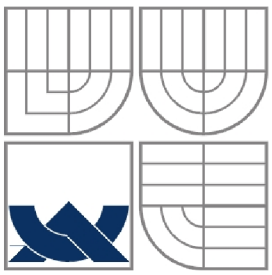


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

NÁVRH DÁLKOVÉHO OVLÁDÁNÍ MOBILNÍHO ROBOTU POMOCÍ MICROSOFT KINECT

THE REMOTE CONTROL DESIGN FOR AUTONOMOUS MOBILE ROBOT BASED ON MICROSOFT KINECT

DIPLOMOVÁ PRÁCE
DIPLOMA THESIS

AUTOR PRÁCE **BC. ADAM BARCAJ**
AUTHOR

VEDOUCÍ PRÁCE **ING. STANISLAV VĚCHET, PH. D.**
SUPERVISOR

BRNO 2012

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav automatizace a informatiky
Akademický rok: 2011/2012

Zadání závěrečné práce

student(ka): Bc. Adam Barcaj

který/která studuje v **magisterském navazujícím studijním programu**

obor: **Aplikovaná informatika a řízení (3902T001)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Návrh dálkového ovládání mobilního robotu pomocí Microsoft Kinect.

v anglickém jazyce:

The remote control design for autonomous mobile robot based on Microsoft Kinect.

Stručná charakteristika problematiky úkolu:
Hlavním cílem práce je návrh způsobu ovládání autonomního mobilního robotu s využitím zařízení Microsoft Kinect. Navržený systém musí podporovat zejména předávaní povelů pomocí předem definovaných gest. Důraz bude kladen zejména na zpracování rozhraní, které bude širšeji využitelné v dalších robotických projektech.

Cíle diplomové práce:

1) Seznamte se s možnostmi Microsoft Kinect
2) Prostudujte dostupné API rozhraní
3) Navrhněte vlastní systém gest rozpoznatelných tímto zařízením.
4) Celý systém zrealizujte a prakticky otestujte.

Seznam odborné literatury:
[1] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series). Intelligent robotics and autonomous agents. The MIT Press, August 2005.
[2] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. Principles of Robot Motion: Theory, Algorithms, and Implementations (Intelligent Robotics and Autonomous Agents). The MIT Press, June 2005.

Vedoucí diplomové práce: Ing. Stanislav Věchet, Ph.D.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2011/2012.

V Brně, dne

L.S.

Ing. Jan Roupec, Ph.D.	prof. RNDr. Miroslav Doupovec, CSc., dr. h. c.
Ředitel ústavu	Děkan fakulty

ABSTRAKT

Tato diplomová práce se zabývá využitím čidla Microsoft Kinect pro dálkové ovládání mobilního robotu. Cílem práce byl návrh způsobu ovládání autonomního mobilního robotu pomocí předem definovaných gest, tyto gesta následně zrealizovat a prakticky otestovat.

ABSTRACT

This diploma thesis deals with application of Microsoft Kinect sensor for remote control of mobile robot. The aim was design and implement methods for remote control of mobile robot with using of predefined gestures and these method practically tested.

KLÍČOVÁ SLOVA

Microsoft Kinect, čidlo, dálkové ovládání, gesta, mobilní robot

KEYWORDS

Microsoft Kinect, sensor, remote control, gestures, mobile robot

PROHLÁŠENÍ O ORIGINALITĚ

Prohlašuji, že jsem uvedenou práci zpracoval samostatně pod vedením Ing. Stanislava Věcheta, Ph.D. a použil jsem pouze literaturu uvedenou v bibliografii.

Květen 2012

Bc. Adam Barcaj …….....

BIBLIOGRAFICKÁ CITACE

BARCAJ, A. *Návrh dálkového ovládání mobilního robotu pomocí Microsoft Kinect.*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2012. 44 s. Vedoucí diplomové práce Ing. Stanislav Věchet, Ph.D..

PODĚKOVÁNÍ

Děkuji vedoucímu bakalářské práce Ing. Stanislavu Věchetovi, Ph. D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

Obsah:

1 ÚVOD

Microsoft Kinect je snímací zařízení původně navržené jako doplňkové zařízení pro herní konzoli XBox 360. Firma Microsoft se nakonec rozhodla podpořit i vývojářské aktivity pro jiné využití vydáním vlastních ovladačů, resp. SDK (Sowtware Development Kit), který zahrnuje ovladače, rozhraní pro komunikaci mezi zařízením a počítačem a ukázky kódů. První SDK bylo vydáno v červnu 2011 jako beta verze a prvních oficiálních ovladačů se vývojáři dočkali v únoru 2012.

MS Kinect umožňuje hraní her nebo ovládání aplikací bez potřeby držení jakéhokoliv ovladače nebo jiného zařízení v ruce a je tedy „hands-free“. Díky jednoduchému propojení a nízké ceně se dostává MS Kinect do různých projektů, které se snaží využít především možnosti hloubkové mapy (získávání vzdálenosti jednotlivých pixelů zorného pole) a rozpoznání „kostry“, kterou detekuje přímo čidlo. Jedním z možných využití je také ovládání autonomních mobilních robotů při umístění na robotu nebo pro manuální ovládání robotu. Člověk ovládající robot pomocí čidla MS Kinect nemusí držet žádné další zařízení nebo sedět u počítače, ale vše záleží na jeho pohybech, či pozici v jaké se nachází.

Hlavním cílem práce bylo navrhnout a zrealizovat systém gest rozpoznatelných čidlem MS Kinect. Byly navrženy čtyři způsoby dálkového ovládání mobilního robotu. Jeden ze způsobů není realizován jako předávání gest, ale jako přímé zadávání rychlosti robotu. Zadávání gest je potom realizováno pohybem, mluveným slovem nebo statickou pózou. Zadávání gest mluveným slovem nevyužívá pouze základního SDK a čidla MS Kinect, ale také doplňkové knihovny firmy Microsoft pro rozpoznání řeči a slovníky. Slova je poté nutné zadávat v anglickém jazyce.

Dalším cílem bylo otestovat systém navržených gest na mobilním robotu. Test proběhl na autonomním mobilním robotu „Leela“, se kterým počítač komunikuje bezdrátově pomocí Bluetooth. Proběhlo otestování všech možností dálkového ovládání pomocí čidla MS Kinect.

2 MICROSOFT KINECT

Microsoft Kinect je snímací zařízení, původně navržené jako doplněk pro herní konzoli XBox 360. Jeho cílem bylo zjednodušit hraní her pomocí pohybů vlastního těla bez potřeby držení jakýchkoliv snímačů. Po uvedení na trh vývojářská komunita vydala vlastní ovladače pro propojení MS Kinect s počítačem, což vedlo k vydání oficiálních ovladačů od Microsoftu v červnu 2011.

Nejdůležitější součástí Kinectu jsou infračervený projektor a senzor, díky nimž dokáže zařízení poměrně přesně určit polohu jakéhokoliv bodu v zorném poli kamer, a algoritmus pro detekování lidského těla, díky němuž je zařízení absolutně „hands-free“. Lze tedy ovládat hry nebo aplikace bez potřeby držení jakéhokoliv snímače.

2.1 Využití

I přes to že byl Kinect původně doplněk pro herní zařízení, ukazuje se, že jeho využití bude mnohem větší. Díky příznivé ceně začíná být Kinect využíván v různých projektech. Možná využití by mohla být například překlad znakové řeči, různá ovládání elektronických přístrojů jako jsou televize, projektory nebo roboty, které by mohly být ovládány pouze pohybem lidí nebo (v případě robotů) rozpoznáváním překážek. Některé již realizované projekty budou popsány v této kapitole.

Firma CHALEARN vypsala soutěž pro toho, kdo vymyslí a zrealizuje nejlepší algoritmus, který by byl schopen se naučit gesta a poté je rozpoznat. Učení pohybů by se měla být aplikace schopna naučit pouze po jednom předvedení a tak si vytvořit databázi gest, které umí rozpoznat.[1]

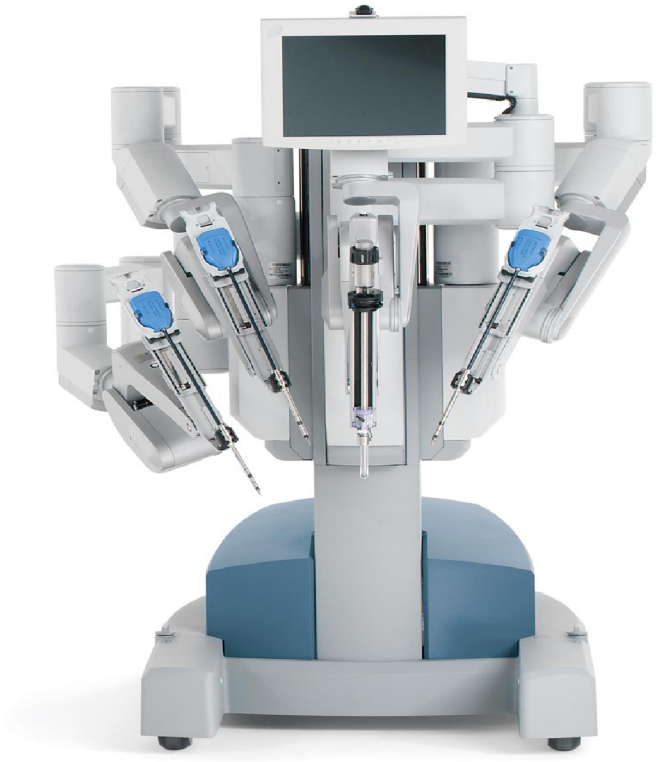
2.1.1 Rozpoznávání kancelářských a domácích objektů

Je to algoritmus, který se učí detekovat různé kancelářské a domácí objekty, které se učí rozpoznat studováním obrazů. Ty jsou označeny klíčovými slovy. Algoritmus poté bere v úvahu i relativní rozmístění objektů a dokáže rozlišit objekty s úspěšností 84 % v případě kancelářských prostor a 74 % v případě domácích. Při umístění zařízeni na robota a správnému naprogramování, by mohl algoritmus řídit roboty bez nutnosti znát prostředí.

Toto řešení vzniklo na Cornell University (Ithaka, NY).[2]

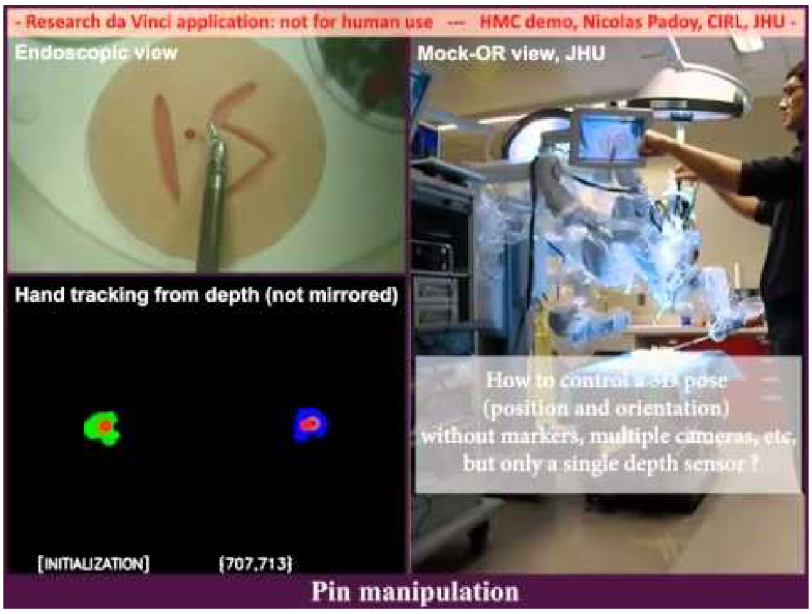
2.1.2 Lékařské využití

Některé projekty ukazují, jak by mohl být MS Kinect využit i v lékařské praxi. Jedním z takových projektů je řešení, které vzniklo na John Hopkins University (Baltimore, MD). MS Kinect umístili na robot „da Vinci“ (viz Obr. 1), který je významným nástrojem pro operace v dutině břišní, hrudní a ústní. Tento robot navazuje na laparoskopické a torakoskopické úkony, všeobecně uznávané jako minimálně invazivní chirurgie. Jedná se o metodu nazývanou Roboticky asistovaná chirurgie a je využíván především jako možnost léčení nádorových onemocnění v oblastech jako jsou gynekologie, urologie, hrudní chirurgie nebo kardiochirurgie.[3]



Obr. 1: Roboticky asistovaná chirurgie – robot „da Vinci“ [3]

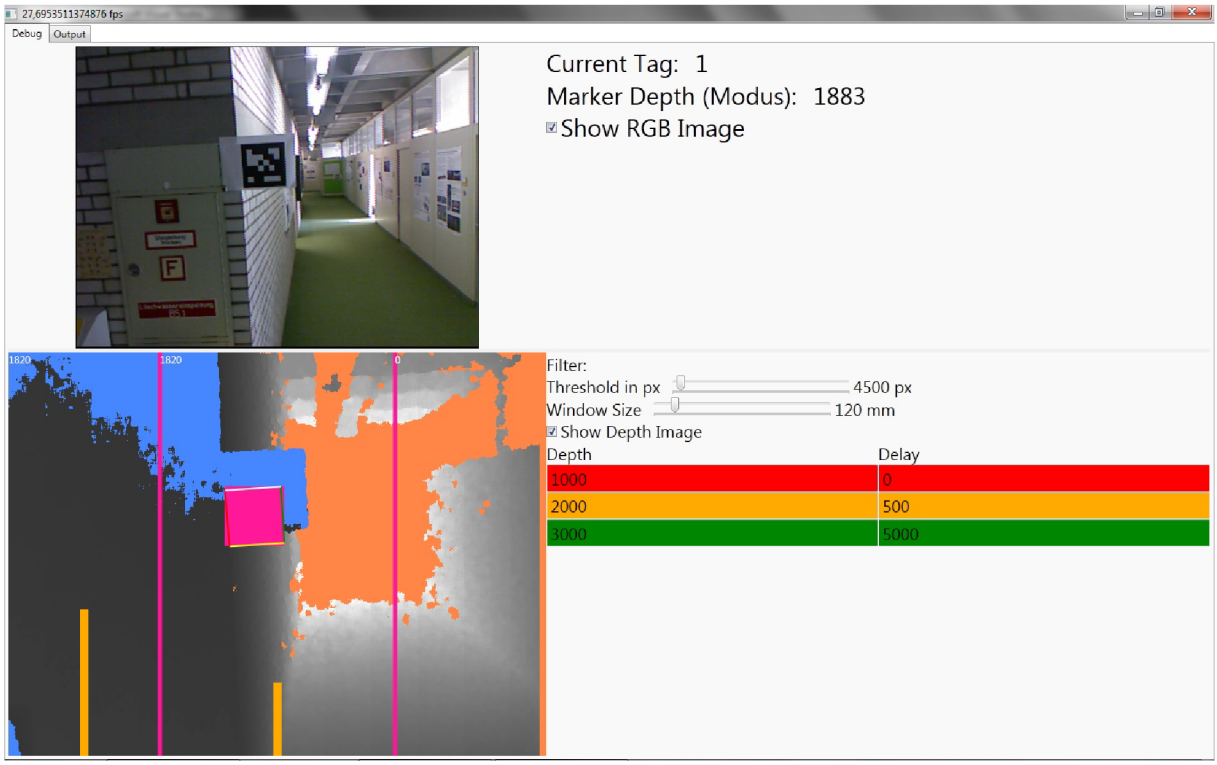
Projekt se zabývá způsobem jak jednoduše a efektivně využít MS Kinect k ovládání robotu „da Vinci“. Ovládání je realizováno pomocí hloubkových kamer a zaznamenávání pohybu pomocí MS Kinect (viz Obr. 2). Testy, které proběhli zatím nejsou praktikovatelné na lidech. Ovládání ještě není naprosto přesné, ale doufejme, že se tak časem stane. Chirurg by poté při operaci nemusel ovládat robot pomocí joysticku, ale pomocí svých vlastních pohybů. [4]



Obr. 2: Ovládání robotu "da Vinci" pomocí MS Kinect. [4]

Další ukázkou z využití v lékařství je projekt studentů z University of Konstanz (GER), kde použili MS Kinect k pomoci zrakově postiženým lidem. Cílem projektu bylo zjednodušit orientaci zrakově postiženým lidem uvnitř budov pomocí vibračního pásu, který by reprodukoval rozvržení místnosti. Na pásu jsou umístěny tři vibrační motory a je využita i reprodukce lidské řeči.

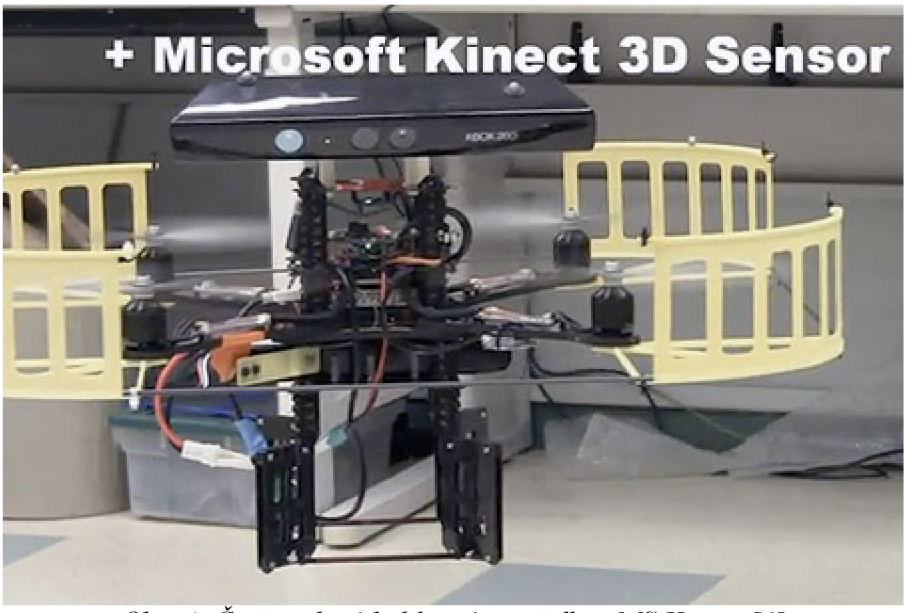
K orientaci jsou využity značky, které poté vykreslují cestu z jedné místnosti do druhé (viz Obr. 3). Značky mohou být umístěny např. na dveřích nebo rozích, které by mohli stát v cestě. Zařízení značku detekuje a pomocí hloubkové kamery zjistí vzdálenost člověka od takovéto značky. Celá navigace poté spočívá v reprodukci hlasu a vibrací na pásu, která osobě řekne nebo dá pocítit, kde se jaká překážka nachází.[5]



Obr. 3: Aplikace detekující značky.[5]

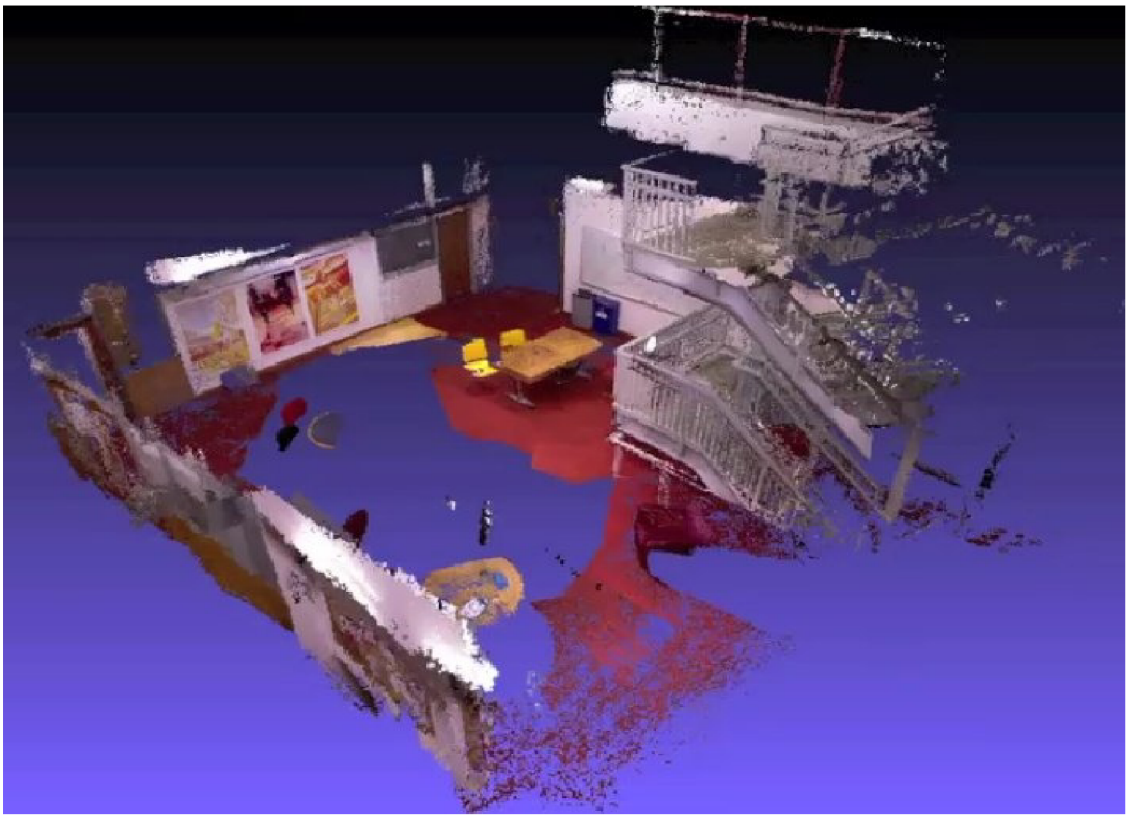
2.1.3 Čtyřvrtulová autonomní helikoptéra

Jde o projekt, kde je čidlo MS Kincet umístěno na čtyřvrtulovou helikoptéru (viz Obr. 4). Ta umožňuje jak manuální, tak plně autonomní mód a využívá hloubkovou mapu. MS Kinect je umístěn na vrchu helikoptéry tak, aby snímal zem a okolí a zajišťuje tak detekování překážek a pomocí detekce země i výšku helikoptéry. V plně autonomním módu helikoptéra prolétává předem danými body a při detekci překážky, která je blíže než jeden metr helikoptéra čeká než je překážka odstraněna mimo její dosah. Projekt vznikl na University of California (Berkeley, CA).[6]



Obr. 4: Čtyřvrtulová helikoptéra s čidlem MS Kinect.[6]

Podobný projekt řeší také na MIT (Boston, MA) ve spolupráci s University of Washintgon. Helikoptéra má plně autonomní mód a dokáže létat uvnitř budov, kde není možné využít výhod systému GPS. Helikoptéra posílá data počítači, který využívá data z RGB kamery a hloubkových čidel. Tím počítač ve spolupráci s helikoptérou mapuje celý vnitřní prostor ve 3D a vytváří tak model dané místnosti. Je schopen odhadnout vzdálenosti od různých překážek a posílá poté helikoptéře informace o její poloze a celou helikoptéru řídí. Na Obr. 5 je vidět 3D model naskenovaný helikoptérou.[7]



Obr. 5: 3D model vytvořený helikoptérou s čidlem MS Kinect.[7]

2.1.4 Autonomní čtyř kolový robot

Jedná se o auto v poměru 1:10 s Kinectem umístěným vepředu a řízené notebookem, který je umístěn v zadní části auta (viz Obr. 6). Notebook pomocí čidla Kinect řídí auto tak, aby nenarazilo do žádné překážky. Řízení není naprosto přesné, i přes to se dokáže auto bez problémů vyhnout překážkám, které detekuje. Toto řešení vzniklo na University of Bundeswehr Münich (GER). [8]

Obr. 6: Čtyřkolový robot s čidlem MS Kinect.[8]

2.1.5 RoboScan

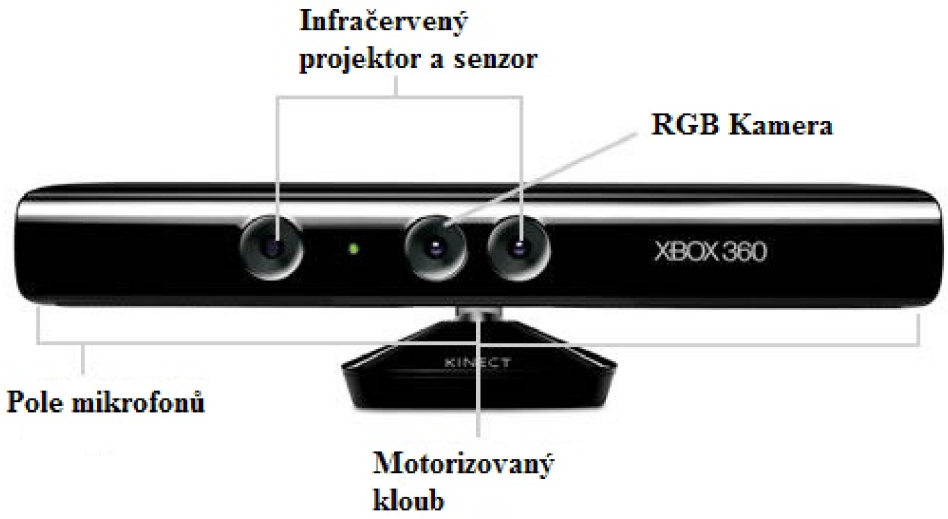
RoboScan je propojení robotické ruky ABB4400 s čidlem MS Kinect pro 3D skenování objektů, jak je ukázáno na Obr. 7. „Mračno bodů“ (Point cloud) je konstruováno v každé poloze robotické ruky. Koordinace bodů je počítána z polohy robotické ruky a vektoru „zápěstí“. Data jsou poté ukládána a zobrazována v reálném čase a vzniká tak 3D model. 3D model může představovat jak nasnímané okolí, tak samozřejmě nějaký objekt.[9]

Obr. 7: Robotická ruka ABB4400 s čidlem Microsoft Kinect a 3D model vytvořený po skenování [9]

2.2 Možnosti

V této kapitole budou popsány základní vlastnosti a možnosti zařízení Microsoft Kinect. Podrobnější vysvětlení a aplikace jednotlivých součástí využitých v této práci bude vysvětleno v kapitole 3.1.

Kinect je s počítačem propojen pomocí sběrnice USB a je napájen externě. Zařízení obsahuje RGB kameru, infračervený projektor a senzor, pole mikrofonů a motorizovaný kloub pro změnu úhlu natočení zařízení vzhledem k podložce (viz Obr. 8).



Obr. 8: Čidlo MS Kinect s popisem.[10]

Všechny součásti, kromě audia, jsou posílány do počítače pomocí *streamů*. Při inicializaci v aplikaci jsou rozpoznány *streamy*, které budou využity, a ty jsou poté otevřeny. U každého takového *streamu* musí být uvedeny doplňkové informace jako rozlišení, typ snímku a počet snímků za sekundu. Aplikace může mít až 4 takové *streamy*.

Zorný úhel	43° vertikálně, 57° horizontálně
Možnost natočení senzoru pomocí kloubu	±28°
Počet snímků za sekundu	30 fps (Frames per second)
Rozlišení hloubkové mapy	QVGA – 320 × 240
Rozlišení kamery	VGA - 640 × 480
Audio formát	16 kHz, 16 bitová mono pulzně kódová modulace (PCM)
Charakteristika vstupu audia	4 mikrofony s 24 bitovým AD převodníkem a zpracování signálu čidlem MS Kinect

Tabulka 1: Technické parametry čidla MS Kinect [11]

2.2.1 Kamera

Data jsou z Kinectu posílány buď ve formátu 32-bitového RGB, nebo 16-bitového komponentního modelu YUV. Model YUV je díky 16 bitům na jeden pixel méně náročný na paměť, oproti RGB ale dokáže snímat pouze 15 snímků za sekundu a pouze v rozlišení 640x480. Oba dva formáty jsou zpracovávány ve stejné kameře, proto reprezentují stejný obrázek.

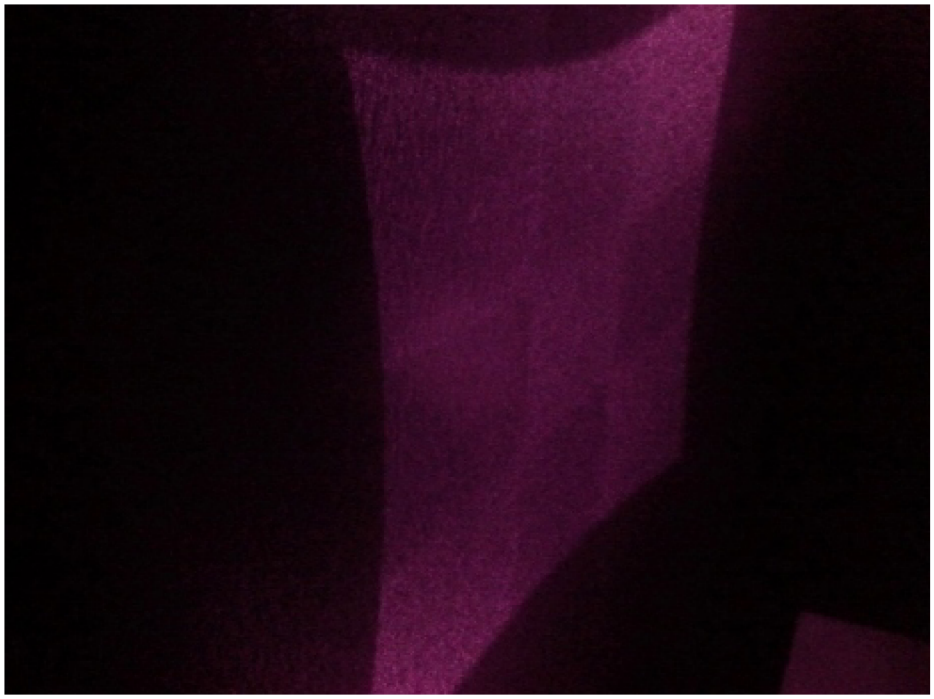
Jak už bylo řečeno, všechny senzory využívají k přenosu sběrnici USB. Toto připojení

poskytuje pouze určitou šířku pásma, proto je obrázek o rozlišení 1280x1024, který poskytuje senzor, vždy komprimován a teprve poté odeslán. Jakmile je obrázek přijat počítačem, je dekomprimován a poté odeslán aplikaci. Použitím komprese je dosaženo posílání 30 snímků za jednu sekundu na úkor kvality obrázku. Na Obr. 9 je ukázán výstup z kamery.[11]

Obr. 9: Ukázka výstupu z kamery.

2.2.2 Hloubková mapa

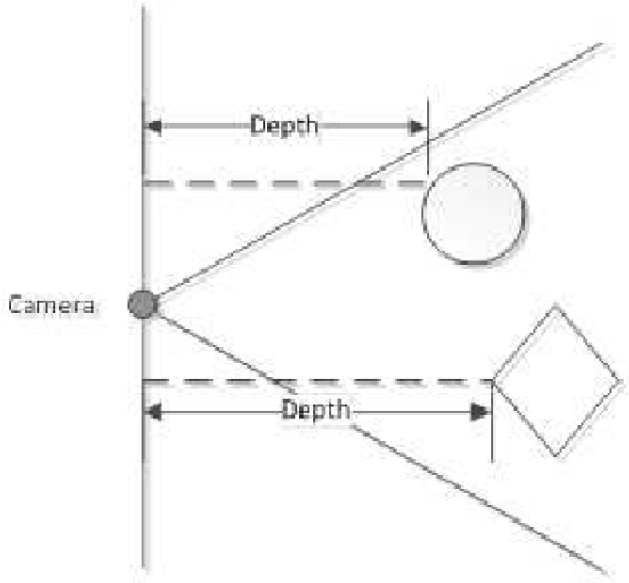
Hloubková mapa je společně s možností rozpoznávat lidskou kostru nejdůležitější součástí Kinectu. Skládá se z infračerveného laserového vysílače a senzoru. Kinect poté pomocí optické triangulace počítá vzdálenost objektu od Kinectu. Na Obr. 10 je vidět infračervené pole, které vyzařuje MS Kinect.



Obr. 10: Infračervené pole, které vyzařuje MS Kinect.

Kinect poskytuje snímek, kde každý pixel představuje vzdálenost od roviny kamery (viz Obr. 11). Na každý pixel se posílají vždy dva byty, uspořádání bitů uvnitř poté záleží na tom, jestli je při inicializaci vybrána pouze hloubková mapa nebo hloubková mapa s detekcí postav. V prvním případě je na vzdálenost vyhrazeno 12 bitů a zbytek zůstává nevyužit. V případě druhém je na vzdálenost vyhrazeno 13 bitů a zbylé tři bity jsou vyhrazeny na index detekovaných postav..

Kinect dokáže detekovat maximálně dvě postavy. Pro každou postavu jsou pak vytvořeny ve snímku segmenty jeho těla, protože pouze pixely, kde je postava detekována, jsou označeny jejím indexem. I přes to, že data pro jednotlivé postavy jsou separovaný snímek, se data hloubkové mapy a postav prakticky spojují do jediného snímku. [11]



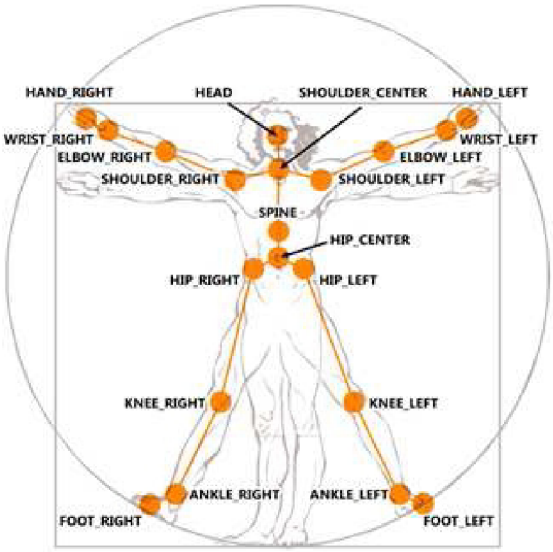
Obr. 11: Vzdálenost objektu od čidla MS Kinect.[11]

Teoreticky změřitelná vzdálenost je 0 – 4095 milimetrů při 12 bitech na pixel

a 0 – 8191 milimetrů při 13 bitech na pixel. Prakticky Kinect neumí měřit vzdálenost bodu, který je blíže než 800 milimetrů.[11]

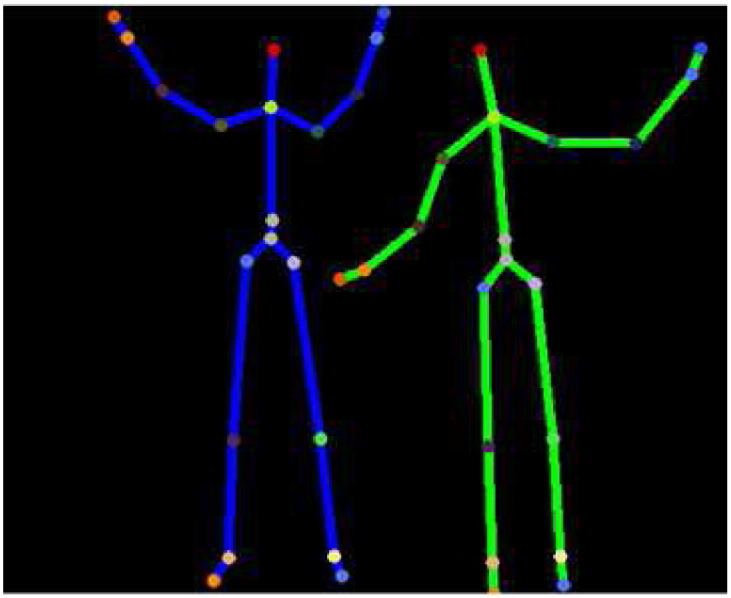
2.2.3 Rozpoznání „kostry“

Od rozpoznání hráče, které bylo uvedeno v kapitole 2.2.2 se výrazně liší. MS Kinect rozpoznává dvacet částí lidského těla (viz Obr. 13), ze kterých lze poté vytvořit jakousi „kostru“ člověka stojícího před Kinectem.



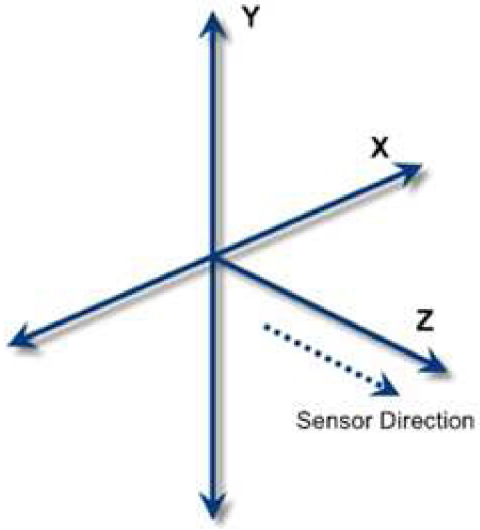
Obr. 13: Body lidského těla, které detekuje čidlo MS Kinect.[11]

O každé postavě jsou poskytnuty informace jako sada bodů. Každý bod je dán vektorem, který určuje přesnou polohu a orientaci a reprezentují aktuální polohu, nebo-li v celkovém měřítku pózu, kterou zaujímá člověk stojící před Kinectem. Ten dokáže detekovat až šest hráčů, z toho o prvních dvou hráčích, které systém najde, jsou poskytnuty kompletní informace (viz Obr. 12) a o zbylých čtyřech jsou poskytnuty pouze informace o těžišti. Každý hráč má garantováno ponechání jeho *ID* dokud neopustí zorné pole. Když hráč ze zorného pole odejde, buď je jeho *ID* nahrazeno novým hráčem, nebo je po vrácení se nazpět detekován dále se stejným *ID*.[11]



Obr. 12: Vykreslování detekované „kostry“ do obrázku.[12]

Každý bod je reprezentován vektorem *(x, y, z, w)*. Na rozdíl od hloubkové mapy, jsou osy orientovány jinak a vzdálenost je dána v metrech. Střed souřadného systému je v infračerveném senzoru (viz Obr. 14) . Celý systém je pravotočivý a závisí na umístění snímače (osa *y* nemusí být rovnoběžná s vektorem gravitace), může se tedy stát, že postavy stojící před snímačem budou na obrázku nakloněny.[11]



Obr. 14: Orientace os při rozpoznávání „kostry“.[11]

2.2.4 Mikrofonové pole

Pole mikrofonů je většinou tvořeno čtyřmi mikrofony, mezi kterými je několik centimetrů místa a jsou umístěny do přímky nebo do tvaru písmene „L“. Takovéto pole nezávislých mikrofonů má oproti jednotlivému několik výhod:

- Zlepšuje kvalitu zvuku. Implementuje se efektivnější algoritmus na ořezávání šumu a ozvěn než je možné u jediného mikrofonu
- Lokalizace zvuku a Beamforming. Při využití faktu, že při jednoduchém zdroji zvuku dorazí do každého mikrofonu zvuk v jiném čase, beamforming umožní aplikacím zjistit směr, odkud zvuk přichází.

Kinect obsahuje čtyřprvkové lineární mikrofonové pole, které využívá 24 bitový AD převodník a poskytuje místní zpracování signálu, včetně ořezání šumu a ozvěn. Pomocí Kinectu lze nahrávat zvuk ve vysoké kvalitě, rozpoznat směr zdroje zvuku a rozpoznávání řeči.[11]

Rozpoznávání řeči není implementováno přímo v Kinectu nebo ovladačích (ať už se jedná o oficiální nebo ne). Protože v práci bylo využito oficiální SDK, je možné použít pro rozpoznání knihovnu od firmy Microsoft.

2.3 Ovladače

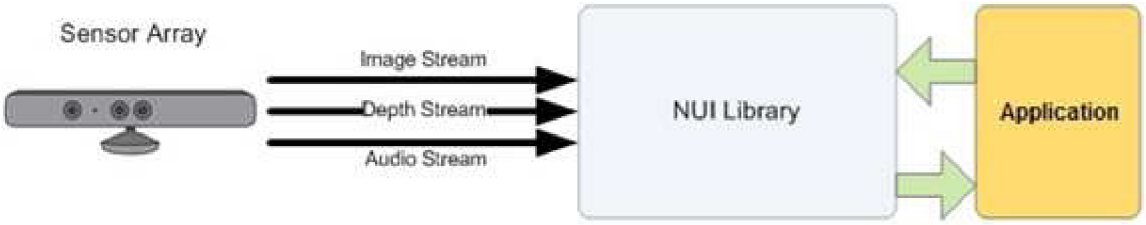
I když byl původní záměr firmy Mirosoft využít Kinect výhradně jako doplňkové zařízení pro herní konzoli XBox 360, vývojáři vydali své vlastní volné ovladače pro propojení čidla s počítačem. příklad takových volných ovladačů je Libfreenect nebo OpenNI – PrimeSense. To vedlo firmu Microsoft k vydání vlastních, oficiálních ovladačů a v červnu 2011 byly vydáno první SDK v1.0 Beta. Postupem času bylo SDK (Sowtware Development Kit) vyvíjeno a v únoru 2012 byly vydány Kinect for Windows SDK.

2.3.1 Microsoft Kinect SDK v1.0 Beta 2

Po stažení SDK (Software Development Kit) obsahuje stažený balíček ovladače pro propojení čidla MS Kinect s počítačem, rozhraní pro komunikaci mezi zařízením a aplikací a ukázky kódů. Po nainstalování SDK se Kinect chová jako multikomponentní zařízení USB. Pro fungování funkcí, které využívají audio je nutné stáhnout přídavné ovladače, které propojí mikrofony s počítačem. Je nutné podotknout, že i při existenci audio ovladačů pro 64-bitovou architekturu nefungují tyto ovladače pro rozpoznávání řeči správně, a proto je doporučeno stahovat ovladače pro 32-bitovou.

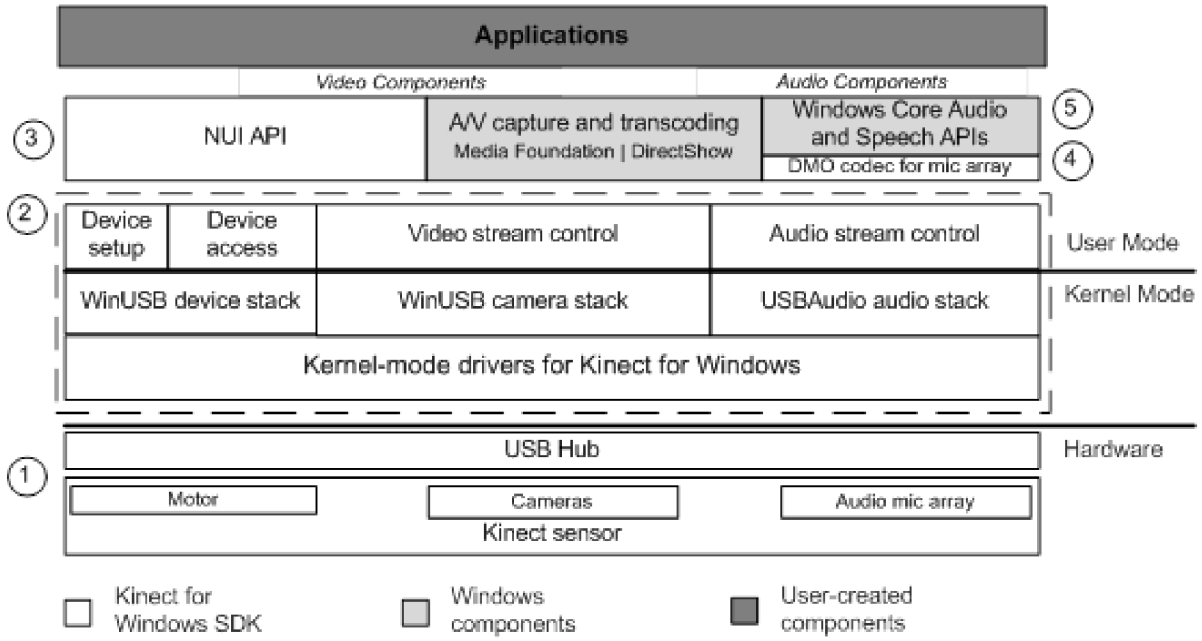
Hardwarové nároky:

- procesor Dual-core, 2,66 GHz nebo rychlejší
- USB 2.0
- 2 GB RAM
- Grafická karta kompatibilní s Windows 7, která podporuje DirectX 9.0c



Obr. 15: Schématické propojení aplikace s čidlem MS Kinect.[11]

Oficiální SDK podporuje programování v jazycích Visual Basic, C++ a C#. Protože bylo využito jazyka C# bude popsán způsob implementace knihoven právě v tomto jazyce v prostředí Visual Studio. Na Obr. 15 je ukázáno schématické propojení aplikace s čidlem MS Kinect a na Obr. 16 jednotlivé součásti SDK.[11]



Obr. 16: Součásti SDK.[11]

3 REALIZACE

Po prostudování možností čidla MS Kinect, jsem přistoupil na samotnou realizaci, která se týkala hlavně propojení Kinectu s počítačem, navržení a zrealizování gest a následné otestování na robotu. Gesta jsou řešena převážně pomocí rozpoznávání „kostry“ a jsou ukázány různé přístupy, od pohybových až po statické pózy.

Nejdříve jsem propojil všechny části Kinectu s aplikací a s jeho pomocí jsem poté realizoval mnou navržená gesta různými způsoby. Způsoby zadávání gest jsou čtyři, i když jeden ze způsobů není vlastně předávání gest, ale přímé posílání rychlosti motorům robotu a tudíž není tak univerzální jako zbylé tři.

Pro propojení Kinectu s počítačem jsem použil oficiální ovladače Microsoft SDK v1.0 Beta 2 a jako programovací prostředí jsem použil Microsoft Visual Studio C# 2010 Express. Celá aplikace je naprogramována v jazyce C#, proto se veškerá syntaxe a části kódu budou týkat výhradně jazyka C#.

3.1 Propojení Kinectu s počítačem

Po nainstalování SDK je nutné v programu Visual Studio přidat knihovnu *Microsoft.Research*. Po přidání této knihovny do vytvořeného projektu je potřeba tyto knihovny využít a propojit s aplikaci viz Obr. 17. Protože bude využito jak obrazových součástí MS Kinect, tak i rozpoznání řeči, jsou do aplikace vloženy obě dostupné knihovny.

zdrojový kód

```
1     using Microsoft.Research.Kinect.Audio;
2     using Microsoft.Research.Kinect.Nui;
```

Obr. 17: *Využití knihoven.*

Aby bylo možné využívat služby (kameru, hloubkovou mapu, rozpoznání kostry a zachycení zvuku), je potřeba si nadeklarovat proměnnou typu *Runtime*, která využije přidané knihovny a spojí MS Kinect s počítačem. Mělo by následovat ověření počtu Kinectů (SDK v1.0 Beta podporuje dva připojené Kinecty zároveň). Protože byla aplikace vytvářena výhradně pro jeden Kinect je ověřena pouze přítomnost a následné naplnění námi deklarované proměnné (v mém případě proměnné *nui*) viz Obr. 18.

zdrojový kód

```
1     if (Runtime.Kinects.Count > 0)
2     {
3           nui = Runtime.Kinects[0];
4     }
```

Obr. 18: Připojení čidla MS Kinect.

Po připojení Kinectu a propojení aplikace s datovým tokem je možno využívat jednotlivé služby, které Kinect poskytuje. Nejdříve je nutno inicializovat všechny služby, které budou v aplikaci využity (viz Obr. 19). Protože má audio, které je možné v Kinectu

zdrojový kód

```
1     nui.Initialize(RuntimeOptions.UseColor |
2           RuntimeOptions.UseSkeletalTracking |
3           RuntimeOptions.UseDepth);
```

Obr. 19: Inicializace služeb poskytovaných čidlem MS Kinect.

využít v jiné knihovně a má deklarovanou jinou proměnnou (viz kapitola 3.2.4).

Kamera a hloubková mapa jsou poskytovány pomocí tzv. *streamů*, které musí být příslušně inicializovány a otevřeny (včetně doplňkových informací jako je rozlišení apod.) (viz Obr. 20). Do těchto *streamů* jsou poté nahrávány data, resp. snímky (v případě hloubkové mapy vzdálenost jednotlivých pixelů v zorném poli, kamery snímek a rozpoznání kostry vektory detekovaných bodů lidského těla), které Kinect posílá.

zdrojový kód

```
1     nui.VideoStream.Open(ImageStreamType.Video, 2,
            ImageResolution.Resolution640x480,ImageType.Color);
2     nui.DepthStream.Open(ImageStreamType.Depth,2,
            ImageResolution.Resolution320x240,ImageType.Depth);
```

Obr. 20: Otevření jednotlivých streamů.

Jednotlivé služby jsou obsluhovány pomocí delegátů a událostí, tzv. *EventHandlery* (viz Obr. 21). Vždy když je snímek připraven, provede se část kódu, která je napsána v implementované metodě k dané události. Je potřeba dbát na rychlost řídících struktur (prováděného kódu), protože při rychlosti 30 snímků za sekundu, musí být počítač schopen provést, všechny napsané příkazy.

zdrojový kód

```
1     nui.DepthFrameReady += new
            EventHandler<ImageFrameReadyEventArgs>(nui_DepthFrameReady);
2     nui.SkeletonFrameReady += new
            EventHandler<SkeletonFrameReadyEventArgs>(nui_SkeletonFrameReady);
3     nui.VideoFrameReady += new
            EventHandler<ImageFrameReadyEventArgs>(nui_VideoFrameReady);
```

Obr. 21: EventHandlery a jejich využití.

3.1.1 Kamera

Jak je uvedeno v kapitole 2.2.1, Kinect posílá obrázek o námi zvoleném rozlišení ve formátu 32-bitového RGB. Nejjednodušší vykreslení obrázku do aplikace je pomocí prvku *PictureBox*. Kinect ale předává data jako *Planar Image [13]*. *Planar Image* je vlastně rovinný obrázek, který v sobě nese jaká je šířka a výška obrázku, kolik se využívá bitů na jeden pixel a samotné bity. Tento obrázek je potřeba převést na *Bitmapu*, protože to je jediný formát, se kterým daný prvek (*PictureBox*) umí pracovat. Tento převod řeší jednoduchá metoda za pomoci vnitřních funkcí Visual Studia. Jediné, co je nutné zvolit, je barevný formát každého pixelu. V případě kamery zvolíme *32bppRgb*, což znamená, že na jeden pixel se využije 32 bitů a barvy budou ve formátu RGB (viz Obr. 22).

Vždy když je snímek připraven, se zavolá tato metoda a obrázek se převede do námi požadovaného formátu. V aplikaci se nakonec nečeká pouze na snímek z kamery, ale čeká se ještě na rozpoznávání kostry, protože detekovaná „kostra“ je vykreslována přímo do tohoto obrázku.

zdrojový kód

```
1     public Bitmap PlanarToBMap(PlanarImage image)
2     {
3           Bitmap bmap = new Bitmap(image.Width, image.Height,
                  PixelFormat.Format32bppRgb);
4           BitmapData bmapdata = bmap.LockBits(new Rectangle(0, 0, image.Width,
                  image.Height), ImageLockMode.WriteOnly, bmap.PixelFormat);
5           IntPtr pointer;
6           pointer = bmapdata.Scan0;
7           Marshal.Copy(image.Bits, 0, pointer, image.Width * image.Height *
                  image.BytesPerPixel);
8           bmap.UnlockBits(bmapdata);
9           return bmap;
10    }
```

Obr. 22: Převod PlanarImage na Bitmapu pro zobrazení obrazu kamery.[14]

3.1.2 Hloubkové čidlo

Stejně jako v předchozí kapitole i tady je potřeba převést snímek, který posílá Kinect, do formátu, se kterým pracuje *PictureBox*. Protože se posílá pouze 13 bitů, není možné využít tu samou metodu jako v případě kamery. Je potřeba zvolit jiný formát pixelů, než u kamery. I když by bylo vhodné využít formát v odstínech šedi, bohužel typ aplikace, ve které je program napsán (*WindowsForm*), tento formát nepodporuje, a proto je nutné vybrat nějaký barevný. Protože je na každý pixel posíláno jen 12 bitů (není využito indexování detekovaných postav) je nejbližší formát *16bppRgb555* (16 bitů na jeden pixel a na každou barvu 5 bitů, tzn. že jeden bit zůstane nevyužit). Při využití tohoto formátu by ale nebyla téměř využita červená barva, proto je potřeba pro každý pixel bity posunout (viz Obr. 23). Na Obr. 25 je vidět rozdíl barev mezi klasickým zobrazením a zobrazením posunutým o dva bity.

zdrojový kód

```
1     public Bitmap MAPA(PlanarImage Image)
2     {
3           int temp = 0;
4           for (int i = 0; i < Image.Bits.Length; i += 2)
5           {
6                 temp = (Image.Bits[i + 1] << 8 | Image.Bits[i]);
7                 temp <<= 2;
8                 Image.Bits[i] = (byte)(temp & 0xFF);
9                 Image.Bits[i + 1] = (byte)(temp >> 8);
10          }
11          Bitmap bmap = new Bitmap(Image.Width, Image.Height,
12          PixelFormat.Format16bppRgb555);
13          BitmapData bmapdata = bmap.LockBits(new Rectangle(0, 0, Image.Width,
                        Image.Height), ImageLockMode.WriteOnly,
                        bmap.PixelFormat);
14          IntPtr pointer = bmapdata.Scan0;
15          Marshal.Copy(Image.Bits, 0, pointer, Image.Width *
                        Image.BytesPerPixel *Image.Height);
16          bmap.UnlockBits(bmapdata);
17          return bmap;
18    }
```

Obr. 23: Převod PlanarImage na Bitmapu pro zobrazení hloubkové mapy.[15]

Pro každý pixel je možnost získat vzdálenost, ve které se nachází od Kinectu. Výpočet vzdálenosti je vlastně jednoduché sečtení bitů, které patří danému pixelu. Aby bylo možné takto sčítat bity, musíme nejdříve vědět, které jsou ty správné. Indexy daných bytů jsou získávány podle vzorců (1) a (2)

$$Byte1[x*(\text{počet bitů na pixel})+y*width*(\text{počet bitů na pixel})] \quad (1)$$

$$Byte2[x*(\text{počet bitů na pixel})+y*width*(\text{počet bitů na pixel})+1] \quad (2)$$

Po nastavení se na byte podle vzorce (2) provede součet jednotlivých bitů a tím se získá vzdálenost v milimetrech (viz Obr. 24).

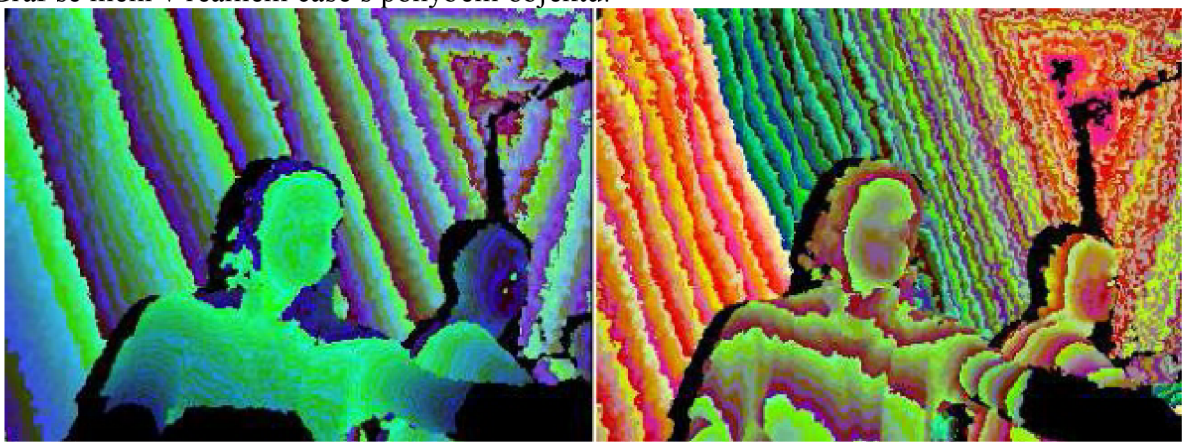
zdrojový kód

```
1     public int vzdalenost(PlanarImage Image, int x, int y)
2     {
3           int d = Image.Bits[x * Image.BytesPerPixel + y * Image.Width *
                  Image.BytesPerPixel + 1];
4           d <<= 8;
5           d += Image.Bits[x * Image.BytesPerPixel + y * Image.Width *
                  Image.BytesPerPixel];
6           return d;
7     }
```

Obr. 24: Algoritmus na výpočet vzdálenosti daného bodu.[15]

Protože není využito možnosti detekovat postavy v hloubkové mapě (tzn. přidělit pixelům detekovaných postav indexy), není potřeba hlídat kolik se přesně sčítá bitů, protože zbylé čtyři bity budou nastaveny na nulu. Kdyby této možnosti bylo využito, nastal by v případě detekce problém s výpočtem vzdálenosti těch bodů, které by měli přiřazeny index postavy, protože by do tohoto výsledku bylo zahrnuto i toto číslo.

V aplikaci je také histogram rozložení pixelů. Histogram je dělán pomocí prvku *Chart* (sloupcový graf) a je to vlastně počet pixelů v dané vzdálenosti. Výpočet je jednoduchým průchodem pole a načítání dané vzdálenosti. Vše je uloženo v poli a poté vykresleno do grafu. Graf se mění v reálném čase s pohybem objektů.



Obr. 25: Rozdíl mezi klasickým zobrazením a zobrazením posunutým o dva bity.

3.1.3 Rozpoznání „kostry“

Rozpoznání „kostry“ je nejdůležitější služba pro detekci pohybů lidského těla, kterou Kinect poskytuje, a byla nejvíce využita v práci. Protože jsou gesta řešena pomocí pohybů nebo polohy jednotlivých částí těla, jedná se o stěžejní službu. Z Kinectu jsou posílána data

o jednotlivých bodech ve formě vektoru (*x, y, z, w*). Ve vektoru jsou vždy souřadnice bodu uvedeny v metrech. Kdybychom chtěli kreslit „kostru“ do samostatného obrázku, mohli bychom rovnou tyto body po přepočtu do os, se kterými pracuje bitmapa, nakreslit. Problém nastává, když chceme „kostru“ nakreslit přímo do obrázku, ve kterém je převeden snímek z kamery. Protože jsou hloubková čidla umístěny vedle RGB kamery, je celá „kostra“ mírně posunuta (viz Obr. 27). Tento problém je řešen pomocí metody, která je přímo v nainstalovaném SDK, i když je potřeba data ještě upravit, než tuto metodu využijeme (viz Obr. 26).

zdrojový kód

```
1     private Point GetPoint(JointID ID, SkeletonData skelDat)
2     {
3           Vector pozice = skelDat.Joints[ID].Position;
4           float xd, yd;
5           nui.SkeletonEngine.SkeletonToDepthImage(pozice, out xd, out yd);
6           xd = Math.Max(0, Math.Min(xd * 320, 320));
7           yd = Math.Max(0, Math.Min(yd * 240, 240));
8           int x, y;
9           ImageViewArea iv = new ImageViewArea();
10          nui.NuiCamera.GetColorPixelCoordinatesFromDepthPixel(ImageResolution
                .Resolution640x480, iv, (int)xd, (int)yd, (short)0, out x,
                out y);
11          return new Point(x, y);
12    }
```

Obr. 26: Přepočet pozice detekovaného bodu do os Bitmapy.44

Metoda vždy vrátí bod v bitmapě s daným rozlišením, kde se bod nachází. Vykreslení proběhne jako pospojování bodů, které se mají spojit. Bohužel zde není realizována nějaká rychlejší forma vykresleni, ale je potřeba postupně napsat všechny spojované body ručně. Vykreslovaní samotné „kostry“ není vlastně ani potřeba, vykreslení již nenese žádná data. Pro lepší přehlednost v aplikaci je ale lepší „kostru“ vykreslit.

Obr. 27: Ukázka detekce „kostry“ a její vykreslení do obrazu kamery

3.1.4 Rozpoznání řeči

Je nutné si uvědomit, že Kinect jako takový řeč rozpoznat neumí. Je potřeba mít doinstalované jak ovladače, které zachycují zvuk, tak knihovny a slovníky pro práci se zachyceným zvukem. Po nainstalování ovladačů je teprve možné propojit audio vstup Kinectu s počítačem (pozor na 32-bitovou architekturu jak je uvedeno v kapitole 2.2.4).

Dále je velký rozdíl při přidávání knihoven do prostředí Visual Studio. Jediná správně fungující knihovna je *Microsoft.Speech*. V možných knihovnách se objevuje i knihovna *System.Speech*, která však nefunguje správně. Po přidání správné knihovny ji využijeme (viz Obr. 28). Tyto dvě knihovny hrají klíčovou roli pro rozpoznání slov.

zdrojový kód

```
1     using Microsoft.Speech.Recognition;
2     using Microsoft.Speech.AudioFormat;
```

Obr. 28: Využití knihoven pro rozpoznání řeči

Nejdříve je potřeba připojit Kinect a hned při vytváření objektu, který pracuje s rozpoznáváním řeči, se vytvoří gramatika. Slova, které se budou rozpoznávat, se do nového objektu vkládají přes konstruktor jako pole řetězců. Poté je vyhodnocen výsledek pomoci *SpeechRecognitionEngine* (reprezentuje hlavní propojení mezi audio vstupem z čidla, rozpoznáním řeči a slovníkem). Když je výsledek pozitivní, tzn. detekoval nějaké slovo, které bylo zadáno v gramatice, s pravděpodobností větší jak 0,5, je výsledek vyhodnocen jako věrohodný a předá se aplikaci právě detekované slovo (viz Obr. 29). Důvěryhodnost v detekované slovo je záměrně volena poměrně malá, protože není předpokládáno zadávání gest rodilým mluvčím. Rozdíl ve výslovnosti rodilého a nerodilého mluvčího je velmi citelný, důvěra v daný výsledek někdy klesá i pod 0,5.

zdrojový kód

```
1     RecognitionResult vysledek = Engine.Recognize();
2     if (vysledek != null && OrderDetected != null && vysledek.Confidence > 0.5)
3     {
4           OrderDetected(vysledek.Text);
5     }
```

Obr. 29: Podmínka pro rozpoznání daného příkazu

3.2 Navržení a realizace gest

Po otevření jednotlivých *streamů* a naprogramování výstupu do aplikace jsem přistoupil k vlastnímu návrhu dálkového ovládání robotu. Dálkové ovládání je ve třech případech řešeno jako předávání gest při určité akci a v jednom případě je řešeno přímým přepočítáváním na rychlosti motorů.

V případě způsobů, které předávají gesta, jsou rychlosti dány „natvrdo“ do programu, aby v případě přeprogramovávání ovládání na jiného robota, bylo toto předělání jednodušší a průhlednější.

číslo gesta	akce
1	doprava
2	doleva
3	stop
4	dopředu
5	dozadu

Tabulka 2: Definovaná gesta.

3.2.1 Pohyb ruky

Obr. 30: Směry detekovaných pohybů

Je naprogramován jednoduchý algoritmus k zachycení pohybů. Algoritmus neumožňuje žádné učení ani složitější pohyby, ale pouze pohyby jak jsou ukázány na Obr. 30. Pro názornost je ukázáno jak se pracuje s body, které jsou spočítány z rozpoznávání „kostry“. Tyto body byly původně využity pro nakreslení kostry do *streamu* RGB kamery. Protože jsou body vlastně umístěním ve snímku, lze jednoduše detekovat pohyb. To samé by bylo možné pro data přímo z rozpoznání kostry“, ty jsou pro názornost ukázány v kapitole 3.2.3.

Detekce pohybu začíná, když levá nebo pravá ruka předpaží. Aplikace pozná, že má začít detekovat pozice předpažené ruky. Pohyb ruky je vyhodnocován každých deset snímků. Po uložení deseti pozic se vyhodnotí, zda ruka byla v pohybu, či nikoliv. Ruka se musí pohnout určitou rychlostí, aby aplikace zaznamenala, že se pohnula. Tento pohyb musí být delší jak pět snímků z deseti. Srovnávání probíhá vždy s předchozím snímkem tak, že se pozice v dané ose jednoduše odečtou. Když je rozdíl větší, resp. menší, jak pět pixelů, resp.

minus pět pixelů, realizuje se přičtení, resp. odečtení. Tento rozdíl je volen intuitivně a hlavně kvůli „šumu“, který vzniká i když se ruka nehýbe, protože není možné držet ruku v naprosto stejné pozici. Veškerý takový pohyb se zaznamenává do proměnné a podle hodnot v této proměnné se poté stanovuje, zda a kam se ruka pohnula. Když je hodnota v proměnné větší nebo menší jak pět, proběhl nějaký pohyb a vyvolá se příslušné gesto. Tato hodnota znamená, že v pěti případech byl rozdíl v pozicích větší jak pět pixelů a představuje jakousi důvěryhodnost v daný pohyb. Pohyb je poté prováděn do té doby, než je detekován pohyb jiný, vysunutá ruka musí zůstat stále vysunuta, jinak se gesto okamžitě přeruší.

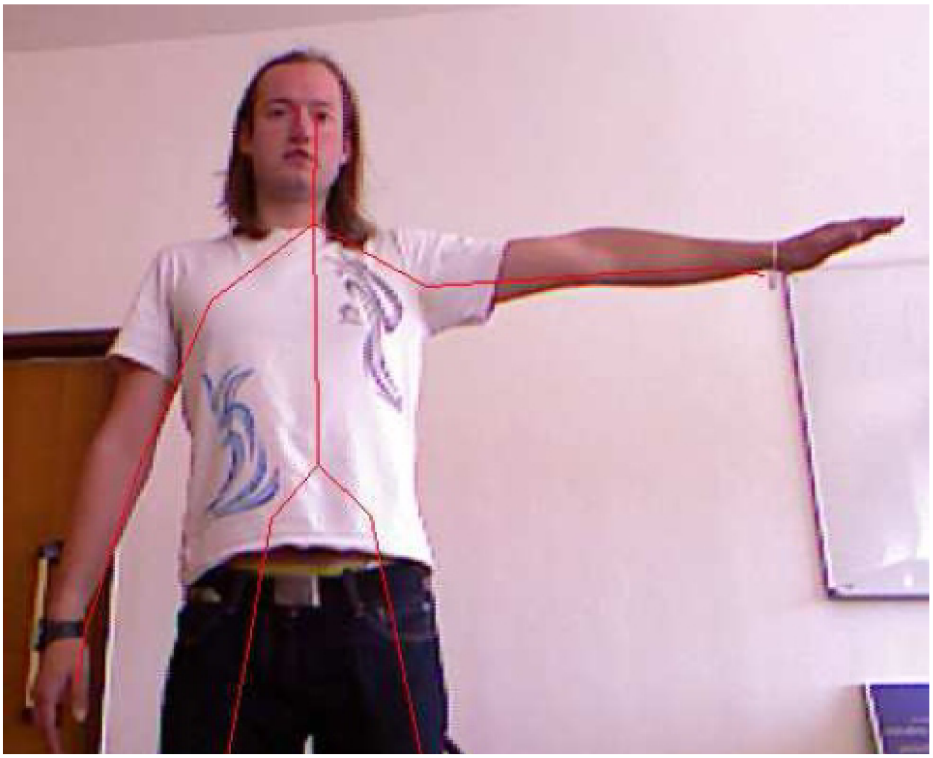
Nadeklarovaná proměnná *XYsmer* určuje kam se ruka pohnula. proměnná je deklarována jako dvourozměrné pole. Index 0 představuje pohyb v ose *x* a index 1 pohyb v ose *y*. Když se ruka pohne doprava do proměnné na index 0 se přičte jednička, když se pohne ruka doleva, jednička se odečte. Podobně pak v ose *y* (viz Obr. 31). Vyhodnocení, zda pohyb nastal nebo ne, poté proběhne v další metodě.

zdrojový kód

```
1     private int[] Kam(Point[] Body)
2     {
3           int[] XYsmer = new int[2]; ;
4           XYsmer[0] = 0;
5           XYsmer[1] = 0;
6           for (int i = 0; i < 9; i++)
7           {
8                 if ((body[i].X+5) < body[i + 1].X)
9                 {
10                      XYsmer[1]++;//doprava
11                }
12                else if ((body[i].X-5) > body[i + 1].X)
13                {
14                      XYsmer[1]--;//doleva
15                }
16                if ((body[i].Y + 3) < body[i + 1].Y)
17                {
18                      XYsmer[0]++;//dolu
19                }
20                else if ((body[i].Y - 3) > body[i + 1].Y)
21                {
22                      XYsmer[0]--;//nahoru
23                }
24          }
25    }
```

Obr. 31: Algoritmus pro detekci pohybu.

3.2.2 Statická póza



Obr. 32: Ukázka statické pózy.

Statická póza je nejjednodušší případ, který může nastat, např. jako na Obr. 32. Díky datům z rozpoznávání „kostry“ je možno rychle a jednoduše naprogramovat jakoukoliv pózu, kde každá bude znamenat určité gesto. V této možnosti ovládání se využívají data přímo z rozpoznávání kostry, protože je pravděpodobné, že bude využito mnoho bodů. Jak je vidět na Obr. 33, je pozice ruky vztažena k páteři. Hodnoty, o které musí být ruka upažena od těla, jsou voleny dle odzkoušení. Není zde předpoklad, že před Kinectem např. stojí člověk, který má ruku kratší než 0,5 metru.

póza	gesto	akce
upažení pravé ruky	1	doprava
upažení levé ruky	2	doleva
upažení	3	stop
vzpažení	4	dopředu

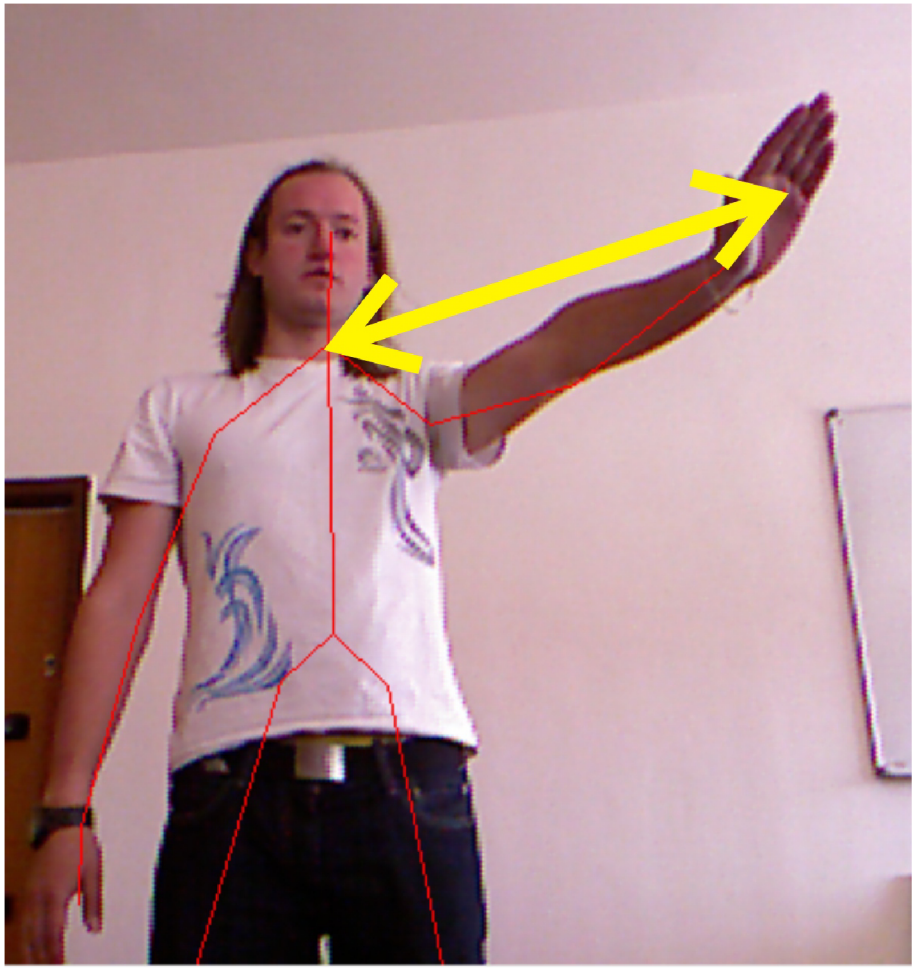
Tabulka 3: Definovaná statické pózy a gesta, která představují.

zdrojový kód

```
1     if ((data.Joints[JointID.HandLeft].Position.X <
                  data.Joints[JointID.Spine].Position.X- 0.5) &&
                  (data.Joints[JointID.HandRight].Position.X >
                  data.Joints[JointID.Spine].Position.X + 0.5))
2           { return 3;}
3     else if (data.Joints[JointID.HandRight].Position.X >
                  data.Joints[JointID.Spine].Position.X + 0.5)
4           { return 1; }
5     else if (data.Joints[JointID.HandLeft].Position.X <
                  data.Joints[JointID.Spine].Position.X - 0.5)
6           { return 2; }
```

Obr. 33: Ukázka definice statické pózy.

3.2.3 Poloha ruky od „středu“



Obr. 34: Poloha ruky od středu

Tento způsob ovládání nejsou vlastně klasická gesta, ale přepočet rychlosti od zvoleného středu. Zde je přepočítávána rychlost od spodní části krku, jak je ukázáno na Obr. 34. Přepočet rychlosti znamená, že čím je ruka více vzdálena od tohoto bodu, tím bude rychlost v daném směru větší. Je zde použito dat z rozpoznávání „kostry“, protože je opět zapotřebí několik bodů, které se srovnávají, a díky tomu je možný přepočet rychlosti.

Protože hodnoty z Kinectu přichází velice rychle a nikdo není schopen držet ruku stále na jednom místě, je ve výseku kolem středu pásmo, kde je rychlost stále nulová. V tomto výseku bychom možná ani nepostřehli nějaký větší pohyb robotu, a proto je vlastně zbytečné přepočítávat rychlost již zde. Podobný výsek je realizován také kolem osy *y*, a to z toho důvodu, že kdybychom se chtěli pohybovat pouze dopředu nebo dozadu, nebylo by to v zásadě možné. Tento výsek není nijak velký, ale pro potřeby pojíždění bez zatáčení naprosto stačí.

Přepočet je potom řešen zvlášť pro pojezd dopředu a dozadu, který je realizován v ose *y*, a zvlášť pro zatáčení. Do metod pro přepočet rychlosti je vkládán parametr *X*, resp. *Y*, což je již vzdálenost ruky od spodní části krku, a maximální hodnota rychlosti. Proměnná *speed* reprezentuje rychlost jednotlivých motorů (0. prvek – levý motor, 1. prvek – pravý motor), jak je vidět na Obr. 35.

zdrojový kód

```
1     private void convert_speed(float Y, int max)
2     {
3           speed[1] = (128 + ((max-128) / 300f * Y));
4           if (speed[1] > max)
5           {
6                 speed[1] = max;
7           }
8           else if (speed[1] < (128 -(max-128)))
9           {
10                speed[1] = 128 - (max - 128);
11          }
12          speed[0] = speed[1];
13    }

14    private void convert_turn(float X, int max)
15    {
16          float turn = (30 / 600f * X);
17          if (Xpom < 0)
18          {
19                if (speed[1] < (max - 30))
20                {
21                      speed[1] -= turn;
22                }
23                else { speed[0] += turn; }
24          }
25          else
26          {
27                if (speed[0] < (max - 30))
28                {
29                      speed[0] += turn;
30                }
31                else { speed[1] -= turn; }
32          }
33    }
```

Obr. 35: Algoritmus pro přepočet rychlosti v závislosti na vzdálenosti

3.2.4 Rozpoznání řeči

Tento způsob ovládání jsou opět klasická gesta, která jsou zadávána slovně. Protože Microsoft při rozpoznávání řeči nepodporuje český jazyk, jsou povely voleny v jazyce anglickém. Slova jsou zadávána v konstruktoru jako pole řetězců (viz Obr. 36). Tím je vytvořena „gramatika“ slov, které se mají detekovat.

zdrojový kód

```
1     ovladaniHlasem = new Hlas("right", "left", "back", "forward", "stop");
```

Obr. 36: Zadávání gramatiky

Po zadání gramatiky je potřeba spustit nahrávání audia vstupu z Kinectu. V tomto nahrávání je již realizováno samotné rozpoznání řeči, které je díky Kinectu a jeho vnitřnímu čištění zvuku poměrně spolehlivé. Je nutné požadované gesta vyslovit správně, protože v opačném případě systém buď slovo vůbec nerozpozná, nebo rozpozná slovo špatné a může

se stát , že je poté gesto interpretováno jako jiné. Důvěryhodnost v rozpoznané slova byla vysvětlena v kapitole 3.1.4.

V Tabulce 4 jsou ukázány anglické slova, které je potřeba zadat, aby byla provedeno gesto.

povel	**gesto**	**akce**
right	1	doprava
left	2	doleva
stop	3	stop
forward	4	dopředu
back	5	dozadu

Tabulka 4: Definované povely a jimi vyvolaná gesta.

4 OTESTOVÁNÍ NA ROBOTU

Po navržení a zrealizování gest jsem přistoupil k samotnému otestování funkčnosti na mobilním robotu. Otestování proběhlo na mobilním robotu „Leela“ (viz Obr. 37). „Leela“ je mobilní robot, který dokáže fungovat jak na manuální, tak autonomní mód (pomocí infračervených majáků a známosti prostředí). V robotu je holonomní diferenciální podvozek. Podvozek obsahu dvě nezávisle řízené nápravy. Do motorů je pomocí Bluetooth posílána rychlost, kterou se mají pohybovat. Hodnoty, které jsou do motorů posílány musí být v intervalu od 0 do 255. Hodnota 0 je nejrychlejší pohyb motoru vzad a 255 nejrychlejší pohyb motoru vpřed. Hodnota 128 je poté hodnota, při které se motory nepohybují vůbec.

Obr. 37: Mobilní robot „Leela“.

4.1 Komunikace

Pro komunikaci s robotem je využito bezdrátové technologie Bluetooth, protože je modul nainstalován v robotu. K využití Bluetooth v počítači bylo využito externího modulu, protože aplikace ve Windows 7, která využívá interního Bluetooth v počítači nesplňovala mé požadavky a není ji možné ze systému odinstalovat. Na spojení s robotem byl nainstalován

software BlueSoleil a poté proběhlo správné spárování s robotem.

Naprogramování komunikace v jazyce C# v prostředí Visual studio je velice jednoduché, protože zde existuje prvek *SerialPort*. Po spuštění aplikace a vybrání portu, který využívá externí modul, a kliknutí na tlačítko *Connect* je aplikace propojena s robotem (viz Obr. 38). Poté již je pouze aktivováno posíláni dat.

zdrojový kód

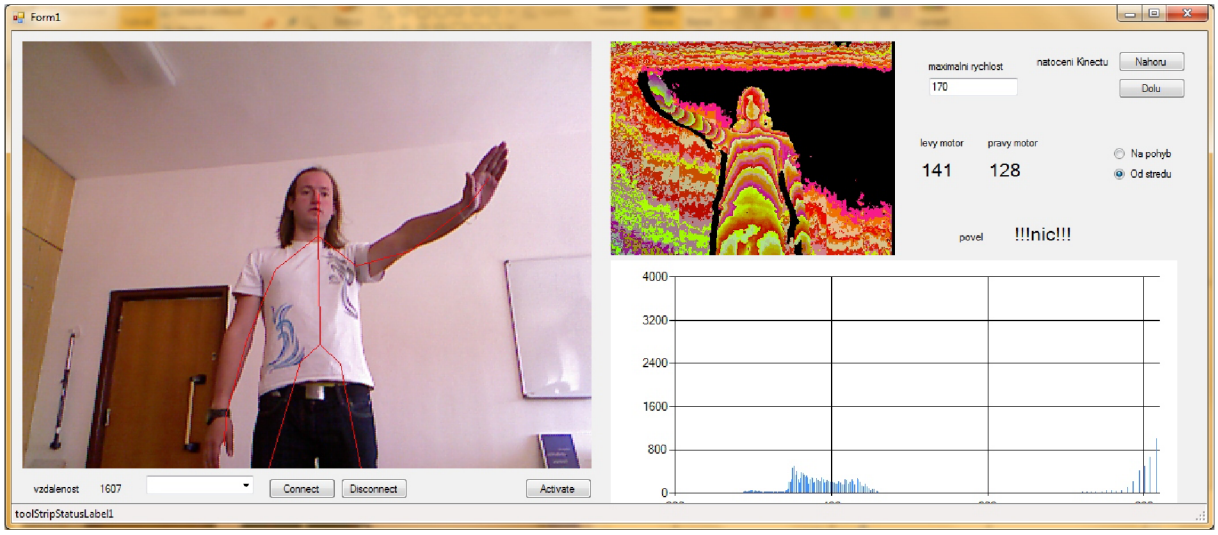
```
1     serialPort1.PortName = comboBox1.Text;
2     serialPort1.BaudRate = 57600;
3     serialPort1.StopBits = StopBits.One;
4     serialPort1.ReadBufferSize = 8;
5     serialPort1.Parity = Parity.None;
6     serialPort1.Handshake = Handshake.None;
7     serialPort1.Open();
```

Obr. 38: Otevření komunikace mezi počítačem a robotem.

4.2 Výsledná aplikace

Výsledná aplikace (viz Obr. 39) dokáže po dálkově ovládat robot pomocí bezdrátové komunikace Bluetooth a Kinectu. Po spojení robotu s aplikací je ho možné ovládat čtyřmi způsoby. všechny způsoby ovládání, které využívají gesta jsou dostupné zároveň, způsob přepočítávání rychlostí funguje pouze zároveň se zadáváním gest mluveným slovem. pro přepínání jsou umístěny v aplikaci tzv. *CheckBoxy*.

Aplikace při zadávání ukazuje, kam se robot po zadání bude pohybovat a jak rychle. V případě přepočítávání rychlosti je možnost zadat i maximální rychlost robotu. Hodnota maximální rychlosti se musí pohybovat v intervalu (*128,255*).



Obr. 39: Výsledná aplikace.

4.3 Otestování

Samotné otestování tedy proběhlo na mobilním robotu Leela. Byly otestovány všechny způsoby dálkového ovládání, které byly navrženy. Na Obr. 40 a 41 jsou ilustrační fotky ovládání robotu. Může se zdát, že zatáčení je znázorněno špatně, ale není tomu tak. Kamerový výstup je totiž zrcadlově, a proto není gesto prováděno levou rukou, jak se může zdát, ale pravou. Kdybychom si ke snímku přiložili zrcadlo, vše by odpovídalo skutečnosti, bohužel zde je levá strana doopravdy pravá a naopak.

Obr. 40: Pohyb robotu v závislosti na vzdálenosti ruky od spodní části krku

Obr. 41: Pohyb robotu v závislosti na pohybu ruky

5 ZÁVĚR

Hlavním cílem práce byl návrh, realizace a následné otestování dálkového ovládání mobilního robotu pomocí předem definovaných gest a čidla MS Kinect. Po propojení čidla s počítačem byly navrženy čtyři způsoby dálkového ovládání pomocí čidla MS Kinect. Dálkové ovládání je realizováno pohybem, statickou pózou, mluveným slovem a přímým přepočítáváním rychlosti motorů v závislosti na poloze ruky. Ovládání pohybem je založeno na jednoduché detekci pohybu. Pohyb lze provádět levou i pravou rukou, která je předpažená. Algoritmus poté vyhodnotí, zda byl detekován pohyb a v kladném případě vyvolá gesto, určené tímto pohybem. Ovládání statickou pózou je nejjednodušší druh dálkového ovládání, který je založen na určité poloze rukou vůči tělu. Ovládání mluveným slovem je realizováno pomocí čidla MS Kinect a externích knihoven firmy Microsoft včetně slovníku. Rozpoznávání hlasu je podporováno pouze v několika světových jazycích, český jazyk není podporován. Povely musí být vysloveny se správnou výslovností jinak nedojde k detekci slova nebo je vyvoláno jiné gesto. Čtvrtý způsob není klasické ovládání pomocí vyvolání určitých gest: Tento způsob je ukázkou využiti rozpoznání „kostry“ a přepočítávání rychlostí motorů v závislosti polohy ruky vůči spodní části krku. Toto dálkové ovládání není tak univerzální řešení jako předchozí tři, protože nejsou vyvolávána gesta, ale jsou posílány přímo rychlosti motorům robotu, tzn. nutnost znalosti konstrukce robotu.

Otestování dálkového ovládání proběhlo na mobilním robotu „Leela“. S robotem počítač komunikuje pomocí bezdrátové komunikace standardu Bluetooth. V robotu je holonomní diferenciální podvozek. Podvozek obsahu dvě nezávisle řízené nápravy. Zatáčení je poté realizováno rozdílnou rychlostí náprav. Všechny čtyři způsoby dálkového ovládání byly otestovány a jsou podrobně popsány v rámci této práci.

SEZNAM POUŽITÉ LITERATURY

[1]HOWARD. Only Kinect: More than a Gaming Gadget. In: *No Free Hunch* [online]. 7 December 2011 [cit. 2012-05-23]. Dostupné z: http://blog.kaggle.com/2011/12/07/only-kinect-more-than-a-gaming-gadget/

[2]KOŘENÁŘ, Patrik. Roboti se pomocí Kinectu učí poznávat svět. In: *TV-Expo.cz* [online]. 8. března 2012 [cit. 2012-05-23]. Dostupné z: http://www.tv-expo.cz/article/104-roboti-se-pomoci-kinectu-uci-poznavat-svet

[3]Co by měl praktický lékař vědět o robotické chirurgii. In: *Hospimed* [online]. 2009 [cit. 2012-05-23]. Dostupné z: http://www.hospimed.cz/wp-content/uploads/files/praktik_lekarska.pdf

[4]Gesture-based Surgical Manipulation of a da Vinci Robot using a Kinect. In: *RobotManiacs.com* [online]. 2011 [cit. 2012-05-23]. Dostupné z: http://robotmaniacs.com/2874/gesture-based-surgical-manipulation-of-a-da-vinci-robot-using-a-kinect/

[5]HEILIG, Mathias. NAVI – Navigational Aids for the Visually Impaired – A student project in the course Blended Interaction: University of Konstanz. In: *Human-Computer Interaction* [online]. Konstanz (GER), March 15, 2011 [cit. 2012-05-23]. Dostupné z: http://hci.uni-konstanz.de/blog/2011/03/15/navi/?lang=en

[6]World's First Autonomous Flying Xbox Kinect Quadrotor. In: *Techeblog* [online]. 12/09/2010 [cit. 2012-05-23]. Dostupné z: http://www.techeblog.com/index.php/tech-gadget/world-s-first-autonomous-flying-xbox-kinect-quadrotor

[7]Visual Odometry For GPS-Denied Flight And Mapping Using A Kinect. In: *Robust Robotics Group | CSAIL* [online]. 2011 [cit. 2012-05-23]. Dostupné z: http://groups.csail.mit.edu/rrg/index.php?n=Main.VisualOdometryForGPS-DeniedFlight

[8]Kinect essential for Autonomous Navigation of Mini Car. In: *KinectHacks.com* [online]. 02/17/2011 [cit. 2012-05-23]. Dostupné z: http://www.kinecthacks.com/kinect-essential-for-autonomous-navigation-of-mini-car/

[9]RoboScan. In: *Sy-lab* [online]. 2011 [cit. 2012-05-23]. Dostupné z: http://sy-lab.net/roboScan

[10]Resources for the classroom. In: *SummitsoftKids* [online]. 2012 [cit. 2012-05-23]. Dostupné z: http://www.summitsoftkids.com/resources.html

[11]MICROSOFT RESEARCH. *Kinect for Windows SDK Beta*: *Programming Guide*. June 16, 2011, 33 s. Dostupné z: http://202.120.53.119/wordpress/wp-content/uploads/2011/06/ProgrammingGuide_KinectSDK.pdf

[12]RAITEN, Shai. Kinect – Getting Started – Become The Incredible Hulk. In: *The Code Project* [online]. 18 Jun 2011 [cit. 2012-05-23]. Dostupné z: http://www.codeproject.com/Articles/213034/Kinect-Getting-Started-Become-The-Incredible-Hulk

[13]FERNANDEZ, Dan. Working with Depth Data (Beta 2 SDK). In: *Channel 9* [online]. Jun 16, 2011 [cit. 2012-05-23]. Dostupné z: http://channel9.msdn.com/Series/KinectSDKQuickstarts/Working-with-Depth-Data

[14]JAMES, MikeGetting started with Microsoft Kinect SDK. In: *I Programmer* [online]. 06 February 2012 [cit. 2012-05-23]. Dostupné z: http://www.i-programmer.info/programming/hardware/2623-getting-started-with-microsoft-kinect-sdk.html?start=1

[15]JAMES, Mike. Getting started with Microsoft Kinect SDK - Depth. In: *I Programmer* [online]. 06 February 2012 [cit. 2012-05-23]. Dostupné z: http://www.i-programmer.info/programming/hardware/2714-getting-started-with-microsoft-kinect-sdk-depth.html

[16]JAMES, Mike. Getting started with Microsoft Kinect SDK - Skeletons. In: *I Programmer* [online]. 06 February [cit. 2012-05-23]. Dostupné z: http://www.i-programmer.info/programming/hardware/3503-getting-started-with-microsoft-kinect-sdk-skeletons.html?start=2