Mendelova univerzita v Brně
Provozně ekonomická fakulta

Návrh a implementace softwarové komponenty pro manipulaci s 3D objekty

Diplomová práce

Vedoucí práce:
Ing. David Procházka, Ph.D.

Bc. Martin Čapek

Brno 2015

Děkuji své rodině, zejména mé mamince, za zázemí, které mi poskytla během celého studia na vysoké škole. Děkuji své přítelkyni, která mi byla oporou od vytvoření zadání této práce. Také děkuji vedoucímu mé diplomové práce, panu Ing. Davidu Procházkovi, Ph.D., za vedení, ochotný přístup, trpělivost a odbornou pomoc během psaní této práce. Nakonec děkuji všem, kteří se podíleli na testování výstupu této diplomové práce.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Návrh a implementace softwarové komponenty pro manipulaci s 3D objekty**

vypracoval samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou Směrnicí o zveřejňování vysokoškolských závěrečných prací. Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 22. května 2015 .....

Abstract

Čapek, Martin. Design and implementation of software component for manipulating 3D objects. Master thesis. Brno, 2015.

This master thesis focuses on the design and implementation of software component for manipulating 3D objects. Work includes a general description of hand, hand gestures aspects of distribution, basic image preprocessing methods, detailed description of the classification methods, description of Kinect device and libraries for image processing. The practical part deals with the design and implementation of software component named Tool into ArcScene application, design and implementation of gestures that emulate a 3D mouse functionality. The component is implemented in C# programming language with Kinect SDK library. This gesture recognition system is based on the FSM method with success 80–97%.

Keywords

Kinect, gesture recognition, Kinect SDK, Natural User Interface, ArcGIS, C#.

Abstrakt

Čapek, Martin. Návrh a implementace softwarové komponenty pro manipulaci s 3D objekty. Diplomová práce. Brno, 2015.

Diplomová práce se věnuje návrhu a implementaci softwarové komponenty pro manipulaci s 3D objekty. Práce obsahuje obecnou rešerši popisu ruky, aspekty rozdělení gest, základní metody předzpracování obrazu, podrobný popis klasifikačních metod, popis zařízení Kinect a knihoven pro zpracování obrazu. Praktická část se zabývá návrhem a implementací softwarová komponenty Tool do aplikace ArcScene, návrh a implementace gest, které emulují funkcionalitu 3D myši. Softwarová komponenta je implementována v programovacím jazyce C# s knihovnou Kinect SDK. Systém pro rozpoznávání gest je založen na metodě FSM s úspěšností 80–97%.

Klíčová slova

Kinect, rozpoznávání gest, Kinect SDK, Přirozené uživatelské rozhraní, ArcGIS, C#.

Obsah

1 Úvod a cíl práce

1.1 Úvod

Interakce člověka s počítačem (*Human-Computer Interaction* - HCI) se vyvíjí přes 2D uživatelská rozhraní, uživatelská rozhraní ovládané hlasem, až k 3D uživatelskému rozhraní. Běžné vstupní zařízení pro 2D uživatelská rozhraní, jako jsou klávesnice a myši, přestávají pro moderní 3D uživatelská rozhraní stačit. 3D uživatelská rozhraní a s nimi související technologie virtuální reality představuje výzvu k vytvoření přirozenější komunikace člověka s počítačem.

Pro tyto systémy již existují různá řešení ke zlepšení komunikace. Například zařízení s haptickou zpětnou vazbou jako jsou datové rukavice *CyberGlove*, nebo exoskelety. Dále se používají platformy pro simulaci pohybu v prostoru, jako je *VirtuSphere*. A nahrazení tradiční 2D myši 3D myší. Všechna tato řešení lze popsat jako kontaktní, člověk pro komunikaci potřebuje další nástroj, který je většinou drahý a nepřirozený nebo omezující. Řešení, které je levnější a pro uživatele příjemnější, je technologie optická. Pomocí této technologie lze bezkontaktně snímat pohyb člověka. Zaměřuje se především na detekování a rozpoznání gest rukou, které jsou přirozené a jednoduché pro použití. Člověk pro komunikaci s dalším člověkem hojně používá gesta rukou. Dobrý příklad je znaková řeč.

Vědci se stále snaží o zlepšení komunikace jak mezi národy pomocí jednoho jazyka, tak i mezi člověkem a počítačem. Inspiraci pro návrh této komunikace vědci především hledají ve skutečném světě. Snaží se převést lidské chování, myšlení a komunikaci do počítače tak, aby tomu rozuměl. Jednou cestou k naplnění této komunikace je rozpoznávání gest člověka pomocí zpracování obrazu. Počítač má k dispozici nějaké zařízení (kameru), které bude snímat okolí. Pomocí metod zpracování obrazu detekuje předměty nebo lidi. Jakmile rozpozná člověka, tak se začne sledovat jeho pohyb a počítač pomocí systému na rozpoznání gest komunikuje s člověkem tak, že na každé jeho rozpoznané gesto, vyvolá nějakou reakci. Tento systém pro rozpoznávání gest (komunikace člověka s počítačem) je náplní této diplomové práce.

1.2 Cíl práce

Cílem této diplomové práce je navrhnout a implementovat softwarovou komponentu, která pomocí zařízení *Kinect* bude manipulovat s 3D objekty. K dosažení hlavního cíle bude nutné splnit cíle dílčí:

- Srovnat vývojové nástroje, metody a knihovny pro zpracování obrazu.
- Prozkoumat možnosti zařízení *Kinect* v rozpoznání pohybů uživatele.
- Prozkoumat již implementovaná řešení rozpoznání pohybů uživatele pomocí zařízení *Kinect*, zejména pro pohyb v 3D prostoru.

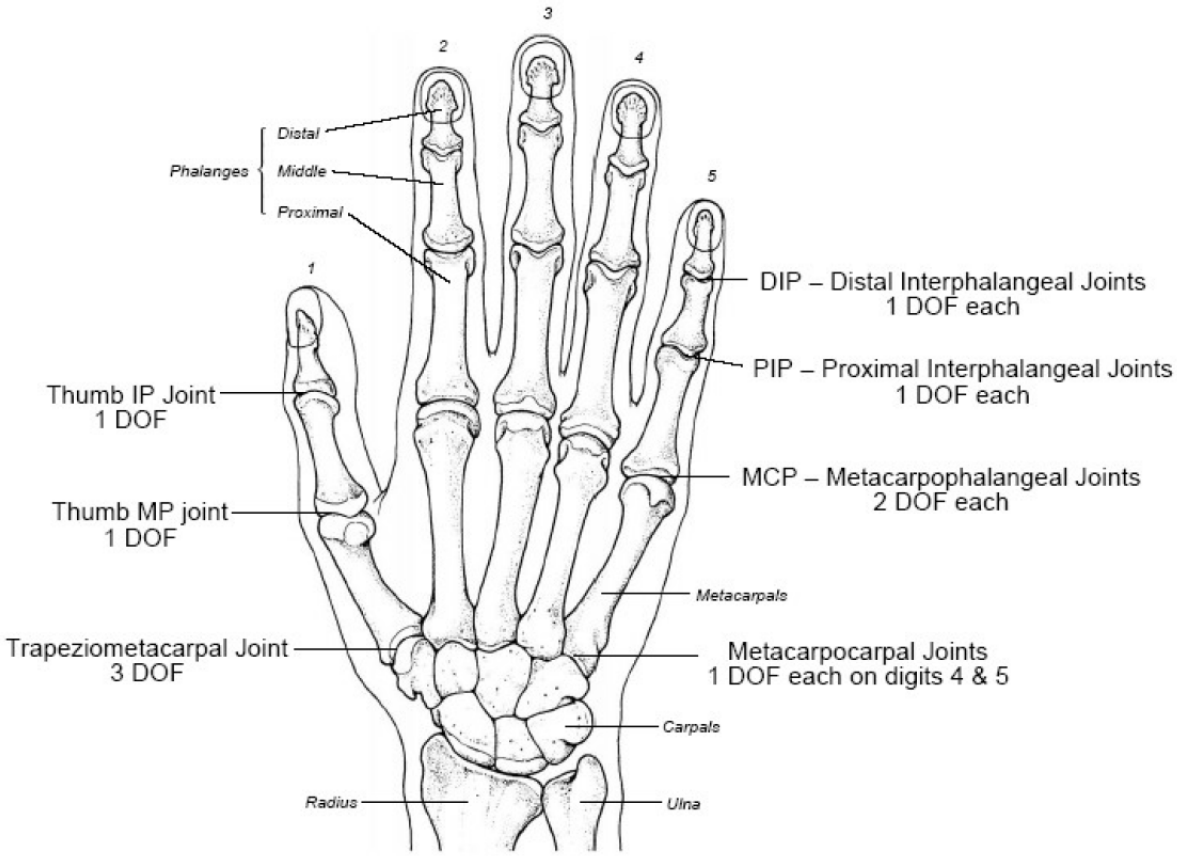
- Navrhnout a implementovat vlastní řešení, které bude pomocí rozpoznání pohybů uživatele umožňovat manipulaci s 3D objekty a nahrazovat funkcionalitu 3D myši, zejména translaci, rotaci a selekci.
- Vyhodnotit a diskutovat výsledné řešení.

2 Používané přístupy pro rozpoznávání gest

Začátek kapitoly se věnuje popisu ruky jako nástroje neverbální komunikace. V práci je využita optická technologie, která je bezkontaktní. Na to navazuje podkapitola Obrazové předzpracování, kde je vysvětleno, jaká data a postupy se používají pro detekci uživatele a extrakce důležitých částí. Výstup dat z obrazového předzpracování jsou vstupní data pro metody rozpoznání a klasifikace gest popsané v podkapitole Metody pro klasifikaci gest a póz těla. Metod pro rozpoznání gest přibývá a v této práci jsou popsané ty nejznámější. V poslední části jsou popsána zařízení, které slouží jako generátory dat pro metody zpracování obrazu. Především je popsáno zařízení *Kinect*, které bylo použito při experimentech.

2.1 Ruka jako nástroj neverbální komunikace

Lidskou ruku lze považovat za složitou konstrukci, která se skládá z mnoha propojených článků a kloubů. Struktura celé ruky, propojených článků a kloubů je znázorněna na obrázku 1.



Obrázek 1: Kostra ruky s viditelnými články a klouby a jejich stupni volnosti. Zdroj: (Qing, 2008)

Každý prst má 3 klouby, které jsou uvedeny i s jejich stupni volnosti (*DOF - degrees of freedom*). Po sečtení všech stupňů volnosti celé ruky vychází číslo 27, tzn. lidská ruka má 27 stupňů volnosti. Vzhledem k tak vysoké stupni volnosti ruky se z rozpoznání gest ruky stává velmi náročný problém. Pro lepší pochopení gest rukou a jejich pohybu jsou dva důležité koncepty, které je nutno vysvětlit:

- Póza ruky: je statický stav, na kterém se nepodílí jakýkoli pohyb.
- Gesto ruky: je posloupnost póz ruky, které jsou spojené pohybem ruky a případně prstů v krátkém čase.

Jinak řečeno póza ruky je například držení ruku v pěst v určité poloze. Gesto ruky je definováno jako dynamický pohyb například mávání. Dynamický pohyb gest zahrnuje dva aspekty: globální pohyby rukou a místní pohyby prstů. Globální pohyb mění pozici a orientaci celé ruky. Místní pohyby prstů v jakémkoli směru nesouvisí s pohybem ani orientací celé ruky. Například pohyb ukazováčkem tam a zpět, lze někomu naznačovat, aby přišel blíž. Gesta ruky ve srovnání s pózami lze chápat jako komplexní kompozici akcí založených na globálním pohybu ruky, místním pohybem prstů a sekvenci póz ruky, které fungují jako přechodové stavy (Qing, 2008).

Gesta a pózy pro komunikaci člověka s počítačem dále jen *HCI* je třeba navrhovat tak, aby je uživatel prováděl intuitivně. Lenman navrhl, že konstrukční prostor pro pohybové příkazy může být charakterizován ve třech bodech: kognitivní aspekt, artikulační aspekt a technologický aspekt (Lenman, Bretzner, Thuresson, 2002).

Kognitivní aspekt
Tento aspekt poukazuje na jednoduchost učení a zapamatování gest. Často se tvrdí, že mají být gesta přirozená a intuitivní. Hlavním cílem je, aby tyto gesta dávali uživateli smysl. Nicméně nemusí existovat stereotypy gest, které by se dali použít pro ovládání s výjimkou specifických situací. Pokud je cílem ovládat nějaké zařízení, není kulturní ani jiný kontext pro většinu funkcí.

Artikulační aspekt
Tento aspekt poukazuje, jak snadno by se daná gesta měla vykonávat a jak jsou únavné pro uživatele. Gesta, která zahrnují složité pózy ruky a prstů je nutno se vyhnout, protože jsou obtížné pro formulaci a pro značnou část populace může být provedení dokonce nemožné. Složitá gesta jsou běžná v současných přístupech pomocí klávesnice nebo myši, protože jsou snadno rozpoznatelná počítačem. Opakovaná gesta vyžadující častý pohyb bez podpory především ramene nejsou vhodná kvůli únavě uživatele.

Technologický aspekt
Poukazuje na vhodnost praktického použití a ne jen ve vizionářských scénářích a kontrolovaných laboratořích. Navržení gest by mělo brát v potaz technologii přítomnosti i blízké budoucnosti.

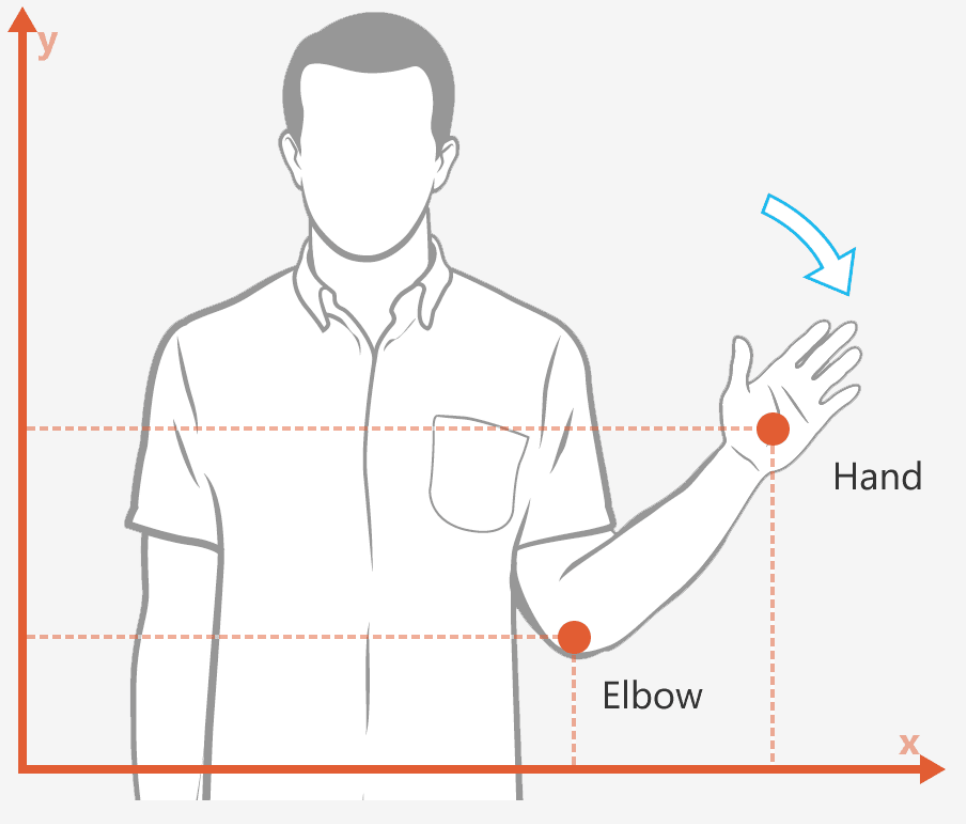
K rozpoznání gest je potřeba dobrá sada charakteristických rysů a znalost vzájemných vztahů, které gesto reprezentují. Pro lepší pochopení bude popsáno gesto mávání. Lidé mávají tak, že zvednou levou nebo pravou ruku a pohybují s ní ze strany na stranu. V průběhu gesta, ruka obvykle zůstává nad loktem a pohybuje se periodicky zleva doprava. Grafické znázornění pohybu je na obrázku 2. Každá pozice (vlevo nebo vpravo) je diskrétní součást gesta. Formálně se tyto části nazývají segmenty.
První segment bude obsahovat podmínky, ruka nad loktem a ruka je vpravo od lokte:

```
Ruka.Pozice.Y > Loket.Pozice.Y AND
Ruka.Pozice.X > Loket.Pozice.X
```

Druhý segment bude obsahovat podmínky, ruka nad loktem a ruka je vlevo od lokte:

```
Ruka.Pozice.Y > Loket.Pozice.Y AND
Ruka.Pozice.X < Loket.Pozice.X
```

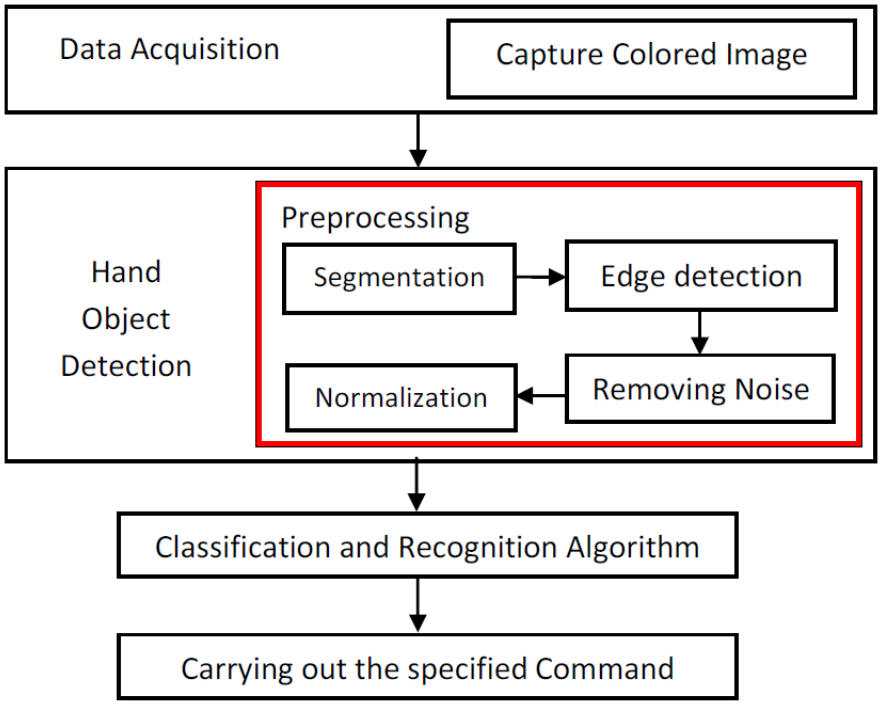
Dále stačí pozorovat tyto segmenty, jak se opakují. Pokud se opakují nejméně třikrát nebo čtyřikrát, pak lze gesto považovat za mávání (Pterneas, 2014).



Obrázek 2: Příklad gesta mávání. Zdroj: (Pterneas, 2014)

2.2 Předzpracování obrazu

Mnoho výzkumů zaměřených na rozpoznávání gest pomocí zpracování obrazu mají různé fáze rozpoznávání, ale shodují se v hlavní struktuře systému pro rozpoznávání gest. Části hlavní struktury jsou následující: segmentace (*segmentation*), detekce hran (*edge detection*), odstranění šumu (*removing noise*), normalizace (*normalization*), rozpoznání (*recognition*) a klasifikace (*classification*) gesta (Khan, Ibraheem, 2012). Tato podkapitola se zabývá především počátečními fázemi až po normalizaci, viz červený rámeček na obrázku 3. Další fáze jsou popsané v podkapitole Metody pro klasifikaci gest a póz těla.



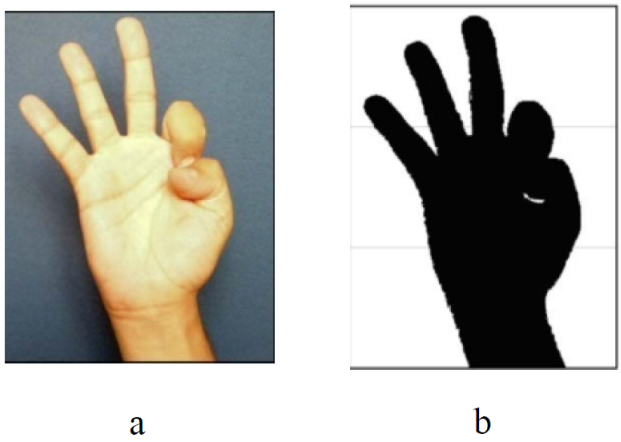
Obrázek 3: Postup v systému rozpoznávání gest. Zdroj: (Khan, Ibraheem, 2012)

Segmentace a odstranění šumu
Vhodná segmentace a odstranění šumu mají vliv na přesnost systému. Cílem segmentace je zjednodušit nebo změnit reprezentaci obrazu do stavu, který je smysluplnější a snadněji se analyzuje. Segmentace obrazu se obvykle používá k vyhledání objektů a hranic (čáry, křivky, atd.). Přesněji řečeno, je to proces přiřazení "značky"každému pixelu v obrazu tak, že pixely se stejnými značkami sdílejí určité vlastnosti (Maire, 2009). Vzhledem k zaměření této práce se segmentace zaměřuje na ruku. Zároveň se segmentací se provádí odstranění šumu. Jedná se o odstranění nežádoucích informací jako je nevhodné pozadí, nesouvisející objekty jako je obličej nebo druhá ruka, pokud má systém používat jen jednu. Prvním krokem je tedy extrahovat oblast ruky ze vstupního obrazu a izolovat ji od pozadí. Existují dvě hlavní metody pro segmentaci objektů v obraze. První metoda je založena na barevném modelu,

který lze získat z existujícího *RGB* modelu a to je například *HSV* barevný model. Druhá metoda je založena na barevném prostoru *YCbCr*, který se zabývá pigmentem lidské kůže na ruce. Významnou vlastností tohoto barevného prostoru je, že různé lidské etnické skupiny mohou být rozpoznány podle sytosti jejich barvy kůže. Poté je oblast ruky izolována ze vstupního gesta s nějakou prahovou hodnotu. Některé normalizace pro segmentovaný obraz vyžadují databázi gest, která by řešila problém natočení, rotace, posunutí a zvětšování (Khan, Ibraheem, 2012).

Použití *HSV* modelu lze najít v publikaci (Hasan, Misra, 2011). Model extrahuje oblast ruky pomocí parametrů pigmentu lidské kůže a používá Laplaceův filtr pro detekci hran. Použití *YCbCr* prostoru lze najít v publikaci (Stergipoulou, Papamarkos, 2009), kde zmíněný prostor použili pro segmentaci ruky. První metodu založenou na barevném modelu *HSV* lze vidět na obrázku 4. Druhou metodu založenou na barevném prostoru *YCbCr* lze vidět na obrázku 5.

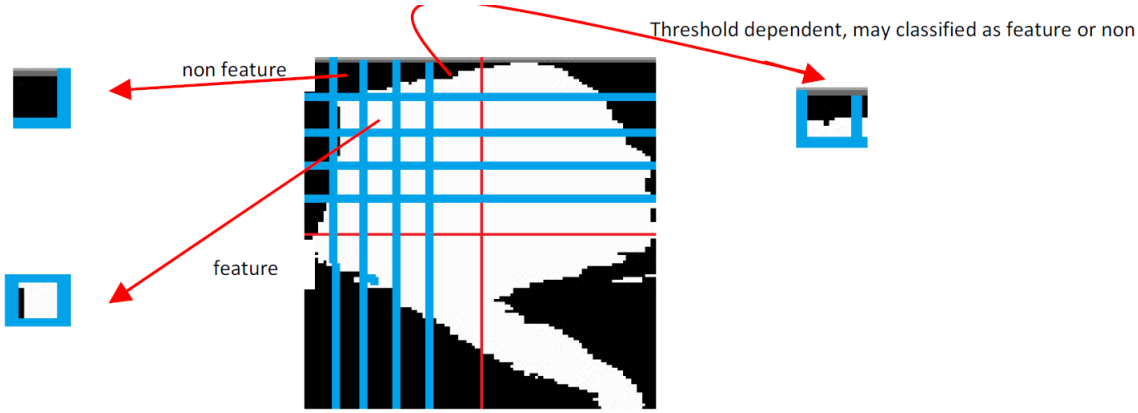
Obrázek 4: Fáze segmentace pomocí *HSV* modelu: a. vstupní obraz, b. segmentovaný obraz, c. detekované hrany. Zdroj: (Hasan, Misra, 2011)



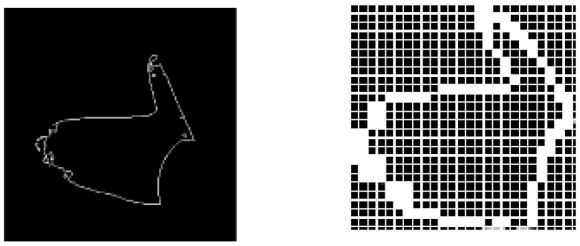
Obrázek 5: Segmentace pomocí *YCbCr* prostoru: a. vstupní obraz, b. segmentovaný obraz. Zdroj: (Stergipoulou, Papamarkos, 2009)

Detekce, extrakce a normalizace Tyto fáze odpovídají za přeměnu segmentované ruky či objektu do číselného formátu tak, aby počítač nebo robot dokázal

zpracovat důležité informace a pochopil tvar ruky. V publikaci (Hasan, Misra, 2011) představili tyto fáze založené na jasu. Jejich databáze vzorků je tvořena prvkovým (*feature*) vektorem pro každý vzorek. Tyto vektory jsou vypočítány podle lokálního jasu bloku ruky v obraze. Každý blok má velikost 5×5 pixelů. Podle jejich použité velikosti obrazů plyne, že každý vektor má velikost 625. Pro úspornější ukládání a vyšší kvalitní zpracování ukládají hodnoty jasu a zanedbávají nejasné rysy. Proces vybírání bloků jasu lze vidět na obrázku 6. Převedení segmentované ruky na jejich vektor, který počítač rozpozná a umí s ním pracovat, lze vidět na obrázku 7.

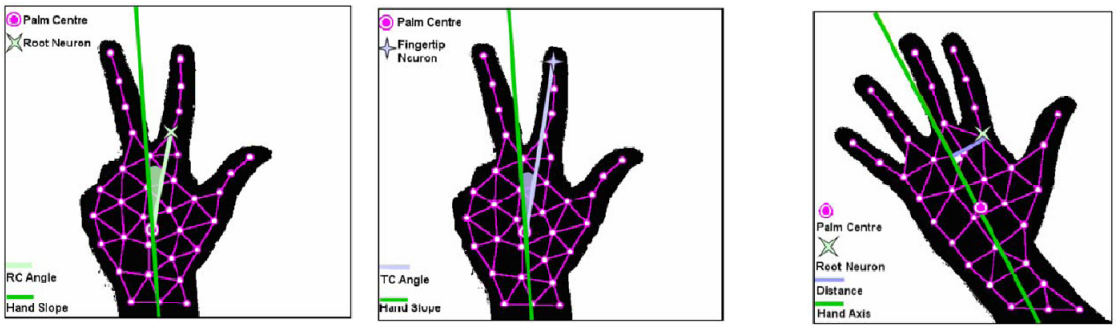


Obrázek 6: Proces vybírání bloků jasu. Zdroj: (Hasan, Misra, 2011)



Obrázek 7: Reprezentace prvkového vektoru jasu: a. segmentovaný obraz, b. prvkový vektor jasu. Zdroj: (Hasan, Misra, 2011)

Jiný přístup lze vidět v publikaci (Stergipoulou, Papamarkos, 2009). Jejich algoritmus využívá *Self-Growing and Self-Organized Neural Gas* (SGONG) neuronovou síť, která extrahuje přesný tvar regionu ruky a určí tři hlavní rysy: region dlaně (*Palm region*), střed dlaně (*Palm center*) a sklon ruky (*Hand slope*). Poté se vypočítají úhly mezi kořenem prstu a středu ruky s názvem *RC Angle*, mezi klouby prstu a středu ruky s názvem *TC Angle* a nakonec úhel a vzdálenost od středu dlaně. Klasifikace jejich systému je popsána v podkapitole Metody pro klasifikaci gest a póz těla Neuronové sítě. Tento proces lze vidět na obrázku 8.



a b c

Obrázek 8: Extrakce ruky pomocí neuronové sítě: a. RC angle, b. TC angle, c. vzdálenost od středu dlaně. Zdroj: (Stergipoulou, Papamarkos, 2009)

2.3 Metody pro klasifikaci gest a póz těla

Klasifikaci gest lze vnímat jako proces, který prochází předzpracovaná data a na základě aplikovaných metod určí, zda se jedná o známé gesto.

2.3.1 Umělé neuronové sítě

Umělá neuronová síť (*Artifical neural network* - ANN) je složena z mnoha propojených neuronů, které pracují na řešení konkrétních problémů. ANN lze nakonfigurovat pro řešení problémů, jako rozpoznání vzorů (*patternt recognition*) nebo dolování dat (*data mining*) prostřednictvím učících modelů. ANN má také schopnosti, jako je adaptivní učení, samoorganizace (Khan, Ibraheem, 2012).

ANN existuje řada typů, přičemž v této podkapitole jsou popsány ty, které se podílí na rozpoznávání gest. Pro lepší pochopení problematiky jsou tyto sítě obecně popsány a následně představeny v publikacích.

V problematice rozpoznávání gest se používají následující typy sítí: Dopředné neuronové sítě (*Feed-Forward Neural Networks*), Vícevrstvé dopředné neuronové sítě (*Multilayer Feedforward Neural Networks*), Opakující se neuronové sítě (*Recurrent Neural Networks*), Elmanovi neuronové sítě (*Elman Neural Networks*), Jordanovi neuronové sítě (*Jordan Neural Networks*), Plně Opakující se neuronové sítě (*Fully Recurrent Neural Networks*), Samorostoucí mapy (*Self-Organizing Maps* - SOM).

Dopředné neuronové sítě

Tyto sítě patří mezi nejoblíbenější a nejpoužívanější modely v mnoha aplikacích. V této architektuře má každý neuron v jedné vrstvě přímé spojení s neurony následné vrstvy. Neexistují žádné vazby mezi neurony ve stejné vrstvě ani s některou z předchozích vrstev. Tok dat má striktně dopředné chování.

Vícevrstvé dopředné neuronové sítě

Vícevrstvá dopředná síť má vrstvenou strukturu. První vrstva je vstupní, kde neu-

rony slouží pouze jako distribuční body, následuje jedna nebo více skrytých vrstev výpočetních neuronů a jedna výstupní vrstva. Jednotky v každé vrstvě obdrží vstup z předchozí vrstvy a jejich výstup pošlou následující vrstvě, což znamená, že se hodnoty pohybují pouze od první vrstvy přes skryté vrstvy až po výstupní vrstvu. Žádné hodnoty se nevrací zpět k předešlým vrstvám. Vícevrstvé neuronové sítě prokázaly svou schopnost řešit mnoho složitých problémů, jako je rozpoznávání vzorů, schopnost extrahovat smysluplnější prvky ze vstupních vzorů pomocí skrytých vrstev.

Opakující se neuronové sítě
Opakující se neuronové sítě byly zajímavou a důležitou součástí výzkumu neuronových sítí v průběhu 90. let. Jsou navrženy tak, aby se naučily sekvenční nebo časově různé vzory a byly použity pro celou řadu problémů, které zahrnují mnoho zajímavých aspektů lidského chování. Dynamické nebo opakující se neuronové sítě se liší od statických neuronových sítí, protože jsou konstruovány se zpětnou vazbou, nebo s opakujícím se spojením mezi vrstvami sítě a v rámci samotné vrstvy. Díky zpětné vazbě má síť lokální paměťové vlastnosti, ve kterých ukládá aktivní vzory a posílá je zpět do sítě více než jednou, což umožňuje síti vlastní aktivaci. V určité době se výstup sítě vypočítá vzorem, který projde celou sítí. Poté se aktivuje zpětná vazba, která vrátí výstupní vzor zpět na začátek celé sítě. Pomocí těchto zpětných vazeb mohou být vytvořeny různé architektury. Zpětné vazby nemusí být vázány přesně na počátek sítě.

Elmanovi neuronové sítě
V roce 1990, Jeff Elman představil rekurentní neuronové sítě, které používají kontextové jednotky. Tato síť je dvouvrstvá. Skrytá vrstva je rekurzivní, každý uzel této vrstvy je spojen zpětnou vazbou se všemi uzly ve stejné vrstvě. V každém časovém kroku jsou výstupy ze skryté vrstvy vypočítány a použity pro výpočet celé sítě. Výstupy ze skryté vrstvy jsou následně uloženy jako „extra vstupy“ (kontextové jednotky). Tyto jednotky se použijí při dalším ovládání sítě. Opakující se kontextové jednotky poskytují vážený součet předchozích hodnot skrytých jednotek ze skryté vrstvy a jsou použity jako vstup skryté vrstvy. Díky tomu síť umožňuje informace o předchozích hodnotách.

Jordanovi neuronové sítě
Tyto sítě jsou podobné Elmanovým sítím v tom, že skrytá vrstva je rekurzivní. Každý kontextový uzel (zkopírovaný uzel) má zpětnou vazbu na všechny ostatní kontextové uzly. Síť má také zpětnou vazbu z výstupní vrstvy na skrytou vrstvu. Aktivita výstupních uzlů je zkopírována zpět do kontextových uzlů. Zpětné vazby mezi kontextovými uzly pomáhají stabilizovat celou síť a poskytují informace o předchozím stavu.

Plně Opakující se neuronové sítě
V tomto typu sítí jsou skryté i výstupní vrstvy rekurzivní. Výstupní vrstvy mají

také zpětnou vazbu na vstupní vrstvu. Vzhledem ke složitější struktuře celé architektury jsou očekávané složitější a náročnější výpočty. Nicméně díky opakujícím se vazbám v síti umožňuje lepší konvergenci optimální hmotnosti, která vede k přesnějšímu rozpoznávání gest.

Samorostoucí mapa
Samorostoucí mapa je typ neuronové sítě, který byl vyvinut Tuevo Kohonenem v roce 1982. První část názvu „Samorostoucí" je z důvodu, že není nutná žádná kontrola a učení tohoto typu sítě je nekontrolované (*unsupervised learning*). Druhá část názvu „mapa" je z důvodu, že se mapa vah snaží namapovat na vstupní data (Khan, Ibraheem, 2012).

V publikaci (Maraqa, Al-Zboun, Dhyabat, Zitar, 2012) testovali všechny výše zmíněné typy sítí pro rozpoznávání gest. Jejich výzkum řešil použití neuronových sítí jako klasifikátorů gest pro arabskou znakovou řeč. Pro vstupní data použili digitální kameru a barevné rukavice. Pro segmentaci ruky použili *HIS* barevný model. Pomocí segmentace rozdělili obraz do šesti barevných vrstev, pět pro prsty a jednu vrstvu pro zápěstí. Pro reprezentaci snímku seskupili a extrahovali třicet rysů ruky a z těchto dat vytvořili vektor. Prsty a zápěstí měřili pomocí úhlů a vzdáleností mezi nimi. Tento vzorový vektor použili jako vstup do neuronových sítí. Trénovacích obrazů vytvořili celkem 900 a 300 testovacích. Výsledky jejich výzkumu pro jednotlivé neuronové sítě jsou popsány dále i s jejich úspěšností.

Dopředná neuronová síť:proces tréninku byl pomalý ve srovnání s opakující se neuronovou sítí. Výsledky rozpoznávání gest nebyly korektní. Jedna třída měla například nulovou přesnost, což je pro systém rozpoznávání gest neakceptující.

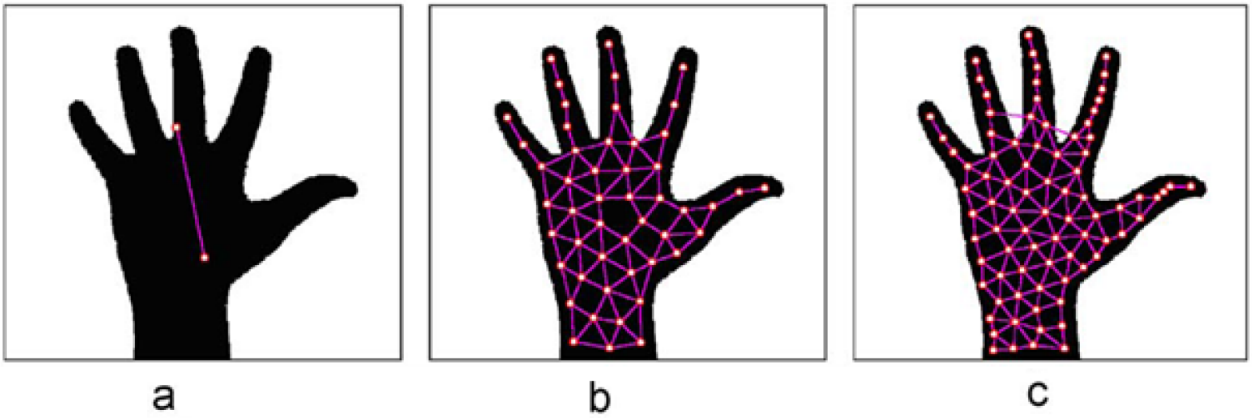
Elmanova neuronová síť: rozdíl oproti dopředné neuronové sítí byl velmi rozeznatelný. Přesnost rozpoznávání gest se zvýšila na 89,66%. Lze si všimnout nedostatků u některých gest a to zejména z důvodu jejich podobností.

Jordanovi neuronové sítě: výsledky při použití této sítě jsou velmi podobné Elmanové neuronové síti i s téměř stejným počtem epoch (počet epoch = počet učení trénovací množiny).

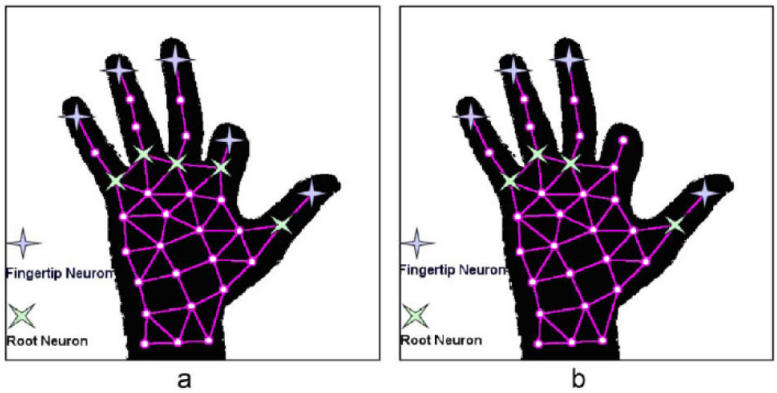
Plně Opakující se neuronové sítě: ze všech vyzkoušených typů sítí má tento typ nejlepší výsledky. Je to zejména z důvodu zpětných vazeb, které vedou ke stabilitě, konvergenci a nejlepší přesnosti.

V publikaci (Stergipoulou, Papamarkos, 2009) představili samoorganizující neuronovou síť přesněji *Self-Growing and Self- Organized Neural Gas* (SGONG). Pro vstupní data použili digitální kameru. Pro detekování ruky použili *YcbCr* barevný prostor viz 2.2 Obrazové předzpracování. SGONG síť využívá porovnávací učící algoritmus

Hebbian[1]. Učení sítě začíná se dvěma neurony a roste do té doby, dokud síť neuronů nepokryje celou oblast ruky viz obrázek 9. Počet zvednutých prstů bylo stanoveno, ale v některých případech vedl algoritmus ke špatné klasifikaci. Problém vyřešili srovnáním střední délky každého prstu viz obrázek 10. Pomocí Gaussova rozdělení pravděpodobnosti[2] se prsty rozdělili do pěti tříd a vypočítaly se rysy ke každé třídě. Nevýhoda tohoto přístupu je, že dva prsty mohou být přidány do stejné třídy. Tento problém vyřešili výběrem nejpravděpodobnějšího výsledku. Jejich systém pro rozpoznávání gest rozpozná 31 předdefinovaných gest s úspěšností 90,45% v rychlosti 1,5 sekund na jedno gesto.



Obrázek 9: Růst SGONG neuronové sítě: a. začíná se dvěma neurony, b. růst se 45 neuronů c. konečný stav, počet neuronů 83. Zdroj: (Stergipoulou, Papamarkos, 2009)



Obrázek 10: Srovnání střední délky prstu: a špatná detekce prstu, (b) správná detekce prstu. Zdroj: (Stergipoulou, Papamarkos, 2009)

[1]Provnávací učící algoritmus Hebian, zdroj: http://www.nbb.cornell.edu/neurobio/linster/lecture4.pdf
[2]Gaussovo rozdělení pravděpodobnosti. zdroj: http://math.feld.cvut.cz/prucha/m3p/u8.pdf

2.3.2 Skryté Markovovské Modely

Skryté Markovovské Modely (*Hidden Markov Models* - HMM) je statistická metoda pro modelování prostorově časové řady. Tyto modely se skládají ze sítě stavů, kde každý stav má pravděpodobnost přechodu na jiný stav a pravděpodobnost tvořit výstup. Existují tři problémy, které se HMM snaží vyřešit:

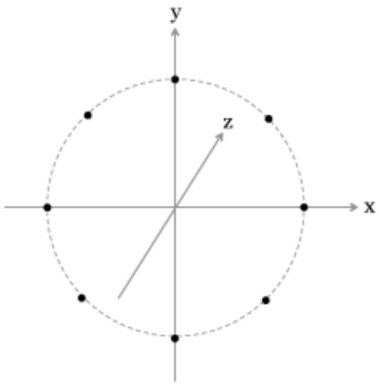
Problém hodnocení – porovnává sekvenci a model, odhaduje pravděpodobnost, jestli je daná sekvence produkována modelem.

Problém dekódování – v dané sekvenci hledá stav, který danou sekvenci generuje.

Problém odhadu – pozoruje množinu sekvencí a zjišťuje parametry modely, které nejlépe popisují tohle pozorování.

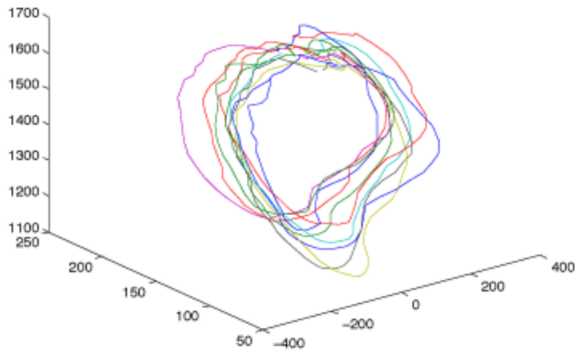
HMM jsou pro systém rozpoznávání gest použity následovně: každé gesto, které má systém rozpoznat, je spojeno s HMM a se souvisejícími parametry ve fázi učení. Tzn. Každé gesto je určitá sekvence reprezentovaná v HMM. Pro každou vstupní sekvenci vytváří všechny HMM pravděpodobnost, zda se jedná o konkrétní sekvenci. Gesto (sekvence) spojené s HMM, které má největší pravděpodobnost, je uznané (Erkan, 2002).

Pro lepší pochopení je popsán příklad s rozpoznáním gesta kruhu. Ideální kruhové gesto je zobrazeno na obrázku 11. Oproti tomu je reálně provedené kruhové gesto uživatelem viz obrázek 12.



Obrázek 11: Ideální gesto kruhu s osmi stavy pro HMM. Zdroj: (Hall, 2011)

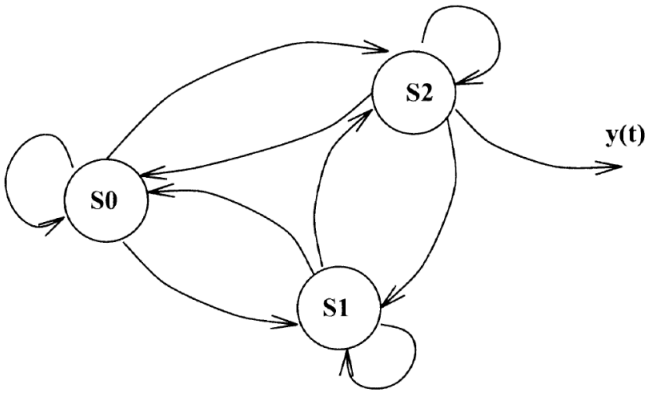
Každý barevný kruh v obrázku 12 představuje instanci skutečného kruhového gesta. Pokud se použije kamera se snímáním 30 snímků za sekundu, gesto se během dvou sekund zachytí v 60 snímcích. Pokud se jedná o hloubkovou kameru, tak lze pozorovat gesto ve třech osách X, Y a Z. Tzn. z každého snímku lze vyčíst pozice ruky. Pokud se má gesto rozpoznat v 60 snímcích, a bude se zvažovat



Obrázek 12: Reálné gesto kruhu provedené desetkrát uživatelem. Zdroj: (Hall, 2011)

osm stavů ideálního kruhového gesta viz obrázek 11, musí se určit pravděpodobnost, že ruka prošla danými osmi stavy předpokládané sekvence. Výstupní konečná množina HMM v tomto případě je osm. Vzhledem k malému počtu stavů HMM, je jich 8, je obtížné kruhové gesto provést správně. Pro zlepšení rozpoznání se HMM rozšíří o další skryté modely, které ulehčí rozpoznání. Tzn. přidají se další body na kruh, kterými má ruka projít a její trajektorie bude porovnávána s více stavy. Pro úspěšnost jich stále stačí osm. Tím pádem je snadnější dané gesto rozpoznat. S akceptující množinou lze manipulovat, tím pádem lze nastavit obtížnost rozpoznávání. V tomto případě HMM není závislé na velikosti ruky, či uživatele, tudíž rozpoznání gesta by mělo fungovat bez ohledu na velikost osoby vykonávající gesto (Hall, 2011).

Základní struktura HMM je znázorněna na obrázku 13, kde *Si* představuje každý stav spojen pravděpodobností do ostatních stavů nebo na sebe a *y(t)* je pozorování každého stavu. Hlavním nástrojem v HMM je *Baum-Welch* (Bishop, 2006) algoritmus pro maximální odhad pravděpodobnosti parametrů modelu (Aggarwal, Cai, 2009).



Obrázek 13: Základní struktura HMM. Zdroj: (Aggarwal, Cai, 2009)

V publikaci (Liu, Lovell, 2003) využili HMM pro rozpoznání gest a porov-

nali *Baum-Welch* algoritmus s algoritmem *Viterbi path accounting* (VPC) (Bishop, 2006). Ve fázi rozpoznávání použili 300 videí. U každého gesta použili 50 videí (25 pro učení a 25 pro testování). Vytvořili osm druhů HMM. Po extrakci sekvence z videa, vypočítali pravděpodobnosti pozorování pro každé HMM. Model s největší pravděpodobností je uznán jako odpovídající gesto. Při testování zjistili, že algoritmus VPC je porovnatelný s algoritmem *Baum-Welch*, tudíž je spolehlivý. Dále zjistili, že jejich model nijak výrazně neovlivnil poměr rozpoznávání. Počet stavů také nemá vliv na poměr rozpoznávání. Větší účinek stavů zatěžuje více *Baum-Welch* algoritmus než VPC.

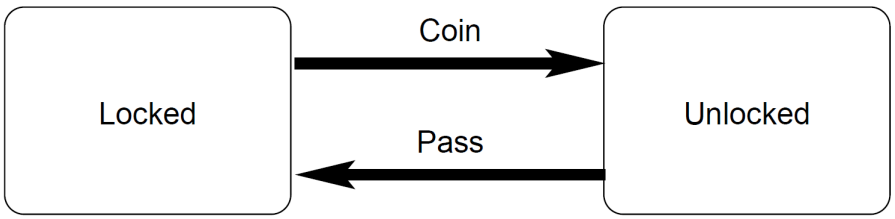
Další příklad využití HMM je v publikaci (Lee, Kim, 1999). Jejich systém zvládne rozpoznat nedefinované vzory. Vytvořili aplikaci s názvem *PowerGesture*, ve které lze ovládat prezentaci pomocí gest rukou místo použití myši. Vytvořili modely gest propojené s HMM a parametry modelů odhadly pomocí *Baum-Welch* algoritmu. Poté vytvořili nový ergodický model odstraněním výstupních vazeb všech modelů a následně je všechny propojili navzájem. V novém modelu se každý stav může dostat do jakéhokoli stavu v jediném přechodu. Pravděpodobnosti každého stavu a jeho přechod sám do sebe zůstávají v novém modelu stejné a pravděpodobnosti výstupů jsou stejně přiděleny. Zachování pravděpodobností stavů a jejich vlastních přechodů sami na sebe vytváří z nového modelu zástupce všech dílčích vzorků referenčních vzorů. Nový ergodický model se dobře porovnává se všemi vzory spojenými z referenčních vzorů v libovolném pořadí. To ukazuje, že výstup z nového modelu může být použit jako adaptivní prahová hodnota pro model gesta a jeho pravděpodobností. Vzhledem k tomu že z každého stavu lze přejít do jakéhokoli stavu, tak je rozpoznávání urychleno.

Rozpoznávání znakové řeči je výzva pro systémy rozpoznávání gest. Je zapotřebí jak trajektorie ruky, tak póza ruky (poloha, orientace, úhly kloubů). Tímto problémem se zabývali autoři v publikaci (Starner, Weaver, Pentland, 1998) a řešili rozpoznávání americké znakové řeči. Využili jednu kameru ve dvou systémech. První systém pozoruje uživatele zepředu (kamera umístěná na stole). Druhý systém používá kameru na čepici pro záznam obrazu. Jejich slovník obsahoval 40 znaků a větnou strukturu, aby byly věty uznány, slova byla omezená na zájmeno, sloveso, podstatné jméno a přídavné jméno. Autoři se nesoustředili na dokonalý tvar ruky, extrahovali vektor s 16 elementy z každé pozice ruky: změna polohy po sobě jdoucích snímcích, oblasti, úhel osy setrvačnosti, délka vektoru a výstřednosti ohraničující elipsu. Přesnost rozpoznávání gest se pohybuje mezi 74,5% a 97,8% v závislosti na poloze kamery a použité gramatiky.

2.3.3 Konečný stavový automat

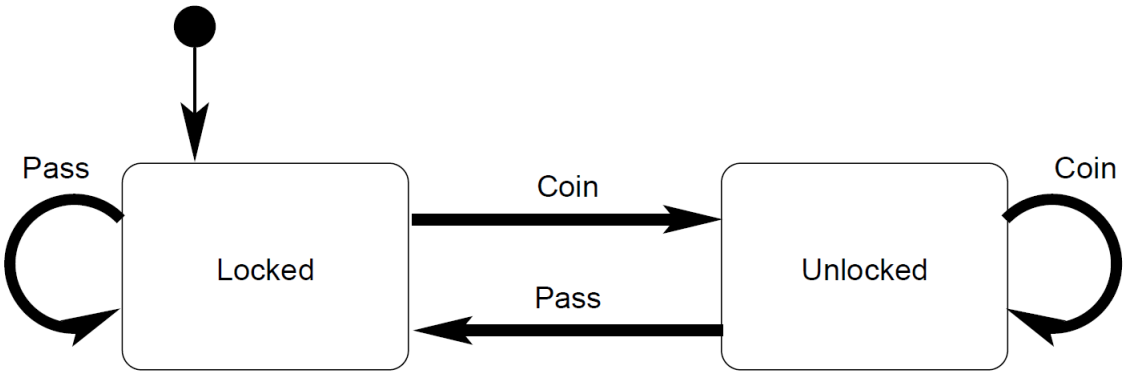
Existuje mnoho způsobů, jak modelovat chování systémů, a použití automatů je jedním z nejstarších a nejznámější. Konečný stavový automat (*Finite State Machine*

– FSM) umožňuje přemýšlet o stavu systému v určitém čase a charakterizuje chování systému na tomto stavu. Pro lepší pochopení je předveden jednoduchý příklad s turniketem. Jedná se o kontrolu turniketu v metru, kde se rotační brány otevřou, jakmile cestující vhodí minci nebo označí jízdenku, tak může projít. Po vhození mince se brána odemkne, ale neotáčí se, dokud do ní cestující nevtlačí a neprojde turniketem. Poté se brána uzamkne. Z tohoto popisu by mělo být jasné, že turniket je vždy v přesně jeden ze dvou možných stavů: zamknuto nebo odemknuto. K dispozici jsou také jen dva přechody: vhození mince, a projití bránou (Wright, 2005). Model tohoto systému lze vidět na obrázku 14.



Obrázek 14: Jednoduchý model FSM turniketu. Zdroj: (Wright, 2005)

I když je to dobré přiblížení chování systému, lze vidět, že některé informace o chování nejsou jasně definovány. Co se stane, když někdo nevhodí minci do turniketu a bude se snažit projít? V tomto případě by měl tento jednoduchý model zůstat ve svém současném stavu (Wright, 2005). Vylepšený model lze vidět na obrázku 15.



Obrázek 15: Vylepšený model FSM turniketu. Zdroj: (Wright, 2005)

Pomocí uvedeného příkladu, lze identifikovat několik klíčových vlastností systému, založeného na FSM:

- Systém musí být popsán konečnou množinou stavů.
- Systém musí mít konečnou množinu vstupů a událostí, které mohou vyvolat přechody mezi stavy.
- Chování systému je určeno aktuálním stavem, vstupem a událostí.

- Chování systému je definováno pro každý možný vstup nebo událost.
- Systém má počáteční stav.

Využití FSM pro rozpoznávání gest lze najít v publikaci (Verma, Dev, 2009). Navrhli přístup založený na FSM, který vymezuje gesto jako posloupnost uspořádaných stavů v konfiguračním prostoru. Každý stav uchovává časoprostorové informace a má následující parametry: poloha ruky v čase a prostoru, dvou dimenzionální rozptyl prostorové matice, minimální a maximální čas restartu FSM. Pohyb a rychlost trajektorie jsou dány časoprostorovými informacemi stavu a sousedního stavu v určitém rozptylu. Stavy pomáhají při segmentaci dat a pro časové zarovnání. Trénovací data jsou zachycena pozorováním gesta, které se několikrát opakuje. Počet opakování vyznačuje počet vstupů pro každý stav. Po skončení tréninku jsou vstupy nahrány do systému a FSM reaguje na všechny známá gesta. Pokud FSM nerozpozná vstup, vrátí se do svého počátečního stavu. Pokud FSM dosáhne konečného stavu, gesto je uznáno. Pokud FSM dosáhne více konečných stavů, vybere se stav, který má nejméně restartů a přechodů mezi stavy.

Další příklad využití FSM lze vidět v publikaci (Hong, Turk, Huang, 2000). Ruku extrahovali a sledovali pomocí barvy kůže a byla ukládána její pozice. Jejich systém dokázal rozpoznat gesta typu: mávání levou rukou, nakreslení kruhu a nakreslení čísla osm. Tyto gesta modelovali pomocí FSM. Jejich metoda modelování gesta umožňovala poloautomatickou cestu pro konstrukci modelů gest. Jednou z výhod, kterou zjistili při použití FSM, je potřeba malého počtu dat pro natrénování gesta. Jako první se algoritmus FSM naučil prostorové informace z kompletních údajů o ruce. Výsledek poskytl podporu pro segmentaci dat a vyrovnání. Poté získal časové informace z vyrovnaných datových segmentů. Pro větší rychlost procesu rozpoznávání autoři začlenili algoritmus KMP (*Knuth Morris Pratt* algoritmus) do FSM postupu (Knuth, Morris, Pratt, 1997). Jejich výpočetní účinnost umožňuje využít velký slovník gest. Navrhovaný přístup testovali pomocí živého videa s úspěšností mezi 90–100%.

2.3.4 Analýza hlavních komponent

Analýza hlavních komponent (*Principal component analysis* – PCA) je matematický postup, který využívá ortogonální (pravoúhlou) transformaci k převodu množiny korelovaných proměnných do množiny lineárně nekorelovaných proměnných nazývaných hlavní komponenty. Počet hlavních komponent je menší nebo rovno počtu originálních proměnných. Tato transformace je definována tak, že první hlavní komponenta má co největší rozptyl. Každá následující komponenta má nejvyšší možný rozptyl s omezením, že musí být kolmá na předcházející komponenty (Gupta, Kundu, Pandey, Ghosh, Bag, Mallik, 2012).

PCA pro rozpoznávání pohybů částí celého těla použili v publikaci (Ju, Black, Dhya-

bat, Yacoob, 1996). Autoři zjistili, že parametry pohybu jsou stabilní v širokém rozsahu pohledů daných činností, aby mohly být reprezentovány několika hlavními směry. Jejich formulace vyžadovala výpočet charakteristické křivky pro každou činnost a část těla pod určitým úhlem. Konstrukce této charakteristické křivky lze dosáhnout sledováním pohybů přes několik subjektů s využitím výše zmíněné PCA metody pro zachycení dominantních prvků křivky.

V publikaci (Chaudhary, Raheja, 1999) autoři využili PCA pro řízení robota v reálném čase. Pomocí PCA extrahovali a rozpoznali gesto z obrazů ve formátu 60×80 pixelů, které měli uložené v databázi. Na každé rozpoznané gesto robot reagoval odpovídající reakcí. Autoři došli k závěru, že metoda PCA je rychlejší než použití neuronových sítí, které vyžadují učící databázi a větší výpočetní výkon.

Použití PCA jako statistického nástroje pro vykonávání učení bez učitele (*unsupervised learning*) a pro vylepšení algoritmu rozpoznávání gest lze najít v publikaci (Gupta, Kundu, Pandey, Ghosh, Bag, Mallik, 2012). Pro extrakci dat pro gesta použili spektrální hustotu výkonu (*Power Spectral Density* – PSD) signálu elektromyografie (*Electromyography* – EMG). EMG se v medicíně používá pro studii kosterního svalstva pomocí elektrických biosignálů, které ze svalů vycházejí (De Luca, Adam, Wotiz, Gilmore, Nawab, 2006). Pomocí Krátkodobé Fourierovy Transformace (*Short-Time Fourier Transformation* - STFT) zachytili nejdůležitější informace z EMG extrahovaných dat pro rozpoznání gest (Gao, Yan, 2010). Vyvinutý algoritmus s pomocí PCA se následně zkoušel na extrahovaných datech upravených STFT.

2.3.5 Fuzzy C-Means Clustering algoritmus

Shlukování dat (*data clustering*) je proces dělení datových prvků do tříd či shluků tak, aby položky ve stejné třídě byly co nejpodobnější, a položky v různých třídách byly co nejvíce odlišné. V závislosti na povaze údajů a účelu použití shlukování se používají různé míry podobnosti pro umístění položek do tříd, kde tyto podobnosti určují, jak jsou vytvořeny shluky. Některé příklady podobností, která mohou být použity pro shlukování, jsou vzdálenost, propojení a intenzita. V silném shlukování (*hard clustering*) jsou položky rozděleny do různých shluků, kde každá položka patří přesně do jednoho shluku. Ve fuzzy shlukování mohou položky patřit do více než jednoho shluku. Každá položka má určitý stupeň příslušnosti ke každému shluku. Tento stupeň je v rozmezí hodnot 0 a 1 včetně obou hraničních hodnot. Fuzzy shlukování je proces přiřazování stupňů příslušnosti a pomocí těchto stupňů přiřazuje položky k jednomu nebo více shlukům. Čím menší je stupeň příslušnosti, tím je položka více u kraje shluku. Jedním z nejčastěji používaných fuzzy algoritmů shlukování je *Fuzzy C-means* (FCM) algoritmus. FCM algoritmus se snaží rozdělit konečnou množinu n prvků $X = (x_1, \cdots, x_n)$ do „c fuzzy“ shluků s ohledem na daná kritéria. Shluků je také konečný počet $C = (c_1, \cdots, c_c)$. Matice rozdělení $W = w_{ij} \in [0, 1]$ kde $i = [1, \cdots, n]$ a $j = [1, \cdots, c]$ popisuje každý prvek wij, který říká: do jaké míry

prvek x_i patří ke shluku c_j (Nock, Nielsen, Pratt, 2006) (Bezdek, 2006). Příklad matice lze vidět v tabulce 1.

Tabulka 1: Ukázka FCM matice

FCM matice	C_1	C_2	C_3	SUM
X_1	0.0	0.0	1.0	1.0
X_2	0.375	0.625	0.0	1.0
X_3	0.5	0.25	0.25	1.0
X_4	1.0	0.0	0.0	1.0
X_5	0.15	0.8	0.05	1.0
X_6	0.3	0.55	0.15	1.0

FCM algoritmus probíhá v iteracích. Každou iterací algoritmu se funkce J minimalizuje viz rovnice 1:

$$J = \sum_{i=1}^{N} \sum_{j=1}^{C} w_{ij} |x_i - c_j|^2 \tag{1}$$

N je číslo položky, C je číslo shluku, c_j je vektor shluku j, a w_{ij} je stupeň příslušnosti položky i do shluku j. Výraz $|x_i - c_j|$ měří podobnost (vzdálenost) položky i od centra vektoru c_j shluku j.

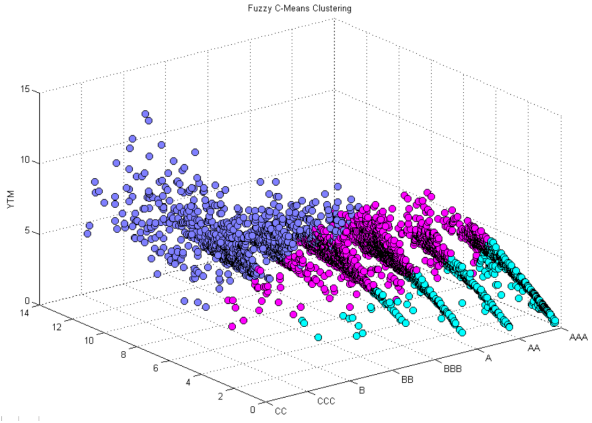
Hodnota stupně příslušnosti w_{ij} se vypočítá pomocí rovnice 2.

$$w_{ij} = \frac{1}{\sum_{k=1}^{C} (\frac{|x_i - c_j|}{|x_i - c_k|})^{\frac{2}{m-1}}} \tag{2}$$

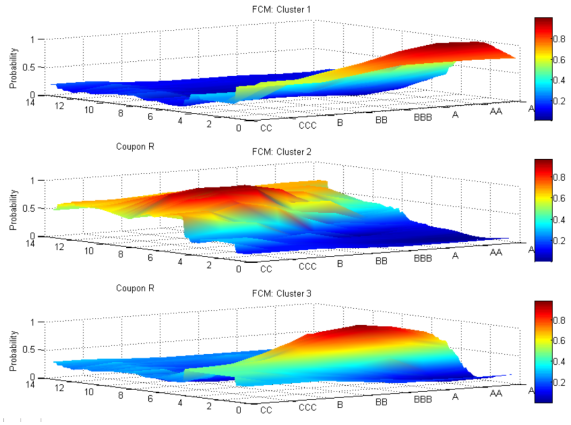
Proměnná m určuje počet shluků, do kterých prvek může patřit. Jinými slovy hodnota m určuje jak moc se shluky mohou překrývat. Čím větší je hodnota m, tím více se shluky překrývají. Pokud je $m = 1$, tak stupeň příslušnosti musí být 0 nebo 1. Těžiště jednotlivých shluků (centrum vektoru) se počítá jako vážený průměr všech položek, kde váha je stupeň příslušnosti k jednotlivým shlukům viz rovnice 3.

$$c_j = \frac{\sum_{i=1}^{N} w_{ij}^m \cdot x_i}{\sum_{i=1}^{N} w_{ij}^m} \tag{3}$$

Příklad obecného FCM lze vidět na obrázcích 16 a 17.



Obrázek 16: Ukázka FCM dat pro shlukování. Zdroj: http://www.mathworks.com/matlabcentral/fileexchange/42744-machine-learning-withmatlab/content/Machine Learning/Clustering/html/Example_Clustering.html



Obrázek 17: Pravděpodobnostní rozdělení FCM shluků. Zdroj: http://www.mathworks.com/matlabcentral/fileexchange/42744-machine-learning-withmatlab/content/Machine Learning/Clustering/html/Example_Clustering.html

V publikaci (Verma, Dev, 2009) využili FCM algoritmus v kombinaci s FSM (*Finite State Machine*) pro rozpoznání gest. Jejich data se skládaly z 2D souřadnic pozic ruky a časových údajů při zachycování provádění gesta. Tyto data definovaly trajektorii jednotlivých gest. Získané body z trajektorie se seskupily pomocí FCM do shluků. Tyto shluky určovaly stavy FSM jehož pomocí se gesto rozpoznalo. Počet shluků ovlivňuje přesnost rozpoznání gesta. Tento postup úspěšně aplikovali na následující gesta: mávání levou rukou, mávání pravou rukou, signalizace k zastavení, vpřed a přetočení. Postup jejich přístupu lze popsat v několika krocích:

1. Zvolí se konečný počet shluků.
2. Každý bod v trajektorii gesta se přidělí do určitých shluků.
3. Druhý bod se opakuje, dokud algoritmus nekonverguje (tzn. Změna mezi dvěma iteracemi není větší než nastavený práh citlivosti).
4. Vypočítá se těžiště pro každý shluk viz rovnice 3.
5. Vypočítá se vzdálenost každého bodu trajektorie gesta od těžiště každého shluku viz rovnice 1 a 2 ($|x_i - c_j|$).
6. Pro každý bod trajektorie gesta se vypočítá stupeň příslušnosti do jednotlivých shluků viz rovnice 2 a vytvoří se FCM matice podobná tabulce 1.

Další přístup pomocí FCM algoritmu lze vidět v publikaci (Li, 2006). Jedná se o systém, který provede akci robota na základě rozpoznání gesta. Pro navrhovanou klasifikaci je FCM algoritmu poslána trénovací množina gest. Těchto gest definovali šest. Každé gesto je reprezentováno vektorem extrahovaným z obrazu. Tato množina vektorů (gest) je seskupena do shluků pro následné použití v systému rozpoznávání. Jakmile jsou shluky vytvořeny, tak jsou pojmenovány ručně podle názvu jednotlivých gest. Po skončení trénování se vytvoří prototypy vektorů jednotlivých shluků. Gesto je klasifikováno podle maximálního stupně příslušnosti jednotlivých vektorů (gest) ke všem shlukům (prototypům). Jejich přesnost klasifikace se pohybuje kolem 85%. Pro vhodnou klasifikaci je u jejich návrhu nutné provést gesto 1 m před kamerou. Jakmile se vzdálenost zvětšuje, přesnost klasifikace rychle klesá.

2.3.6 Genetické algoritmy

Genetické algoritmy (GA) jsou třídy stochastické evoluční vyhledávací techniky, které efektivně řeší různé optimalizační problémy. Algoritmy využívají náhodné operátory, které jsou inspirovány procesy jako je výběr (*selection*), dědičnost (*inheritance*) a mutace (*mutation*), jež jsou součástí biologické evoluce. GA začínají se souborem jedinců nazývaným populace. Každý jedinec má svoji kondiční hodnotu, která je skalární představující dobro jedince, pokud jde o funkci, která má být optimalizována. Cílem je změnit jedince a dostat filtrovanou generaci tím, že stochasticky vytvoří potomka za použití výše uvedených operátorů. Algoritmus pokračuje, dokud není vytvořen jedinec s požadovanou přesností, nebo je dosaženo maximálního počtu generací(Erkan, 2002).

Genetické algoritmy pro klasifikaci gest použili autoři v publikaci (Wu, Huang, 1999). Pro modelování gesta využili 3D modely. Model ruky aproximovali jako abstraktní kostru, která se skládá z 16 menších modelů a kinematickými vazbami mezi těmito modely. Informace o držení ruky rozdělili na dvě části: globální a lokální. Globální část řeší otáčení ruky a celkový pohyb ruky. Lokální část řeší stav prstů. Pro rozpoznání gesta použili dvě optimalizační metody. První metoda se nazývá *least median*

squares (LMS) (Clark, 1997), kterou využili pro opakované hledání globálních parametrů pohybu ruky. Druhá metoda je založená na GA, kterou využili pro nalezení jednotlivých úhlů. Chromozom v druhé metodě je reprezentován jako binární řetězec každého úhlu z kinematických vazeb 3D modelu ruky. Cílem je minimalizovat rozdíly mezi získaným obrazem a předpokládaným modelem. Nicméně, jejich práce byla založena na předpokladu, že jsou všechny prsty viditelné a mohou být detekovány v obraze. To je velmi omezující předpoklad, pokud se bere v úvahu komplikovaná struktura celé ruky. Kromě toho jim algoritmus selhal, pokud se ruka velmi rychle pohybovala.

V publikaci (Lien, Huang, 1999) použili GA pro porovnávání modelů. Pro detekci ruky používali speciální rukavice se sedmi barevnými značkami, pět na konečcích prstů, jednu na zápěstí a ve středu dlaně. Na základě předpokladu, že vztah mezi parametry prstu a úhlech tohoto prstu lze převést na analytické funkce ploch a křivek, trénovali v režimu *offline* s velkým množstvím dat odpovídající různým polohám ruky. Později našli výše očekávané analytické funkce reprezentující vztah mezi parametry ruky. Pomocí těchto vztahů a dat o 3D poloze špičkách prstů, které získali pomocí stereo kamerového systému, vyřešili inverzní kinematiku. Pro proces porovnávání zápěstí použili GA. Kondiční hodnota používaná pro zápěstí je vážený součet (součin váhy a hodnoty dané veličiny) objemů čtyř prstů, který lze vidět v rovnici 4. Chromozom je kódován jako plovoucí (*float*) hodnota pro rotační úhly.

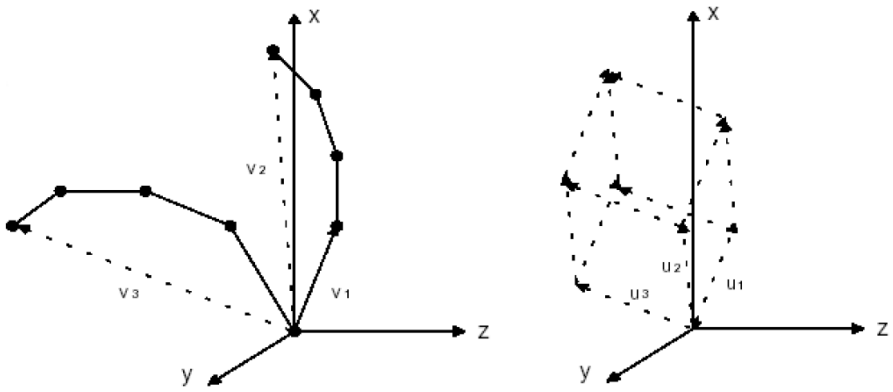
$$V_{(n_x,n_y)} = w_1V_1 + w_2V_2 + w_3V_3 + w_4V_4 \tag{4}$$

Kde *w1 – w4* jsou váhy a *V1 – V4* jsou objemy čtyř prstů bez palce. Definování objemů lze vidět v rovnici 5 a na obrázku 18.

$$V = |(u_1 \times u_2) \cdot u_3| \tag{5}$$

u_1 – jednotkový vektor, který směřuje od prvního kloubu prstu ke druhému kloubu prstu (první článek)
u_2 - jednotkový vektor, který směřuje od prvního kloubu prstu ke špičce prstu v modelu
u_3 - jednotkový vektor, který směřuje od prvního kloubu prstu ke špičce prstu reálné ruky

Jakmile se objem prstu snižuje, podobnost mezi modelem a skutečnou rukou se zvyšuje.



Obrázek 18: Objem použitý pro měření podobnosti v GA. Zdroj: (Lien, Huang, 1999)

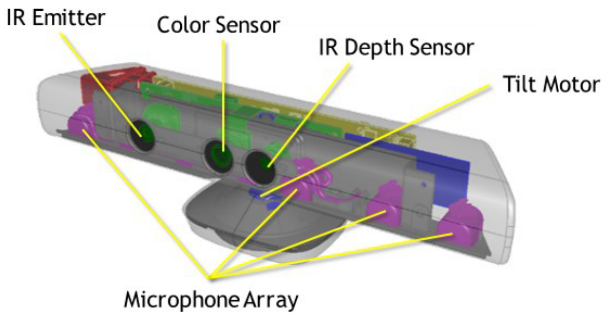
2.3.7 Shrnutí klasifikačních metod

Umělé neuronové sítě se v klasifikaci hojně využívají, ale vyžadují vyšší výpočetní výkon. Jejich úspěšnost je vysoká, ale hodí se pouze pro statická gesta. ANN jsou výborné pro rozpoznání znakové řeči. Oproti ANN jsou HMM použitelné pro rozpoznání dynamických gest, jak je například gesto kruhu zmíněné v podkapitole Skryté Markovovské Modely. Implementace této metody není tak náročná jak je tomu u ANN. Na internetu lze najít i pár návodů, kdežto pro ANN je potřeba větší teoretický základ. Metoda FSM je podobná metodě HMM. Lze ji použít jak na dynamická, tak na statická gesta. Lze říct, že většina systémů pro rozpoznávání se zakládá na této metodě. I když se třeba nejedená o rozpoznávání, metoda FSM je nejlépe pochopitelná i pro začátečníka. To nelze říct o metodách PCA, FCM ani GA. Tyto metody jsou založené na velkém teoretickém základu stejně jako metoda ANN. Z posledních tří metod lze je nejjednodušší metoda FCM. K Metodě FCM jsou k dispozici dobré zdroje pro pochopení a dokonce i ukázky řešení.

2.4 Kinect

Zařízení *Kinect* bylo původně známé pod označením „Projekt Natal“. Jedná se o zařízení pro detekci pohybu a hlasu uživatele, které bylo původně vyvinuto pro herní konzole *Xbox 360*. Po uvedení zařízení *Kinect* na trh pro konzole *Xbox 360*, měli vědci a vývojáři zájem o připojení zařízení k počítači, aby na něm mohli zkoumat a vyvíjet bezdotykovou interakci s počítačem. Proto firma Microsoft následně uvedla na trh *Kinect for Windows*. Rozdíl mezi těmito zařízeními je zmíněno dále v této kapitole. *Kinect* nabízí přirozené uživatelské rozhraní (*Natural user interface* – NUI) pro interakci pomocí pohybu těla, gest a hlasovými příkazy. *Kinect* je horizontální zařízení s infračerveným vysílačem (IR - *Infrared emitter*), hloubkovým senzorem

(IR – *Infrared depth sensor*), barevnou kamerou (*Color sensor*) a sadou mikrofonů (*Microphone array*). Všechny zmíněné komponenty jsou zabalené do malé ploché plastové krabice. Tato krabice je připojená k malému motoru (*Tilt motor*), který umožňuje sklápění zařízení ve vertikálním směru (Abhijit, 2012). Složení zařízení *Kinect* lze vidět na obrázku 19.

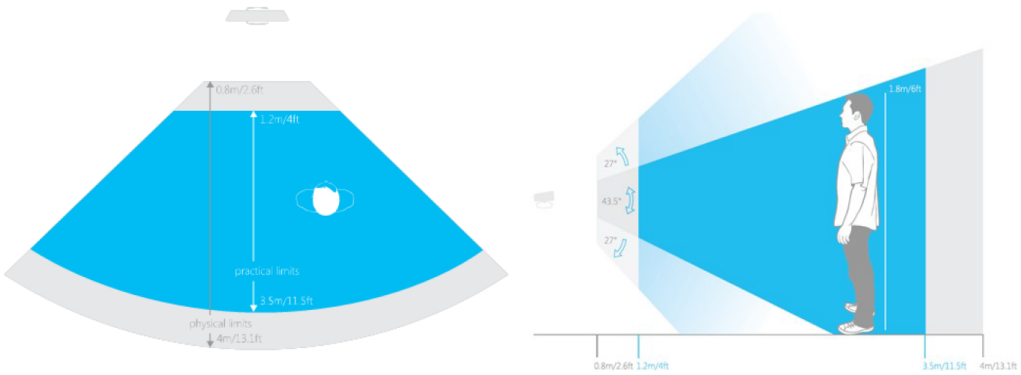


Obrázek 19: Zařízení *Kinect*. Zdroj: https://msdn.microsoft.com/en-us/library/jj131033.aspx

Kromě výše zmíněných komponent má zařízení *Kinect* napájecí adaptér pro externí napájení a USB adaptér pro připojení k počítači. Vzhledem k cíli této diplomové práce budou popsány jen komponenty sloužící k zachycení obrazu.

Barevná kamera

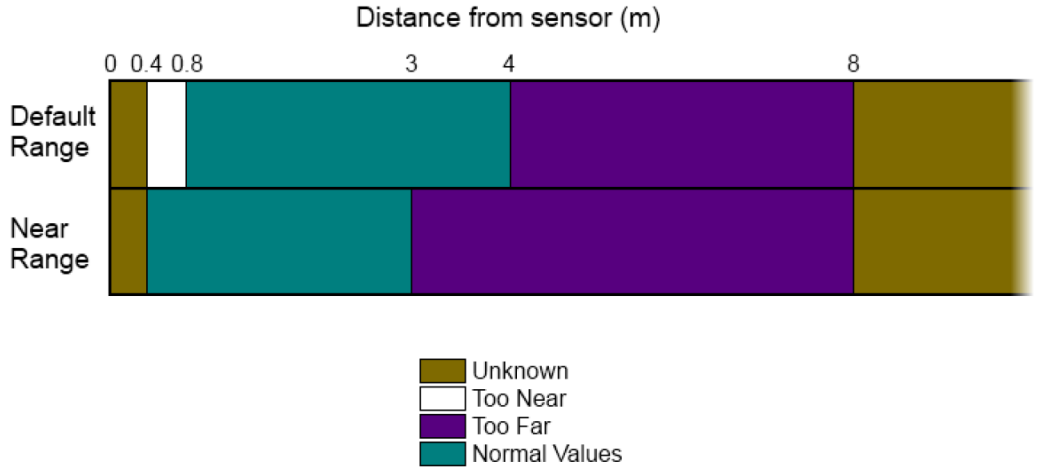
Barevná kamera je zodpovědná za zachycení a *streamování* barevných snímků v barevném modelu RGB. Kamera podporuje rychlost 30 snímků za sekundu (*fps*) v rozlišení 640 × 480 pixelů a při maximálním rozlišení 1280 × 960 pixelů podporuje rychlost 12 *fps*. Hodnota rychlosti snímků za sekundu se může lišit v závislosti na použitém rozlišení (Abhijit, 2012). Viditelný rozsah kamery je 43 stupňů vertikálně a 57 stupňů horizontálně viz obrázek 20.



Obrázek 20: Snímací rozsah zařízení *Kinect*. Zdroj: https://msdn.microsoft.com/en-us/library/hh973074.aspx

Infračervený vysílač a hloubkový senzor

Infračervený vysílač vypadá z vnějšku podobně jako kamera, ale je to reflektor, který neustále vysílá infračervené světlo do „pseudo-náhodného bodového vzoru“ před zařízení *Kinect*. Tyto body jsou neviditelné pro člověka, ale je možné zachytit jejich hloubku pomocí hloubkového senzoru. Body se odráží od různých objektů, hloubkový senzor je snímá a převádí na hloubkové informace změřením vzdálenosti mezi snímačem a objektem, odkud byl infračervený bod snímán. Dosah snímání hloubkových dat zařízení *Kinect* lze rozdělit na dva módy: výchozí vzdálenost (*Default range*), blízká vzdálenost (*Near Range*) viz obrázek 21. Pro zařízení *Kinect* pro konzole *Xbox 360* nelze nastavit jiný mód než *Default Range*. Zařízení *Kinect for Windows* může snímat v obou módech (Abhijit, 2012).



Obrázek 21: Hloubkový rozsah zařízení *Kinect*. Zdroj: https://msdn.microsoft.com/en-us/library/hh973078.aspx#Depth_Ranges

Hloubkový *stream* podporuje rozlišení 640×480 pixelů, 320×240 pixelů a 80×60 pixelů. Viditelný rozsah hloubkového senzoru je stejný jako u barevné kamery viz obrázek 20.

Zpracování hloubkových dat

Vyvinutí technologie zpracování hloubkových dat v zařízení *Kinect* zajistila firma *PrimeSense*. Pokud je potřeba snímat hloubková data, *PrimeSense* čip vyšle signál do IR vysílače pro vysílání infračerveného světla a vyšle další signál do hloubkového senzoru pro zahájení snímání hloubkových dat. *PrimeSense* čip analyzuje snímaná data a vytvoří hloubkový snímek. Každý pixel v hloubkovém snímku obsahuje vzdálenost v milimetrech v kartézské soustavě souřadnic nejbližšího objektu. Souřadnice v hloubkovém snímku neodpovídají skutečným souřadnicím objektů ve snímaném prostoru, ale představují umístění pixelu v hloubkovém snímku. Po vytvoření hloubkového snímku ho *PrimeSense* čip pošle na výstup (Abhijit, 2012).

Kinect for Window vs Kinect for Xbox

I když jsou *Kinect for Windows* a *Kinect for Xbox* v mnoha ohledech podobné, existuje několik rozdílů z pohledů vývojářů. Hlavním cílem pro *Kinect for Xbox* bylo hraní her. Nebylo tedy navrženo pro vyvíjení aplikací pro počítač. Oproti tomu *Kinect for Windows* je především zařízení pro vývoj aplikací a ne pro hraní her. Pro vyvíjení aplikací lze použít oba, ale *Kinect for Xbox* byl navržen tak, aby detekoval uživatele ve vzdálenosti 80–400 cm (*Default range* – viz obrázek 21). *Kinect for Windows* má nový „firmware“, který umožňuje sledovat uživatele ve vzdálenosti 40–300 cm (*Near range* – viz obrázek 21). Při používání *Kinect for Xbox* pro počítačové aplikace může nastat problém připojením a detekcí zařízení. Největší rozdíl mezi *Kinect for Xbox* a *Kinect for Windows* je v tom, že *Kinect for Windows* lze použít pro komerční použití, kdežto *Kinect for Xbox* jen pro výuku a výzkumné účely.

Výhody

- sledování kostry člověka (*skeleton tracking*)
- sledování obličeje člověka
- použití více zařízení zároveň
- přístup k surovým datům
- ovládání hlasem
- *open-source* knihovny pro práci se zařízením

Nevýhody

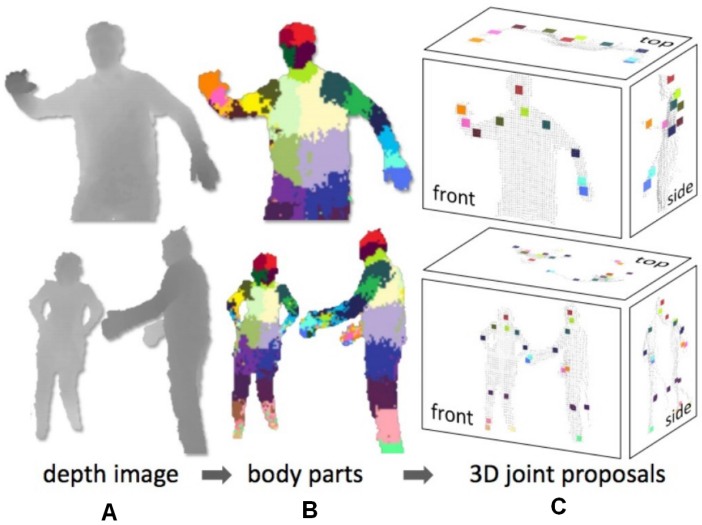
- nutný napájecí kabel
- velké množství generovaných dat zahlcuje USB spojení
- malé rozlišení snímání, neumožňuje sledování prstů
- zařízení lze použít jen pro *desktopové* aplikace pro Windows 7 a lepší

2.4.1 Sledování kostry člověka

Zařízení *Kinect* vrací surová hloubková data viz výše Zpracování hloubkových dat, ze kterých lze identifikovat pixely, které představují uživatele. Sledování kostry člověka je obtížné z několika důvodů: různé pózy těla (jedná část těla se může pohnout různými směry a způsoby), velikost člověka, výška člověka, atd. K překonání těchto problémů zařízení *Kinect* používá proces „vykreslovací rouru“ (*rendering pipeline*), kde se vstupní data shodují se vzorovými natrénovanými daty. Algoritmus pro rozpoznání těla a póz těla člověka používá charakteristické modely (natrénovaná data), které se liší výškou, velikostí, oblečením a několika dalšími faktory. Těchto modelů má zařízení *Kinect* ve své vnitřní databázi 20 milionů s 200 různými pózami. V modelech jsou označeny jednotlivé části těla a srovnány se vstupními hloubkovými daty, které určí do jaké části těla daná data patří.

V počátečních fázích procesu „vykreslovací roury“, hloubkový senzor identifikuje tělo člověka, jako kterýkoli jiný objekt. Senzor nerozpozná, jestli se jedná o tělo člověka. Následující obrázek 22a ukazuje, jak vypadá tělo člověka v hloubkovém snímku. Tělo je rozpoznáno jako velký objekt. Pro rozpoznání těla člověka začne zařízení *Kinect* porovnávat vstupní hloubková data s natrénovanými daty. Tohle porovnání se provádí s velmi vysokou rychlostí zpracování. Další krok v rozpoznávání je označení částí těla vytvořením segmentů viz obrázek 22b. Vytvoření segmentů závisí na porovnání vstupních a natrénovaných datech. Zařízení *Kinect* používá trénovací stromovou strukturu (*decision tree*) k porovnání určitých částí těla. Každý natrénovaný model představuje jednu stromovou strukturu, dohromady tvoří rozhodovací les (*decision forest*). Všechny uzly v rozhodovacím stromu jsou rozdílné údaje modelu a každý je označený názvem části těla. Každý pixel vstupních dat prochází tímto stromem a porovnává se s natrénovanými daty. Proces porovnávání trvá tak dlouho, dokud nenajde shodu. Vždy, když se najde shoda, se pixely označují částmi těla a vytváří se segmenty.

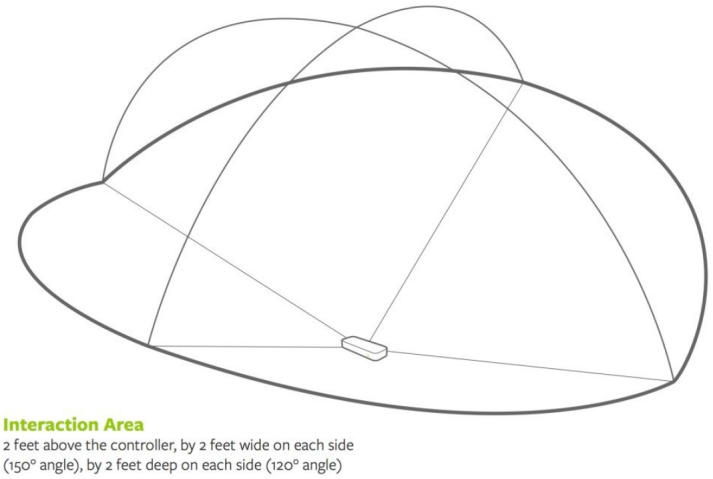
Jakmile jsou identifikovány jednotlivé části těla, umístí se na jejich pozice značky. Tyto značky představují především klouby jako ramena, lokty, zápěstí, ale také jiné části těla jako hlavu, střed ramen, střed těla. Pozice těchto značek se měří třemi souřadnicemi X, Y a Z, kde X a Y určují pozici v obraze a Z určuje vzdálenost od zařízení *Kinect*. Pro získání přesnějších souřadnic zařízení *Kinect* vypočítá tři pohledy z jednoho snímku: pohled zepředu (daný snímek), pohled ze stany (zleva) a pohled ze shora. Těmito pohledy zařízení *Kinect* vytvoří návrh 3D proporce těla viz obrázek 22c. S pomocí identifikovaných částí těla a jejich pohybu lze sledovat pohyb člověka (Abhijit, 2012).



Obrázek 22: Proces vytvoření skeletonu. Zdroj: http://research.microsoft.com/pubs/145347/BodyPartRecogni

2.4.2 Alternativa - Leap motion

Jedná se o malé zařízení, které sleduje ruce a prsty uživatele a je připojené k počítači. Vzhledem k tomu, že se jedná o téměř nové zařízení, tak software technologii nelze konkrétně popsat, tzn. algoritmy pro rozpoznávání prstů, gest a zpracování obrazu. Hardware zařízení *Leap Motion* se skládá ze dvou kamer a tří infračervených LED diod. Tyto diody vysílají infračervené světlo o vlnové délce 850 nm a to je mimo viditelné světelné spektrum pro člověka. Pomocí širokoúhlým objektivům má zařízení interakční prostor 8 krychlových stop ve tvaru obrácené pyramidy viz obrázek 23.



Obrázek 23: Snímací rozsah zařízení Leap Motion. Zdroj: http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/

Rozsah viditelnosti nad zařízením je zhruba 2 stopy (60 cm). Tento rozsah je omezen z důvodu šíření světla pomocí LED diod. Intenzita LED diod je omezena maximálním proudem, který může být čerpán přes USB připojení. Pomocí řadiče USB na zařízení čte data ze senzorů do vlastní lokální paměti a provede všechny nezbytné úpravy rozlišení. Tyto data jsou následně přes USB připojení přeneseny do sledovacího programu zařízení *Leap Motion* v počítači. Snímky posílané do sledovacího programu jsou ve stupních šedi. Intenzivní zdroje nebo odrazy infračerveného světla mohou ztížit rozlišení ruk a prstů. *Leap Motion* negeneruje hloubkové snímky, místo toho využívá pokročilé algoritmy pro zpracování surových dat ze senzorů. Pro správný provoz zařízení *Leap Motion* musí být nainstalována služba *Leap Motion Service*, která zpracovává obraz. Po kompenzaci objektů v pozadí (jako jsou hlavy) a okolního osvětlení v oblasti interaktivního prostředí, jsou obrazy analyzovány s cílem rekonstruovat 3D zobrazení toho, co zařízení vidí. Sledovací vrstva porovnává data pro sledování prstů nebo nástrojů (tužka, ukazovátko). Algoritmus pro sledování interpretuje 3D data a zajímá se pouze o plné (uzavřené) objekty. *Leap Motion* také používá filtrační techniky k zajištění časových soudržností údajů. Jakmile zaří-

zení detekuje a rozpozná prsty nebo nástroje, pošle svoje výsledky přes transportní protokol. Pomocí tohoto protokolu služba (*Leap Motion Service*) komunikuje s kontrolním panelem (*Leap Motion Control Panel*) i s klientskými knihovnami. Klientská knihovna organizuje data do objektově orientované API struktury a poskytuje pomocné funkce a třídy. Příklad detekce a rozpoznání prstů lze vidět na obrázku 24.

Obrázek 24: Ukázka snímání rukou pomocí Leap Motion. Zdroj: http://theprezenter.com/hints-and-tips/prezi-and-leap-motion

Výhody

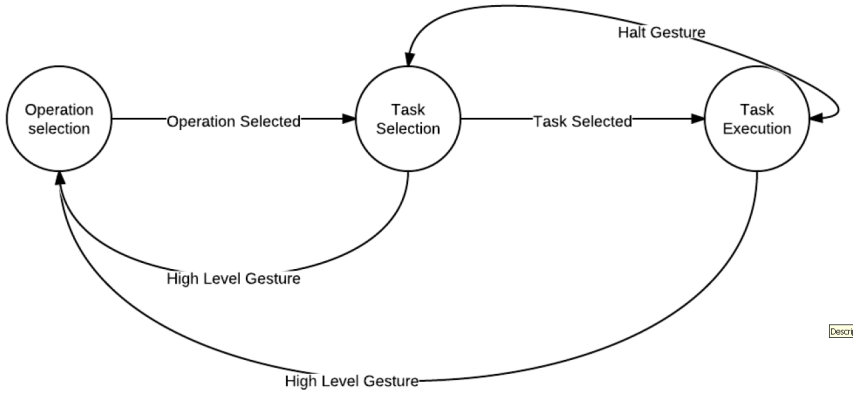
- rychlé a přesné rozpoznání prstů (*finger tracking*)
- velmi malé a oproti konkurenci levné
- „Airspace application store“ umožňuje vývojářům distribuci aplikací
- podpora frameworku například *.NET, Cinder*
- funkčnost na *Mac OS* i *Windows*

Nevýhody

- snímací rozsah je poměrně omezený
- neumožňuje sledování kostry ani obličeje člověka
- žádný přístup k surovým datům

2.4.3 Přístup rozpoznávání gest zařízením Kinect

V publikaci (Dave, Chowriappa, Kesavadas, 2012) použili zařízení *Kinect* jako nástroj pro modelování 3D modelů v aplikaci *Protean Clay*. Navrhli a implementovali gesta pro změnu měřítka, rotaci a posunutí. Jejich systém mohl ovládat pouze jeden uživatel, ale pro budoucí vylepšení chtějí implementovat ovládání pro více uživatelů. Navržená gesta rozdělili do tří úrovní: selekce operace, selekce úkolu, provedení úkolu. Když uživatel provede gesto pro selekci operaci, tak systém čeká až se operace vybere. Po vybrání operace se přejde na stav vybrání úkolu. Po vybrání úkolu systém provede vybraný úkol, například rotaci. Pro rozpoznávání využili metodu FSM, kterou lze vidět na obrázku 25.



Obrázek 25: Schéma použitého FSM. Zdroj: (Lien, Huang, 1999)

2.5 Knihovny pro zpracování obrazu

2.5.1 Kinect SDK

Knihovna *Kinect SDK* je určená pro přímé použití ve spojení se zařízením *Kinect*. Je dokonce zakázané ji použít s jiným zařízením. Knihovna je napsaná v jazyce *C#* a lze ji používat v jazycích *C#, C++, VB*. Volně dostupná je pouze pro akademické účely. *Kinect SDK* má kvalitní dokumentaci a spoustu návodů řešení i podporu vývojářů firmy Microsoft. Knihovna má určité verze. Verze 1.0–1.8 se používají se zařízením *Kinect for Xbox* a *Kinect for Window* V1 a lze je použít jen v operačním systému Windows 7 a vyšším. Verze knihovny 2.0 a vyšší je určena pro zařízení *Kinect V2* a lze je použít pouze v operačním systému Windows 8 a vyšším. V této práci je použito zařízení *Kinect* V1, tudíž se další popis knihovny změří na verze 1.0–1.8. Od verze 1.5 lze stáhnout s knihovnou také *toolkit*, který například řeší modelování 3D modelů pomocí *Kinect Fusion*, nebo rozpoznat interakce pomocí *Kinect Interaction*. *Kinect SDK* umožňuje detekovat šest uživatelů, ale jen dva sledovat.

2.5.2 OpenNi

Knihovna *OpenNi* (*Open Natural interface*) je volně dostupná knihovna od společnosti *PrimeSense*, která vyvinula algoritmy sledování kostry uživatele pro zařízení *Kinect*. Tato knihovna lze použít se zařízením *Kinect for Xbox* i pro *Kinect for Windows*. Knihovna je napsaná v jazyce *C++* a ve spojení s knihovnou *Nite* dokáže sledovat až šest koster uživatelů ve stejnou dobu. První verze knihovny se již nevyvíjí a dokumentace k ní není kvalitní. Druhá verze knihovny je také od společnosti *PrimeSense*, ale nekomunikuje se zařízením *Kinect*. Společnost *PrimeSence* ji vytvořila pro použití se zařízením *ASUS Xtion* a jinými, které používají infračervenou kameru s výjimkou právě zařízení *Kinect*.

2.5.3 OpenCV

Knihovna *OpenCV* (*Open Computer Vision*) je volně dostupná knihovna, která se používá pro zpracování obrazu. Řeší především předzpracování obrazu. Obsahuje algoritmy, které umí nalézt hrany, což se používá například u rozpoznávání vzorů (*pattern recognition*). Pomocí této knihovny lze detekovat obličej uživatele. Má velice kvalitní dokumentaci i se vzorovými příklady. Na internetu je hodně návodů různých řešení a využití. Tato knihovna je multiplatformní, lze ji používat například v operačních systémech *Window, Linux, Mac OS*, a další. Knihovna je napsaná v programovacím jazyce *C++*. Použít ji lze v různých prostředích například *MS Visual Studio, Matlab, Octave*, atd. Knihovna *OpenCV* lze využít v kombinaci se zařízením *Kinect*, ale dokáže ho sama o sobě použít jen jako *RGB* kameru. Pro detekování a sledování kostry uživatele je zapotřebí jedna z výše zmíněných knihoven.

3 Metodika práce

Pro návrh a implementaci softwarové komponenty pro manipulaci s 3D objekty budou nutné následující kroky:

Vybrat prostředí pro manipulaci 3D objektů
Prostředí pro manipulaci 3D objektů by mělo být udržované, rozšířené a především funkční. Bude nutné nalézt způsob přidání softwarové komponenty do tohoto prostředí. Může se jednat o knihovnu, *Add-in*, *plug-in*. Softwarovou komponentu by mělo být jednoduché připojit k existujícímu prostředí zvolené verze a vyšší. Prostředí musí umožňovat manipulaci alespoň jednoho 3D objektu pomocí funkcionalit 3D myši. Prostředí musí být spustitelné v operačním systému Windows 7 a vyšší, z důvodu použití zařízení *Kinect*.

Vybrat knihovnu, která dokáže komunikovat se zařízením *Kinect*
Bude nutné vybrat knihovnu, která dokáže komunikovat se zařízením *Kinect*. Knihovna musí být napsaná ve stejném programovacím jazyce, ve kterém lze vyvíjet softwarovou komponentu pro vybrané prostředí. Vybraná knihovna musí umožňovat detekci uživatel. Musí umožňovat i sledování různých částí těla, především ruce. Vybraná knihovna musí být stabilní, udržovaná a mít kvalitní dokumentaci. Knihovna by měla umožňovat přístup i k surovým datům, nejen komunikovat se zařízením *Kinect*.

Navrhnout množinu gest, které emulují funkcionalitu3D myši
Bude nutné navrhnout gesta, které nahrazují jednotlivé funkce 3D myši pro manipulaci 3D objektu. Tyto funkce jsou: rotace ve všech osách, posun v ose X a Y, změna měřítka objektu (*zoom*). Návrh gest by se měl inspirovat vzory ze skutečného světa. Gesta ze skutečného světa jsou lidem nejblíže, tudíž jsou intuitivní a jednoduchá pro provedení. Vzor pro návrh lze také hledat u mobilních platforem, které jsou v aktuální době velice rozšířené. Gesta na mobilních platformách jsou jednoduchá pro provedení a již vyřešená pro implementaci. K návrhu gest by se mělo přistupovat i podle jejich častého využití, aby z nich uživatel nebyl příliš unavený. Návrh gest není omezený pouze funkcionalitou 3D myši, mohou se navrhnout i gesta, která mají odlišné reakce.

Implementovat navržené řešení
Před implementováním navržených gest se musí implementovat propojení mezi softwarovou komponentou a zvoleným prostředím. Poté se musí nalézt či implementovat funkce, které ve zvoleném prostředí manipulují s 3D objektem. Tyto funkce budou použity jako reakce na navržená gesta. Po úspěšném propojení softwarové komponenty s prostředím a po nalezení funkcí pro manipulaci s 3D objektem se musí přejít k implementaci spuštění zařízení *Kinect* a ke sběru potřebných dat, které budou základem pro detekci uživatele a rozpoznání gest. Po úspěšném implementování

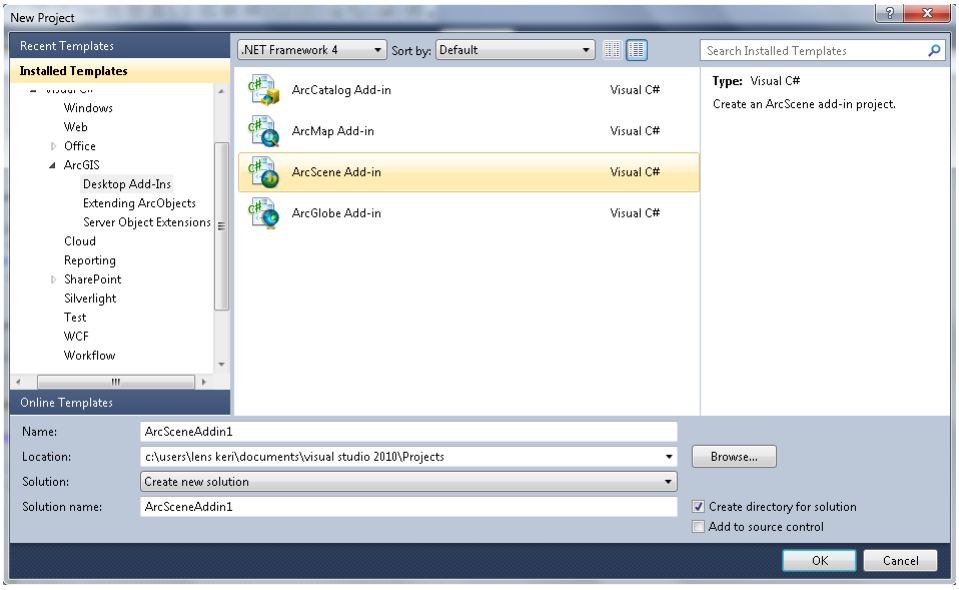
spuštění zařízení *Kinect* a sběru dat, se musí implementovat detekce uživatele. Rozpoznání uživatele by mělo zvládat i více uživatelů ve scéně a měl by být umožněn výběr jednoho z nich pro manipulaci se systémem. Po detekování a výběru jednoho uživatele by se měli implementovat navržená gesta. K implementaci navržených gest by se mělo přistupovat podle využití. Jako první by se mělo implementovat gesto pro translaci (posun) objektu a po něm všechna ostatní. Při implementaci gest i při jejich návrhu by se mělo dbát na kolizi při provádění gest. To znamená, aby gesta nebyla natolik podobná, že by je bylo obtížné rozpoznat od sebe navzájem. Implementace gest by neměla příliš zpomalovat vykreslování či jiné výpočty v prostředí. Po úspěšné implementaci navržených gest se musí dbát i na selhání připojení zařízení *Kinect* či vypnutí softwarové komponenty. Musí se zajistit stálá plynulost prostředí.

Testovat navržené řešení
Navržené a implementované řešení je třeba otestovat. Testovat je potřeba nasazení softwarové komponenty do zvoleného 3D prostředí. Nasazení musí proběhnout bez jakýchkoli problémů. Testovat se musí správná detekce uživatele i s více uživateli ve scéně. Dále je potřeba testovat zejména správné navržení gest. Po zjištění špatného rozpoznávání, kde je například příliš kolizí s jinými gesty, by se měl návrh a implementace gest zopakovat. Musí se otestovat spouštění, průběh a vypnutí softwarové komponenty. Při těchto akcích je nutné odchytávat veškeré výjimky a zařídit plynulost prostředí. Při detekci výjimky ovlivňující chod softwarové komponenty se musí softwarová komponenta vypnout.

4 Praktická část

4.1 Použité nástroje

Hlavním nástrojem pro zpracování této diplomové práce a pro dosažení jejího cíle je zařízení *Kinect*. Program či prostředí, který umí manipulovat s 3D objekty, byl po konzultaci s vedoucím práce zvolen *ArcScene*. Program *ArcScene* je aplikace systému *ArcGIS*. Systém *ArcGIS* umožňuje vytvářet a používat mapy, spravovat geografická data, analyzovat mapové informace a mnoho dalšího. Aplikace *ArcScene* umožňuje vizualizaci 3D mapy/modelu na základě modelu/modelů nebo geografických dat, manipulovat s touto 3D mapou/modelem a mnoho dalšího. Aplikace *ArcScene* byla vybrána pro možnost jednoduché softwarové komponenty typu *Add-in* a také kvůli rozšířenosti celého systému *ArcGIS*. Tento *Add-in* funguje na základě tzv. *ArcGIS Engine*, což je produkt pro vývojáře pro vytvoření GIS aplikací pro desktop platformy. *ArcGIS Engine* zahrnuje hlavní množinu komponent nazvanou *ArcObjects*. V této množině lze najít například komponentu *Button*, *Tool*, apod. Pro vytvoření softwarové komponenty byla zvolena komponenta *Tool* z *ArcObjects*. Prostředí *ArcGIS* bylo zvoleno také z důvodu kvalitní dokumentace pro vývojáře i kvalitní množství vzorů řešení. *ArcGIS Engine* lze instalovat při instalaci celého systému *ArcGIS*. Pro vývoj softwarové komponenty bylo použito prostředí *MS Visual Studio 2010*. Při instalaci *ArcGIS Engine* se vytvořilo nastavení pro tvorbu projektů či komponent jako je *Add-in* přímo do prostředí *MS Visual Studio 2010*. Při vytváření softwarové komponenty stačilo vytvořit projekt v prostředí *MS Visual Studio 2010* pro zvolenou aplikaci *ArcScene* viz obrázek 26.



Obrázek 26: Vytvoření softwarové komponenty Add-in pro aplikaci ArcScene. Zdroj: Bc. Martin Čapek

Vytvoření projektu pro *Add-in* provází podrobný *wizard*. *ArcGIS Engine* podporuje celou řadu programovacích jazyků jako je *C# .NET, Java, C++*. Podle této podpory byla zvolena knihovna pro práci se zařízením *Kinect* a to *Kinect SDK*. Popis knihovny lze najít výše v kapitole *Kinect SDK*. Při sestavování softwarové komponenty *Add-in* se sama nasazuje do systému *ArcGIS*. Softwarová komponenta lze nasadit do systému *ArcGIS* i na jiném počítači. Popis nasazení softwarové komponenty *Add-in* je popsáno níže v podkapitole Nasazení softwarové komponenty do systému *ArcGis - ArcScene*.

4.2 Implementace softwarové komponenty

4.2.1 Překryté (override) metody komponenty Tool z ArcObjects

Při kliknutí na vytvořený *Tool* se spustí metoda *OnActivate()*. Při spuštění se vytvoří nová kamera, do které se vloží kamera aktuálního pohledu scény, nastaví se projekce a jako poslední se spustí zařízení *Kinect*. Metoda *StartSensor()* je popsána dále.

```
protected override void OnActivate()
{
  base.OnActivate();
  // nová instance kamery
  cam = new CameraClass();
  // předaní kamery z aktuálního pohledu
  cam = (ICamera3)ArcScene.Scene.SceneGraph.CurrentViewer.Camera
      ;
  // předání celé scény ArcScene
  scene = ArcScene.Scene.SceneGraph;
  scene.SetupProjection(cam); // nastavení projekce
  StartSensor(); // spuštění zařízení Kinect
}
```

Další metoda, která se překryje sama při vytvoření projektu *Add-in* pro *ArcScene* je *OnUpdate()*. Jedná se o metodu, která se spustí při spuštění aplikace. Tato metoda zajišťuje stav *enable* pro daný *Tool*, tzn. na daný *Tool* lze kliknout a používat ho.

```
protected override void OnUpdate()
{
  Enabled = ArcScene.Application != null;
}
```

Při implementaci je velmi důležité si tyto metody neplést. Spuštění zařízení *Kinect* není dobré provést v metodě *OnUpdate()*, protože se zařízení nepoužívá. Při testování se zařízení *Kinect* spouštělo zároveň v obou metodách a kvůli tomu aplikace nefungovala správně a po nějaké době se ukončila.

Poslední metodu, kterou je důležité překrýt je metoda *OnDeactivate()*. Pomocí této metody se ukončí veškeré sledování pomocí zařízení *Kinect* a je možné aplikaci dále

používat, případně znovu zapnout ovládání pomocí zařízení *Kinect*. Metoda zjišťuje, jestli zařízení *Kinect* v aplikaci je přiřazeno, pokud je zařízení přiřazeno, tak se vypne.

```
protected override bool OnDeactivate()
{
  if (this.sensor != null)
  this.sensor.Stop(); // zastavení zařízení Kinect
  return base.OnDeactivate();
}
```

Lze přepsat ještě další metody například *OnKeyUp()* nebo *OnMousePressed()*. Metoda *OnKeyUp()* byla implementována pouze pro testování funkcí pro manipulaci 3D mapy, ale vzhledem k zaměření této práce nebude popsána.

4.2.2 Spuštění zařízení Kinect

Metoda *StartSensor()* zajišťuje přiřazení detekovaného zařízení *Kinect*, které následně bude ovládat manipulaci 3D mapy. V první části metody se prochází potencionální zařízení, a pokud je nějaké zařízení *Kinect* připojeno a funkční, tak se uloží do proměnné *sensor*. Pokud není připojeno žádné zařízení, tak se komponenta *Tool* deaktivuje.

```
private void StartSensor()
{
  foreach (var potentialSensor in KinectSensor.KinectSensors)
  {
    if (potentialSensor.Status == KinectStatus.Connected)
    {
      // vybrání prvního zapojeného zařízení Kinect
      this.sensor = potentialSensor;
      break;
    }
  }

  if (this.sensor == null)
  {
    MessageBox.Show("zadny kinect v dosahu");
  }
```

Pro zajištění správné funkčnosti detekování a sledování uživatele je potřeba spustit datové proudy pro hloubkové mapy (*DepthStream*) a pro sledování kostry člověka (*SkeletonStream*). Proud pro hloubkové mapy potřebuje rozlišení, ve kterém má snímat obraz. Rozlišení si vývojář může vybrat dle dostupného formátu *DepthImageFormat*. Vybrané rozlišení má vliv na rychlost a přesnost detekce člověka, závisí i na hardwarových parametrech počítače, na kterém je aplikace spuštěna. Proud pro sledování kostry člověka může přebírat parametr *TransformSmoothParameters*. Jedná se o nastavení vyhlazování pozice kloubů (*joints*) podle daných parametrů. Pokud se při spuštění *SkeletonStream* neuvede parametr, tak se použije výchozí nastavení pro vyhlazování. Nastavení výchozího vyhlazování lze vidět v tabulce 2.

Tabulka 2: Nastavení výchozího vyhlazování

Parametr / překlad	Popis	Hodnota
Smoothing / vyhlazení	Zvýšení hodnoty parametru vede k lepšímu vyhlazení skeletonu.	0.5f
Correction / korekce	Nižší hodnoty zajišťují vyšší korekci a lepší vyhlazení, ale zhoršení výkonu.	0.5f
Prediction / predikce	Hodnota = počet snímku pro predikci.	0.5f
JitterRadius / poloměr chvění	Hodnota je v metrech pro redukci chvění. Jakékoli chvění větší než daný poloměr je připnuto k poloměru.	0.05f
MaxDeviationRadius / Maximální odchylka poloměru	Maximální poloměr v metrech, který určuje, jak moc se můžou filtrovaná data lišit od nezpracovaných dat.	0.04f

Po pozitivním spuštění proudů se u každého z nich nastaví mód vzdálenosti snímání. Povolená vzdálenost snímání záleží na připojeném zařízení *Kinect*, jestli se jedná o *Kinect for Xbox* nebo o *Kinect for Windows*. Pokud se jedná o *Kinect for Windows*, tak se vzdálenost snímání nastaví na mód *Near*.

```
try
{
  // spuštění proudů DepthStream a SkeletonStream
  this.sensor.DepthStream.Enable(DepthImageFormat.
     Resolution640x480Fps30);
  this.sensor.SkeletonStream.Enable();

  try
  {
    // nastavení módu Near pro Kinect for Windows
    this.sensor.DepthStream.Range = DepthRange.Near;
    this.sensor.SkeletonStream.EnableTrackingInNearRange = true;
  }
  catch (InvalidOperationException)
  {
    // nastavení módu Default pro Kinect for Xbox
    this.sensor.DepthStream.Range = DepthRange.Default;
    this.sensor.SkeletonStream.EnableTrackingInNearRange = false
       ;
  }
}
catch (InvalidOperationException)
```

```
{
  OnDeactivate(); // po chybě v bloku try deaktivuje komponentu
}
```

Před spuštěním zařízení se ještě musí zaregistrovat události pro detekování kostry člověka *SkeletonStreamReady* a pro reakce na gesto mávání *GestureRecognized*. Událost *SkeletonStreamReady* je popsána v podkapitole o sběru dat ze zařízení *Kinect*. Události *GestureRecognized* jsou popsány v podkapitole Gesto mávání.

```
// registrování události pro proud SkeletonStream
this.sensor.SkeletonFrameReady += this.SensorSkeletonFrameReady;
// registrování události pro gesto mávání levou rukou
m_waveLeftHand.GestureRecognized += new EventHandler(
    m_waveLeftHand_GestureRecognized);
// registrování události pro gesto mávání pravou rukou
m_waveRightHand.GestureRecognized += new EventHandler(
    m_waveRightHand_GestureRecognized);
```

Spuštění toků a nastavení snímací vzdálenosti je potřeba implementovat v příkazu *try* a odchytávat výjimku *InvalidoperationException* z důvodu, že se spouštění proudů nemusí zdařit a aplikace by se mohla neočekávaně ukončit.

V poslední fázi metody *StartSensor()* se zařízení pokusí spustit, pokud se nespustí, tak se komponenta *Tool* deaktivuje.

```
try
{
 this.sensor.Start(); // spuštění zařízení Kinect
}
catch (IOException)
{
  this.sensor = null;
  MessageBox.Show("nepodarilo se nastartovat kinect");
  OnDeactivate(); // po nezdařilém spuštění deaktivuje
      komponentu
}
```

4.2.3 Sběr dat ze zařízení Kinect

Pro sběr dat ze zařízení *Kinect* se používají datové proudy. Datové proudy jsou pro: *SkeletonStream*, *DepthStream*, *ColorStream*, *InteractionStream*. V této práci byly použity všechny kromě *ColorStream*. Jako první je popsán proud *SkeletonStream* a jeho zaregistrovaná událost *SensorSkeletonFrameReady*, jako druhý je popsán proud *DepthStream*, který nemá událost, ale má implementovanou metodu *ProcessDepthFrame*(*long skeletonTimestamp*), jako poslední je popsán proud *InteractionStream* a jeho zaregistrovaná událost *InteractionFrameReady*. Pro správné detekování *skeletona* s interakcemi je zapotřebí snímek pořízený ve stejný čas, tzn. proudy *SkeletonStream* a *DepthStream* musí pracovat právě se stejným časovým snímkem pro zajištění správného chodu programu. Při implementaci nastal problém

s proudem *DepthStream*. Jakmile se pro něj zaregistrovala událost, tak snímky neměli stejnou časovou hodnotu a aplikace se neočekávaně ukončila. Řešením je vytvoření metody *ProcessDepthFrame*, které se předal časový parametr snímku z proudu *SkeletonStream*.

SkeletonStream a událost SensorSkeletonFrameReady

Spuštění a nastavení proudu *SkeletonStream* je popsáno v metodě *StartSensor()*. Tento proud zajišťuje detekci pozice jednotlivých kloubů a vazeb kostry člověka i s úhly mezi klouby. Pozice jsou v trojrozměrném souřadném systému, tudíž jsou k dispozici tři souřadnice: na ose X, na ose Y a na ose Z. Jakmile zařízení *Kinect* rozpozná kostru nějakého člověka, tak se vyvolá událost *SensorSkeletonFrameReady*, která příjme v druhém parametru (*SkeletonFrameReadyEventArgs e*) data o všech rozpoznaných kostrách.

```
private void SensorSkeletonFrameReady(object sender,
    SkeletonFrameReadyEventArgs e);
```

Podobný problém jako u proudu *DepthStream* nastal u proudu *InteractionStream*. Jakmile se vytvořil a spustil v metodě *StartSensor()* jako ostatní proudy, tak aplikace nefungovala správně a interakce se nechtěly rozpoznat. Řešením je vytvoření, spuštění a zaregistrování události až v události *SensorSkeletonFrameReady*. Všechny tyto akce se provedou, jen pokud je proměnná *interactionStream* prázdná.

```
if (interactionStream == null)
{
  // vytvoření proudu InteractionStream
  interactionStream = new InteractionStream(this.sensor, new
      MyInteractionClient());
  // zaregistrování události proudu InteractionStream
  interactionStream.InteractionFrameReady += new EventHandler<
      InteractionFrameReadyEventArgs>(
      interactionStream_InteractionFrameReady);
}
```

Po vyřešení proudu *InteractionStream* se vynuluje pomocné pole *skeletons* typu *Skeleton[]*. V sekci *using* se otevře aktuální snímek, ze kterého se zkopírují všechna data o kostrách do pole *skeletons*. Pro každou rozpoznanou kostru se provede rozpoznání interakcí s pomocí hloubkových map. Vzhledem k použití *Kinect SDK 1.7* lze detekovat šest lidí, ale sledovat pouze dvě kostry.

```
skeletons = null;
using (SkeletonFrame skeletonFrame = e.OpenSkeletonFrame())
{
  if (skeletonFrame != null)
  {
    // nové pole detekovaných skeletonů
    skeletons = new Skeleton[skeletonFrame.SkeletonArrayLength];
    // kopírování dat o skeletonech do pole
    skeletonFrame.CopySkeletonDataTo(skeletons);
```

```
        // poslání dat o skeletonech z proudu SkeletonStream do
           proudu
        interactionStream.ProcessSkeleton(skeletons, this.sensor.
           AccelerometerGetCurrentReading(), skeletonFrame.
           Timestamp);  InteractionStream
        // zavolání metody pro zpracování hloubkového snímku
        ProcessDepthFrame(skeletonFrame.Timestamp);
      }
    }
```

Po zpracování všech proudů nastává vybrání nejbližšího uživatele, na kterém se testuje viditelnost všech potřebných části těla pro rozpoznání gest. Pokud jsou všechny potřebné části viditelné, tak se přejde k rozpoznávání implementovaných gest.

```
// vybrání nejbližšího skeletona
m_closestSkeleton = skeletons.Where(s => s.TrackingState ==
    SkeletonTrackingState.Tracked).OrderBy(s => s.Position.Z).
    FirstOrDefault();

if (m_closestSkeleton == null)
  return; // pokud není detekován žádný skeleton, metoda končí

// testování viditelnosti potřebných částí těla
if (CheckJointsForTracking(m_closestSkeleton))
{
  ProcessGestures(); // zavolání metody pro detekci gest
}
```

DepthStream a metoda ProcessDepthFrame

Proud *DepthStream* zajišťuje hlavní detekci uživatele. Pomocí tohoto proudu lze detekovat uživatele i jeho interakce. V kombinaci s proudem *SkeletonStream* představují vstupní data do proudu *InteractionStream* pro rozpoznání sevření ruky. Jak už bylo řečeno dříve, proud *DepthStream* musí poslat do proudu *InteractionStream* data se stejnou časovou složkou jako proud *SkeletonStream*. To byl hlavní důvod vytvoření metody *ProcessDepthFrame* místo události. Data z proudu *DepthStream* obsahují hloubkové informace nejbližších objektů před zařízením *Kinect* a pomocí tzv. *player segmentation* přiřadí indexy rozpoznaným uživatelům. Díky této segmentaci lze vybrat a sledovat dvě kostry uživatele ve stejný čas. Metoda *ProcessDepthFrame* se stará o zpracování hloubkových informací pořízené ze zařízení *Kinect* a posílá je do proudu *InteractionStream* pro další zpracování.

```
private void ProcessDepthFrame(long skeletonTimestamp)
{
  using (DepthImageFrame depthframe = this.sensor.DepthStream.
     OpenNextFrame(0))
  {
    if (depthframe != null)
    {
      // vytvoření pole pixelů
```

```
            DepthImagePixel[] dPixels = new DepthImagePixel[depthframe
                .PixelDataLength];
            // kopírování dat z proudu do pole pixelů
            depthframe.CopyDepthImagePixelDataTo(dPixels);
            // poslání hloubkové mapy z proudu DepthStream do proudu
                InteractionStream
            interactionStream.ProcessDepth(dPixels, skeletonTimestamp)
                ;
        }
      }
    }
```

InteractionStream a událost InteractionFrameReady

Tento proud je nový od verze *Kinect SDK 1.7* v knihovně *Toolkit.Interaction*, který je použit v této práci. Slouží pro rozpoznání efektu tlačítka a sevření rukou. V této práci bylo použito pouze rozpoznání sevření ruky. Pro rozpoznání efektu tlačítka, nebo sevření ruky potřebuje proud *InteractionStream* výstupní data z proudů *SkeletonStream* i *DepthStream*. Pokud nemá oba se stejnou časovou hodnotou, tak se aplikace neočekávaně ukončí. Při vytvoření proudu *InteractionStream* je nutné implementovat třídu, která dědí z rozhraní *IInteractionClient*. Tato třída musí implementovat metodu rozhraní *GetGetInteractionInfoAtLocation*, která vrací informace o interakcích daného uživatele.

```
public class MyInteractionClient : IInteractionClient
{
  public InteractionInfo GetInteractionInfoAtLocation(int
      skeletonTrackingId, InteractionHandType handType, double x
      , double y)
  {
    return new InteractionInfo
    {
      IsPressTarget = false, // nerozpoznává efekt tlačítka
      IsGripTarget = true, // rozpoznává sevření ruky
    };
  }
}
```

Událost *InteractionFrameReady* lze rozdělit na dvě části. První část prochází zpracovaná data proudem. Tyto data popisují interakce jednotlivých uživatelů pro každou jejich ruku. Pro jednotlivé uživatele se data o interakci uloží do pole *userInfos* typu *UserInfo[]*.

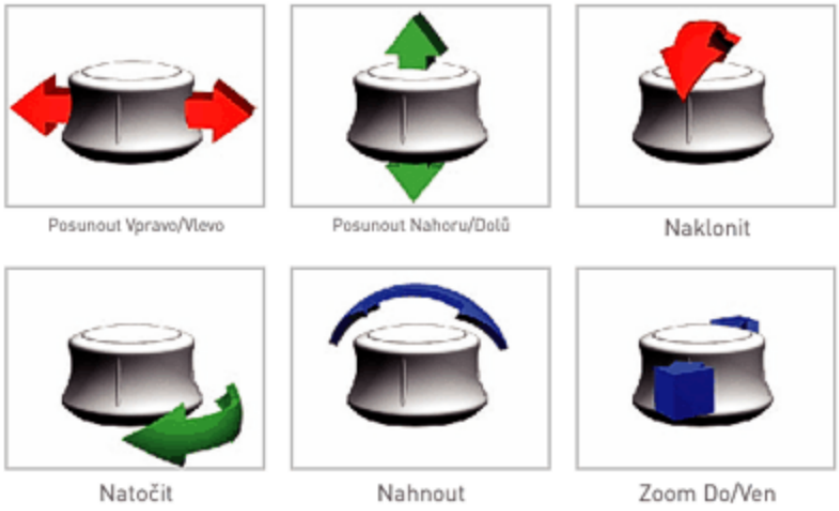
```
using (InteractionFrame frame = e.OpenInteractionFrame())
{
  if (frame != null)
  {
    if (this.userInfos == null)
    {
      // vytvoření pole pro data interakcí jednotlivých
          skeletonů
```

```
            this.userInfos = new UserInfo[InteractionFrame.
                UserInfoArrayLength];
          }
          // kopírování dat z proudu InterationStream do pole
              interakcí
          frame.CopyInteractionDataTo(this.userInfos);
        }
        else
        {
          return;
        }
      }
```

Druhá část události prochází pole *userInfos* a nastavuje hodnoty proměnných *m_leftGrip*, *m_rightGrip* typu *bool*, které určují sevření jednotlivých rukou.

4.2.4 Návrh gest a emulované funkce 3D myši

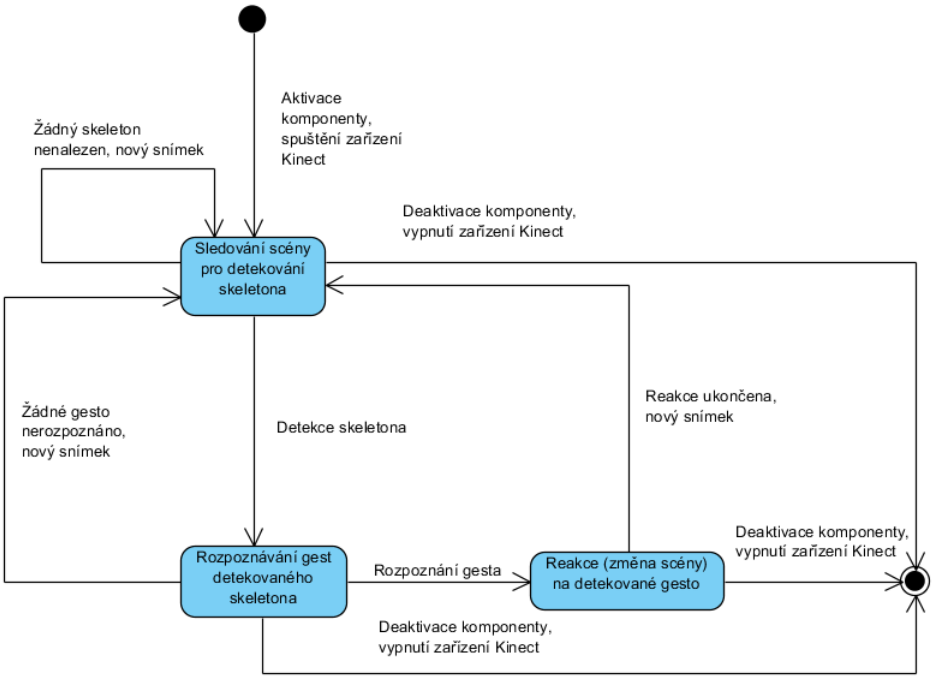
Návrh gest pro manipulaci 3D mapy v programu *ArcScene* se zakládají na funkcionalitě 3D myši, která již s tímto programem pracuje. Funkcionality 3D myši jsou následující: rotace ve všech osách, posun nahoru, posun dolů, posun doleva, posun doprava a změna měřítka. Selekci čili vybrání určitého objektu ve scéně zajišťuje vybrané prostředí pro manipulaci 3D objektů *ArcScene*, ve kterém je možné manipulovat pouze jedním objektem (scénou), který se vybere při otevření projektu. Tuto funkcionalitu znázorňuje obrázek 27.



Obrázek 27: Funkcionality 3D myši. Zdroj: http://www.cadstudio.cz/prod/3dmouse.asp

Po nastudování předešlých metod pro klasifikaci gest, byla zvolena metoda FSM. Jedná se o jednoduchou metodu, kde hlavní roli hrají stavy, vstupy a události. Stavy určují sekci v kódu, která se právě provádí. Výchozí stav je sledování scény pro detekování *skeletona*. Tento stav se provádí stále dokola, pokud není detekovaný

žádný uživatel. Po detekci *skeletona* se přejde do stavu rozpoznávání gest. Tento stav provádí rozpoznávání všech implementovaných gest. Jakmile se rozpozná alespoň jedno gesto, tak se přejde do stavu reakce na detekované gesto. V tomto stavu se testují pozice potřebných částí těla pro nastavení pomocných proměnných, které slouží jako parametry pro provedení dané reakce. V jakémkoli stavu lze deaktivovat softwarovou komponentu a tím vypnout zařízení *Kinect* a rozpoznávání gest. Stavový diagram použité metody FSM lze vidět na obrázku 28.



Obrázek 28: Stavový diagram systému pro rozpoznávání gest. Zdroj: Bc. Martin Čapek

V kapitole o proudu *SkeletonStream* je popsána událost *SensorSkeletonFrameReady*, ve které se volá po určitých splněných podmínkách metoda *ProcessGestures()*. Tato metoda zajišťuje výběr gest pro manipulaci 3D mapy v programu *ArcScene*. Funkcionalita 3D myši je převedena na gesta následovně:

- rotace ve všech osách → metoda rotation()
- posun (nahoru, dolů, doleva, doprava) → metoda moveXYZ()
- změna měřítka → zoom()

Do proměnné *m_gestures* typu *int* se ukládá právě prováděné gesto s výjimkou gesta mávání. Je to z důvodu podobnosti gest, které emulují funkcionalitu 3D myši. Proměnná *m_gestures* může nabývat pouze těchto hodnot:

- hodnota 0 → neprovádí se žádné gesto
- hodnota 1 → provádí se gesto translace nebo rozhlížení

- hodnota 2 → provádí se gesto pro změnu měřítka
- hodnota 3 → provádí se gesto rotace

Tato gesta mají podobné podmínky pro rozpoznání, tudíž pokud se detekuje například gesto posunu (metoda *moveXYZ()*), tak se další neprovádějí kvůli možné chybné detekci a šetření výkonu počítače.

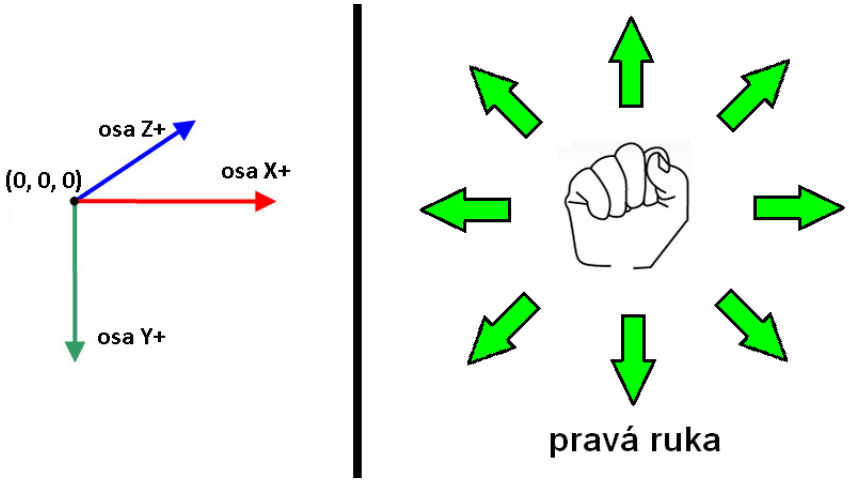
```
private void ProcessGestures()
{
  if (m_gestures == 0 || m_gestures == 1)
    moveXYZ(); // detekce gesta pro translaci
  if (m_gestures == 0 || m_gestures == 2)
    zoom(); // detekce gesta pro změnu měřítka
  if (m_gestures == 0 || m_gestures == 3)
    rotation(); // detekce gesta pro rotaci

  // detekce gesta mávání levou rukou
  m_waveLeftHand.Update(m_closestSkeleton, m_leftGrip);

  // detekce gesta mávání pravou rukou
  m_waveRightHand.Update(m_closestSkeleton, m_rightGrip);
}
```

Metoda *moveXYZ()* řeší posunutí v mapě po všech osách oproti funkcionalitě 3D myši, která objekt posunuje pouze v ose X a Y. Metoda *moveXYZ()* také umožňuje funkci rozhlížení, jak je tomu ve *First-Person* hrách, pouze v horizontálním směru. Podrobnější popis metody *moveXYZ()* lze najít v podkapitole Gesto pro translaci. Metoda *rotation()* řeší rotaci pouze v ose Y a X oproti funkcionalitě 3D myši, která dokáže objekt rotovat ve všech osách. Podrobnější popis metody *rotation()* lze najít v podkapitole Gesto pro rotaci. Reakce na gesta, které emulují funkcionalitu 3D myši mohou být statická, nebo dynamická. Tuto statiku a dynamiku určuje globální proměnná *dynamicGestures* typu *bool*. Popis reakcí je popsán v jednotlivých podkapitolách Gesto pro translaci, Gesto pro rotaci, Gesto pro změnu měřítka. Gesto mávání zajišťuje pomocnou funkcionalitu při manipulaci 3D mapy a má dvě použití viz podkapitola Gesto mávání. Je nutné dodat, že se sevření ruky vyskytuje ve dvou navržených gestech a to v metodě *moveXYZ()* a *rotation()*. Známá chyba při použití proudu *InteractionStream*, který zajišťuje detekci sevření rukou viz podkapitola Sběr dat ze zařízení *Kinect*, uvedená firmou Microsoft je, že detekování sevření levé ruky má horší úspěšnost než pravé ruky.

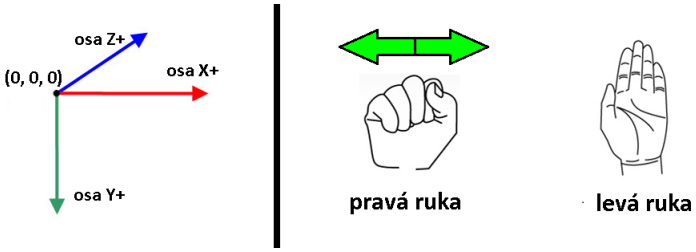
4.2.5 Gesto pro translaci a rozhlížení

První navržené a implementované gesto bylo pro translaci (posun) 3D mapy. Jako první bylo navrženo a implementováno z důvodu časté použitelnosti při manipulaci s 3D mapou. Častá použitelnost vedla k co nejjednoduššímu návrhu gesta pro jednoduché a intuitivní použití. Jak ve skutečném světě lze pohnout objekty sevřením ruky a posunutím na jiné místo, tak stejně funguje navržené gesto translace. Pro

rozeznání posunutí objektu či 3D mapy je potřeba stanovit určité podmínky. Pokud se člověk prochází, tak je přirozené mít ruce u těla. Pokud chce člověk něco vzít nebo přesunout, tak natáhne ruku, uchopí objekt a provede akci. Ve skutečnosti mohou být ruce stále u těla, jak je tomu při procházení. Tento problém je třeba eliminovat podmínkou, že uchopit objekt lze jen tehdy, pokud je ruka natažená. Z toho plyne hlavní podmínka: ruka musí být nad loktem. Pomocí určení této podmínky lze vidět, které části těla jsou potřeba pro detekci: loket a ruka. Tohle gesto je určeno pouze pro pravou ruku, z důvodu kombinace dalších gest a také z důvodu špatné detekce sevření levé ruky. Způsob provedení gest lze vidět na obrázku 29 a 30.



Obrázek 29: Gesto pro translaci. Zdroj: Bc. Martin Čapek



Obrázek 30: Gesto pro rozhlížení. Zdroj: Bc. Martin Čapek

Rozpoznání gesta pro translaci a rozhlížení řeší metoda *moveXYZ()*. Na začátku této metody se uloží potřebné části těla do lokálních proměnných. Jak již bylo řečeno, důležité části těla jsou: pravá ruka a pravý loket. Vzhledem k funkčnosti této metody, jsou potřeba ještě části: levá ruka a levý loket. První důvod je kvůli podmínce, kde se testuje, jestli jsou obě ruce nad svými lokty. Výsledky těchto podmínek se uchovávají v lokálních proměnných *left* a *right* typu *bool*. Při splnění podmínky, že jsou obě ruce nad svými lokty a levá ruka je sevřená (proměnná *m_leftGrip*), algoritmus končí a vynulují se všechny pomocné proměnné určující poslední pozici ruky a proměnnou *m_gestures*, určující právě prováděné gesto.

```
// deklarace potřebných částí těla a pomocných proměnných
var leftHand = m_closestSkeleton.Joints[JointType.HandLeft];
var rightHand = m_closestSkeleton.Joints[JointType.HandRight];
var leftElbow = m_closestSkeleton.Joints[JointType.ElbowLeft];
var rightElbow = m_closestSkeleton.Joints[JointType.ElbowRight];
bool left = false;
bool right = false;

// testování pozic rukou
if (leftHand.Position.Y > leftElbow.Position.Y)
  left = true;
if (rightHand.Position.Y > rightElbow.Position.Y)
  right = true;
if ((left && m_leftGrip) && right)
{
  m_lastPos.x = 0.0;
  m_lastPos.y = 0.0;
  m_lastPos.z = 0.0;
  m_gestures = 0;
  return;
}
```

Po úspěšném splnění podmínek, že jsou obě ruce nad svými lokty a levá ruka není sevřená, se algoritmus dostane k další podmínce, kde se testuje sevření pravé ruky. Pokud není pravá ruka sevřená, opět se všechny pomocné proměnné vynulují. Po splnění této podmínky se nastaví proměnná *m_gestures* na hodnotu 1 a testuje se, jestli poslední pozice pravé ruky je nulová. Pokud je poslední pozice nulová, tak se nastaví na aktuální a metoda končí. Pokud poslední pozice ruky není nulová, tak se přejde na nastavení dalších lokálních proměnných: *x*, *y*, *z*, *pom*. Proměnné *x*, *y* a *z* určují rozdíl aktuální pozice od poslední pozice pravé ruky. Proměnná *pom* je pomocná proměnná, která je použitá při posunu pouze po ose Z. Podle hodnoty výše zmíněné proměnné *dynamicGestures*, která určuje, zda reakce na gesta mají být dynamická či nikoli, se přepíše proměnná *pom* a proměnná z. Dynamika tohoto gesta spočívá pouze v posunu po ose Z. Pokud gesto začne a je dynamické, tak se do proměnné *z* ukládá rozdíl mezi úplně první detekovanou pozicí pravé ruky a aktuální pozicí pravé ruky. Tzn. jakmile se pravá ruka vzdaluje od první pozice, tak se posun po ose Z zrychluje.

```
// testování sevření pravé ruky pro gesto translace
if (right && m_rightGrip)
{
  m_gestures = 1;
  if (m_lastPos.x != 0.0 && m_lastPos.y != 0.0 && m_lastPos.z !=
      0.0)
  {
    // pomocné proměnné pro reakci na gesto translace
    double x = m_lastPos.x - (double)rightHand.Position.X;
    double y = m_lastPos.y - (double)rightHand.Position.Y;
    double z = m_lastPos.z - (double)rightHand.Position.Z;
    int pom = 105;
```

```
      if (dynamicGestures)
      {
        z = m_moveZStart - (double)rightHand.Position.Z;
        pom = 10;
      }
      ...
    }
    ...
  }
```

Po nastavení všech pomocných a lokálních proměnných se přechází na konečnou reakci pro dané gesto. Reakce jsou dvě a záleží po již splněných podmínkách na tom, pokud je levá ruka nad levým loktem. Pokud je jen pravá ruka nad svým loktem a levá ruka je u těla, tak se provede výše zmíněný posun po všech osách X, Y a Z zavoláním pomocné metody *Move()* popsané dále. Druhá reakce při splněných podmínkách, že jsou obě ruce nad svými lokty a pravá ruka je sevřená, je tzv. „rozhlížení“ po scéně v horizontální rovině. K této reakci slouží metoda kamery *HTurnAround()*, která přijímá parametr úhlu, o který se má kamera otočit kolem své osy.

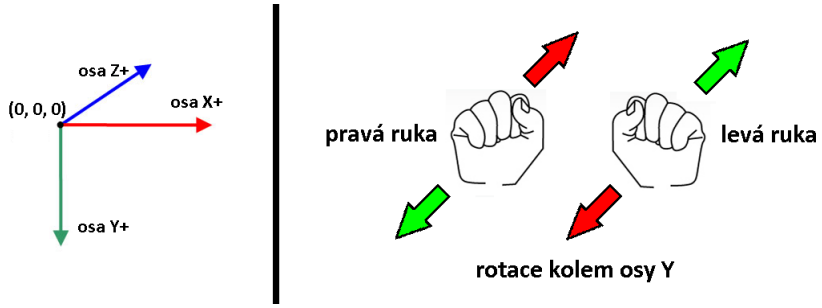
```
if (!left)
{
  // reakce na gesto translace
  Move(x, esriCameraMovementType.esriCameraMoveLeft,
      esriCameraMovementType.esriCameraMoveRight, 105);
  Move(y, esriCameraMovementType.esriCameraMoveDown,
      esriCameraMovementType.esriCameraMoveUp, 105);
  Move(z, esriCameraMovementType.esriCameraMoveAway,
      esriCameraMovementType.esriCameraMoveToward, pom);
}
else
{
  int angle = 0;
  if (x > 0.0)
    if (x > 0.005)
      angle = 5;
  else
    if (x < -0.005)
      angle = -5;

  // reakce na gesto pro rozhlížení
  cam.HTurnAround(angle);
}
```

Pomocná metoda *Move()* čtyři parametry, které určují posun 3D mapy. První parametr *axis* typu *double* určuje o kolik se má mapa posunout na dané ose. Druhý a třetí parametry typu *esriCameraMovementType* určují směr (osu), po které se mapa posune. Poslední parametr *ratio* typu *int* je pomocné číslo, kterým se násobí posun. Blok celé metody se skládá z podmínky, která určuje, o kolik se ruka posunula. Podle kladnosti parametru *axis* se určí i směr a délka posunutí.

```
private void Move(double axis, esriCameraMovementType direction1
    , esriCameraMovementType direction2, int ratio)
{
  // pomocná metoda pro reakci na gesto translace
  if (axis > 0.0)
    if (axis > 0.005)
     cam.Move(direction1, 0.005 * (ratio * axis));
  else
    if (axis < -0.005)
      cam.Move(direction2, 0.005 * (ratio * axis * -1));
}
```

4.2.6 Gesto pro rotaci

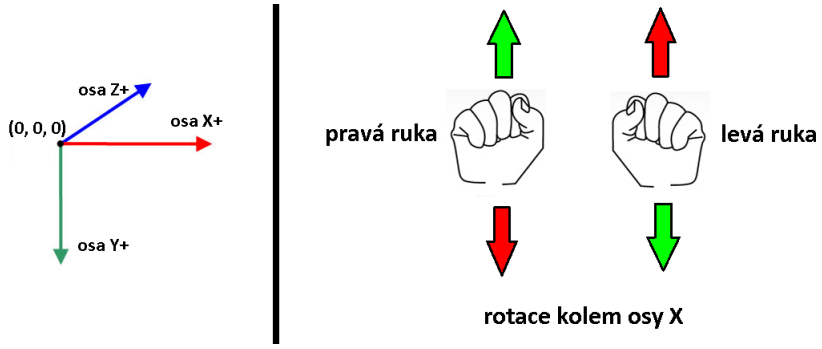
Vzor návrhu gesta pro rotaci lze najít také ve skutečném světě. Jakmile člověk potřebuje otočit nějakým objektem, tak se objekt sevře do obou rukou a následně jím otočí v určité ose. Ve skutečném světě záleží na velikosti objektu, pokud by byl malý a lehký, tak lze objekt uchopit a otočit pouze jednou rukou. Jednu ruku již není možné v navrženém systému použít pro jednoznačnou detekci. Vzorem pro rozpoznání gesta je tedy uchopení a natočení objektu oběma sevřenýma rukama. Proto je nutné detekovat a sledovat obě ruce. Pro správné provedení gesta je nutné mít obě ruce nad svými lokty. Tato samotná podmínka nesplňuje návrh, takže je potřeba detekovat i sevření obou rukou. Tyto dvě podmínky již splňují návrh gesta, který se podobá provedení ve skutečném světě. Provedení gesta lze vidět na obrázcích 31 a 32.



Obrázek 31: Gesto pro rotaci kolem osy Y. Zdroj: Bc. Martin Čapek

Rozpoznání gesta pro rotaci řeší metoda *rotation()*. Na začátku této metody se uloží potřebné části těla do lokálních proměnných. Důležité části těla jsou: obě ruce a oba lokty. Po uložení potřebných částí těla se testují podmínky pro každou ruku, jestli je každá nad svým loktem a jestli jsou obě sevřené. Pokud podmínky neuspějí, tak se pomocné proměnné *m_rotY*, *m_rotX* a *m_gestures* vynulují. Proměnná *m_rotY* a *m_rotX* jsou typu *double* a určují poslední rozdíl mezi souřadnicemi mezi oběma rukama, popsáno dále.

```
// deklarace potřebných částí těla a pomocných proměnných
```



Obrázek 32: Gesto pro rotaci kolem osy X. Zdroj: Bc. Martin Čapek

```
var leftHand = m_closestSkeleton.Joints[JointType.HandLeft];
var rightHand = m_closestSkeleton.Joints[JointType.HandRight];
var leftElbow = m_closestSkeleton.Joints[JointType.ElbowLeft];
var rightElbow = m_closestSkeleton.Joints[JointType.ElbowRight];
bool left = false;
bool right = false;

// testování pozic rukou
if (leftHand.Position.Y > leftElbow.Position.Y && m_leftGrip)
  left = true;
if (rightHand.Position.Y > rightElbow.Position.Y && m_rightGrip)
  right = true;
if (!left || !right)
{
  m_rotY = 0.0;
  m_rotX = 0.0;
  m_gestures = 0;
  return;
}
```

Po splnění hlavních podmínek pro provedení gesta se nastaví proměnná *m_gestures* na hodnotu 3. Přejde se k testování pomocné proměnné *m_rotY*. Pokud je nulová, tak se nastaví na rozdíl mezi souřadnicemi pravé a levé ruky na ose Z. Pokud proměnná *m_rotY* není nulová, tak se přejde k nastavení pomocných proměnných *y*, *x*, *azimuth* a *inclin*, které určují vlastnosti reakce.

```
m_gestures = 3;
if (m_rotY != 0.0)
{
  // pomocné proměnné pro reakci na gesto rotace
  double y = (double)rightHand.Position.Z - (double)leftHand.
     Position.Z;
  double x = (double)rightHand.Position.Y - (double)leftHand.
     Position.Y;
  double azimuth = 0.0;
  double inclin = 0.0;
  ...
}
```

Příklad reakce je popsán pouze pro osu Y, tzn. rotace kolem osy Y. Reakce pro rotaci kolem osy X je stejná až na použitou proměnnou pro její stupeň otočení. Proměnná *azimuth* určuje otočení kolem osy Y a proměnná *inclin* určuje otočení kolem osy X. Již lze vidět dynamičnost reakce na gesto. Pokud je proměnná *dynamicGestures* hodnoty *true*, tak se otáčení zrychluje a zpomaluje podle velikosti výše zmíněných rozdílu mezi souřadnicemi rukou.

```
// výpočet pomocných proměnných pro reakci na gesto rotace
if (y > m_rotY + 0.007)
{
  if (dynamicGestures)
    azimuth = y * 10;
  else
    azimuth = 5;
}
else
  if (y < m_rotY - 0.007)
  {
    if (dynamicGestures)
      azimuth = y * 10;
    else
      azimuth = -5;
  }
```

Po nastavení všech proměnných pro rotaci se zavolá metody kamery *PolarUpdate()*. Tato metoda přijímá čtyři parametry. První parametr je typu *double* a určuje faktor vzdálenosti kamery od objektu. Faktor vzdálenosti v tomto případě znamená, že když je parametr nastaven na hodnotu 1, tak se vzdálenost nemění. Druhý parametr je též typu *double* a určuje otočení kolem osy Y daného objektu. Jinak řečeno, kamera se otočí kolem osy Y celé 3D mapy. Třetí parametr je také typu *double* a určuje otočení kolem osy X. Stejně jako u rotace kolem osy Y se kamera otáčí kolem osy X celé 3D mapy. Poslední čtvrtý parametr je typu *bool* a určuje omezení pro rotaci kolem osy X. Pokud je parametr nastaven na hodnotu *true*, tak lze otočit kameru kolem osy X jen o 180°. Po zavolání metody *PolarUpdate()* je dobré překreslit celou scénu v aktuálním pohledu metodou *Redraw()*. Tato metoda přijímá parametr typu *bool*, který určuje zda se kamera pohnula či nikoli.

```
// reakce na gesto rotace
cam.PolarUpdate(1, azimuth, inclin, true);
scene.CurrentViewer.Redraw(true);
```

Hlavní dynamičnost reakce na gesto rotace je změna role proměnných *m_rotY* a *m_rotX*. Pokud je proměnná *dynamicGestures* hodnoty *true*, tak se proměnné *m_rotY* a *m_rotX* změní z posledního rozdílu souřadnic na úplně první. Pokud je v tom případě aktuální rozdíl větší než úplně první, tak se po každém pozitivním rozpoznání gesta pro rotaci kamera otočí. S větším rozdílem se rychlost otáčení zvětšuje.

```
if (!dynamicGestures)
{
```

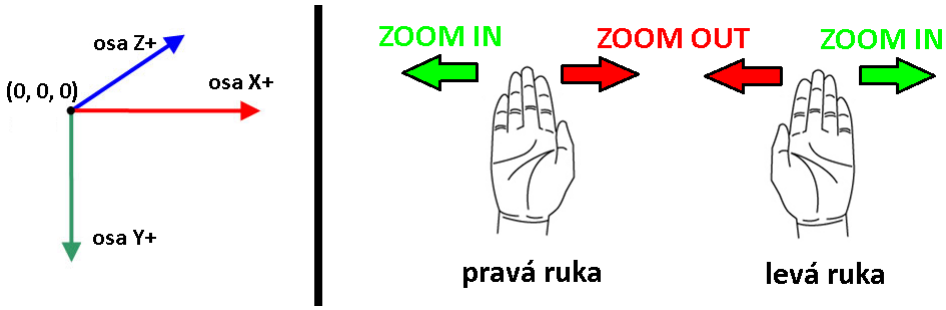
```
        m_rotY = y;
        m_rotX = x;
    }
```

Pokud je proměnná m_rotY nulová, jak je zmíněno výše, tak se obě proměnné m_rotY a m_rotX nastaví rozdílem souřadnic. Pro proměnnou m_rotY je to rozdíl souřadnic pravé a levé ruky na ose Z. Pro proměnnou m_rotX je to rozdíl souřadnic pravé a levé ruky na ose Y.

```
m_rotY = (double)rightHand.Position.Z - (double)leftHand.
    Position.Z;
m_rotX = (double)rightHand.Position.Y - (double)leftHand.
    Position.Y;
```

4.2.7 Gesto pro změnu měřítka

Vzor pro návrh gesta, které mění měřítko objektu, lze najít u mobilních aplikací. Pro uživatele je jednoduché a intuitivní pro použití. Jedná se o tzv. „zoom efekt“. Na mobilních platformách gesto funguje se dvěma prsty. Uživatel je odtahuje nebo přibližuje. Při odtahování prstů se objekt (obrázek v mobilní aplikaci) zvětšuje a při přibližování prstů se naopak zmenšuje. Zařízení *Kinect for Xbox* ani *Kinect for Windows* nedokáže rozlišit jednotlivé prsty, tudíž přesná kopie gesta z mobilních aplikací není jednoduchá a kvůli tomu vznikla jiná alternativa. Místo prstů budou stačit obě ruce, které se budou chovat podobně jako dva prsty u mobilních platforem. Tzn. pro dané gesto bude nutné detekovat a sledovat obě ruce. Jak již bylo zmíněno výše, tak gesta, která emulují funkci 3D myši, mají podobné podmínky. Tohle gesto není výjimka. Pro správné rozpoznání gesta musí být obě ruce nad svým loktem. Při použití jen této podmínky se při testování vyskytl problém. Jakmile chtěl uživatel provést gesto pro rotaci, tak musel zvednout obě ruce také nad lokty, ale sevřel je až po nějaké době. Což znamená, že se omylem předtím detekovat gesto pro změnu měřítka, i když ho uživatel provést nechtěl. Částečným řešením tohoto problému je další podmínka, která určuje, že pro správnou detekci gesta pro změnu měřítka musí být obě ruce dále od těla. Tato podmínka problém neeliminuje. Stále se může stát nechtěná detekce. Způsob provedení gesta lze vidět na obrázku 33.



Obrázek 33: Gesto pro změnu měřítka. Zdroj: Bc. Martin Čapek

Rozpoznání gesta pro změnu měřítka řeší metoda zoom(). Na začátku této metody se uloží potřebné části těla do lokálních proměnných. Oproti předchozím metodám se uloží i část těla *ShoulderCenter* (střed ramen). Po nastavení lokálních proměnných se testuje vhodná pozice obou rukou vzhledem k lotům. Obě ruce musí být nad svými lokty a také v určité vzdálenosti od středu ramen. Důležitou podmínkou je i to, že obě ruce nesmí být sevřené, jinak by se jednalo o gesto pro rotaci.

```
// deklarace potřebných částí těla a pomocných proměnných
var leftHand = m_closestSkeleton.Joints[JointType.HandLeft];
var rightHand = m_closestSkeleton.Joints[JointType.HandRight];
var leftElbow = m_closestSkeleton.Joints[JointType.ElbowLeft];
var rightElbow = m_closestSkeleton.Joints[JointType.ElbowRight];
var shoulderCenter = m_closestSkeleton.Joints[JointType.
    ShoulderCenter];
bool left = false;
bool right = false;

// testování pozic rukou
if (leftHand.Position.Y > leftElbow.Position.Y && leftHand.
    Position.Z < shoulderCenter.Position.Z - 0.23 && !m_leftGrip
    )
  left = true;
if (rightHand.Position.Y > rightElbow.Position.Y && rightHand.
    Position.Z < shoulderCenter.Position.Z - 0.23 && !
    m_rightGrip)
  right = true;
```

Po splnění předchozích podmínek se jedná o gesto pro změnu měřítka a nastaví se proměnná *m_gestures* na hodnotu 2. Pro provedení reakce se testuje hodnota proměnné *m_zoom* typu *double*, jestli není nulová. Proměnná *m_zoom* nabývá absolutní hodnoty rozdílu souřadnice pravé ruky na ose X a souřadnice levé ruky na ose X. Úplně stejně nabývá hodnoty lokální proměnná x typu *double*, která se používá při porovnání s proměnnou *m_zoom*. Proměnná x představuje aktuální absolutní rozdíl mezi souřadnicemi pravé a levé ruky na ose X. Proměnná *m_zoom* představuje poslední absolutní rozdíl mezi souřadnicemi pravé a levé ruky na ose X. Jakmile se aktuální absolutní rozdíl změní s vůlí vyšší než 0.009, tak se přepíše lokální proměnná *pom* typu *double*, která určuje, na kolik procent velikosti se má aktuální objekt změnit. Po testování rozdílů mezi proměnnými x a *m_zoom* se přechází k samotné reakci zavoláním metody Zoom dané kamery. Tato metoda přijímá parametr typu *double* a jedná se o výše zmíněnou procentuální změnu velikosti objektu. Po provedení této metody je dobré překreslit aktuální scénu stejně jako u gesta pro rotaci.

```
m_gestures = 2;
if (m_zoom != 0.0)
{
  // pomocné proměnné pro reakci na gesto pro změnu měřítka
  double x = Math.Abs((double)rightHand.Position.X - (double)
      leftHand.Position.X);
```

```
    double pom = 1.0;
    if (x > m_zoom + 0.009)
      pom = 0.975;
    else
      if (x < m_zoom - 0.009)
        pom = 1.025;

    // reakce na gesto pro změnu měřítka
    cam.Zoom(pom);
    scene.CurrentViewer.Redraw(true);
...
}
```

Dynamičnost reakce tohoto gesta spočívá v tom, že pokud je nastavená proměnná *dynamicGestures* na hodnotu *true*, tak se proměnná *m_zoom* neaktualizuje a její role se změní z posledního absolutního rozdílu souřadnic pravé a levé ruky na úplně první absolutní rozdíl souřadnic pravé a levé ruky na ose X. Tím vzniká podobný efekt jako u předchozích gest s tím rozdílem, že se hodnota procentuální změny nezmění. Pokud je aktuální absolutní rozdíl odlišný, tak se stále mění měřítko.

```
...
if(!dynamicGestures)
  m_zoom = x;
...
```

4.2.8 Gesto mávání

Logika gesta mávání je jiná než u předchozích gest. Pomocí představení implementace tohoto gesta od (Pterneas, 2014), se vytvořila podobná alternativa. Jednoduchý popis tohoto gesta lze vidět v kapitole Ruka jako nástroj neverbální komunikace. Pro implementaci se použil základ této logiky. Pro mávání je typické, že se mění souřadnice ruky oproti svému loktu na ose X. Tím lze vytvořit segmenty pro detekování. První segment má souřadnice ruky na ose X menší než souřadnice loktu na ose X. Druhý segment je opačný, souřadnice ruky na ose X je větší než souřadnice loktu na ose X. Po srovnání a testování mávání různých lidí se ukázalo, že tahle logika není dostatečná. Někteří lidé mávají pouze svoji dlaní a hýbou zápěstím. Tím se logika gesta mávání rozšířila o další podmínku. V obou segmentech spolu s testováním souřadnic ruky a loktu, se testují souřadnice ruky a zápěstí stejným způsobem. V prvním segmentu se testuje, jestli je souřadnice ruky na ose X menší než souřadnice zápěstí na ose X. V druhém segmentu je to opět opačné. Souřadnice ruky na ose X musí být větší než souřadnice zápěstí na ose X. Provedení gesta mávání lze vidět na obrázku 2. Pro rozpoznání gesta mávání se vytvořily následující třídy a instance:

- abstraktní třída *GestureSegment*
- třídy *WaveSegment1* a *WaveSegment2*, které dědí z třídy *GestureSegment*
- třída *WaveGesture*

- dvě statické instance třídy *WaveGesture -> m_ waveLeftHand* a *m_ waveRightHand*

Abstraktní třída GestureSegment
Abstraktní třída *GestureSegment* určuje proměnné a metodu, kterou musí potomci implementovat. Proměnné určují ruku, která mává, její loket a zápěstí. Potřeba jsou také druhá ruka s loktem pro určení podmínek. Metoda *Update()* přijímá dva parametry a vrací hodnotu typu *bool*. První parametr je typu *Skeleton* a jedná se sledovanou kostru nejbližšího uživatele. Druhý parametr je typu *bool* a určuje sevření ruky, která mává.

```
public abstract class GestureSegment
{
  protected JointType m_hand;
  protected JointType m_wrist;
  protected JointType m_elbow;
  protected JointType m_handWrong;
  protected JointType m_elbowWrong;
  public abstract bool Update(Skeleton skeleton, bool gripped);
}
```

Třídy WaveSegment1 a WaveSegment2
Třídy *WaveSegment1* a *WaveSegment2* mají totožný konstruktor, kde přebírají potřebné části těla a ukládají je do výše zmíněných proměnných. První část metody *Update()* mají obě třídy stejnou. Jedná se testování, zda je ruka, která má mávat, nad svým loktem a ruka, která mávat nemá, pod svým loktem. Také se testuje sevření ruky, která mává. Pokud je tato ruka sevřená, tak podmínka neprojde a metoda *Update()* vrátí hodnotu *false*.

```
if(skeleton.Joints[m_hand].Position.Y > skeleton.Joints[m_elbow
   ].Position.Y &&
skeleton.Joints[m_handWrong].Position.Y < skeleton.Joints[
   m_elbowWrong].Position.Y &&
!gripped)
...
```

Po splnění předchozí podmínky se první větev příkazu *if* v jednotlivých třídách liší. Ve třídě *WaveSegment1* se testuje podmínka, zda má ruka větší souřadnici na ose X než její loket, nebo její zápěstí. Pokud je podmínka splněna, tak metoda *Update()* vrátí hodnotu *true* a tento segment je splněn.

```
if (skeleton.Joints[m_hand].Position.X + 0.04 > skeleton.Joints[
   m_elbow].Position.X ||
skeleton.Joints[m_wrist].Position.X < skeleton.Joints[m_hand].
   Position.X)
{
  return true;
}
```

Ve třídě *WaveSegment2* je to přesně opačně. Testuje se podmínka, zda má ruka menší souřadnici na ose X než její loket, nebo její zápěstí. Po splnění této podmínky opět metoda *Update()* vrací hodnotu *true* a segment je splněn.

```
if (skeleton.Joints[m_hand].Position.X - 0.04 < skeleton.Joints[
    m_elbow].Position.X ||
skeleton.Joints[m_wrist].Position.X > skeleton.Joints[m_hand].
    Position.X)
{
  return true;
}
```

Třída WaveGesture

Třída *WaveGesture* slouží pro rozpoznání gesta mávání. K tomu jsou potřebně atributy *m_segments* typu *GestureSegment[]*, *m_maxNumberOfFrames* typu *int*, *m_currentSegment* typu *int*, m_frameCount typu *int* a událost *GestureRecognized*. Atribut *m_segments* je pole výše zmíněných segmentů, které určují počet segmentů a jejich kombinaci. Toto pole se inicializuje v konstruktoru a kombinuje tři instance *WaveSegment1* a tři instance *WaveSegment2*, které se střídají při testování. Proměnná *m_maxNumberOfFrames* určuje počet snímku, ve kterých se je možné detekovat jeden segment. Proměnná *m_currentSegment* určuje index aktuálního testovaného segmentu z pole *m_segments*. Proměnná *m_frameCount* určuje počet již testovaných snímků pro aktuální segment. Událost *GestureRecognized* slouží pro reakci na mávání. Tato událost se musí zaregistrovat při vytvoření instance třídy *WaveGesture*, aby existovala reakce na mávání.

```
GestureSegment[] m_segments;
readonly int m_maxNumberOfFrames = 60;
int m_currentSegment = 0;
int m_frameCount = 0;
public event EventHandler GestureRecognized;
```

Metoda *Update()* přijímá dva parametry stejně jako metoda abstraktní třídy. Tyto parametry se následně předají metodě *Update()* aktuálního testovaného segmentu.

```
public void Update(Skeleton skeleton, bool gripped)
{
  bool result = m_segments[m_currentSegment].Update(skeleton,
      gripped);
  ...
}
```

Pokud testovaný segment vrátí hodnotu *true*, tak se testuje, zda se jedná o poslední segment v poli *m_segments*. Pokud se nejedná o poslední segment v poli, tak se atribut *m_currentSegment* zvýší o jedna, *m_frameCount* nastaví na nulu a při dalším testování mávání se testuje následující segment, který má k dispozici opět přednastavený počet snímků k detekování. Pokud se jedná o poslední segment v poli *m_segments*, tak se vyvolá událost *GestureRecognized* a zavolá se metoda *Reset()*, která vynuluje atributy *m_currentSegment* a *m_frameCount*. Pokud testovaný segment vrátí hodnotu *false*, tak se testuje rovnost hodnot *m_frameCount* a *m_maxNumberOfFrames*. Pokud se rovnají, tak se zavolá metoda *Reset()* a

rozpoznávání gesta mávání začíná znovu. Pokud je *m_frameCount* menší než *m_maxNumberOfFrames*, tak se atribut *m_frameCount* zvýší o jedna.

```
if (m_currentSegment + 1 < m_segments.Length)
{
  m_currentSegment++;
  m_frameCount = 0;
}
else
{
  if (GestureRecognized != null)
  {
    GestureRecognized(this, new EventArgs());
    Reset();
  }
}
```

Reakce na mávání

Vzhledem k tomu, že lze mávat každou rukou zvlášť, tak jsou navrženy dvě různé reakce na mávání každé ruky. Pro detekování mávání obou rukou jsou v komponentě *Tool*, vytvořeny dvě statické instance třídy *WaveGesture*. První instance *m_waveLeftHand* pro mávání levou rukou a druhá instance *m_waveRightHand* pro mávání pravou rukou.

```
static WaveGesture m_waveLeftHand;
static WaveGesture m_waveRightHand;
```

Pro každou instanci je potřeba zaregistrovat výše zmíněnou událost *GestureRecognized*. Tato registrace je provedena v metodě *StartSensor()* v podkapitole Spuštění zařízení Kinect. Událost pro instanci *m_waveLeftHand* je metoda *m_waveLeftHand_GestureRecognized()*. Tato metoda přijímá dva parametry. První parametr je typu *object* a určuje třídu, která tuto metodu vyvolala. Druhý parametr je typu *EventArgs* a určuje parametry pro vyvolanou událost. Reakce, kterou gesto mávání levou rukou vyvolá, je vrácení scény do původního pohledu zavoláním metody kamery *SetDefaultsMBB()*. Tato metoda přijímá jeden parametr, který popisuje pohled, který má kamera vidět.

```
void m_waveLeftHand_GestureRecognized(object sender, EventArgs e
    )
{
  cam.SetDefaultsMBB(scene.Extent);
}
```

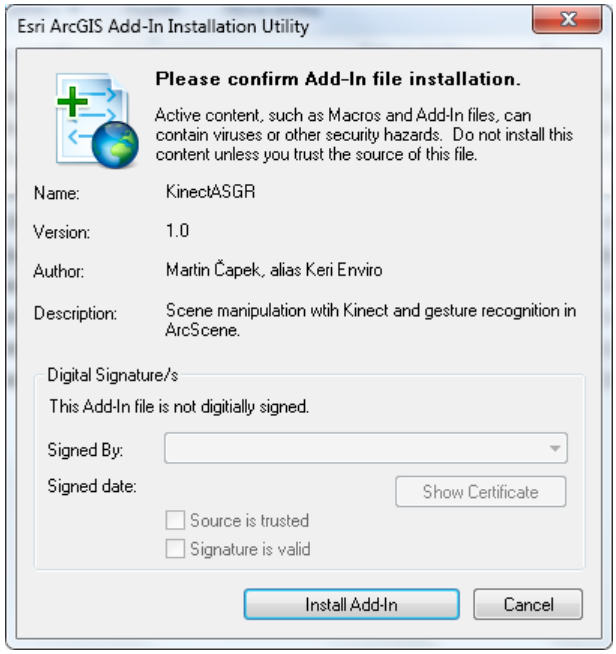
Událost pro instanci *m_waveRightHand* je metoda *m_waveRightHand_Gestu-reRecognized()*, která přijímá stejné parametry jako metoda *m_waveLeftHand_Ge-stureRecognized()*. Reakce, kterou vyvolává gesto mávání pravé ruky, je změna dynamiky. Při mávnutí pravou rukou se přepisuje hodnota proměnné *dynamicGestures*, která určuje dynamické reakce výše zmíněných gest, které emulují funkcionalitu 3D myši.

```
void m_waveRightHand_GestureRecognized(object sender, EventArgs
    e)
```

```
{
  if (dynamicGestures)
    dynamicGestures = false;
  else
    dynamicGestures = true;
}
```

4.3 Nasazení softwarové komponenty do systému ArcGIS - ArcScene

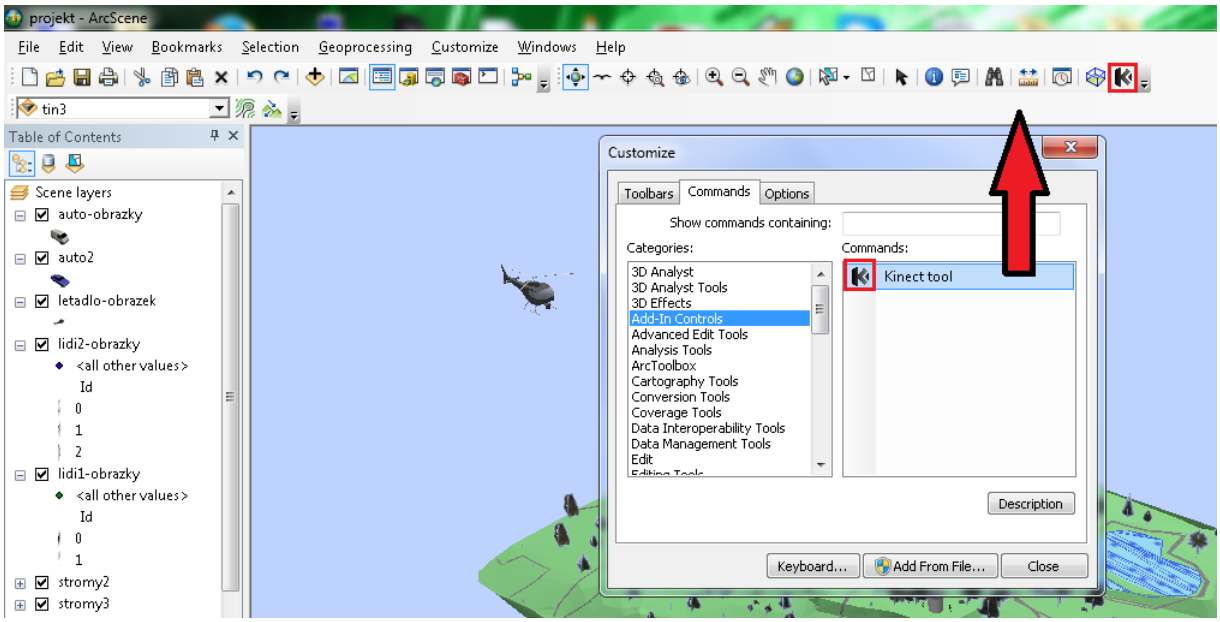
Pro nasazení softwarové komponenty je třeba mít nainstalovaný systém *ArcGIS* verze 10.2 nebo vyšší. Soubor implementované softwarové komponenty má název *KinectASGR* a musí mít příponu *esriAddIn*. Instalace je jednoduchá, stačí dvakrát kliknout na soubor s příponou *esriAddIn*, kliknout na tlačítko „Install AddIn“ (viz obrázek 34) a softwarová komponenta se sama nainstaluje do systému *ArcGIS*.



Obrázek 34: Instalační okno softwarové komponenty. Zdroj: Bc. Martin Čapek

Softwarová komponenta je nyní nainstalována, ale nepojede správně. Je nutné ještě zkopírovat soubory pro *KinectInteraction170_32.dll* a *KinectInteraction170_64.dll* do složky *bin*, příklad cesty ke složce *bin* je „C:/Program Files/ArcGIS/Desktop10.2/bin“. Zkopírováním těchto souborů se zajistí správné detekování sevření ruky, bez ní by softwarová komponenta nefungovala správně a aplikace *ArcScene* by se mohla neočekávaně ukončit. Po nainstalování a zkopírování všech potřebných souborů se spustí aplikace *ArcScene*, klikne se na *toolbar* na tlačítko „Customize“ a v rozvinutém listu se klikne na tlačítko „Customize Mode...“.

Objeví se dialog s názvem „Customize“. V komponentě *tab* se přepne na stránku „Commands“, kde se načtou všechny dostupně softwarové komponenty. Softwarová komponenta založená na rozpoznání gest pomocí zařízení *Kinect* je v kategorii „Add-In Controls“. Ikona softwarové komponenty se přetáhne na *toolbar* aplikace a je připravená k použití. Zařízení *Kinect* se zapojí do počítače a po kliknutí na ikonu softwarové komponenty se rozpoznávání gest spustí. Po opětovném kliknutí se rozpoznávání gest zastaví. Postup lze vidět na obrázku 35.



Obrázek 35: Nalezení a přetáhnutí softwarové komponenty do palety příkazů. Zdroj: Bc. Martin Čapek

5 Diskuze

Kapitola se bude zabývat vyhodnocením realizovaného projektu, zejména možným omezením dané hardwarem, knihovnami pro zpracování obrazu. Následně bude popsán návrh pro budoucí vylepšení.

5.1 Vyhodnocení realizované softwarové komponenty

Navržená softwarová komponenta detekuje nejbližšího uživatele a pomocí jeho gest manipuluje s 3D mapou (scénou) v aplikaci *ArcScene*. Tato manipulace se odvíjí od funkcí 3D myši. Navržený systém funguje správně podle cíle práce a má i několik funkcí navíc. Například funkci vrácení se do původního pohledu či dynamiku reakcí. Při navrhování gest, které mají emulovat funkce 3D myši, bylo nejtěžší navrhnout gesta tak, aby se navzájem nekryla či špatně rozpoznala. Konečný návrh má za vzor lidská gesta ze skutečného světa. Návrh gest byl také omezen knihovnou *Kinect SDK*, která neumožňuje rozpoznání prstů, tudíž by se museli použít sofistikovanější algoritmy, které přesahují náročnost této práce. Softwarová komponenta se nedá využít venku, kde by slunce rušilo detekování zařízení *Kinect*. Použití je určeno výhradně místnostech s minimálním venkovním osvětlením. Při testování se zařízení *Kinect* posouvalo a různě natáčelo (vertikálně), což se ukázalo také jako problém. Čím bylo větší natočení, tím menší byla úspěšnost rozpoznávání. A to z důvodu, že si zařízení *Kinect* nijak neuchovává svoje natočení ke skutečné horizontální rovině, tzn. výsledná kostra člověka byla deformovaná (špatně natočená). Detekce uživatele i rozpoznání gest je nejlepší ve vzdálenosti od zařízení *Kinect* v rozmezí 1–2.5 m a když zařízení *Kinect* míří kolmo na uživatele. Systém testovalo kolem deseti uživatelů. Gesto pro translaci mělo nejvyšší úspěšnost, téměř kolem 100%. Gesto pro rotaci bylo rozpoznáno s nejhorší úspěšností kolem 80%, je to zejména z důvodu špatné detekce sevření levé ruky. Ostatní gesta se více blíží úspěšnosti gesta pro translaci. Velkou úspěšnost mělo gesto mávání levou rukou s reakcí vrácení pohledu do výchozího stavu.

5.2 Návrh pro budoucí vylepšení

Přestože, že softwarová komponenta funguje správně, stále je co zlepšovat. Například problém s natáčením zařízení *Kinect* lze vyřešit konfiguračním souborem, ve kterém by byla jedna hodnota o úhlu natočení zařízení *Kinect*. Softwarová komponenta by se nemusela znovu sestavovat, ale po každém natočení zařízení *Kinect* by se musel konfigurační soubor přepsat. Aplikace *ArcScene* má mnoho funkcí, které pracují s mapou. Například lze exportovat aktuální pohled scény do souboru jako obrázek. Navrhnout gesto, které by využívalo tuhle funkci, by vylepšilo stávající řešení. Další funkce, která by se mohla využít je natáčení vide scény. Pro začlenění těchto funkcí by byl potřeba komplexnější návrh veškerých gest. Některá gesta by měla různé reakce závisle na módu aplikace atd. Vzhledem k problémům detekce sevření levé

ruky, by bylo dobré buď navrhnout nové gesto, které sevření levé ruky nepoužívá a je stále intuitivní pro uživatele, nebo nahradit zařízení *Kinect V1* novým zařízením *Kinect V2*. *Kinect V2* potřebuje verzi *Kinect SDK* nejméně 2.0, což by znamenalo přepsání veškerého kódu.

6 Závěr

Cílem této diplomové práce bylo navrhnout a implementovat softwarovou komponentu pro manipulaci s 3D objekty. Pro splnění hlavního cíle bylo nutné splnit cíle dílčí. Srovnat vývojové nástroje, metody a knihovny pro zpracování obrazu. Prozkoumat možnosti zařízení *Kinect* v rozpoznání pohybů uživatele. Prozkoumat již implementovaná řešení rozpoznání pohybů uživatele pomocí zařízení Kinect, zejména pro pohyb v 3D prostoru. Navrhnout a implementovat vlastní řešení, které bude pomocí rozpoznání pohybů uživatele umožňovat manipulaci s 3D objekty a nahrazovat funkcionalitu 3D myši, zejména translaci, rotaci a selekci.Vyhodnotit a diskutovat výsledné řešení.

V teoretické části práce je popsána ruka jako nástroj neverbální komunikace a její stupeň volnosti. Následoval popis gest, které člověk běžně provádí a byly popsány jejich aspekty, které jsou důležité při jejich návrhu. Následně bylo popsáno předzpracování obrazu, kde byly představeny základní metody pro segmentaci, detekce hran, odstranění šumu a normalizace. Výstup předzpracování obrazu používají klasifikační metody, které byly následně podrobně popsány. Ke každé klasifikační metodě jsou uvedené alespoň dva příklady praktického použití. Jedná se o metody ANN, HMM, FSM, PCA, FCM a GA. Poté bylo popsáno zařízení *Kinect*, které bylo stěžejní pro tuto práci. Důraz byl kladen na sledování kostry člověka (*skeleton tracking*) a na možnosti snímání. Byla představena také alternativa zařízení *Leap Motion*. Poslední sekce teoretické části se zabývá knihovnami pro zpracování obrazu, které dokáží komunikovat se zařízením *Kinect*.

Podle nabytých zkušeností byla stanovena metodika práce, která vedla ke splnění cíle této diplomové práce. Metodika popisuje výběr prostředí, ve kterém lze manipulovat s 3D objekty. Dále popisuje výběr knihovny, která musí komunikovat se zařízením *Kinect*. Potom popisuje návrh množiny gest, které emulují funkce 3D myši. A jako poslední popisuje implementaci a testování navrženého řešení.

Podle metodiky práce se přistoupilo k praktické části. Zvolilo se prostředí *ArcScene* pro manipulaci 3D mapy. Pomocí knihovny *Kinect SDK* se vyřešila detekce a rozpoznávání gest uživatele. Návrh implementovaných gest používal jako vzor gesta skutečného světa a mobilních platforem. Navržená gesta emulují funkce 3D myši a to rotace, posun a změnu měřítka. Dále je implementováno gesto mávání levou rukou pro vrácení pohledu do výchozího stavu a gesto mávání pravou rukou pro změnu dynamičnosti reakcí gest. Na závěr praktické části se diskutovaly dosažené výsledky a problémy, pro které se navrhlo zlepšení.

Jsem přesvědčen, že cíl této diplomové práce byl splněn. Softwarová komponenta je funkční. Systém funguje na principu FSM a rozpoznání gest se příliš nekříží. Optimální vzdálenost pro rozpoznání gest je 1–2.5 m od zařízení *Kinect*. Je nutno dodat, že neexistuje řešení v rozpoznávání gest s úspěšností 100%, kde by se navržená gesta nemusela opakovat.

7 Literatura

ABHIJIT, Jana. *Kinect for Windows SDK programming guide: build motion-sensing applications with Microsoft's Kinect for Windows SDK quickly and easily.* Birmingham, UK: Packt Pub., 366 s., 2012..

AGGARWAL, J. K., CAI, Q.. *Human Motion Analysis: A Review.* Computer Vision and Image Understanding. 1999, Vol. 73, No. 3, pp. 428–440. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.333.2315&rep=rep1&type=pdf.

BEZDEK, James C.. *Pattern recognition with fuzzy objective function algorithms.* New York, Plenum Press, 1981, xv, 256 s., ISBN 0306406713..

BISHOP, Christopher M.. *Pattern recognition and machine learning.* Springer, 2006, 738 s. Information science and statistics. ISBN 0-387-31073-8.

DAVE, Dipen, CHOWRIAPPA, Ashirwad, KESAVADAS, Thenkurussi. *Gesture Interface for 3D CAD Modeling using Kinect.* CAD Solutions, Computer-Aided Design & Applications, 9(a), Virtual Reality Lab., University at Buffalo, Buffalo, NY, 2012.

DE LUCA, Carlo J., ADAM, Alexander, WOTIZ, Robert, GILMORE, L. Donald, NAWAB, S. Hamid. *Decomposition of Surface EMG Signals.* Journal of Neurophysiology, vol. 96, issue 3, 1646-1657 s, 2006, DOI: 10.1152/jn.00009.2006. http://www.delsys.com/decomp/Deluca_etal_2006.pdf.

ERKAN, Ayase Naz. *MODEL BASED THREE DIMENSIONAL HAND POSTURE RECOGNITION FOR HAND TRACKING.* Bogazici University, 2002. Dostupné z: http://cs.nyu.edu/~naz/docs/ayse-erkan-ms-thesis.pdf.

GAO, Robert X. GAO, YAN, Robert X.Ruqiang. *Wavelets: theory and applications for manufacturing.* Springer, 2010. ISBN 9781441915443.

GUPTA Sauvik Das , Souvik KUNDU, Rick PANDEY, Rahul GHOSH, Rajesh BAG, Abhishek MALLIK. *Hand Gesture recognition and classification by Discriminant and Principal Component Analysis using Machine Learning techniques.* International Journal of Advanced Research in Artificial Intelligence, Vol. 1, No. 9, 2012. Dostupné z: http://thesai.org/Downloads/IJARAI/Volume1No9/Paper_8-Hand_Gesture_recognition_and_classification_by_Discriminantand_Principal_Component_Analysis_using_Machine_Learning_techniques.pdf.

HALL. Jonathan C.. *How to Do Gesture Recognition With Kinect Using Hidden Markov Models (HMMs).* 2011. Dostupné z: http://www.creativedistraction.com/demos/gesture-recognition-kinect-with-hidden-markov-models-hmms/.

HASAN, MOKHTAR M., MISRA, PRAMOUD K.. *BRIGHTNESS FACTOR MATCHING FOR GESTURE RECOGNITION SYSTEM USING SCALED NORMALIZATION*. International Journal of Computer Science & Information Technology (IJCSIT). 2011 Dostupné z: http://www.airccse.org/journal/jcsit/0411csit03.pdf.

HONG, PENGYU, TURK, MATTHEW, HUANG, THOMAS S.. *Constructing Finite State Machines for Fast Gesture Recognition*. In Proc. 15th ICPR, 691-694, 2000 https://www.cs.ucsb.edu/~mturk/Papers/ICPR2000.pdf.

CHAUDHARY, ANKIT, J.L. RAHEJA, KAREN DAS, SONIA RAHEJA. *A Survey on Hand Gesture Recognition in Context of Soft Computing*. CCSIT 2011, Part III, CCIS 133, pp. 46–55, 2010..

JU SHANON X., MICHAEL J. BLACK, YASER YACOOB. *Cardboard People: A Parameterized Model of Articulated Image Motion*. Proc. Second Int. Conf. on Automatic Face and Gesture Recognition, 1996. http://cs.brown.edu/~black/Papers/fg96.pdf.

KHAN, RAFIQUL ZAMAN, IBRAHEEM, NOOR ADNAN. *Survey on Gesture Recognition for Hand Image Postures*. Computer and Information Science. 2012.

KHAN, RAFIQUL ZAMAN, IBRAHEEM, NOOR ADNAN. *Vision Based Gesture Recognition Using Neural Networks Approaches: A Review*. International Journal of human Computer Interaction (IJHCI), Volume 3, Issue 1, 2012..

KNUTH, DONALD E., MORRIS, JAMES H., PRATT, VAUGHAN R.. *FAST PATTERN MATCHING IN STRINGS*. SIAM J. COMPUT, Vol. 6, No. 2, 1977. http://www.cin.ufpe.br/~paguso/courses/if767/bib/KMP_1977.pdf.

LEE, H. AND J. KIM. *n HMM-Based Threshold Model Approach for Gesture Recognition*. IEEE Transactions on Pattern Analysis and Machine Intelligence , Vol. 21, pp. 961-973, 1999..

S. LENMAN, L. BRETZNER, AND B. THURESSON. *Using marking menus to develop command sets for computer vision based hand gesture interfaces*. Second Nordic Conference on Human-Computer Interaction, 2002. Dostupné z: ftp://ftp.nada.kth.se/CVAP/tmp/LenBreThu-NCHI02.pdf.

LI, XINGYAN. *Gesture recognition based on fuzzy C-Means clustering algorithm.*. Department Of Computer Science The University Of Tennessee Knoxville, 2003. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.136.3935&rep=rep1&type=pdf.

LIEN, C., HUANG C.. *The Model-Based Dynamic Hand Posture Identification Using Genetic Algorithm*. Machine Vision and Applications, pp. 107-121, 1999..

LIU, Nianjun a LOVELL, Brian C.. *Gesture Classification Using Hidden Markov Models and Viterbi Path Counting*. Proc. VIIth Digital Image Computing: Techniques and Applications. Sydney, 2003..

MAIRE, Michael Randolph. *Contour Detection and Image Segmentation*. University of California, Berkeley. 2009 Dostupné z: http://ttic.uchicago.edu/~mmaire/papers/pdf/mmaire_thesis.pdf.

MARAQA Manar, Farid AL-ZBOUN, Mufleh DHYABAT, Raed Abu ZITAR. *Recognition of Arabic Sign Language (ArSL) Using Recurrent Neural Networks*. Journal of Intelligent Learning Systems and Applications, 4, 41-52, 2012..

NOCK Richard, NIELSEN Frank. *On Weighting Clustering*. IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 28, NO. 8, Srpen 2006. http://www1.univ-ag.fr/~rnock/Articles/Drafts/tpami06-nn.pdf.

OLSON, Clark F.. *An Approximation Algorithm for Least Median of Squares Regression*. Information Processing Letters, 36: 237-241 s., 1997. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.54.2655&rep=rep1&type=pdf.

PTERNEAS, Vangos. *Implementing Kinect gestures*. Dostupné z: http://pterneas.com/2014/01/27/implementing-kinect-gestures/.

QING, Chen. *Real-Time Vision-Based Hand Tracking and Gesture Recognition*. Ottawa, 2008. Dostupné z: http://www.discover.uottawa.ca/~qchen/my_papers/phd-thesis.pdf. Doctor of Philosophy. University of Ottawa.

STARNER, T., J. WEAVER, and A. PENTLAND. *Language Recognition Using Desk and Wearable Computer Based Video*. IEEE Transactions on Pattern Analysis and Machine Intelligence , pp. 1371-1375, 1998..

STERGIOPOULOU, E. a N. PAPAMARKOS. *Hand gesture recognition using a neural network shape fitting technice*. Engineering Applications of Artificial Intelligence 22. 2009, 1141–1158. ISSN: 0952-1976. Dostupné z: http://www.papamarkos.gr/uploaded-files/Handgesturerecognitionusinganeuralnetworkshapefittingtechnique.pdf.

VERMA Rohit, Ankit DEV. *Vision based Hand Gesture Recognition Using Finite State Machines and Fuzzy Logic*. International Conference on Ultra Modern Telecommunications & Workshops, IEEE, 2009. .

WRIGHT, David R.. *Finite State Machines*. 2005, 28 s. http://www4.ncsu.edu/~drwrigh3/docs/courses/csc216/fsm-notes.pdf.

WU, Y. AND T. S. HUANG. *Capturing Human Hand Motion a Divide and Conquer Approach*. Proceedings of IEEE International Conference of Computer Vision , pp. 606-611, Corfu, Greece, 1999.