohybové“ uživatelské rozhraní v anglosaských zdrojích označováno termínem *kinetic user interface* (*KUI*). *Microsoft* sám však dává rozhraním využívajícím *Kinect* nálepku *natural user interface* (*NUI*), tj. rozhraní, která jsou intuitivní (přirozená) tak, že uživatel aplikace založené na *NUI* ji takřka ihned či za krátký čas dokáže pochopit a plně využívat [2][23]. Tuto nálepku přelepil *Microsoft* i na označení jádra *Kinect for Windows API*, skrze nějž lze přistupovat k jednotlivým komponentám *Kinect*u [8].

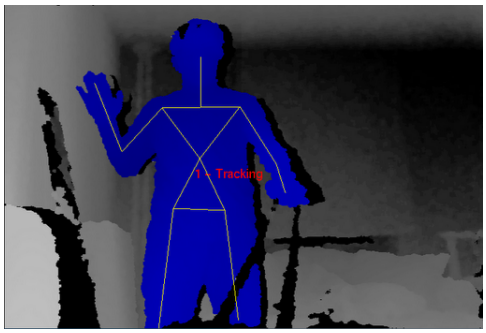
Jednou z nejdůležitějších informací při vytváření interaktivního rozhraní využívajícího *Kinect* se tak stává údaj o poloze částí těla osoby. *Kinect* sám o sobě žádnou podobnou informaci neposkytuje a vše je tak nutné řešit na úrovni softwaru. Bylo by ale nepohodlné a neefektivní, kdyby byl programátor například při vývoji aplikace, kde se kurzor hýbe podle pravé ruky uživatele, nucen vytvořit ze znalosti dat hloubkové mapy svůj vlastní detekční algoritmus pravé ruky. Naštěstí je tato funkce v některých knihovnách zaimplementována. Obvykle se mezi komunitou označuje jako *Skeleton* nebo *Skeletal Tracking* (*detekce kostry*).

[3] *Proprietární USB port* - port vycházející z klasického *USB portu*. *USB port* rozšiřuje o další vlastnosti dle potřeb zařízení využívající komunikaci skrze USB, s čímž souvisí například přidání pinů či změna tvaru.

V různých knihovnách se podoba oné kostry může lišit, ale stručně řečeno jde o balík cca. 20 bodů označovaných jako *joints*. Každý *joint* je reprezentován třemi souřadnicemi, které určují jeho polohu v prostoru před *Kinect*em.

Se *Skeleton Tracking*em jde ruku v ruce i druhá zajímavá funkce a tou je tzv. *segmentace hráče* (*Player Segmentation*). V *hloubkové mapě* udává o každém pixelu informaci, zda je v něm obsažen hráč, či nikoliv. Při vykreslování této mapy si je pak indexované pixely možno např. obarvit stejnou barvou, čímž vzniká jakási silueta. Dále se v případě některých knihoven pracujícími s *Kinect*em můžeme setkat s detekčními algoritmy ruky, gest apod.

Pro příklad využití *Skeleton Tracking*em získaných bodů pospojovaných úsečkami v kombinaci s pixely označenými pomocí *segmentace hráče* vykreslenými v *hloubkové mapě* viz Obrázek 4.



Obrázek 4: *Skeleton Tracking* a *segmentace hráče* v praxi.

Některé vývojářské balíky navíc přidávají i možnosti práce s výše zmíněným mikrofonním polem či dalšími komponentami *Kinect*u jako je motor pro vertikální pohyb hlavy zařízení a signalizační LED dioda na předním panelu.

Dále je ještě nutné zaměřit pozornost na požadavky, které by mělo takové uživatelské rozhraní splňovat. Tím nejdůležitějším požadavkem je intuitivnost, byť je zřejmé, že přísné podmínky *NUI* se ve většině případů dodržet nepodaří. Uživatel pomyslného rozhraní by však měl být schopen svými pohyby a gesty aplikaci plně a pohodlně ovládat, z čehož plyne další bod a tím je efektivita. Je totiž dobré myslet i na ergonomii budoucího uživatele a s tím související pohodlí. Nebylo by příliš výhodné, kdyby uživatel musel ze své pozice kdesi v prostoru před *Kinect*em neustále chodit něco přepínat pomocí klávesnice nebo kdyby byl nucen aplikaci ovládat až příliš gymnastickými pohyby. Též je dobré nezapomínat ani na

lidskou hravost a naplnit tak v rámci svého rozhraní nějakým způsobem i ono základní heslo celé filozofie *Kinect*u – bavit se.

2.4 Zkoumané vývojářské možnosti

Je-li již zřejmé, jak by mělo ono interaktivní rozhraní využívající *Kinect* vypadat, a hlavně, které funkce při jeho vývoji by bylo dobré použít, nastává problém výběru vhodného vývojářského balíku, jenž by obsahoval knihovnu (knihovny), která by se při vytváření takového interaktivního rozhraní hodila nejvíce.

Termínem *knihovna* se v informatice označuje soubor funkcí a procedur, který může být sdílen více počítačovými programy skrz aplikační rozhraní (pro přesnější definici viz [24]). V našem případě budeme požadovat, aby nám byla prostřednictvím možností takovýchto knihoven umožněna v první řadě komunikace s ovladači zařízení *Kinect* a dále, aby bylo možné využít již zmíněných funkcí pro sledování části či celého lidského těla.

Tato práce, jak bylo nastíněno dříve, se zaměřuje na framework *OpenNI* a vývojářský balík *Kinect for Windows SDK*. Pro její potřeby je proto nejprve nezbytné obě možnosti pečlivě porovnat. Na základě porovnání vybrat vhodného kandidáta pro aplikační část práce. Následně se podívat na zkoumané balíky poněkud obecněji, a poté se věnovat především tomu, jak je možné využít v nich vnořených funkcí pro sledování lidského těla v praxi.

2.4.1 Základní porovnání OpenNI a Kinect for Windows SDK

Tabulka 1 dokumentuje a porovnává možnosti využití frameworků na různých platformách, podporované programovací jazyky a pro účely práce velice důležité algoritmy umožňujících sledování celého či částí lidského těla. Dále je v ní uveden i seznam rozšiřujících funkcí knihoven nabízených v rámci prvního či druhého vývojářského balíku.

	OpenNI	Kinect for Windows SDK
Operační systém	Windows XP/Vista/7(x86&x64), Linux, MacOS	Windows 7/8(x86&x64)
Programovací jazyky	C++, C#, java, python	C++, C#, Visual Basic
Čtení dat z RGB kamery	ano	ano
Čtení dat z hloubkové kamery	ano	ano
Skeletal Tracking	ano	ano
Hand Tracking	ano	ne
Face Tracking[4]	ne	ano
Player Segmentation	ano	ano
Další funkce	+ rozeznávání gest + přístup k mikrofonům	+ podpora rozpoznávání řeči + přístup k motoru + přístup k signalizační LED diodě + dobře zpracovaná dokumentace

Tabulka 1: Základní porovnání vývojářských balíků *OpenNI* a *Kinect for Windows SDK* [1][2].

Knihovny obsažené v SDK od *Microsoft*u přináší větší počet funkcí než je tomu v případě *OpenNI*. V její další prospěch hovoří i fakt, že má daleko propracovanější dokumentaci a začínající programátor v oblasti *Kinect* se tak bude mít od samého začátku o co opřít.

Součástí balíku je i dostatečné množství příkladů.

Framework *OpenNI*, který je na rozdíl od *Kinect for Windows SDK* možné využít na různých operačních systémech, disponuje bohužel horší kvalitou dokumentace. Tu je však možné suplovat dostatečnou podporou komunity v rámci webových fór.

Jelikož vývoj aplikace probíhal stejně ve vývojovém prostředí *Visual Studio 10*, nebyl striktní požadavek *Microsoft*u na volbu vývojového prostředí pro aplikační část práce nějak svazujícím (pro připomenutí – podpora pouze pro *Visual Studio 10*), byť je zřejmé, že i toto omezení může při soudu pro či proti *Kinect for Windows SDK* sehrát svou roli.

Programovací jazyk, který byl k implementaci zvolen, tj. *C#*, je též v obou případech podporován a u obou zkoumaných možností je součástí základního balíku několik stručných příkladů v jeho syntaxi.

Využití *OpenNI* si žádá 1GHz procesor s 512Mb RAM pamětí. Je tak méně náročné než použití *Kinect for Windows SDK*. SDK od *Microsoft*u má minimální hardwarové požadavky stanovené na dvoujádrový 2,66GHz procesor s alespoň 2Gb RAM pamětí [1][2].

Z důvodu nižší hardwarové náročnosti a přenositelnosti na různé platformy byl pro samotnou aplikační část práce zvolen framework *OpenNI*. Výhodné bylo navíc i to, že *OpenNI* disponuje funkcí *Hand Tracking* pro přesnější detekci ruky, které bylo využito v rámci tvorby interaktivního rozhraní výsledné aplikace.

2.4.2 Kinect for Windows SDK

Nezbývá než se podívat na obě porovnávané možnosti a na jimi poskytované funkce umožňující sledování lidského těla hlouběji.

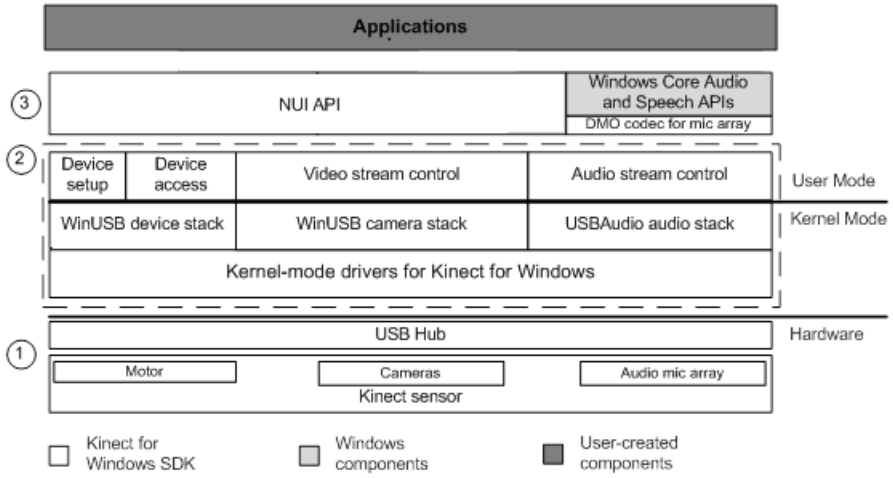
Nejprve budou uvedeny informace o vývojářském balíku *Kinect for Windows SDK*. Vzhledem k dostatečné kvalitě dokumentace a podpory ze strany *Microsoft*u nebudou však tak podrobné, jako tomu bude v případě *OpenNI*.

Dostupnost *Kinect for Windows SDK* je dobře přístupná. Lze pozorovat, že se nejspíše jedná o úmysl *Microsoft*u propagovat svůj produkt na vývojářské úrovni a tlačit tak zkušené i méně zkušené programátory k vývoji nových aplikací, což s sebou může přinést více potenciálních zákazníků.

Zprovoznění probíhá pomocí jednoduché instalace. Instalátor *Kinect for Windows SDK* je možné získat na webu tohoto projektu viz [2] a jeho spuštěním se nainstaluje vše potřebné včetně ovladačů.

[4] *Face Tracking* (viz Tabulka 1) - funkce vložená do Kinect for Windows SDK od verze 1.5. Umožňuje detekovat pohyby obličeje v reálném čase.

Struktura Obrázek 5 stručně zachycuje jednotlivé vrstvy struktury celého systému pracujícím s *Kinect for Windows SDK*.



Obrázek 5: Struktura systému s *Kinect for Windows SDK* [8].

- 1.vrstva – je vrstvou fyzickou. Jedná se o kompletní hardware, tj. od senzorů *Kinect*u až po USB hub, skrze který je *Kinect* připojen k počítači.
- 2.vrstva – jedná se o ovladače zařízení, které jsou součástí instalace celého balíku. Ty umožňují operačnímu systému pracovat se samotným hardwarem.
- 3.vrstva - je již samotné *API*, které může být dále využíváno programátorem vytvořenými aplikacemi. *Microsoft* jej označuje jako *NUI*, o čemž již byla zmínka dříve.

Sledování lidského těla Nejpodstatnější problematikou této práce nadále zůstává, jakým způsobem je vlastně možné sledovat lidské tělo. *Kinect for Windows SDK* k tomu poskytuje zejména funkci *Skeletal Tracking*.

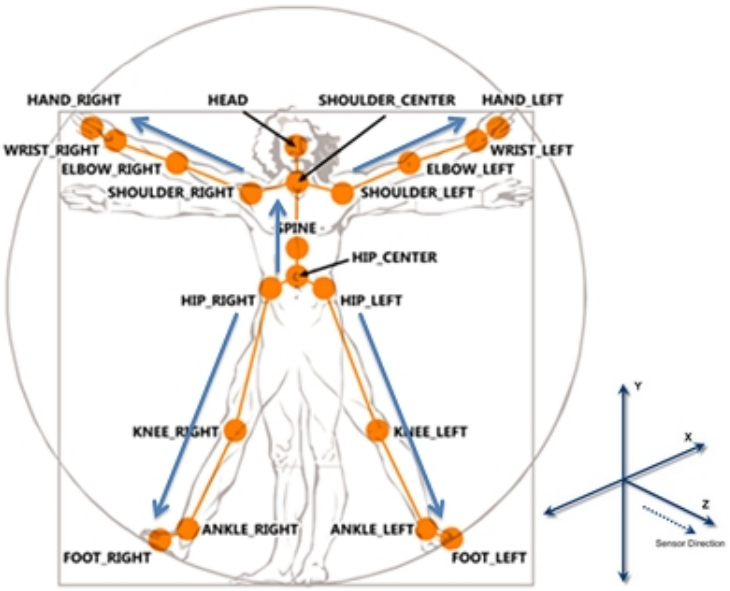
Algoritmus *Skeletal Tracking*u probíhá v *Kinect for Windows SDK* jako zaznamenání 20 bodů (dále *joint*ů) na těle osoby stojící před *Kinect*em. Tyto *joint*y mají své jednoznačné umístění a označení. Například *joint Head* označuje bod, který odpovídá středu hlavy před *Kinect*em stojící osoby, apod. (všechny *joint*y viz Obrázek 6). Verze 1.5 (nejaktuálnější v

době psaní této práce) umožňuje navíc u sedící nebo kvůli nějaké překážce z dolní poloviny neviditelné osoby zaznamenávat pouze 10 *joint*ů odpovídajících viditelné horní polovině těla. Jedná se o tzv. *Skeletal Tracking* v módu *seated* (sedící) [8].

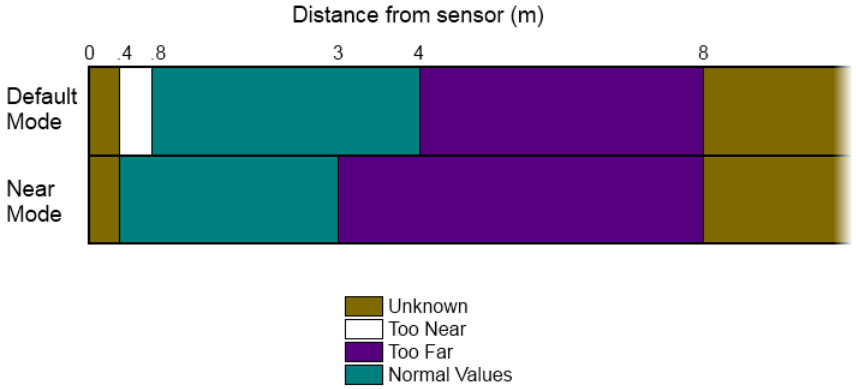
Každý joint, v *SDK* reprezentovaný stejnojmennou strukturou, je mimo dalších členů určen trojicí číslic určujících jeho polohu v prostoru před *Kinect*em. Ty jsou ukryty ve třídě *Position*. Třída *Position* symbolizuje souřadnice [x,y,z] v pravotočivém systému souřadném. V základním tvaru udávají tyto souřadnice vzdálenost příslušného *joint*u od osy *Kinect*u v metrech. *Kinect for Windows SDK* umožňuje sledovat najednou až šest osob a na dvě z nich aplikovat *Skeletal Tracking*.

Verze 1.5 s sebou přináší i možnost nastavení filtrace informace o poloze *joint*u jako například úrovně potlačení chvění, vyhlazení trajektorie apod. *Skeleton*y je pak možné v této verzi zaznamenávat ve dvou režimech snímání – v režimu *Near Depth Range* (pro snímání v blízkosti *Kinect*u) a nebo klasickém režimu *Default Depth Range*. O kolik vylepšuje *Near Depth Range* režim schopnost Kinectu sledovat osoby stojící blíže k němu viz Obrázek 7.

Závěrečné poznámky Část textu práce věnující se *Kinect for Windows SDK* slouží pouze jako stručný úvod do problematiky. Jelikož tuto problematiku samotná práce již dále nerozšiřuje, nebyly zde uvedeny bližší informace. Motivem toho, proč zde byl balík *Microsoft*u zmíněn, se stalo to, aby čtenář získal i možnost srovnání *OpenNI* s jinými v mnohém odlišnými vývojářskými balíky. Vše potřebné pro hlubší studium samotného *Kinect for Windows SDK* lze nalézt v kvalitní dokumentaci viz [8].



Obrázek 6: *Skeletal Tracking* v *Kinect for Windows SDK* a orientace souřadného systému [8].



Obrázek 7: Rozdíl mezi režimem *Near Depth Range* a *Default Depth Range* [8].

2.4.3 OpenNI

OpenNI (*Open Natural Interaction*) je volně šiřitelný (podléhající licenci GNU LGPL) framework vytvořený stejnojmennou neziskovou organizací, která byla založena několika společnostmi, mezi nimiž figurují společnosti jako *Prime Sense*, *Asus*, *Side-Kick* atd. Obsahuje soubor APIs pro psaní aplikací založených na *Natural Interaction* (*přirozená interakce* – pojem, kterým organizace *OpenNI* označuje právě způsoby komunikace s počítačem skrze pohyby či řeč; ekvivalent k *NUI* používané *Microsoft*em).

Dalším úkolem *OpenNI* je vytvoření standardizovaných APIs, která umožní jednak komunikaci s obrazovými a zvukovými senzory skrz ovladače a také komunikaci s tzv. *middleware komponentami*. *Middleware komponenty* jsou dle *OpenNI* softwarové komponenty, jež umožňují analyzování a „pochopení“ zaznamenávaných obrazových nebo zvukových dat. V případě analyzování dat z *Kinect*u je touto *middleware komponentou NITE* vytvořená společností *Prime Sense*. Ta umožňuje přístup k senzorům *Kinect*u a obsahuje algoritmy pro sledování lidského těla. Proto se lze setkat i s označením *OpenNI/NITE*.

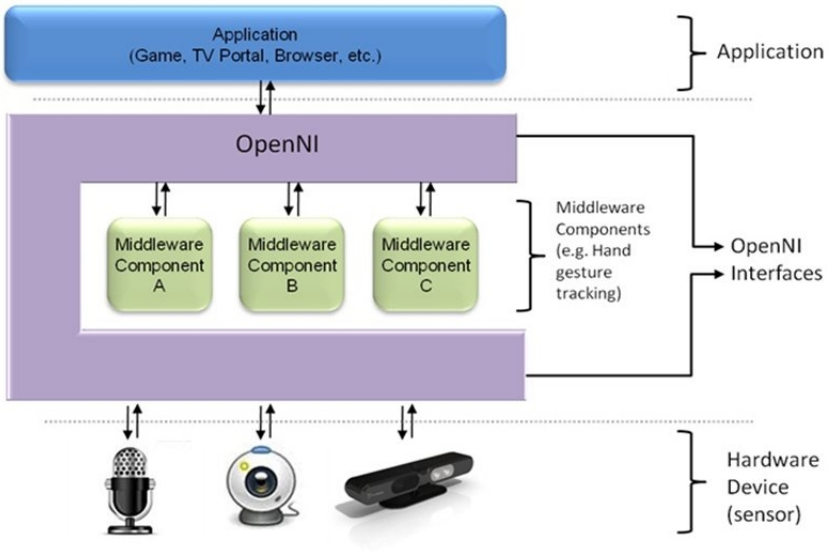
Stručně řečeno – *OpenNI* by mělo touto svou filozofií zajistit jakousi nezávislost mezi senzory a *middleware komponentami*, čímž by mělo být zároveň dosaženo možnosti distribuovat jednu aplikaci vyvinutou v *OpenNI* pro různé senzory a další zařízení. Například byla-li vyvinuta aplikace pro *Kinect* v *OpenNI*, neměl by ji být problém přenést na podobné zařízení, pokud toto zařízení bude přímo podporovat nebo mít svou *middleware komponentu* mající stejné funkce jako ty, kterými disponuje *NITE* [1].

Dostupnost Jak bylo nastíněno dříve, zprovoznění *Kinect*u s ovladači od *Prime Sense* a s frameworkem *OpenNI* je o něco málo komplikovanější než pouhé stažení a spuštění instalátoru, byť i ten se pro případ spolupráce *Kinect* a *OpenNI* již „neoficiálně“ na webu objevil viz [18].

Klasickým postupem však zůstává stáhnout jednotlivé části odděleně. Odkazy na ně lze nalézt na záložce downloads v sekci *Open Sources* na webu *OpenNI* viz [1] – jedná se o *PrimeSense Sensor Module for OpenNI* a samotný framework *OpenNI*. Dále si je nutné ještě ze sekce *OpenNI Modules* obstarat příslušnou *middleware komponentu*, tj. *NITE*, a ovladač od *Prime Sense*.

Struktura Obrázek 8 znázorňuje jednotlivé vrstvy struktury celého systému pracujícím s *OpenNI*. Je možné vypozorovat, že se v tomto případě již nedočkáme obdobného striktního rozvrstvení jako u *Kinect for Windows SDK*, což plyne z oné filozofie *OpenNI*, o které už byla řeč v úvodu do tohoto frameworku.

Jednotlivé vrstvy lze i přes tuto skutečnost rozdělit například na spodní, střední a horní vrstvu.



Obrázek 8: Struktura systému s *OpenNI* [1].

- Spodní vrstva – opět se jedná o fyzickou vrstvu.
- Střední vrstva – reprezentuje framework *OpenNI*, který svými rozhraními zajišťuje komunikaci jak se senzory, tak s *middleware komponentami*.
- Horní vrstva – jedná se o vrstvu aplikační. Software, který dále využívá možností *OpenNI*.

Production Nodes a Capabilities Dříve než bude pozornost zaměřena na funkce umožňující sledování lidského těla, je nutné vysvětlit pojem *Production Node*.

OpenNI definuje *Production Node* (dále jen *node*) jako sadu komponent, které mají produktivní roli při vytváření dat nezbytných pro tvorbu aplikací založených na *Natural Interaction*. Každý *node* tedy zajišťuje funkčnost týkající se generování jemu odpovídajících dat a je základním prvkem *OpenNI* API [1].

Vlastní logika generování dat je však odkázána na *middleware komponenty*, takže pokud je požadováno například využít možnosti sledování lidské ruky, logika tohoto procesu přichází právě z *middleware komponenty* vnořené do *OpenNI*.

Production nodes jsou rozděleny do dvou kategorií:

- *Sensor-Related Production Nodes*(*Production Nodes* související se senzory) - poskytují data přímo ze senzoru. Pro účely práce je nejdůležitějším *node* této kategorie *node Depth Generator*, jenž umožňuje generování *hloubkové mapy*.
- *Middleware-Related Production Nodes*(*Production Nodes* související s *middleware komponentami*) - poskytují data z *middleware komponenty*. Pro účely práce jsou zásadní:
  - *User Generator* - generuje reprezentaci těla nebo jeho částí.
  - *Hands Generator* - podporuje detekci pozice ruky.
  - *Gesture Generator* - podporuje detekci gest.

Byli-li zmíněny *Production nodes*, nelze zapomenout ani na tzv. *Capabilities*. *Capabilities* v řeči *OpenNI* API znamenají další nepovinná rozšíření. *OpenNI* tak reaguje na různé možnosti senzorů či *middleware komponent* reprezentovaných pomocí *Production Nodes*. *Production Node* je tak v případě volání nějaké specifické *Capability* otázán, zda ji umožňuje či nikoliv.

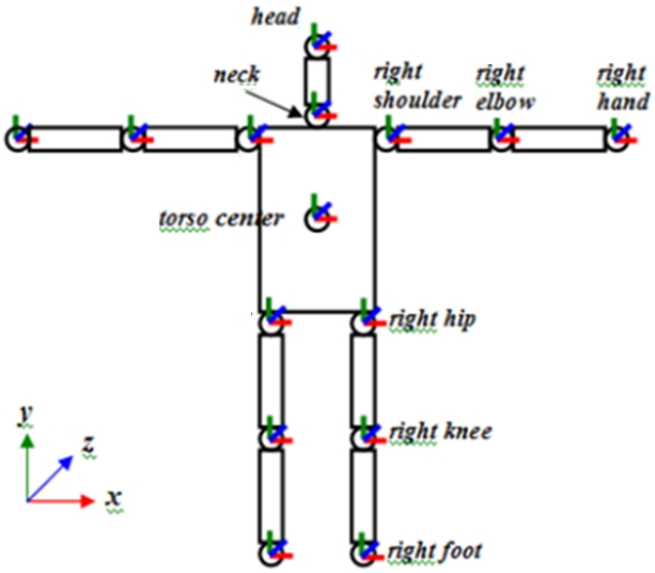
Pro potřeby této práce byly nejdůležitější následující *Capabilities*:

- *Pose Detection* - umožňuje, aby *node User Generator* dokázal rozpoznat, zda uživatel stojí v tzv. kalibrační póze viz *OpenNI/Skeletal Tracking*.
- *Skeleton* - umožňuje, aby node *User Generator* zaznamenal příslušná data ze *Skeletal Tracking*u viz *OpenNI/Skeletal Tracking*.

Poznámka: Koncept *Production nodes* a *Capabilities* je v OpenNI realizován stejnojmennými třídami.

Sledování lidského těla V rámci přípravy práce bylo využito tří funkcí pro sledování lidského těla nabízenými *OpenNI/NITE*. Prvním z nich je již mnohokrát zmiňovaný *Skeletal Tracking*. Zbývajícími dvěma funkcemi je pak *Hand Tracking* a *Gesture Detection*.

Skeletal Tracking Na rozdíl od *Kinect for Windows SDK* *OpenNI/NITE* zachytává „pouze" 15 *joint*ů, jejichž pozice udává v základním tvaru vzdálenost od osy *Kinect*u, přesněji řečeno – osy *hloubkové kamery*, v milimetrech v pravotočivém systému souřadném. Jednotlivé *joint*y včetně jejich pozice na těle a názvu viz Obrázek 9.

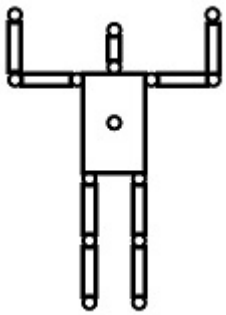


Obrázek 9: *Skeletal Tracking* v *OpenNI/NITE* a orientace souřadného systému [1].

Padla-li poznámka o funkci *Skeletal Tracking* je nutno ještě dodat, že starší verze *OpenNI/NITE* neposkytovaly možnost přímého rozpoznávání jednotlivých *joint*ů na těle osoby, aniž by se tato osoba postavila do tzv. *kalibrační pózy* (někdy též označované jako Ψ – póza dle jejího tvaru). Od verze *OpenNI* 1.4.0.2 je však možné tuto nutnost vynechat, což se hodí především při implementaci softwaru pro rehabilitační účely, kde se začíná možností *Kinect*u hojně využívat. Osoby se sníženou pohyblivostí by mohly mít s uvedením se do tvaru pózy problém. Kalibrační póza viz Obrázek 10.

Hand Tracking a Gesture Detection Důvod toho, proč funkce *Hand Tracking* a *Gesture Detection* nemá smysl od sebe oddělovat, vyplyne vzápětí.

Hand Tracking umožňuje sledování pohybu ruky, která se hýbe v prostoru před *Kinect*em a též ji vrací ve stejné struktuře jako *Skeletal Tracking*. Využití *Skeletal Tracking*u pro sledo-



Obrázek 10: *Kalibrační póza* v *OpenNI/NITE* [1].

vání ruky, respektive jím získaného *joint*u odpovídajícího ruce, nevede obvykle k tak přesným výsledkům jako v případě detekčních algoritmů pro ruku.

Jelikož *node Hands Generator* sám neposkytuje žádný mechanismus k počátečnímu určení pozice ruky, je nutné odněkud tuto informaci získat. Častým scénářem získání primární informace o pozici ruky je využití funkce *Gesture Detection*, v OpenNI zpřístupněné pomocí *node Gesture Generator*, která umožňuje rozpoznat ve scéně před *Kinect*em určité gesto. Pod pojmem *gesto* se v rámci *OpenNI* rozumí specifický pohyb ruky jako například mávnutí apod. Zároveň umožňuje určit konečnou pozici ruky, která gesto právě provedla, čímž je tedy možné získat i onu primární informaci.

Lze tedy tvrdit, že *Hand Tracking* a *Gesture Detection* jsou v *OpenNI* v obdobném vztahu jako *Skeletal Tracking* a *kalibrační póza*.

Příklad implementace Dále již bude přikročeno k samotnému modelovému příkladu práce s *OpenNI* na vývojářské úrovni v prostředí .NET a jazyku C#. Jelikož pro implementaci aplikační části práce stačilo využít sledování uživatelovy ruky, uvedeme si příklad použití *Hand Tracking*u v kombinaci s *Gesture Detection*.

Implementace *Skeletal Tracking*u se nese ve stejném duchu, ale vzhledem k větší rozsáhlosti problému a tudíž i kódu, by ji pro modelový příklad nebylo příliš efektivní využít.

Úvod Nejprve je nutné přidat do svého projektu referenci na knihovnu OpenNI.net.dll, která je součástí celého balíku *OpenNI*.

Následně je zapotřebí nastavit tzv. *Context*, což je vstupní bod do *OpenNI* reprezentující

všechny *Production nodes*, které mají být zpřístupněny. *Context* se nastavuje pomocí skriptu z XML souboru. V případě *Kinect*u postačí soubor SamplesConfig.xml, který je opět součástí balíku *OpenNI* a zpřístupňuje všechny *Production Nodes* umožňující práci s *Kinect*em v *OpenNI* (SamplesConfig.xml viz Příloha).

Pokud proběhlo nastavení *Context*u, je žádoucí pro potřeby *Skeletal* a *Hand Tracking*u v *OpenNI* zpřístupnit *node Depth Generator*, neboť obě funkce jsou přímo závislé na znalosti *hloubkové mapy*. Následující kód ukazuje způsob, jak nastavit *Context* a zpřístupnit *Depth Generator* – deklarace a inicializace jsou odděleny znakem //.

```
private string XML_FILE = "config_xml_file_adress";
private Context context;
private ScriptNode scriptNode;
//
this.context = Context.CreateFromXmlFile(XML_FILE, out scriptNode);
this.depth = context.FindExistingNode(NodeType.Depth) as DepthGenerator;
```

Gesture Detection Jak již bylo zmíněno při popisu této funkce v odstavci Hand Tracking a Gesture Detection, je nyní nutné zpřístupnit *Gesture Generator*, který umožní detekci gesta. *Gesture Generator* a další *Middleware-Related Production Nodes* jsou zaváděny do projektu jako instance stejnojmenných tříd, jejichž parametrem je *Context*.

```
private GestureGenerator gesture;
//
this.gesture = new GestureGenerator(this.context);
```

Pomocí metody AddGesture(**string** gesture) třídy GestureGenerator se zadává gesto, které má být rozpoznáno - například mávnutí. Toto gesto se označuje řetězcem "Wave". Další gesta viz [1].

```
this.gesture.AddGesture("Wave");
```

Algoritmus detekce gesta se spouští pro *Gesture Generator*, stejně jako pro další *Middleware-Related Production Nodes*, metodou StartGenerating().

```
this.gesture.StartGenerating();
```

Je-li určeno, které gesto má být rozpoznáno, je nutné jej nějak odchytit. To lze zajistit pomocí události GestureRecognized, která bude obsloužena například metodou s názvem gestureRecognized, o jejíž funkci bude zmínka později. Tato událost obsahuje ve svých dodatečných informací mimo jiné pro *Hand Tracking* tolik důležitou informaci o poloze ruky, která dané gesto právě provedla.

```
this.gesture.GestureRecognized += gestureRecognized;
```

Hand Tracking Jelikož i *Hands Generator* umožňující sledování uživatelovy ruky je *Middleware-Related Production Node*, tak se jeho zavedení do projektu řídí obdobnými pravidly jako u *Gesture Generator*u.

```
private HandsGenerator hand;
//
this.hand = new HandsGenerator(this.context);
this.hand.StartGenerating();
```

Dále je opět dobré za pomoci událostí odchytit následující tři případy (obslužné metody jsou pro jednoduchost pojmenovány obdobně jako v případě metody gestureRecognized):

- událost rozpoznání ruky (HandCreate) - Tato událost je vyvolána pouze v případě primárního rozpoznání uživatelovy ruky, tj. zavoláním metody StartTracking(Point3D point), o níž bude zmínka níže.

  ```
  this.hand.HandCreate += handCreate;
  ```

- událost změny polohy ruky (HandUpdate) - Tato událost je vyvolána při změně pozice uživatelovy ruky, a tudíž právě z její dodatečných informací bude čerpána informace o poloze uživatelovy ruky.

  ```
  this.hand.HandUpdate += handUpdate;
  ```

- událost ztráty ruky (HandDestroy) - Tato událost je vyvolán při ztrátě uživatelovy ruky.

  ```
  this.hand.HandDestroy += handDestroy;
  ```

Funkce metod handCreate a HandDestroy bude především informativní, tj. jejich výstupem by měla být informace pro uživatele, zda již byla jeho ruka rozpoznána nebo zda se jeho ruka *Kinect*u ztratila. Tudíž jejich tělo není nutné nějak blíže zkoumat.

Podstatnější je získat z dat události GestureRecognized primární informaci o pozici ruky a z ní spustit *Hand Tracking* pomocí metody StartTracking(Point3D point). Argumentem této metody je pozice reprezentována strukturou Point3D, jež v *OpenNI* udává souřadnice.

```
private void gestureRecognized(object sender, GestureRecognizedEventArgs e)
{
  this.hand.StartTracking(e.EndPosition);
}
```

Hlavním úkolem obslužné metody handUpdate je získat po každém vyvolání události HandUpdate aktuální polohu uživatelovy ruky, kterou lze například zpracovat tak, že se vždy bude uchovávat v proměnné handPosition typu Point3D.

```
private void handUpdate(object sender, HandUpdateEventArgs e)
{
  this.handPosition = e.Position;
}
```

Závěr Na závěr implementace je ještě nutné nastavit určitým způsobem obnovování všech *nodes*, aby docházelo k získávání stále nových dat z *Kinect*u. To se provádí obvykle zařazením smyčky, která bude probíhat dokud nemá být čtení dat z *nodes* ukončeno (v příkladu níže je toto ukončení závislé na booleovské proměnné shouldRun).

```
while (this.shouldRun)
{
  this.context.WaitAndUpdateAll();
}
```

Závěrečné poznámky Část práce věnující se *OpenNI* opět slouží jako základní úvod do problematiky a zdaleka nepokrývá všechny možnosti tohoto frameworku. Příklad implementace je věnován *Hand Tracking*u použitému v samotné aplikační části a obsahuje pouze základní kostru kódu. Bližší seznámení se s principy *OpenNI* je možné pročtením dokumentace projektu, která je k dispozici na oficiální webu této neziskové organizace viz [1]. Dále je možné čerpat informace z přiložených souborů k této práci.

3 3D editor pro plánování trajektorie

Další část textu popisuje formu komunikace s *3D editorem pro plánování trajektorie*. Komunikace s *editorem* se týkají i další důležité problematiky práce a to možnosti implementace *WebSocket* klienta v .NET a serializace dat za použití protokolu *JSON*.

3.1 Představení editoru

3D editor pro plánování trajektorie (dále jen *editor*) je nástroj vyvinutý členy Katedry kybernetiky Fakulty aplikovaných věd Západočeské univerzity sloužící jako interface při řízení lanových robotů, kde jsou uživatelsky definované body trajektorie interpolovány *NURBS*(*non-uniform rational B-spline*) křivkou pátého řádu.

Editor je rozdělen na klientskou a serverovou část, které však mohou být společně umístěny na jednom počítači.

- Klientská část - přináší grafické rozhraní editoru (viz Obrázek 12) implementované v prostředí webového prohlížeče použitím moderních webových technologií. Pro každý požadavek, který má být ošetřený serverovou částí, tento klient vygeneruje požadavek v *JSON* formátu a předá je prostřednictvím asynchronní *WebSocket* komunikace serveru. Více o formátu *JSON* viz 3.3.1.

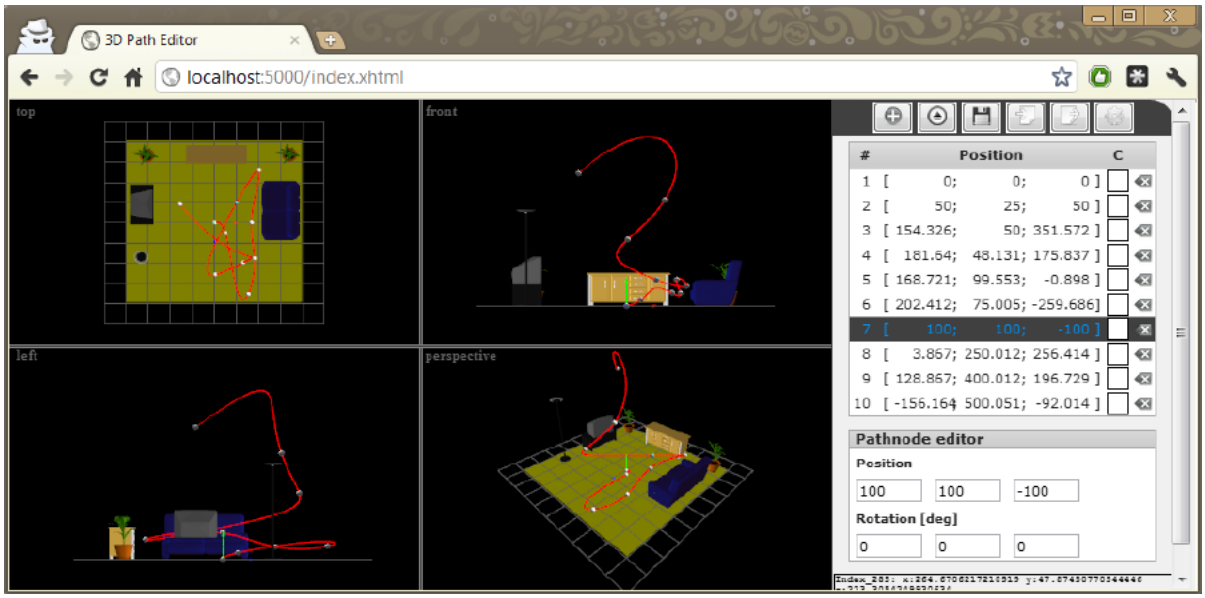
- Serverová část - řeší zmíněné požadavky klienta. Je odpovědná především za všechny výpočty a náročnější úkoly. Na server se pak může nahlížet jako na pomyslný spoj mezi prezentační a aplikační vrstvou.

V současné implementaci je serverová část *editoru* přístupná z bloku MC_MovePath řídicího systému *REX* [3][4] po stisknutí tlačítka *Special edit*. Webového klienta je na stejném počítači po spuštění serveru možno otevřít zadáním adresy http://localhost:5000/index.xhtml do webového prohlížeče podporujícího webové technologie, které tato část *editoru* využívá.

Ovládání webového klienta není po uživatelské stránce nikterak složité. Po zadání alespoň šesti bodů se objeví trajektorie, která tyto body interpoluje *NURBS* křivkou pátého řádu. Body je možné libovolně přidávat, přemisťovat a odstraňovat.

Aplikačním cílem této práce je vytvoření doplňku k *editoru*, který by umožnil ovládat jeho klientskou část interaktivním rozhraním využívajícím *Kinect*.

Pro bližší seznámení s *editorem* a prozkoumání jeho principů a možností viz [5][9].



Obrázek 11: Grafické rozhraní *editoru* [5].

3.2 WebSocket

Dalším mezníkem práce se tak stala problematika vytvoření *WebSocket* klienta v .NET tak, aby bylo možné komunikovat s *editorem* pomocí navržené aplikace.

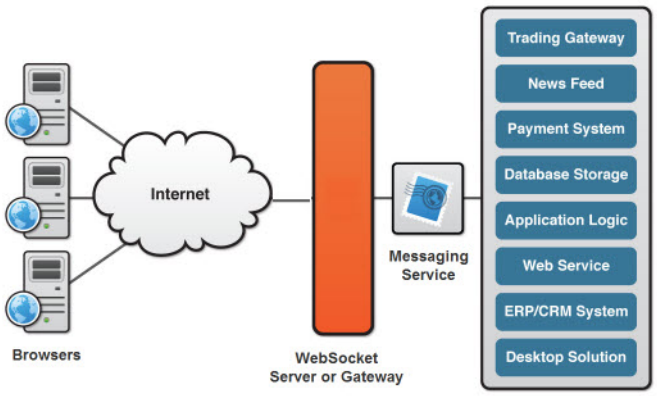
3.2.1 Princip technologie

WebSocket je technologie umožňující obousměrnou komunikaci mezi klientem a serverem.

Samotný *WebSocket* protokol se skládá se z otevíracího *handshake*[5], po němž následuje výměna dat založena na klasickém protokolu TCP. Cílem technologie *WebSocket* by mělo být poskytnutí komunikace, která nebude závislá na otevírání velkého počtu http připojení a nahradí tak starší webové technologie, jakými jsou streaming, pooling či long-pooling.

Jakmile je spojení skrz *handshake* navázáno, je klasické http spojení nahrazeno *WebSocket* spojením přes stejné TCP/IP spojení, po němž si klient se serverem mohou nezávisle na sobě vyměňovat data. *Handshake* se v případě klienta skládá z URI adresy (adresa URI v případě

[5] *handshake* - tímto pojmem se označuje proces, při kterém dochází mezi účastníky komunikace k dohodnutí o podmínkách společné komunikace.



Obrázek 12: Architektura technologie *WebSocket* [12].

WebSocket serveru má pefix ws:// nebo wss:// v případě zabezpečené verze), ke které se chce onen klient připojit, a z dalších položek, upřesňujících způsob vzájemné komunikace. V případě serveru především z klíče, jenž mu byl klientem poslán a jehož prostřednictvím si klient ověří, zda zpráva přichází ze správného serveru.

Data jsou sdružována do celků (tzv. *messages*), které se dále rozdělují do rámců (tzv. *frames*). Výhodou uvedeného postupu je především fakt, že server tak získává možnost posílat jednotlivé rámce dříve než má celou zprávu k dispozici, což s sebou přináší hlavně zrychlení komunikace a možnost obsluhy většího počtu klientů.

Jednou z dalších výhod technologie *WebSocket* je možnost jejího využívání přímo na úrovni webové stránky umožněné díky zavedení standardu HTML5, který je již podporován v mnoha současných populárních prohlížečích.

K podrobnějšímu nahlédnutí do problematiky technologie viz [12][13][25].

3.2.2 Příklad implementace WebSocket klienta v .NET

V souvislosti se zmíněným standardem HTML5 by bylo dobré se podívat na to, zda vůbec existuje nějaká komponenta, která by nám pomohla vytvořit jednoduchého *WebSocket* klienta v .NET obdobně jednoduše, jako je to možné díky HTML5 v prostředí webových stránek.

Tyto požadavky plně pokrývá knihovna *WebSocket4Net* zpřístupněná jako open source viz [11], která umožňuje využívat všechny základní prvky týkající se komunikace s *WebSocket*

serverem, tj. otevřít a ukončit spojení s *WebSocket* serverem, posílat zprávy na tento server a obsloužit všechny asynchronní události, které mohou ve zmíněné komunikaci nastat.

Následující příklad (uvedený i jako democode na stránkách projektu *WebSocket4Net* [11]) demonstruje použití knihovny *WebSocket4Net* v syntaxi jazyka C#.

Po přidání reference na WebSocket4Net.dll je prvním krokem vytvoření instance třídy WebSocket, jejíž konstruktor obsahuje vždy adresu *WebSocket* serveru, ke kterému se má klient připojit a může obsahovat i další řetězce, které odpovídají dalším komunikačním protokolům.

```
WebSocket websocket = new WebSocket("ws://localhost:2012/");
```

Dále je nutné obslužnými metodami (jejichž tělo zde nebude uvedeno; jméno ve tvaru websocket_JménoUdálosti) obsloužit následující události:

- Otevření komunikace - událost, která se vyvolá vždy při navázání komunikace s *WebSocket* serverem.

  ```
  websocket.Opened += new EventHandler(websocket_Opened);
  ```

- Chyba v komunikaci - událost, která se vyvolá, pokud nastane chyba v komunikaci s *WebSocket* serverem, indikována příjetím chybové zprávy.

  ```
  websocket.Error += new EventHandler(websocket_Error);
  ```

- Ukončení komunikace - událost, která se vyvolá, pokud dojde k ukončení komunikace s *WebSocket* serverem, indikována příjetím zprávy o ukončení spojení.

  ```
  websocket.Closed += new EventHandler(websocket_Closed);
  ```

- Obdržení zprávy - událost, která se vyvolá, pokud dojde k přijetí zprávy z *WebSocket* serveru.

  ```
  websocket.MessageReceived += new EventHandler(websocket_MessageReceived);
  ```

Jsou-li obslouženy všechny zmíněné události, zbývá už jen vložit do příkladu následující tři základní metody:

- websocket.Open(); - nutné zavolat vždy. Tato metoda otevírá komunikaci s *WebSocket* serverem.
- websocket.Close(); - Tato metoda ukončí komunikaci s *WebSocket* serverem.
- websocket.Send(message); - Tato metoda pošle na *WebSocket* server zprávu. Obvykle ve formě řetězce.

3.3 Komunikace s editorem

Vzhledem k tomu, že již bylo vysvětleno, co vše je potřebné k implementaci klienta, může být dále přikročeno k samotné formě komunikace s *editorem*.

Komunikace s editorem probíhá odesíláním a přijímáním dat, která jsou zabalena v rámcích ve formátu *JSON*.

3.3.1 JSON

JSON je zkratkové slovo pro *JavaScript Object Notation*. Jde o odlehčený formát pro výměnu dat, který je jednoduše čitelný i zapisovatelný člověkem a snadno analyzovatelný i generovatelný strojově, z čehož je zřejmé, že data jsou po tzv. serializaci reprezentována řetězci.

Obecně je *JSON* založen na dvou základních strukturách, jež symbolizují univerzální datové struktury, které se v podstatě v nějaké formě objevují ve všech moderních programovacích jazycích [14].

Jedná se o struktury:

- Kolekce párů název-hodnota – ta bývá v rozličných jazycích realizována například jako objekt, záznam, struktura apod.
- Tříděný seznam hodnot – obvykle realizován například jako pole, seznam, vektor apod.

Jak taková serializace objektu podle *JSON* vypadá demonstruje následující příklad:

Příklad serializace Jako příklad serializace bude uvedena serializace třídy reprezentující osobu, u které zkoumáme tři charakteristiky - křestní jméno, příjmení a věk:

```
public class Person
{
  public string FirstName { get; set; }
  public string LastName { get; set; }
  public int Age { get; set; }
}
```

Pro ilustraci nechť je vytvořena jedna konkrétní osoba - například osoba Pavel Novák, kterému je 45 let:

```
Person person = new Person();
Person.FirstName = "Pavel";
Person.LastName = "Novak";
Person.Age = 45;
```

Výstup serializace, tj. zápis person ve formátu *JSON* bude mít následující tvar:

```
{
  "firstName":"Pavel",
  "secondName":"Novak",
  "age":45
}
```

Poznámka K realizaci serializací dat ve formátu *JSON* bylo v aplikační části práce využito knihovny JSON.NET [15], která díky funkcím SerializeObject a DeserializeObject celý proces serializace a deserializace dat výrazně ulehčuje.

3.3.2 Popis komunikace

Základní formát zprávy obousměrné komunikace klient-*editor*, který respektuje formát *JSON*, má následující tvar:

```
{"command": "command", "parameters": {objekt s parametry}}
```

command je příkaz, který má být proveden. Pod parameters se skrývají parametry nutné k provedení tohoto příkazu.

V současné podobě lze jednotlivé příkazy popsat následovně (*in* - tvar zprávy, která může být posílána ze zařízení(klienta); *out* - tvar zprávy, kterou zařízení(klient) po poslání dané „*in*" zprávy obdrží):

- init - inicializuje spojení. Komunikace v případě úspěšného spojení s editorem proběhne následovně:

  out: {command:"init",parameters:{}}
  in: {command:"init",parameters:{},errorcode:0}

- addNode - přidá do *editoru* další bod. Povinným parametrem jsou v tomto případě souřadnice bodu, který má být přidán. Nepovinným parametrem je parametr index, který udává, na jakou pozici se má přidávaný bod v poli všech bodů udávajících trajektorii zařadit.

  out: {command:"addNode",parameters:{coordinates:[coX,coY,coZ], index:ind}}
  in: –

- updateNode - aktualizuje bod, jehož pozice v poli všech bodů je dána indexem. Aktualizovaný bod se přemístí na nově zadané souřadnice. V případě, že do index je poslána hodnota -1, zobrazí se na zadaných souřadnicích ukazatel, kterého je možné opět vypnout zasláním zprávy s hodnotou indexu -1, ale tentokrát bez souřadnic.

  out:{command:"updateNode","parameters":{index: ind, coordinates:[coX,coY,coZ]}}
  in: –

- removeNode - odebere bod, jehož pozice v poli všech bodů je dána indexem.

  out: {command:"removeNode","parameters":{ index: ind}}
  in: –

- removeAllNodes - odebere všechny body.

  out: **out**: {command:"removeAllNodes","parameters":{}}
  in: –

- getNodes - načte všechny body z *editoru*.

  out: {command:"getNodes",parameters:{}}
  in: {command:"getNodes",parameters:{nodes:[[coX,coY,coZ],[...],...}}

3.3.3 Realizace komunikace

Komunikaci s editorem je možné realizovat vytvořením třídy, která bude reprezentovat zprávu se všemi potřebnými atributy. Tu následně serializovat (v případě zasílání zprávy do *editoru*) či deserializovat (v případě příjmu zprávy z *editoru*) a data z ní zpracovat. Samotné odesílání a příjem zpráv pak zařídit pomocí *WebSocket* klienta.

Příklad třídy reprezentující zprávu obousměrné komunikace klient-*editor*:

```
class Message
{
  public string command { get; set; }
  public int errorcode { get; set; }
  public Parameters parameters { get; set; }
}
//
public class Parameters
{
  public int index { get; set; }
  public double[] coordinates { get; set; }
```

```
  public double[][] nodes { get; set; }
}
```

4 Aplikační část práce

Hlavním cílem práce bylo především navržení aplikace založené na vhodném interaktivním rozhraní, které využije možnosti *Kinect*u. Následující řádky dokumentují, jakým způsobem bylo k tomuto cíli přikročeno. Popis postupu návrhu aplikace by měl odpovídat postupu plynoucímu z předchozích kapitol práce.

4.1 Definice cíle

Cíl práce se týká vytvoření aplikace, která komunikuje s *editorem*. Pomocí interaktivního rozhraní aplikace by tak mohlo být dosaženo toho, že uživatel bude při ovládání *editoru* odkázán pouze na pohyby částí svého těla, popřípadě nějaký hardware, který bude mít v prostoru před *Kinect*em u sebe.

Dále se bude muset aplikace vypořádat s problémem vymezení prostoru. Bylo by dobré, kdyby měl uživatel možnost nějak fyzicky vymezit prostor, jenž by pak odpovídal prostoru v *editoru*. Toho by se dalo dobře využít v praxi, neboť každý bod *editoru* by pak odpovídal své reálné pozici. Například - jednalo-li by se o divadelní scénu, mohl by režisér po vymezení divadelní scény, nad kterou by byl umístěn *editorem* řízený lanový robot, pouze ukázat přímo na scéně, kudy má výsledná trajektorie robota procházet a aplikace v kombinaci s *editorem* by jeho požadavkům vyhověla (samozřejmě se jedná pouze o modelový příklad a rozmezí viditelnosti jednoho *Kinect*u celou divadelní scénu nepokryje).

4.2 Řešení

Máme-li již definován cíl práce, podívejme se nyní na to, jak bylo přistoupeno k jeho řešení.

4.2.1 Komunikace s editorem

Řešení komunikace s editorem plyne přímo z podkapitoly 3.3.

4.2.2 Interaktivní rozhraní aplikace

Z podkapitoly 2.3 plyne, jak by mělo interaktivní rozhraní využívající možnosti *Kinect*u vypadat, tj. že by se měl programátor vždy pokusit o intuitivnost a efektivitu. V případě aplikační části práce byl pro dodržení těchto podmínek zvolen následující přístup.

Uživatel bude pro plné ovládání používat jednak své ruky (viz *Hand Tracking* v *OpenNI/NITE*) a jednak bude ve své druhé, pasivní, ruce využívat možnosti prezentéru či jiného podobného zařízení. Důvod použití dodatečného hardwaru plyne ze zmíněné efektivity ovládání. Zajišťuje pohodlí pro uživatele a vylepšuje přesnost ovládání.

Základním prvkem rozhraní je sledování pozice uživatelovy ruky v prostoru před *Kinect*em, která bude po určitém vymezení prostoru odpovídat svými souřadnicemi souřadnicím v *editoru*.

Dalším nástrojem rozhraní je detekce stisknutí dvou různých kláves (odpovídajícím dvou různým tlačítkům prezentéru). První bude umožňovat „položení“ bodu, tj. přidání nového bodu do seznamu doposud zaznamenaných bodů editoru určujících výslednou trajektorii. Pokud se uživatelova „ovládací“ ruka přiblíží svou polohou k poloze určitého, již zaznamenaného, bodu trajektorie a uživatel stiskne druhé tlačítko, dojde k vybrání tohoto bodu. Vybraný bod je pak možné přemístit na jinou pozici opět pohybem ruky a stiskem prvního tlačítka na signalizaci toho, že na aktuální pozici ruky se má tento bod přemístit, nebo je jej možné ze seznamu bodů trvale odstranit – opětovné stisknutí druhého tlačítka.

Takto navržené rozhraní se zdá být dostatečně intuitivní, byť jeho využití bude potřebovat i jistý návyk z uživatelovy strany.

Poznámka pro zpětnou vazbu, tj. aby měl uživatel o svých úkonech náležitý přehled, je v samotné aplikaci i editoru podpořeno grafické rozhraní, které plně podporuje výše zmíněné principy.

4.2.3 Vymezení prostoru (kalibrace)

Důležitým úkolem je vymezení části prostoru před *Kinect*em, která má být následně ztotožněna s *prostorem editoru*. Toto vymezení nechť bude dále označováno jako kalibrace.

Dříve než začne popis samotného principu návrhu kalibrace, je nutné vytyčit, co se uvažuje pod pojmem *prostor editoru*. Jelikož má *editor* nastavenou fixní velikost scény na 1000 pixelů, lze si představit jako *prostor editoru* krychli o hraně délky 1000. Pro určení souřadnice bodu ležícího v tomto prostoru je ve středu pomyslné krychle umístěn osový kříž, tj. limitní hodnota těchto navzájem ortogonálních os x, y, z je ± 500. K lepšímu nahlédnutí viz Obrázek 13.

Z podkapitoly 2.4.3 plyne, že *Hand Tracking*em získaná pozice ruky odpovídá její vzdálenosti od osy *hloubkové kamery* v milimetrech a respektuje pravotočivý systém souřadný, tj. pohybuje-li se uživatelova ruka napravo od osy, je získána kladná x-ová souřadnice, pohybuje-li se směrem nahoru od osy, je získána kladná y-ová souřadnice a naopak. Hloubka, tj. z-ová souřadnice, je vždy obdržena pouze v kladném tvaru – vzdálenost od *Kinect*u v milimetrech.

Navržený postup kalibrace Uživatel bude dříve než začne ovládat editor vyzván k udání vrcholů obecného kvádru (tj. sady 8 bodů), který vymezuje tu část prostoru před *Kinect*em, jež má odpovídat prostoru *editoru*.

Tyto body budou pro následující úvahy označeny po řadě, tj. dle směru kalibrace, $B_1, B_2, \ldots, B_8$. Pozici bodů určují souřadnice získané pomocí *Kinect*u $B_1 = [x_{k1}; y_{k1}; z_{k1}], B_2 = [x_{k2}; y_{k2}; z_{k2}], \ldots, B_8 = [x_{k8}; y_{k8}; z_{k8}]$.

Je zřejmé, že body $B_1, B_2, \ldots, B_8$ mají být nějakým způsobem použity pro získání obecných transformačních vztahů, které by pozici pravé ruky zaznamenávanou *Kinect*em přetransformovali do *prostoru editoru*. Nabízí se tak využít například lineární regresi.

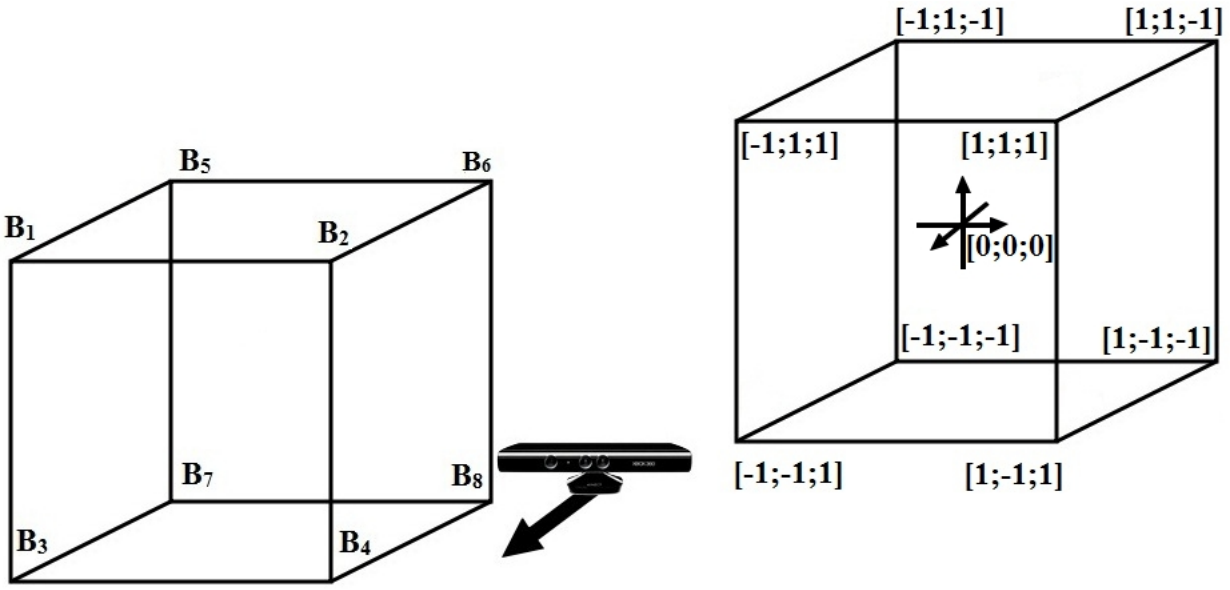
Výše zmíněné body jsou tedy vyjádřeny tak, aby jejich hodnoty odpovídaly hodnotám jim odpovídajících bodů v *prostoru editoru*, tj. aby odpovídaly vrcholům krychle, která *prostor editoru* vymezuje. Vzhledem k lepšímu následnému použití se nebere v úvahu krychle délky 1000, nýbrž krychle o délce 2 (hodnoty souřadnic tak získáme v intervalu $< -1; 1 >$, což zlepší přepočty pro různé limitní hodnoty podobně vymezených prostorů - bude je stačit pouze vynásobit touto limitní hodnotou. Například pro případ *prostoru editoru* se vynásobí vždy každá nově získaná souřadnice bodu hodnotou 500).

Obrázek 13 podpořený rovnicemi dokumentuje, jak jsou uživatelem postupně zadávané body, které vymezují prostor před *Kinect*em ve tvaru obecného kvádru, porovnávány s odpovídajícími hranami krychle.

$$\begin{aligned} B_1 = [x_1; y_1; z_1] &= [-1; 1; 1] \\ B_2 = [x_2; y_2; z_2] &= [1; 1; 1] \\ &\vdots \\ B_8 = [x_8; y_8; z_8] &= [1; -1; -1] \end{aligned} \tag{1}$$

Následně budou tyto hodnoty souřadnic pro každý bod vyjádřeny jako lineární kombinace původních souřadnic získaných pro tento bod *Kinect*em, čímž pro každý bod získáme sadu tří rovnic o devíti neznámých koeficientech.

$$x_n = r_1 \cdot x_{kn} + r_2 \cdot y_{kn} + r_3 \cdot z_{kn} \tag{2}$$



Obrázek 13: Body zadané postupně při kalibraci a s nimi porovnávané vrcholy krychle.

$$y_n = s_1 \cdot x_{kn} + s_2 \cdot y_{kn} + s_3 \cdot z_{kn} \tag{3}$$

$$z_n = t_1 \cdot x_{kn} + t_2 \cdot y_{kn} + t_3 \cdot z_{kn}, \tag{4}$$

kde koeficienty $r_1, r_2, r_3, s_1, s_2, s_3, t_1, t_2, t_3 \in \Re$ a $n = 1, \ldots, 8$.

Jednotlivé koeficienty je možné získat metodou nejmenších čtverců, ale dříve než bude metoda aplikována, je nutné ještě trochu poupravit souřadnice bodů získaných *Kinect*em, kvůli tomu, že z-ová souřadnice nabývá pouze kladných hodnot, a tudíž nepřepočítáním z-ové souřadnice by byla celá úloha špatně podmíněná.

Přepočet se provádí odečtem z-ové souřadnice středu uživatelem zadaného obecného kvádru. Tento střed je určen v souřadnicích bodů získaných *Kinect*em dle následujícího postupu (zapsán pro ilustraci bodově):

$$S = [x_{ks}; y_{ks}; z_{ks}] = \frac{\frac{\frac{B_1+B_2}{2} + \frac{B_3+B_4}{2}}{2} + \frac{\frac{B_5+B_6}{2} + \frac{B_7+B_8}{2}}{2}}{2} \tag{5}$$

Přepočet jednotlivých souřadnic:

$$x'_{kn} = x_{kn} - x_{ks};\ y'_{kn} = y_{kn} - y_{ks};\ z'_{kn} = z_{kn} - z_{ks}, \tag{6}$$

kde $n = 1, \ldots, 8$.

Takto upravenou úlohu hledání optimálních koeficientů lze přepsat do maticového tvaru:

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_8 \end{bmatrix} = \begin{bmatrix} x'_{k1} & y'_{k1} & z'_{k1} \\ x'_{k2} & y'_{k2} & z'_{k2} \\ \vdots & \vdots & \vdots \\ x'_{k8} & y'_{k8} & z'_{k8} \end{bmatrix} \cdot \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix} \tag{7}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_8 \end{bmatrix} = \begin{bmatrix} x'_{k1} & y'_{k1} & z'_{k1} \\ x'_{k2} & y'_{k2} & z'_{k2} \\ \vdots & \vdots & \vdots \\ x'_{k8} & y'_{k8} & z'_{k8} \end{bmatrix} \cdot \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} \tag{8}$$

$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_8 \end{bmatrix} = \begin{bmatrix} x'_{k1} & y'_{k1} & z'_{k1} \\ x'_{k2} & y'_{k2} & z'_{k2} \\ \vdots & \vdots & \vdots \\ x'_{k8} & y'_{k8} & z'_{k8} \end{bmatrix} \cdot \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \tag{9}$$

Nyní již lze využít *metody nejmenších čtverců* k určení jednotlivých koeficientů, tj. obecně je hledáno optimální řešení rovnice $c = A \cdot b$ ve smyslu kriteriální funkce $F(b) = ||Ab - c||^2$ v závislosti na vektoru neznámých neboli řešením je takový vektor b, pro který kriteriální funkce F(b) nabývá svého minima, tj. pokládá se $\frac{\partial F}{\partial x} = 2A^TAb - 2A^Tc$ rovno nule, z čehož plyne, že optimálním řešením ve smyslu kriteriální funkce $F(b)$ je $b = (A^TA)^{-1}A^Tc$. Výraz $A^+ = (A^TA)^{-1}A^T$ se pak označuje jako *Moore-Penroseova pseudoinverze*. Pro podrobnější odvození *metody nejmenších čtverců* viz [26].

Nechť je zavedena matice M jako:

$$\begin{bmatrix} x'_{k1} & y'_{k1} & z'_{k1} \\ x'_{k2} & y'_{k2} & z'_{k2} \\ \vdots & \vdots & \vdots \\ x'_{k8} & y'_{k8} & z'_{k8} \end{bmatrix} = M \tag{10}$$

Jednotlivá řešení rovnic (7), (8) a (9) v duchu *metody nejmenších čtverců*, tj. hodnoty

vektorů koeficientů $r = [r_1\ r_2\ r_3]^T$, $s = [s_1\ s_2\ s_3]^T$ a $t = [t_1\ t_2\ t_3]^T$ budou získány z:

$$r = M^+ \cdot x \tag{11}$$
$$s = M^+ \cdot y \tag{12}$$
$$t = M^+ \cdot z \tag{13}$$

Jakmile jsou určené koeficienty, lze již provádět transformaci každého bodu získaného pomocí *Kinect*u ($B_{kinect} = [x_{kinect}; y_{kinect}; z_{kinect}]$) následujícím způsobem:

$$x_{transform} = r_1 \cdot x_{kinect} + r_2 \cdot y_{kinect} + r_3 \cdot z_{kinect} \tag{14}$$

$$y_{transform} = s_1 \cdot x_{kinect} + s_2 \cdot y_{kinect} + s_3 \cdot z_{kinect} \tag{15}$$

$$z_{transform} = t_1 \cdot x_{kinect} + t_2 \cdot y_{kinect} + t_3 \cdot z_{kinect} \tag{16}$$

Je tak získán transformovaný bod $B_{transform} = [x_{transform}; y_{transform}; z_{transform}]$, jehož souřadnice mají hodnoty spadající do intervalu $< -1; 1 >$. Například pro transformaci do *prostoru editoru* je tak ještě nutné vynásobit každou souřadnici 500.

Poznámka Minimální počet bodů, které je nutné zaznamenat při takto navrženém postupu kalibrace, jsou tři. Pro získání lepších výsledků a i jistou lepší přehlednost pro uživatele byla kalibrace navržena s osmi body symbolizujícími ony vrcholy obecného kvádru, jenž vymezuje prostor, který se ve výsledku transformuje do *prostoru editoru*.

4.3 Výsledná aplikace

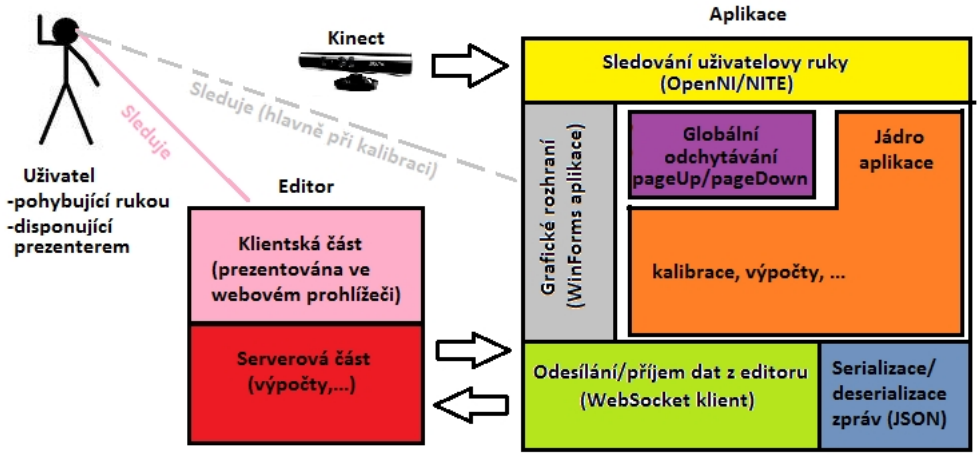
Po nastínění řešení zbývá vysvětlit, jak byla výsledná aplikace implementována.

4.3.1 Struktura aplikace

Obrázek 14 demonstruje základní strukturu a pozici aplikace v systému uživatel-*Kinect*-aplikace-*editor*.

Aplikaci lze rozdělit na šest základních oddílů. Třem z nich, tj. problematice sledování uživatelovy ruky, implementaci *WebSocket* klienta a formátu zpráv *JSON*, byly již věnovány kapitoly a podkapitoly 2.4.3, 3.3 a 3.3.1.

Jádro aplikace pak tvoří vesměs postupy navržené v rámci řešení funkčních stránek aplikace a jedná se o nejrozsáhlejší část.



Obrázek 14: Struktura aplikace a pozice aplikace v systému uživatel-*Kinect*-aplikace-*editor*.

Aplikace byla vyvinuta ve vývojovém prostředí *Visual Studio 10* a to jako klasická WinForms aplikace, což už částečně předurčuje její podobu, tj. její grafické rozhraní.

Poslední oddíl je věnován odchytávání stisknutí dvou klíčových kláves (tlačítkům) (důvod plyne z podkapitoly 4.2.2), které je dobré zajistit nějak globálně, neboť jedině tak bude možné aplikaci minimalizovat a zachovat její funkčnost, což se hodí právě při sledování klientské části *editoru*.

Z implementačního hlediska je aplikace strukturována do následujících tříd (u každé třídy bude vysvětleno, jaké místo v celém projektu zaujímá):

- Calculations.cs - třída, která obsahuje metody odpovědné za všechny důležité výpočty. Vzhledem k nutnosti implementace složitějších výpočtů bylo v této třídě využito open source knihovny *Math.NET Numerics* [16].
- Calibration.cs - třída disponující metodami odpovědnými za postup *kalibrace*.
- Client.cs - třída navržená tak, aby mohla být použita pro realizování komunikace s *editorem* a to i v rámci jiných projektů (společně s třídou Message.cs). Kromě funkce klasického *WebSocket* klienta obsahuje i metody umožňující následnou komunikaci s *editorem*. Jejich názvy pak odpovídají názvům jednotlivých příkazů (například init(), updateNode(**int** index, **double**[] node) apod.).

- Communication.cs - obsahuje dodatečné úkoly, které je nutné dořešit pro komunikaci aplikace-*editor*.
- Form1.cs - obsahuje tzv. kód pod formulářem, tj. kód, který leží pod grafickou reprezentací formuláře, a z tohoto důvodu se jedná o jakousi hlavní třídu celého projektu. Na úrovni této třídy je dále řešena veškerá spolupráce s *Kinect*em.
- globalKeyboardHook.cs - třída, která řeší globální odchytávání kláves. Není autorská - k dispozici ji je možné získat jako kód podléhající *The Code Project Open License*(CPOL) [17].
- Message.cs - třída, jejíž tvar byl diskutován v rámci podkapitoly 3.3.3. S výše zmíněnou třídou Client.cs může tvořit samostatný celek, který zprostředkovává komunikaci s *editorem*.
- Painting.cs - třída, která obsahuje metody pro vykreslení různých komponent grafického rozhraní aplikace.
- Program.cs - třída obsahující statickou metodu main(), tj. vstup do aplikace.

Poznámka Další třídy a soubory projektu odpovídají souborům nutným k realizaci formulářové aplikace navržené ve *Visual Studio 10*.

4.3.2 Příklady implementace

Na tomto místě uvedeny příklady samotné implementace některých problémů, o kterých byla zmínka, v programovacím jazyku C#.

V přehledu tříd projektu byla diskutována role tříd Calibration.cs a Message.cs. Následující příklad dokumentuje, jak je realizována metoda AddNode(**double**[] node), která má za úkol přidat do editoru nový bod:

```
public void AddNode(double[] node)
{
  Message message = new Message();
  Parameters parameters = new Parameters();
  message.command = "addNode";
  parameters.coordinates = node;
  message.parameters = parameters;
  string output = JsonConvert.SerializeObject(message, Formatting.None, new
  JsonSerializerSettings { NullValueHandling = NullValueHandling.Ignore,
```

```
  DefaultValueHandling = DefaultValueHandling.Ignore });
  websocket.Send(output);
}
```

Další příklad se věnuje problému stisknutí klíčové klávesy (v případě současné implementace aplikace - PageDown) proběhla-li již kalibrace (kód byl pro tento příklad redukován pouze na posílání klientských požadavků):

```
private void pageKeyDown(object sender, KeyEventArgs e)
{
  ...
  if (e.KeyValue == Convert.ToInt32(Keys.PageDown) && this.amITracking &&
      this.amICalibrated&&!this.calibration.outOfAxes
      (this.calibration.newPoint3D(this.handPosition)))
        {
     if(this.holdingPoint)
       {
          ...
          if (this.amIConnected) this.client.updateNode(this.holdingIndex,
              this.calibration.point3DToEditor(this.handPosition));
        }
      else
         {
           ...
           if(this.amIConnected)this.client.AddNode
           (this.calibration.point3DToEditor(this.handPosition));
         }
     }
  ...
}
```

Jelikož příklad implementace *Hand Tracking*u byl již uveden v rámci 2.4.3, je věnován poslední příklad kalibraci a to konkrétně metodě, která za kalibraci odpovídá (samozřejmě zde nebudou uváděny všechny kroky kalibrace):

```
public bool calibrateMe(Graphics graphics, int step, Point3D handPosition)
{
  ...
  switch (step)
  {
    case 0:
      this.calibrationStatus = "Enter the right upper corner.";
      this.painting.drawTheBlinkingPoint(graphics, painting.x1);
      this.rightUpperCorner1 = handPosition;
      this.painting.drawTheCube(graphics);
```

```
      return false;
    ...
    case 7:
      this.calibrationStatus = "Enter the left bottom corner.";
      this.painting.drawTheBlinkingPoint(graphics, painting.x8);
      this.leftBottomCorner2 = handPosition;
      this.painting.drawTheCube(graphics);
      return false;

    case 8:
      if (forFirstTime)
      {
        this.center = calculations.centerOfBox(//all points);
        ...
        this.coeficientsX = calculations.calculateCoeficientsForX(
        //all points minus center);
        ...
        forFirstTime = false;
      }
      this.painting.drawTheCube(graphics);
      return true;
  }
}
```

4.3.3 Uživatelská stránka aplikace

Následující řádky mohou sloužit i jako jakýsi uživatelský manuál či dokumentace.

Aplikace je spustitelná standardně z exekutivy EditorWithKinect.exe. Její korektní běh je podmíněný nainstalováním všech důležitých komponent pro *Kinect* popisovaných v podkapitole 2.4.3 a připojením *Kinect*u. Dále se předpokládá, že má uživatel po ruce nějaký prezentér či jiný podobný hardware pro vzdálené ovládání mapující se na klávesy PageDown a PageUp (samozřejmě je možné ovládat i stiskem příslušných kláves přímo na klávesnici).

V případě, že se chce uživatel připojit k *editoru*, učiní tak zadáním IP adresy a portu do příslušných kolonek. Přednastaveny jsou hodnoty pro připojení k *editoru* na stejném počítači, na kterém běží i aplikace. Připojení však pro běh aplikace není nějak nutné – aplikaci je možné vyzkoušet i bez něj.

V zorném poli *Kinect*u již stojící uživatel aplikace je vyzván k vykonání gesta (mávnutí). To provede rukou, kterou chce dále využívat pro určování polohy. Gesto je nutné vykonat kvůli tomu, aby mohla být zjištěna startovací pozice ruky pro *Hand Tracking*. Více o gestu viz podkapitola 2.4.3.

Provedl-li uživatel gesto, přišel čas na kalibraci. Uživatel je pomocí grafiky veden k postupnému zadání osmi vrcholů obecného kvádru, kterými vytyčí prostor, jenž bude odpovídat *prostoru editoru* viz 4.2.3. Kvádr pak může být vůči *Kinect*u libovolně pootočen – použitá metoda výpočtu transformace to umožňuje. Jednotlivé body kalibrace zadává uživatel stiskem klávesy na prezentéru či jiném podobném zařízení, která odpovídá PageDown.

Pokud je aplikace připojena k *editoru*, zůstává na rozhodnutí uživatele, kterému grafickému rozhraní dá přednost – zda *editoru* či aplikaci. V případě grafického rozhraní aplikace vidí uživatel před sebou krychli odpovídající prostoru, který nakalibroval. Přiblíží-li se polohou ruky, která je *Kinectem* sledována, k tomuto prostoru zobrazí se ukazatel ve formě plně vybarveného kruhu. Ten odpovídá pozici ruky v nakalibrovaném prostoru – je-li jeho ruka někde blízko tomuto prostoru, ale nikoliv uvnitř – má červenou barvu, je-li uvnitř má barvu zelenou a po hranách zobrazované krychle začnou jezdit ukazatelé ve formě úseček. Ty mají za úkol zlepšit uživatelovu orientaci v onom prostoru.

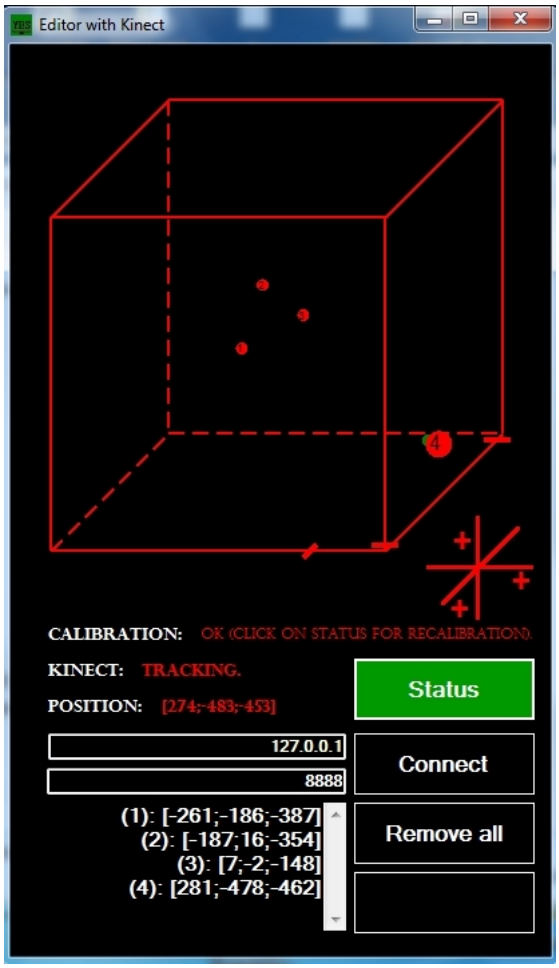
V prostředí *editoru* se zobrazuje výsledná trajektorie. Též se zde zobrazuje ukazatel ve formě kruhu a dodatečné úsečky, které upřesňují polohu (příkazy pro realizaci ukazatele viz 3.3.2 Popis komunikace).

Ovládání aplikace se nese v duchu navrženého rozhraní viz podkapitola 4.2.2. Nové body trajektorie lze zaznamenat pohybem „ovládající“ ruky na pozici, kde má být bod umístěn a stiskem tlačítka odpovídajícího PageDown. Pro přemístění či odstranění bodu trajektorie je nejprve nutné, aby uživatel umístil svoji ruku do okolí bodu. U aplikace i editoru se pak bod, ke kterému se uživatel přiblížil, zvýrazní. To je ten správný okamžik pro jeho výběr – stisk PageUp. Vybraný bod je pak možné přemisťovat. Dostane-li uživatel bod na pozici, na kterou jej chtěl přemístit – stiskne PageDown. Odstranění bodu se provádí opětovným stiskem PageUp.

Kdykoliv během procesu *Hand Trackingu* se může stát, že *Kinect* ruku tzv. ztratí. Nastane tedy nutnost pro opětovné provedení gesta. Aplikace indikátor ztráty obsahuje dokonce i pro případ, kdy má uživatel aplikaci minimalizovanou. Indikace této ztráty je provedena změnou ikony spuštěné aplikace na hlavním panelu ze zelené na červenou (dobře viditelné hlavně v prostředí Windows 7).

Opětovná kalibrace se provádí stiskem tlačítka Status.

Nároky na hardware se odvíjí od použití *Kinect*u a OpenNI viz podkapitola 2.4.1.



Obrázek 15: Grafické rozhraní navržené aplikace.

Požadavky na operační systém plynou z použití grafického rozhraní formulářové aplikace systému Windows (Windows XP & vyšší). Taková implementace byla zvolena kvůli jednoduchosti tvorby grafické stránky aplikace, na jejíž podobu nemusel být brán vzhledem k dostatečnému grafickému rozhraní editoru větší zřetel.

4.3.4 Zhodnocení aplikace

Podařilo se vytvořit aplikaci založenou na dostatečně intuitivním interaktivní rozhraní.

Míra intuitivnosti aplikace bude záviset na posouzení uživatelů, kteří budou systém *Kinect*-aplikace-*editor* využívat. Lze předpokládat, že se najdou tací, kterým námi navržené interaktivní rozhraní nebude vyhovovat, ale tento případ nastává i pro jiná, daleko profesionálnější uživatelská rozhraní. V mnoha ohledech záleží na úrovni počítačové gramotnosti uživatele, schopnosti učit se a jeho, což pro vše týkající se *Kinect*u platí dvojnásob, hravosti.

Co se rychlosti komunikace s *editorem* týče - záleží především na rychlosti připojení. Interval zasílání požadavků v případě komunikace s *editorem* je plně závislý na rychlosti *Hand Tracking*u, který souvisí s frekvencí získávání *hloubkové mapy* z *hloubkové kamery Kinect*u (30Hz, tj. interval zhruba 0,033s) viz 3.3.2.

Na testovacím zařízení, tj. notebook HP Pavilion dv6000 - procesor Intel Celeron M 530 1,73GHz, 1024Mb RAM, se hodnota *fps* zdaleka nepřibližuje číslu 30. Naopak - je pohyblivá a pohybuje se v rozmezí 15 - 21*fps*, z čehož lze vypozorovat i jistou vazbu použití funkce *Hand Tracking* na hardware počítače.

4.4 Další příklad aplikace - dvojkolka

Jako další příklad interaktivního rozhraní, které využívá zařízení *Kinect*, a též k demonstraci některých, v této bakalářské práci pouze okrajově zmíněných, funkcí frameworku *OpenNI*, byla vytvořena aplikace pro řízení dvojkolky pomocí *Kinect*u. Podívejme se proto krátce na realizaci této aplikace.

Dvojkolka (viz Obrázek 16) je příklad dvoukolového inverzního kyvadla, tj. inverzního kyvadla, k němuž je z každé strany připevněn motor, na jehož hřídeli je kolo. Jedná se tedy o jakési automaticky řízené vozítko Segway, jež se stalo inspirací k jejímu sestavení. Pro další informace o dvojkolce viz [10].

Celý systém dvojkolky je řízený řídicím systémem REX [3][4], jehož součástí je i konzolová aplikace *RexWS*, která slouží jako *WebSocket* server pro zpřístupnění dat ze systému REX. *RexWS* se připojuje přímo k běžícímu RexCore.

To ve výsledku umožňuje aplikovat základní poznatky získané z této práce i na problém řízení dvojkolky pomocí *Kinect*u.

Komunikace s RexWS Forma komunikace s *RexWS* je od formy komunikace s editorem odlišná, avšak základní principy zůstávají zachovány a to včetně použití formátu *JSON* pro serializaci a deserializaci zpráv mezi klientem a serverem.

Obrázek 16: Dvojkolka [10].

Základní formát zprávy pro *RexWS* má v současné implementaci následující tvar:
`{cmd: int, header: {header}, data: {data}}`

- cmd - příkaz diagnostického protokolu.
- header - hlavička - obsahuje klientem libovolně generované id, server vrací id, lokální chyby a chyby serveru:
  - header klient-server: {id: **int**}; id - klientem libovolně generované id.
  - header server-klient: {id: **int**, srverr: **string**, svrerrno: **int**, err: **string**, errno: **int**}; id - klientem libovolně generované id, errno - číslo lokální chyby (RexWS), err - text lokální chyby (RexWS), svrerrno - číslo chyby serveru (RexCore), svrerr - text chyby serveru (RexCore).
- data - formát dle příkazu

Seznam příkazů (forma zápisu jako u podkapitoly 3.3.2):

- XDG_INIT - inicializace spojení.

```
in: {host: "hostname", port:port}
out:
```

- XDG_EXIT - ukončení spojení.

```
in:
out:
```

- XDG_GET_VERSION - zjištění verze REXu.

```
in:
out: {version: {hi: int, lo: int, rel: int, rev: int, day: int, month: int,
     year: int}}
```

- XDG_ADD_GROUP - vytvoří novou skupinu.

```
in: {names: [string array]}
out: {ids: [int array], groupid: int}
```

- XDG_READ_GROUP - čtení dat.

```
in:  {groupid: int}
out:  {values: [value array], groupid: int}
```

- XDG_WRITE_GROUP - zápis dat.

```
in:  {groupid: int, values: [value array]}
out:
```

Po navázání spojení se serverem je třeba poslat příkaz XDG_INIT. Dále je nutné vytvořit tzv. skupinu, tj. skupinu dat ve spuštěné úloze systému REX, která mají být měněna - v případě dvojkolky půjde hlavně o rychlost otáčení a rychlost pohybu v tasku DK_State_FB. Opět byly pro potřeby řízení dvojkolky prostřednictvím *Kinect*u realizovány (obdobným postupem jako u *editoru*) dvě klientské třídy umožňující komunikaci s *RexWS* - třída klientská RexWSclient.cs a třída reprezentující zprávu RexWSmessage.cs.

Následující příklad demonstruje průběh komunikace se serverem použitím metod třídy RexWSclient.cs:

```
rws = new RexWSclient(ip, port);
//
rws.ConnectToTarget(); //init
//
rws.AddGroup(new string[] { "DK_State_FB.rychlost_otaceni:ycn",
                            "DK_State_FB.rychlost_pohybu:ycn" }));
```

Server pak vrátí id skupiny a již lze do této skupiny zapisovat nebo z ní číst.

```
rws.ReadGroup(id);
//
rws.WriteGroup(id, new double[] {0.0,0.0}));
//dvojkolka stoji (rychlost otaceni i pohybu na 0)
```

Interaktivní rozhraní Interaktivní rozhraní aplikace využívající možnosti *Kinect*u slouží jako příklad funkce *Skeletal Tracking*.

Funguje jako jakýsi imaginární volant, který uživatel svírá ve svých rukou. Rychlost pohybu směrem dopředu, popř. dozadu, se ovlivňuje příslušným pohybem paží vpřed, popř. vzad. Na začátku řízení je uživatel nejprve vyzván k uvedení se do *kalibrační pózy*. Jakmile je rozpoznán a začne *Skeletal Tracking*, musí uživatel před řízením ještě vyrovnat ruce - dát je vodorovně do polohy, kde chce mít nulovou hodnotu rychlosti, a setrvat cca tři vteřiny v této poloze – kvůli tomu, aby bylo jasné, že chce dvojkolku skutečně řídit.

K tomu, aby mohlo takové rozhraní vzniknout, je zapotřebí čtyř *joint*ů - LeftHand, RightHand, Torso, Head, reprezentujících po řadě levou ruku, pravou ruku, střed těla a hlavu.

Joint Head je nutný k tomu, aby se zabránilo začátku odpočítávání už při ukončení kalibrační pózy (mohlo by se stát, že by si uživatel, byť je mu to oznámeno skrze grafické rozhraní aplikace, nevšiml, že je již „trackován“ a jeho ruce by zůstali v *kalibrační póze* vyrovnány v pozici nad hlavou, což by započalo neplánovaně vlastní proces řízení). Sleduje se tak y-ová souřadnice tohoto *joint*u.

*Joint*u Torso je využito k umožnění zmíněné logiky řízení pohybu dvojkolky směrem dopředu a vzad – uživatel se tak může libovolně přemisťovat v zorném poli *Kinect*u s tím, že pozice obou rukou pro nulovou rychlost bude zachována.

*Joint*y udávající pozici rukou pak slouží k natáčení, tj. udávají rychlost otočení. Úhel natočení imaginárního volantu je získáván pomocí funkce tangens příslušným využitím x-ové a y-ové souřadnice pravé ruky a x-ové a y-ové souřadnice středu úsečky vzniklé mezi pravou a levou rukou. Tento úhel je pak lineárně přepočítáván na hodnotu rychlosti otočení.

Grafické rozhraní Grafické rozhraní je zde, stejně jako v případě *editoru*, založené na formulářové aplikaci operačního systému Windows.

Co je ale zajímavější – byla zde pro zpětnou vazbu k uživateli použita funkce *Player Segmentation* zmíněná v podkapitole 4.2.2. Pro náhled grafického rozhraní aplikace viz Obrázek 17.

Závěrečné poznámky Vzhledem k tomu, že se v aplikaci počítá pouze s jedním uživatelem (řidičem), nastává případ, kdy *Kinect* jako první osobu určí nějakou překážku či část dalších osob ve svém zorném poli - je tedy nutné ze strany uživatele vyčkat po určitý časový okamžik nebo se pokusit nějak pohnout, aby jej *Kinect* rozpoznal a aby jej indexoval jako v pořadí prvního uživatele. Tento postup byl v implementaci aplikace zvolen kvůli zajištění větší bezpečnosti.

Interaktivní rozhraní aplikace pro řízení dvojkolky vyhovuje základním požadavkům na interaktivní rozhraní zmíněných v textu práce.

5 Závěr

Práce byla od svého začátku koncipována tak, aby pouze neinformovala své čtenáře o aplikační činnosti, ale aby je i nějakým způsobem seznámila s problematikou, s níž se třeba doposud nesetkali.

Po přečtení práce by měl být čtenář seznámen především s možnostmi využití zařízení *Kinect* pro sledování lidského těla a získat povědomí o komunikaci s *editorem* tak, že by i on sám mohl nabyté vědomosti aplikovat nebo se alespoň orientovat v této problematice do takové míry, aby si dokázal načerpat dodatečné informace z literatury či internetu.

Co se aplikační části týče, nebylo primárním účelem práce nějak rozsáhle popisovat kódovou stránku problému, ale spíše se zaměřit a dostatečně vysvětlit základní úvahy, které vedly k jeho řešení, tj. realizaci výsledné aplikace. Vše bylo podáno tak, aby se čtenář i v případě této kapitoly a hlavně v její zvolené struktuře, tj. definice problému, řešení problému a výsledná aplikace, dostatečně orientoval.

Obrázek 17: Grafické rozhraní aplikace pro řízení dvojkolky prostřednictvím *Kinect*u.

Literatura

[1] OPENNI. *OpenNI: open source framework*. [online]. [cit. 2012-04-19]. Dostupné z: <http://www.openni.org/>.

[2] Microsoft. *Kinect for Windows SDK*. [online]. [cit. 2012-04-19]. Dostupné z: <http://www.microsoft.com/en-us/kinectforwindows/>.

[3] REX CONTROLS. *Stručný popis systému REX*. Plzeň, 2011.

[4] REX CONTROLS. *Funkční bloky systému REX*. Referenční příručka. Plzeň, 2011.

[5] O. SEVERA, R. PIŠL, M. ČECH, M. GOUBEJ, M. ŠTĚTINA, M. SCHLEGEL. *New 3D HMI tool for robot path planning based on latest W3C standards*. Západočeská univerzita v Plzni, 2012.

[6] J. KRAMER, N.BURRUS, F.ECHTLER, D. HERRERA C., M. PARKER. *Hacking the Kinect*. Apress, 2012.

[7] Prime Sense. *Oficiální webové stránky společnosti*. [online]. [cit. 2012-04-19]. Dostupné z: <http://www.primesense.com>.

[8] MSDN. *Kinect for Windows SDK*. [online]. [cit. 2012-07-16]. Dostupné z: <http://msdn.microsoft.com/en-us/library/hh855347>.

[9] ZČU. *Publikace - Universal 3D trajectory planning editor*. [online]. [cit. 2012-07-21]. Dostupné z: <http://www.zcu.cz/research/publications/?id=43897490>.

[10] JÁGER A. *Návrh řídicího systému robustní stabilizace dvojkolky*. Diplomová práce, ZČU, 2011.

[11] WebSocket4Net. *A .NET websocket client implementation*. [online]. [cit. 2012-07-24]. Dostupné z: <http://websocket4net.codeplex.com/>.

[12] WebSocket.org. *Are you plugged in?*. [online]. [cit. 2012-07-23]. Dostupné z: <http://www.websocket.org/>.

[13] I. FETTE,A. MELNICOV, Google Inc. *The WebSocket Protocol*. [online]. [cit. 2012-07-23]. Dostupné z: <http://tools.ietf.org/html/rfc6455>.

[14] JSON.org. *Úvod do JSON*. [online]. [cit. 2012-07-26]. Dostupné z: <http://www.json.org/json-cz.html>.

[15] JSON.NET. *Serialize all the things*. [online]. [cit. 2012-07-26]. Dostupné z: <http://json.codeplex.com/>.

[16] Math.NET Numerics. *Open source numerical library for the .NET*. [online]. [cit. 2012-07-30]. Dostupné z: <http://mathnetnumerics.codeplex.com/>.

[17] The code project. *A Simple C# Global Low Level Keyboard Hook*. [online]. [cit. 2012-07-30]. Dostupné z: <http://www.codeproject.com/Articles/19004/A-Simple-C-Global-Low-Level-Keyboard-Hook>.

[18] Brekel. *All-in-one OpenNI Kinect Auto Driver Installer*. [online]. [cit. 2012-07-18]. Dostupné z: <http://www.brekel.com/?page_id=160>.

[19] Howstuffworks. *How Microsoft Kinect Works*. [online]. [cit. 2012-04-19]. Dostupné z: <http://electronics.howstuffworks.com/microsoft-kinect.htm>.

[20] ABC Linuxu . *Kinect pro Xbox 360 a GNU/Linux – rozlousknutí komunikace*. [online]. [cit. 2012-05-05]. Dostupné z: <http://www.abclinuxu.cz/clanky/kinect-pro-xbox-360-a-gnu-linux-rozlousknuti-komunikace>.

[21] Wikipedia. *Kinect*. [online]. [cit. 2012-04-19]. Dostupné z: <http://en.wikipedia.org/wiki/Kinect>.

[22] Wikipedia. *Structured-light 3D scanner*. [online]. [cit. 2012-05-05]. Dostupné z: <http://en.wikipedia.org/wiki/Structured-light_3D_scanner>.

[23] Wikipedia. *Natural user interface*. [online]. [cit. 2012-05-05]. Dostupné z: <http://en.wikipedia.org/wiki/Natural_user_interface>.

[24] Wikipedia. *Library*. [online]. [cit. 2012-07-19]. Dostupné z: <http://en.wikipedia.org/wiki/Programming_library>.

[25] Wikipedia. *WebSocket*. [online]. [cit. 2012-07-23]. Dostupné z: <http://en.wikipedia.org/wiki/WebSocket>.

[26] Wikipedia. *Metoda nejmenších čtverců*. [online]. [cit. 2012-07-29]. Dostupné z: <http://cs.wikipedia.org/wiki/Metoda_nejmensích_čtverců>.

Příloha

Příklad Příklad konfiguračního XML souboru zmíněného v podkapitole 2.4.3:

```
<OpenNI>
        <Licenses>
                <!-- Add licenses here
                <License vendor="vendor" key="key"/>
                -->
        </Licenses>
        <Log writeToConsole="false" writeToFile="false">
                <!-- 0 - Verbose, 1 - Info, 2 - Warning, 3 - Error (default) -->
                <LogLevel value="3"/>
                <Masks>
                        <Mask name="ALL" on="true"/>
                </Masks>
                <Dumps>
                </Dumps>
        </Log>
        <ProductionNodes>
                <Node type="Depth" name="Depth1">
                        <Configuration>
                                <Mirror on="true"/>
                        </Configuration>
                </Node>
                <Node type="Image" name="Image1" stopOnError="false">
                        <Configuration>
                                <Mirror on="true"/>
                        </Configuration>
                </Node>
                <!--
                <Node type="Audio" name="Audio1"/>
                -->
        </ProductionNodes>
</OpenNI>
```