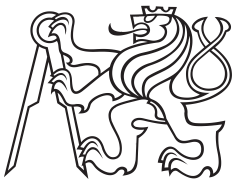


České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Sledování reálných objektů ve virtuální realitě

Jan Lazarek

Školitel: Ing. David Sedláček Ph.D.
Květen 2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Lazarek** Jméno: **Jan** Osobní číslo: **474599**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Studijní obor: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Sledování reálných objektů ve virtuální realitě

Název bakalářské práce anglicky:

Real object tracking for Virtual Reality

Pokyny pro vypracování:

Analyzujte možnosti snímání rukou ve virtuální realitě (VR) a dalších předmětů, které může uživatel držet nebo jinak používat ve VR.
Ve spolupráci s vedoucím vyberte postupy pro experimenty a porovnejte je z hlediska přesnosti a robustnosti řešení (minimálně přesnost odhadu pozice a orientace objektu, prodlevu systému).
Navrhněte a implementujte prototyp aplikace, kde uživatel bude moci ve VR pracovat s tužkou a pravítkem, s tím, že bude držet vhodně upravené reálné objekty v rukou.
Prototyp otestujte s uživateli.

Seznam doporučené literatury:

[1] Joseph J. LaViola, Jr. et all. 3D User Interfaces: Theory and Practice, second edition. 2017. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.
[2] Practical Augmented Reality: Steve Aukstakalnis, 2017, Addison Wesley
[3] T. Nakamura, 'Real-time 3-D object tracking using Kinect sensor,' 2011 IEEE International Conference on Robotics and Biomimetics, Karon Beach, Phuket, 2011, pp. 784-788.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. David Sedláček, Ph.D., katedra počítačové grafiky a interakce FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **10.02.2020** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **30.09.2021**

Ing. David Sedláček, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

Datum převzetí zadání Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Poděkování

Děkuji svému vedoucímu tohoto projektu Ing. Davidu Sedláčkovi Ph.D. za podporu při tvorbě této Bakalářské práce a Českému vysokému učení technickému v Praze, speciálně fakultě Elektrotechnické za zapůjčení prostředků a prostor umožňujících vypracování této práce.

Prohlášení

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, května 22, 2020

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 22. May 2020

Abstrakt

Předmětem této práce je zkoumání způsobů využití rozpoznávání a sledování barevně odlišených předmětů pomocí hloubkové kamery pro VR aplikace. V práci jsem využil hloubkovou kameru Microsoft Kinect v2 a Azure Kinect a knihovny OpenCV a PCL.

Klíčová slova: bakalářská práce, VR, virtuální realita, rozpoznávání obejktů, sledování objektů, Kinect

Školitel: Ing. David Sedláček Ph.D.

Abstract

The subject of this work is a research of possibilities of usage of recognition and tracking of color-coded objects by depth camera and usage of the tracking data in VR applications. For this task, I used Microsoft Kinect v2, Azure Kinect depth cameras, and OpenCV and PCL libraries.

Keywords: bachelor thesis, VR, virtual reality, object reckognition, object tracking, Kinect

Title translation: Real object tracking for Virtual Reality

Obsah

1 Motivace 1

2 Teorie a analýza 3
2.1 Virtuální realita 3
2.2 Počítačové vidění 3
2.3 Analýza problematiky 4
2.3.1 Dvojrozměrná a hloubková data 4
2.3.2 Stereoskopické kamery 5
2.3.3 Optical tracking 6
2.3.4 Structured light 7
2.3.5 Time-of-flight 7

3 Návrh řešení 9
3.1 Testované technologie záznamu scény 9
3.1.1 Microsoft Kinect v1 9
3.1.2 Microsoft Kinect v2 10
3.1.3 Creative BlasterX Senz3D . . . 10
3.1.4 Microsoft Azure Kinect Developement Kit 11
3.2 Záznam člověka 12
3.3 Příprava scény a rozestavení zařízení 13
3.3.1 Poloha kamery 13
3.3.2 Sledované objekty 14
3.4 Zpracování obrazu 16
3.4.1 Zpracování vstupních dat . . . 16
3.4.2 Použité knihovny 17
3.4.3 Barevné modely 17
3.4.4 Práce s hloubkovými daty . . . 19

4 Implementace 21
4.1 Práce s 2D RGB obrazem 21
4.1.1 Využití OpenCV 21
4.2 Práce s hloubkovými daty 22
4.2.1 Rozeznávání na základě barvy 22
4.2.2 Implementace rozpoznávání . 23
4.2.3 Zpracování dat 24
4.2.4 Fitování objektů 24
4.2.5 Komunikace s Unity 25
4.2.6 Unity 25

5 Testování 27
5.1 První fáze testování 27
5.2 Druhá fáze testování 29
5.2.1 Rozložení scény 29
5.2.2 Barevná kalibrace 30
5.2.3 Testování přesnosti 32
5.2.4 Testování rychlosti 33
5.2.5 Testování sledování jednotlivých dvou předmětů 34
5.2.6 Uživatelské testování celé aplikace 35

6 Závěr 37
6.1 Shrnutí postupu 37
6.2 Závěrečné zhodnocení 38

A Literatura 39

B Struktura přiložených souborů 41

Obrázky

2.1 Vizualizace rekonstrukce povrchu na základě hloubkových dat pořízených Kinectem první generace 7
2.2 Vizualizace hloubkových dat pořízených Azure Kinectem 8

3.1 Microsoft Kinect, společně s označením rozložení kamer 9
3.2 Microsoft Kinect v2, společně s označením rozložení kamer 10
3.3 Hloubková kamera Creative BlasterX Senz3D 11
3.4 Schéma Microsoft Azure Kinectu 12
3.5 Leap Motion 13
3.6 Kamery Leap Motionu 13
3.7 Schéma úhlu záběru Kinectu. Vlevo pohled y boku, kdy je Kinect uměn 45 cm nad podložkou. Modře je vyznačeno jeho pohledové frustum. V pravo je znázorněn pohled shora. Červeně je vyznačen průnik pohledového frusta Kinectu a desky stolu. Na vodorovné ose jsou vyznačeny vzdálenosti od Kinectu. 14
3.8 HTC Vive s upěvněným Leap Motionem a Vive tracker 15
3.9 Fotografie sledovaných objektů . 16
3.10 Vizualizace RGB ve formě jednotkové krychle 18
3.11 Kuželovitá vizualizace HSV . . . 18
3.12 Válcovitá vizualizace HSV 19

4.1 Fotografie sledovaných objektů . 23

5.1 Plastový kelímek a velké množství chybně zaznamenané body 28
5.2 Papírový kelímek, profiltrované mračno bodů a mračno bodů, které které označil RANSAC jako náležící kelímku 28
5.3 Fotografie testovacího rozložení. 29
5.4 Snímek obrazovky Azure Kinect Vieweru společně s vhodně nastavenými parametry barevné kamery. 31
5.5 Screenshot objektů po barevné segmentaci 32

Tabulky

3.1 Technická specifikace Kinect v1 . . 9
3.2 Technická specifikace Kinect v2 . 10
3.3 Technická specifikace Creative BlasterX Senz3D 11
3.4 Technická specifikace Kinect Azure 11

4.1 Tabulka parametrů RANSAC algoritmu pro jednotlivé předměty 25

5.1 Hodnoty fps - množství vykreslených snímku za sekundu v závislosti na velikost vykreslovaného mračna a aplikovaných výpočtů . . . 28
5.2 Defaultní hodnoty nastavení barevné kamery Azure Kinectu . . . 31
5.3 Měření polohy tužky a hodnoty odchylek těchto měření. Hodnoty měření v metrech 32
5.4 Měření polohy trojúhelníku a hodnoty odchylek těchto měření. Hodnoty měření v metrech 33
5.5 Měření stability estimace jednotlivých rotací. Udávané hodnoty jsou jednotlivé složky směrových vektorů. 33
5.6 Počet vypočítaných estimací poloh pro jednotlivé objekty za jednu sekundu. 34

Kapitola 1

Motivace

S rostoucími možnostmi využití virtuální reality napříč různými obory v posledních letech roste i poptávka po jejím dalším využití v oborech stávajících i v oborech, kde dosud využívána nebyla. S tím souvisí i snaha o neustálé zdokonalování a rozšiřování této technologie. V současné době se virtuální realita využívá v odvětvích jako je například marketing, vojenství, zdravotní terapie, trénování lékařů a v neposlední řadě má také své využití v herním a zábavním průmyslu.

Virtuální realita se snaží nahradit základní vjemy člověka vjemy umělými, případně vjemy skutečnými, ale určitým způsobem modifikovanými. Využitím různých výpočetních technologií se dá vytvořit iluze skutečného světa. Základní lidské vjemy, které jsou většinou manipulovány jsou zrak a sluch. Vjemovou oblastí, která je předmětem zájmu této práce, je však především hmat a možnost haptické interakce s předměty reálného světa. Jako příklad je možné uvést situaci, kdy uživatel uchopí do ruky skutečnou tužku, kterou pak může využít jako předmět ve svém virtuálním světě. V současné době již existují různé ovladače, které jsou speciálně využívané pro virtuální realitu. Jsou to ale většinou zařízení plnící pouze tento jeden účel a pro nezkušeného uživatele virtuální reality jsou tedy úplně nové. Tím mohou představovat další překážku pro rychlé sžití se s touto technologií. Hlavním přínosem této práce je tedy možnost využití uživatelem již dávno známých předmětů denní potřeby jako je třeba pravítko nebo tužka ve virtuální realitě.

Kapitola 2
Teorie a analýza

2.1 Virtuální realita

Virtuální realita je soubor technologií umožňujících přenesení jejího uživatele do umělého, počítačem generovaného prostředí. Za tímto účelem je využíváno množství technologií, pomocí kterých jsou jednotlivé přirozené lidské vjemy nahrazovány vjemy uměle vytvořenými. Standardně se jedná o nahrazování vjemů vizuálních a zvukových. Ve spojení s virtuální realitou je často možné zaznamenat také výraz mixed reality, neboli česky smíšená realita. Od reality virtuální se zpravidla liší tím, že vjemy uživateli předávané, jsou částečně založené na vjemech skutečných, které jsou pouze do určité míry manipulovány. Většinou se tedy nejedná o přímé přenesení uživatele do jiné reality, ale pouze o upravení skutečnosti. Smíšená realita se dělí na rozšířenou realitu a rozšířenou virtualitu podle poměru množství reálných a virtuálních vjemů. Může se tak dít přímo nebo nepřímo.

Smíšené reality se dá docílit dvěma základními metodami, přímou a nepřímou. Nepřímá metoda využívá záznamové zařízení, například mobilní telefon, který pomocí kamery na své zadní straně snímá scénu. Obraz je následně zařízením zpracován, obohacen o prvky dané rozšířeně realitní aplikace a pak je celá scéna finálně zobrazována na displej. Oproti tomu přímá metoda nezpracovává celou scénu, ale přidaná generovaná informace je uživateli zobrazována například na poloprůhledném displeji[con19].

2.2 Počítačové vidění

Počítačové vidění je vědní obor počítačových věd, zabývající se technologiemi, které umožňují počítačům porozumění reálnému světu na základě obrazové informace podobným způsobem, jaký využívají i lidé. Základními úlohami tohoto odvětví jsou získání, zpracování, analýza a v závěru porozumění obrazovému signálu [BJ00]. Výsledkem těchto operací jsou pak vysokoúrovňová data v kontextu dané scény podobná tomu, jak by scénu byl schopen rozpoznat a vnímat člověk. Celková snaha se tedy upírá k tomu, aby počítače byly schopny vnímat jednotlivé objekty a jejich vzájemný kontext podobným

způsobem, jak to dělá lidské vidění.

2.3 Analýza problematiky

Předmětem této práce je rozpoznávání a sledování předmětů každodenního užití za účelem jejich využití ve virtuální realitě. Konkrétně se bude jednat o pravoúhlý trojúhelník a tužku. Pomocí metod počítačového vidění bude potřeba získaný obraz scény převézt do formátu, se kterým je možné dále pracovat. Těmto datům bude nutno nejprve porozumět, na základě čehož je následně možné odfiltrovat data nepotřebná či chybná. To vše bude činěno za účelem minimalizace případné možnosti chyb u dalších výpočtů a zároveň aby množství dat, se kterými se bude dále pracovat, bylo co nejmenší. Toto omezení plyne z limitací výkonu současné výpočetní techniky. Ve výsledku se má jednat o virtuálně realitní aplikaci. Zde je žádoucí dosažení co největší plynulosti a rychlosti zobrazení, a to z důvodu omezení možnosti vzniku kinetózy. Kinetóza jako taková je fyzickým stavem jedince, při kterém vjem pohybu získaný vestibulárním systémem nesouhlasí s vizuálními vjemy.[con18] Na druhou stranu, ale množství dat nemůže být ani příliš malé, aby bylo vůbec možné dosáhnout relevantních výsledků.

Po profiltrování získaných dat přichází na řadu analýza obrazové informace. Zde budou aplikovány algoritmy počítačového vidění, které budou schopné efektivně a v reálném čase rozpoznat podle určitých pravidel jednotlivé objekty ve scéně. Dále bude třeba zjistit přesné pozice jednotlivých objektů a následně i jejich rotaci. Zároveň bude potřeba řešit problematiku zastínění jednoho objektu druhým, případně zastínění části objektu vlivem jeho uchopení uživatelem.

Ve finální aplikaci již bude možné na základě získaných polohových dat jednotlivé předměty vykreslovat na jejich příslušné pozice. Pro zvýšení imerze uživatele a věrnosti prostředí, bude potřeba ještě určitým způsobem získávat data o poloze uživatele, a hlavně pak jeho rukou. Získání dat o pozici je nutné proto, aby mohl být uživatel ve virtuálním světě vykreslován, čímž se docílí lepší orientace a ukotvení ve scéně. Zároveň tak bude pro uživatele jednodušší pochopit a uvěřit vztahu mezi jeho virtuálním já a reálnými předměty před ním.

2.3.1 Dvojrozměrná a hloubková data

Pro účely této práce připadají v úvahu v podstatě dva způsoby snímání scény a v důsledku toho i dva fundamentální způsoby získávání informací o scéně. Většina uživatelů je v digitálním světě zvyklá využívat pouze plošná data. Těmi mohou být například fotografie nebo videozáznam, konkrétně třeba film. Tento druh záznamu je standardně získáván fotoaparátem, případně kamerou, které oba obsahují jeden záznamový čip - snímač. U těchto technologií tedy dochází k promítání prostoru na plochu daného čipu. Důsledkem toho ale

dochází k degradaci dat, kdy z prostorové informace přicházíme o hloubková data a zůstává nám pouze plochý obraz. To většinou nevadí, protože takto pořízené záznamy mají z pravidla čistě informativní hodnotu a člověk jim je schopen, na základně předchozích zkušeností a znalostí, porozumět. Tato data jsou v digitálním světě standardně ukládána do obrazové matice. Zde počet řádků a sloupců definuje velikost - rozlišení obrazu a hodnota v každém políčku definuje hodnotu obrazové informace v daném pixelu fotografie nebo rastrového obrázku. Takováto data jsou ukládána například do formátu Portable Network Graphics (.png) [con20c].

Oproti těmto, někdy nazývaným plošným datům, jsou data hloubková neboli hloubkový obraz. Zde jsou v podstatě dvě možnosti, jak lze počítači předat hloubková data. Prvním z nich je využití přímo hloubkové kamery. Zde existuje více technologií, které uvedu později v kapitole 2.3.2. Společné ale mají to, že vracejí v podstatě podobnou obrazovou matici jako standardní barevné kamery, avšak s jedním podstatným rozdílem. Místo hodnoty barevné informace je v jednotlivých políčkách uložena vzdálenost toho daného bodu od snímače kamery. Vizualizaci takových dat je možné vidět na obrázku 2.2, kde je vzdálenost bodu znázorněna barvou bodu. Nejbližší body jsou vykresleny modrou barvou, která s rostoucí vzdáleností bodu od kamery postupně přechází v barvu zelenou a žlutou. Je však potřeba odlišit hloubkový obraz od mračna bodů. Mračno bodů je většinou neuspořádaná množina všech bodů dané scény, která je převážně získávána 3D laserovými scannery. V této struktuře jsou uloženy jednotlivé body ve formátu x,y,z, kde tyto hodnoty definují polohu každého jednoho bodu v souřadném systému scény. Existují různé variace této datové struktury. Například je možné k prostorovým souřadnicím bodu přidat ještě hodnoty barevných složek, čímž získáme barevné mračno bodů[RC11].

Druhou možností získání hloubkových dat je využití většího množství dat plošných. Existují sice způsoby, jak se dají z jedné standardní kamery získat hloubková data, jedná se ale o poměrně složité metody, které v této práci nebudu rozebírat. Přímočařejší cesta, jak z barevné kamery získat hloubková data, je využití více kamer a principu triangulace. Ten je v počítačovém vidění popisován jako proces určování polohy bodu v 3D prostoru na základě jeho projekcí na dva a více čipů záznamových zařízení. Jinak řečeno, jedná se o určování polohy bodu na základě záznamu scény z alespoň dvou různých úhlů. Pro výpočet polohy je pak potřeba znát matice projekce 3D scény na 2D roviny čipů jednotlivých kamer. Algoritmus vychází z úvahy, že bod v 3D prostoru a bod promítnutý na čip definují přímku. Pokud máme několik záznamových zařízení v různých pozicích, tak získáme několik takových přímek. Bod v 3D prostoru se pak nachází v průsečíku těchto přímek [con20e].

2.3.2 Stereoskopické kamery

Jako první možnost se nabízí využití stereoskopické kamery. Jako stereoskopické vidění je označován princip, na kterém funguje třeba i lidský zrak. Jsou

zde používána dvě záznamová zařízení, v případě člověka jsou to oči. Tento druh záznamu využívá skládání obrazu ze dvou snímačů, které jsou podobně jako lidské oči na trochu jiných místech. Každý ze snímačů tedy "vidí"scénu z trochu jiného úhlu. Následně, pokud známe přesnou polohu a naklonění snímačů vůči sobě, je možné data z obou zařízení spojit a získat mimo barevné také prostorovou informaci o scéně[con20a]. Zařízení, která takto zpracovávají obraz, existují, ale často je problém, že se z nich v reálném čase data špatně získávají, nebo je získávat vůbec nelze. Navíc se tato data většinou nehodí pro další zpracovávání, přesněji řečeno knihovny, které by s nimi uměly pracovat, nejsou příliš rozšířené. Jedním příkladem z této kategorie jsou zařízení typu Windows Mixed reality, kde se jedná o náhlavní displeje určené pro virtuální případně smíšenou realitu, vybavené stereoskopickými kamerami, konkrétně třeba Acer Windows Mixed Reality Headset.

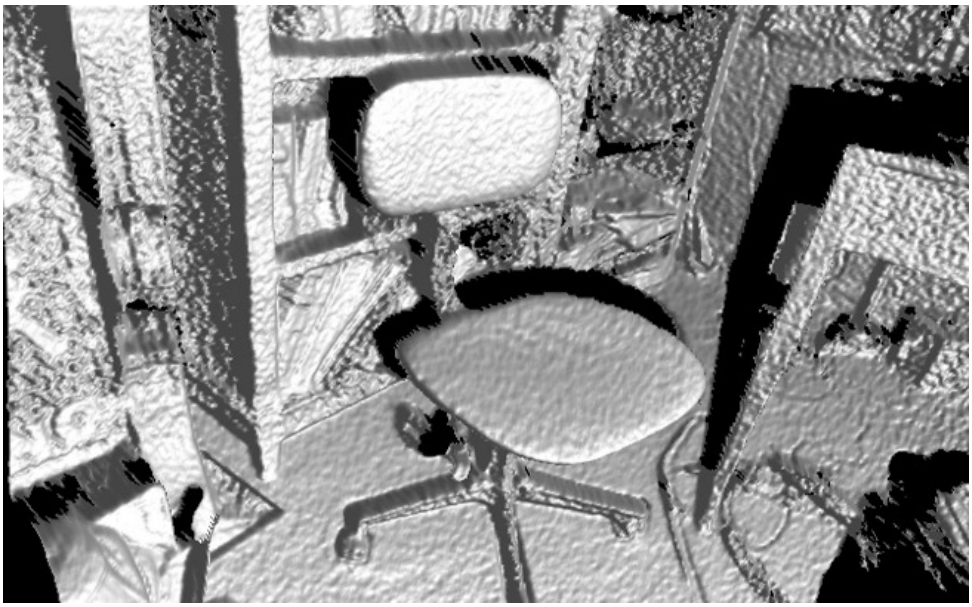
2.3.3 Optical tracking

Dále by bylo možné použít optical tracking neboli optické sledování se značkami. Tato technologie využívá sledování speciálních předmětů obsahujících aktivní případně pasivní infračervené značky. Zpravidla se využívá několika kamer, které z různých úhlů sledují scénu, ve které se nachází sledovaný předmět s infračervenými značkami. Počítač zároveň zná přesné rozložení značek na daném předmětu. Data získaná z kamer srovnává s předdefinovaným objektem a tím je možné získat přesnou pozici a rotaci daného předmětu ve scéně. Infračervené značky mohou být dvou tipů, pasivní a aktivní. Pasivní jsou pouze vizuální značky určitého tvaru a případně barvy. Nemusí se jednat pouze o infračervené značky, ale pro sledování se používají třeba i QR kódy jako v případě platformy Vuforia. V případě pasivních infračervených značek jsou u kamer zároveň infračervené vysílače svítící do scény a poloha značek je zjišťována pomocí odraženého infračerveného záření. Tuto technologii využívá například OptiTrack. Aktivní značky přímo vysílají infračervené záření a kamery tedy pouze sledují, odkud přesně toto záření vychází.

Rád bych na tomto místě ještě vysvětlil anglický pojem "outside-in tracking", neboli česky sledování zvenku dovnitř. Jedná se o způsob sledování polohy předmětu za použití vnějšího pozorovatele. Pozorovatelem je myšleno zařízení, které snímá scénu a objekty v ní. Konkrétně u HTC Vive jsou za tímto účelem používány standardně dvě základní stanice obsahující kamery zaznamenávající intenzitu infračerveného záření. Pro zvýšení přesnosti sledování a minimalizaci možnosti zákrytu sledovaných objektů je ale možné základních stanic využít více. Oproti tomuto přístupu existuje i přístup ve své podstatě opačný, kde není zapotřebí žádný vnější pozorovatel. V tomto případě se samotné sledované zařízení snaží samo určit svou polohu v prostoru. Za tímto účelem se zpravidla využívá kombinace čidel, kterými bývají gyroskop a akcelerometr v kombinaci se snímáním okolního prostředí. Optické senzory zaznamenávají takzvané features okolí, neboli specifické rysy okolí a podle jejich pohybu vypočítávají pohyb a orientaci zařízení samotného.

2.3.4 Structured light

Další užívaná technologie se nazývá structured light. Zde se využívá promítání konkrétního obrazce do scény, většinou se jedná o mřížku. Scéna, na kterou je obrazec promítán, je snímána kamerou. Vlivem překážek a nerovností ve scéně, pokud se tedy jedná o neplochou scénu, je promítaný obraz vůči pohledu kamery nějakým způsobem zdeformován. Hloubková data v jednotlivých místech jsou poté získávána srovnáním původního promítaného a nasnímaného zdeformovaného obrazu[con20d]. Tato technologie je většinou používána pro 3D laserové skenování objektů a prostorů. V těchto aplikacích často nejde o rychlost snímání, ale většinou se využívá takzvané registrace mračen bodů. Jako registrace je označován proces, kdy dochází ke spojování většího množství mračen do mračna jednoho. V praxi se používá například při skenování objektu z více různých úhlů. Existují však zařízení, která tuto technologii používají pro poskytování real-time hloubkových dat. Jedním takovým je Microsoft Kinect první generace. Na obrázku 2.1 je rekonstrukce scény vytvořená na základě hloubkových dat pořízených Kinectem. Na první pohled je vidět, jak se zdá většina povrchů do určité míry zvlněná. Toto naznačuje znatelnou míru nepřesnosti měření prováděných tímto zařízením. Z tohoto důvodu jsem se rozhodl s touto kamerou dále nepracovat.

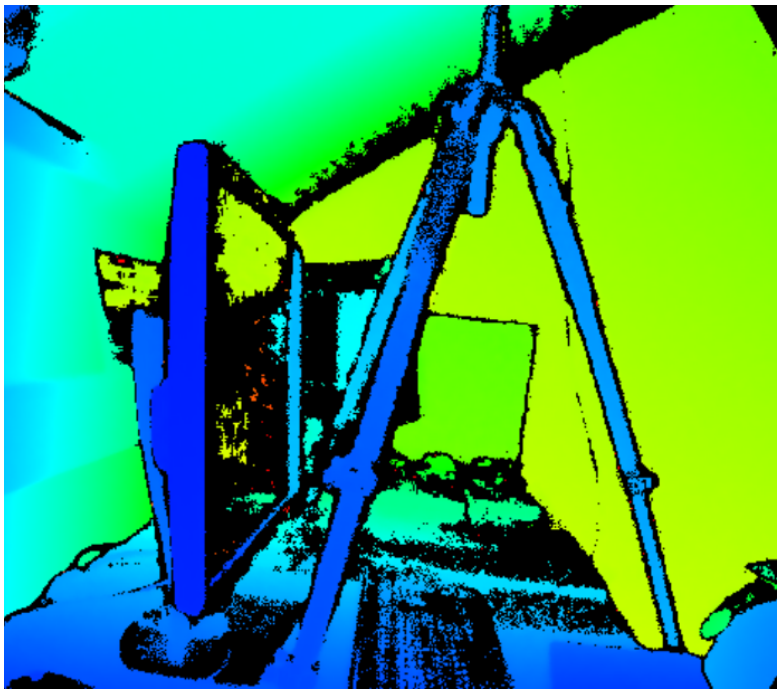


Obrázek 2.1: Vizualizace rekonstrukce povrchu na základě hloubkových dat pořízených Kinectem první generace

2.3.5 Time-of-flight

Poslední a zároveň v současné době asi nejpoužívanější technologií je time-of-flight technologie, která odkazuje na dobu letu světla. Technologie využívá měření délky letu částice světla. Pro zjištění vzdálenosti jednotlivých

bodů ve scéně využívá měření doby, za jakou se paprsek světla vyslaný do scény, odrazí od objektů a vrátí se zpět k jeho vysílači. Výsledná vzdálenost bodu označovaná `d` je vypočítána jako `(t * c)/2`, kde `t` je doba od vyslání světelného pulzu do jeho návratu a `c` je konstanta rychlosti světla. Konkrétně je tato technologie využívána u zařízení Kinect v2 a Azure Kinect. Tato technologie má však také své nedostatky, které je možné vidět na obrázku 2.2 pořízeném Azure Kinectem. Zde je barvou pixelu znázorněna vzdálenost daného bodu od kamery. Tmavě modré body jsou nejblíže ke kameře a s rostoucí vzdáleností od ní se barva postupně mění přes zelenou až ke žluté. Na obrazovce monitoru v popředí je možno vidět zelenou barvu. Toto není způsobeno tím, že by byl monitor průhledný, ale dochází zde k absolutnímu odrazu paprsků vyslaných kamerou, a tak se v monitoru v podstatě zrcadlí zadní stěna místnosti. Další pozorovatelnou chybou je šikmý pruh černých pixelů v horní části. Zde se jedná o situaci, kdy se v daném místě odrazil infračervený paprsek vysílače kamery od jedné stěny na druhou. Z tohoto místa tedy kamera zaznamenala dvoje hloubková data o jedné oblasti, což vedlo k zneplatnění výsledku měření. Poslední pozorovanou chybou je černá linie okolo modrého stativu uprostřed. Toto zneplatnění měření vzniklo získáním duplicitního měření, kdy se do kamery vrátil jednak paprsek odražený od stativu a zároveň i od pozadí [TS19]. Tuto chybu je možné pozorovat ještě výrazněji při snímání pohybujících se předmětů.



Obrázek 2.2: Vizualizace hloubkových dat pořízených Azure Kinectem

Kapitola 3

Návrh řešení

3.1 Testované technologie záznamu scény

3.1.1 Microsoft Kinect v1

Na výběr jsem měl ze čtyřech zařízení. Prvním je Microsoft Kinect první generace, který je vidět na obrázku 3.1. Zařízení pracuje na technologii structured light, která využívá promítání infračerveného obrazce a snímání scény a pomocí monochromatického CMOS senzoru.Podrobněji jsem fungování této technologie popisoval v 2.3.4 Hloubková kamera pracuje na rozlišení 640x480 pixelů[WRA14]. Další technické specifikace jsou uvedeny v tabulce 3.1. Jelikož se jedná o starší zařízení, které nedosahuje takové obrazové kvality jako zařízení novější, rozhodl jsem se ho dále netestovat.

Rozlišení barevné kamery Obnovovací frekvence	460 x 480 pixelů 30Hz
Rozlišení hloubkové kamery Obnovovací frekvence	512 x 424 pixelů 30Hz
Pracovní vzdálenost	0,8 - 4m
Zorné pole vertikálně Horizontálně	43° 57°
Použitá technologie	Structured Light

Tabulka 3.1: Technická specifikace Kinect v1

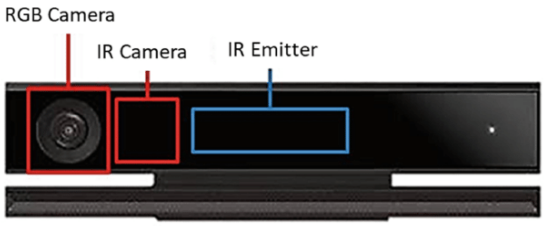
Obrázek 3.1: Microsoft Kinect, společně s označením rozložení kamer

3.1.2 Microsoft Kinect v2

Další možností byl Kinect v2 (druhá verze), na obrázku 3.2. Toto zařízení oproti první verzi pracuje na mírně menším rozlišení 512x424 pixelů, ale využívá o něco přesnější time-of-flight technologii, kterou jsem popisoval výše v části 2.3.5. Tato technologie umožňuje přesnější výpočet vzdálenosti a oproti Kinectu v1, který zvládá 2048 diskrétních hodnot vzdálenosti, Kinect v2 jich zvládá 8192. Hloubkovou kameru doprovází barevná kamera pracující na rozlišení 1920 x 1080 pixelů při obnovovací frekvenci 30 Hz. Další specifikace je obsažena v tabulce 3.2.

Rozlišení barevné kamery Obnovovací frekvence	1920 x 1080 pixelů 30 Hz
Rozlišení hloubkové kamery Obnovovací frekvence	512 x 424 pixelů 30 Hz
Pracovní vzdálenost	0,5 - 4,5 m
Zorné pole vertikálně Horizontálně	60° 70°
Použitá technologie	time-of-flight

Tabulka 3.2: Technická specifikace Kinect v2



Obrázek 3.2: Microsoft Kinect v2, společně s označením rozložení kamer

3.1.3 Creative BlasterX Senz3D

Dalším zařízením, které jsem měl k dispozici, je BlasterX Senz3D vyobrazený na obrázku 3.3. Jedná se o hloubkovou kameru pracující opět na time-of-flight technologii. Jak je vidět v tabulce 3.3, ve většině technických parametrů překonává oba přenos kandidáty. Zároveň tomuto zařízení stačí pouze jeden jediný kabel pro jeho napájení a posílání dat. V kombinaci s tím, že se jedná o nejmenší kameru, kterou jsem měl k dispozici, se jedná o jednoznačného favorita. Pro komunikaci s kamerou je však, bohužel, využívána Intel Real Sense technologie a v době tvorby této práce nebyly k dispozici žádné vhodné nástroje, které by mi umožňovaly z této kamery získávat data.

Rozlišení barevné kamery Obnovovací frekvence	1920 x 1080 pixelů 30 Hz
Rozlišení hloubkové kamery Obnovovací frekvence	640 x 480 pixelů 60 Hz
Pracovní vzdálenost	0,2 - 1,5 m
Zorné pole vertikálně Horizontálně	85° 85°
Použitá technologie	time-of-flight

Tabulka 3.3: Technická specifikace Creative BlasterX Senz3D

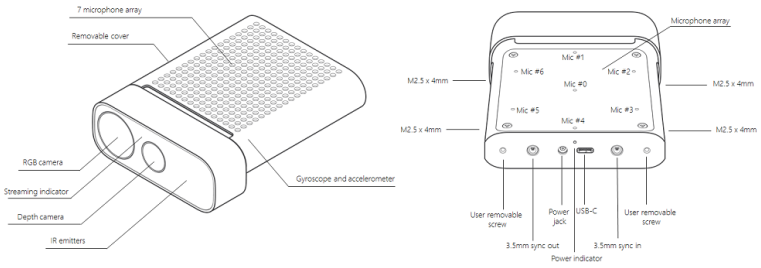
Obrázek 3.3: Hloubková kamera Creative BlasterX Senz3D

3.1.4 Microsoft Azure Kinect Developement Kit

Posledním kandidátem je Azure Kinect na obrázku 3.4. Jedná se o nejnovější z testovaných zařízení speciálně zaměřené na vývoj rozličných aplikací a s podporou pro rozsáhlé využití umělé inteligence. Zařízení je součástí Azure platformy od Microsoftu, která poskytuje mnoho nástrojů pro vývoj a správu aplikací. Toto zařízení podporuje množství konfigurací kamer a senzorů, které se definují při inicializaci zařízení. V tabulce 3.4 níže jsou uvedeny parametry konfigurace, které jsem používal při testování. Jedná se o konfiguraci úzkého zorného pole hloubkové kamery bez rozdělení na intervaly. Co se počtu úrovní hloubky týče, tak jsem žádná data nenalezl, ale už v úvodu testování se ukázalo, že data z tohoto zařízení se zdála nejpřesnější.

Rozlišení barevné kamery Obnovovací frekvence	1920 x 1080 pixelů 30Hz
Rozlišení hloubkové kamery Obnovovací frekvence	640 x 576 pixelů 60Hz
Pracovní vzdálenost	0,5 - 3,86m
Zorné pole vertikálně Horizontálně	65° 75°
Použitá technologie	time-of-flight

Tabulka 3.4: Technická specifikace Kinect Azure



Obrázek 3.4: Schéma Microsoft Azure Kinectu

3.2 Záznam člověka

Aby byla pro uživatele tato virtuálně realitní aplikace co nejvíce pohlcující a aby nedocházelo ke konfliktu vizuálních a hmatových vjemů. Bude potřeba mimo manipulovaných předmětů také nějakým způsobem vyobrazit uživatele samotného. Hlavně se bude jednat o vyobrazení jeho rukou, a to proto, aby vůbec mohl odhadovat, v jaké poloze vůči předmětům je a tím pádem, aby nesahal okolo sebe vlastně úplně naslepo.

Existuje množství zařízení, která bez větších modifikací nebo softwarových úprav člověka sledovat dokáže. Předmětem této práce je primárně zkoumání toho, zda je vůbec možné uskutečnit takovýto virtuálně realitní zážitek. Z tohoto důvodu jsem se rozhodl využít již existující řešení a technologie.

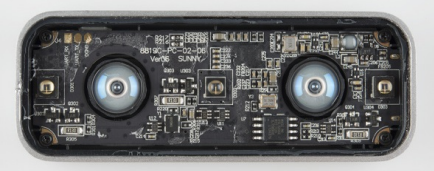
V první řadě jsou zde zařízení Kinect od Microsoftu. Jedno z nich bude pravděpodobně použito pro sledování předmětů. Pokud by se dalo použít i pro sledování člověka, výrazně by to ulehčilo situaci v tom, že by nebylo třeba řešit umístění a správu dalšího zařízení. Všechny tři Kinecty, které jsem zmínil v 3.1, podporují rozpoznávání a sledování lidské postavy. Co se první generace Kinectu týče, tak ta, bohužel, zvládá sledovat pouze polohu základních částí končetin, konkrétně tedy paži, předloktí a dlaň, což pro tuto aplikaci není dostatečné. Druhá generace Kinectu již zvládá rozpoznávat pozici palce a konce zbylých prstů. Dokonce také dokáže rozpoznávat základní gesta, ale nezvládá vracet přesnou pozici jednotlivých prstů v reálném čase [LRD16]. Toto je sice posun, ale pro danou aplikaci to stále není dostatečné.

Podobně jako Kinect v2 funguje i Kinect Azure. Existují sice projekty, které využívají toto zařízení pro sledování pohybu jednotlivých prstů, ale jedná se spíše o experimenty, než o široce používaná řešení. Následně se objevila myšlenka, že by Kinect šlo používat alespoň pro sledování zbytku těla uživatele. S tím se však váže problém, že k zařízení není možné přistupovat z více programů najednou. Není tedy možné v jednom programu získávat data pro sledování objektů a v herním enginu pak získávat data pro renderování správné polohy těla.

Jako jediné možné řešení se tedy jeví použití Leap Motionu. Jedná se o specializované vstupní zařízení, ovladač určený pro počítač, který využívá

jako vstupní data polohu uživatelových rukou a prstů. Toto zařízení by se dalo analogicky přirovnat k počítačové myši, avšak princip jeho fungování je diametrálně odlišný. Zařízení je vidět na obrázku 3.5. Obrázek 3.6 odhaluje, že pod černým sklem se nachází dvojice kamer s infračervenými vysílači. Jde tedy o stereoskopickou kameru určenou k sledování polohy rukou a prstů pro virtuální realitu [con20b]. Zároveň jeho obrovskou výhodou je jak po softwarové, tak hardwarové stránce, dobrá kompatibilita s HTC Vive, a tím i velmi dobré propojení s Unity.

Obrázek 3.5: Leap Motion

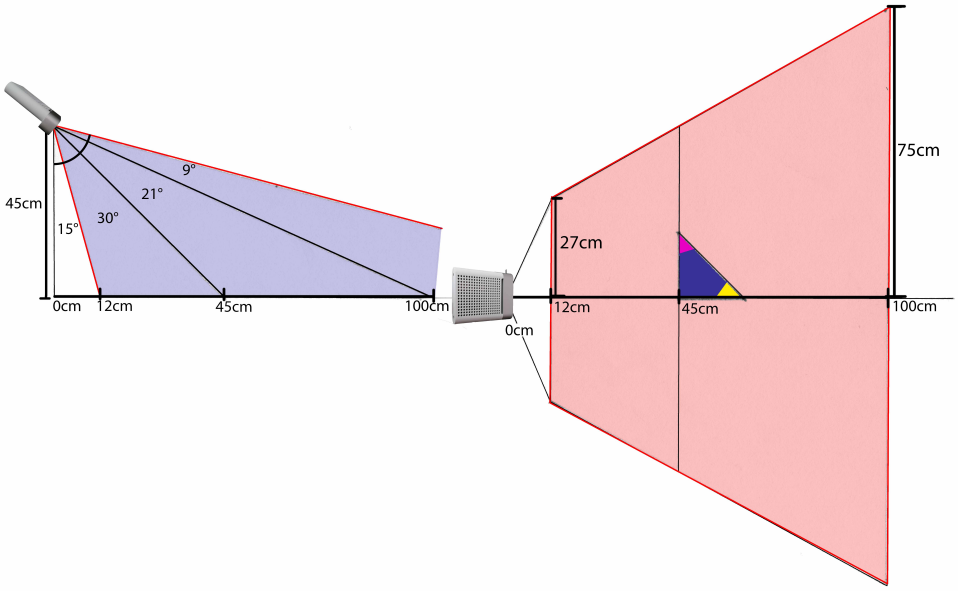


Obrázek 3.6: Kamery Leap Motionu

3.3 Příprava scény a rozestavení zařízení

3.3.1 Poloha kamery

Pro úvodní návrh rozložení scény jsem se rozhodl využít technické specifikace Kinectu v2, který se z důvodu nedostupnosti Azure Kinectu jevil jako hlavní kandidát pro použití. V obrázku 5.3 je vidět náčrt rozestavení Kinectu, který je ve výšce 45 cm nad deskou a je sklopen na stůl pod úhlem 45°. V pravé části je znázorněn průnik pohledového frusta hloubkové kamery Kinectu v2 a desky stolu, která má předpokládanou hloubku 100 cm. Tento červený čtyřúhelník tedy ukazuje velikost snímané plochy na úrovni stolu. Je vidět, že zadní hrana aktivní plochy má zhruba 55 cm a přední hrana zhruba 150 cm, přičemž hloubka aktivní plochy je necelých 90 cm. Neuvádím zde úplně přesná, ale spíše orientační čísla, a to ze dvou důvodů. Jednak proto, že měření na okraji pohledového frusta nemusejí být úplně přesná, a proto nemusejí být použitelná. Druhým důvodem je skutečnost, že se poloha a náklon Kinectu bude moci měnit a aplikace se podle toho bude sama při spuštění znovu kalibrovat. V levé části obrázku 3.7 je znázorněna poloha Kinectu a pohledového frusta vyznačeného modrou barvou z boku. Kinect je zde umístěn ve výšce 45 cm nad deskou stolu a nad desku nakloněn pod úhlem 45°. Červené linie potom znázorňují vrchní a spodní hrany frusta.



Obrázek 3.7: Schéma úhlu záběru Kinectu. Vlevo pohled y boku, kdy je Kinect uměn 45 cm nad podložkou. Modře je vyznačeno jeho pohledové frustum. V pravo je znázorněn pohled shora. Červeně je vyznačen průnik pohledového frusta Kinectu a desky stolu. Na vodorovné ose jsou vyznačeny vzdálenosti od Kinectu.

V této části je také potřeba zmínit celkovou kalibraci scény, která bude probíhat ve třech fázích. První část je provedena při instalaci základních stanic systému HTC Vive a je provedena z velké části automaticky a za drobného přispění uživatele, a to v prostředí SteamVR. Druhá část kalibrace je manuální. Jelikož je velká část rozpoznávání objektů závislá na barvě, je proto potřeba kalibrovat i kameru samotnou. Zde se jedná o nastavení délky expozice, kompenzace protisvětla a nastavení vyvážení bílé a to na základě okometrického měření. Verifikace této části kalibrace se dá provézt za běhu aplikace. Celý postup této procedury budu ještě popisovat v kapitole 5.

Třetí a poslední typ kalibrace bude probíhat za běhu programu, kdy bude průběžně sledována poloha Leap Motionu a Kinectu. V testovací fázi bude sledování obou zařízení prováděno za pomocí HTC Vive trackerů. Jedná se o drobná zařízení, která jsou sledována základními stanicemi HTC Vive a poskytují data o své přesné pozici. V další fázi se počítá s připevněním Leap Motionu přímo na Vive headset, který bude sám zajišťovat sledování polohy. HTC Vive s připevněným Leap Motionem a Vive tracker je možno vidět na obrázku 3.8.

3.3.2 Sledované objekty

V této fázi vývoje bylo potřeba také důkladně rozvážit, s jakými předměty konkrétně se bude v aplikaci pracovat. V souladu se zadáním práce a předpokládanými softwarovými nástroji, které budu popisovat v kapitole

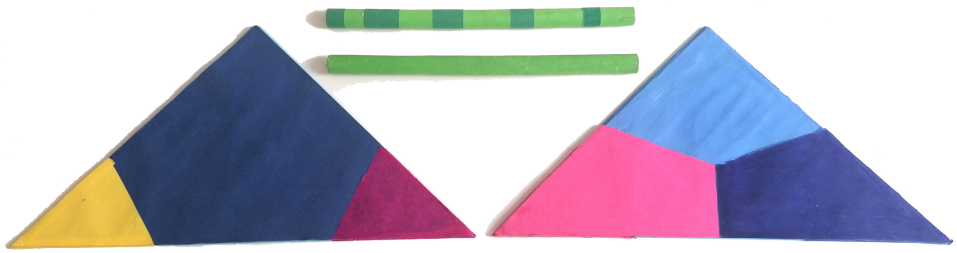
Obrázek 3.8: HTC Vive s upěvněným Leap Motionem a Vive tracker

4.1, bylo potřeba se držet několika omezení. V první řadě tím, že se má jednat o základní kancelářské potřeby tj. předměty, které jsou uživateli již předem známy. Zároveň by bylo vhodné, aby tvar předmětů byl dostatečně obecný, aby se jim v případě potřeby dalo připodobnit třeba i větší množství konkrétních nástrojů. Kupříkladu, že podlouhlý váleček může být jak tužka, tak propiska, nebo třeba i štětec. Druhé omezení plyne z užité knihovny a postupu. Předměty se musí skládat z určitých základních těles a tvarů jako je válec, kruh, plocha a koule. Je zde sice i potenciál možnosti nadefinovat vlastní tvary, ale toto řešení by pravděpodobně bylo poměrně složité na implementaci a i výpočetně velmi náročné.

Brzy na začátku testování kamer bylo jasné, že bude třeba použít předměty z materiálů, které budou mít co nejmatnější povrch. U lesklých předmětů totiž dochází k odrazu laserových paprsků vyslaných time-of-flight senzorem a důsledkem toho nastává chyba v měření, případně dochází ke kompletnímu zneplatnění měření v daném bodě, což jsem ukazoval v části 2.3.5. Zároveň u lesklých předmětů dochází k problémům, kdy se v nich může odrážet příliš velké množství světla, případně i barva jiných předmětů. Toto může vézt k špatnému určení barvy daného předmětu, které je pro další části programu kritické, což budu dále rozebírat v části 4.2. Nakonec byly jako nejvhodnější předměty vybrány tužka a pravoúhlý trojúhelník. Aby bylo možné dobře experimentovat s konkrétními tvary a vlastnostmi předmětů, bylo třeba vytvořit vlastní modely. První nápad byl 3D tisk modelů. Tato metoda by ulehčila práci tím, že by bylo mnohem jednodušší tvořit větší množství standardizovaných modelů, které by měly poměrně přesný tvar a rozměry. Tento přístup má ale i své nevýhody. První je problematičnost tisku vícebarevných modelů. Druhým problémem je materiál, kdy se jedná o plast, který by byl bez dalšího zpracování lesklý, a tudíž pro tuto aplikaci nepoužitelný. Zároveň by předměty měly bez dalšího opracování ono pro 3D tisk charakteristické vroubkování, které je pro sledování hloubkovou kamerou rovněž nežádoucí.

Nakonec jsem se rozhodl pro tvorbu modelů z barevných kartonů. Toto rozhodnutí mělo hned několik důvodů. V první řadě jednoduchá tvorba modelů. Za druhé, kartony se poměrně málo lesknou, bude tedy jednoduší je snímat hloubkovou kamerou a v neposlední řadě byla rozhodující dostupnost barevných papírů a množství barevných odstínů. Na obrázku 3.9 jsou vyfoceny vyrobené modely. Jedná se o dva modely pravoúhlého trojúhelníku o délce odvěsen 18 cm a přepony 25,5 cm. Jejich tloušťka je zhruba 3 mm. V horní části jsou pak válečky imitující tužky. Jejich rozměry jsou 18 cm na délku a průměr činí 1 cm.

V této fázi práce jsem zároveň provedl několik výpočtů za účelem vhodných rozměrů sledovaných těles. Jelikož hloubková kamera má poměrně malé rozlišení, konkrétně v případě Kinectu v2 512 x 424 pixelů, tak je potřeba předměty udělat dostatečně veliké, aby je bylo možné spolehlivě rozpoznat. Když se vrátím k trojúhelníku zmíněném výše, který je také znázorněn v obrázku 3.7, tak ve vzdálenosti, kde je trojúhelník znázorněn, tedy 63,5 cm od kamery, jeho 1 cm odpovídá 5,7 pixelům. Pokud tedy odvěsna měří 18 cm, tak bude odpovídat zhruba 103 pixelům, což by mělo být naprosto dostatečné rozlišení pro estimaci polohy trojúhelníku. Pokud trojúhelník posuneme na přední hranu stolu, tedy přímo před uživatele, obdobným výpočtem zjistíme, že 1 cm odpovídá 3,4 pixelům. Odvěsna tohoto trojúhelníku pak bude zabírat 61 pixelů, což je opět množství pixelů, které by mělo stačit pro bezproblémové rozpoznání objektu.



Obrázek 3.9: Fotografie sledovaných objektů

3.4 Zpracování obrazu

3.4.1 Zpracování vstupních dat

Po výběru záznamového zařízení a rozmyšlení scény je potřeba ze zařízení získat obraz. Za tímto účelem bývají pro hloubkové kamery vydávány oficiální SDK (Software Developement Kit), což jsou sady vývojových nástrojů, poskytujících rozhraní pro komunikaci se zařízením. V případě kamery dávají vývojáři prostředky k její inicializaci, nastavení a v neposlední řadě dodávají rozhraní pro získávání video streamu ze zařízení. Pro Kinect existuje Kinect for Windows SDK 2.0[Mic14] respektive 1.8 pro první generaci Kinectu. Toto

SDK zároveň obsahuje Kinect Studio, což je aplikace pro správu kamery a vizualizaci získaných dat z ní. Podobně k BlasterX Senz3D je možné přistupovat přes Intel RealSense Software Development Kit, který je unifikovaný pro všechny RealSense kamery. Tato rozhraní nám dávají přístup k proudu hloubkových dat z kamery, kterému se říká mračno bodů. Zároveň tato zařízení disponují i RGB kamerou, ke které se dá přistupovat buď separátně jako ke každé jiné RGB kameře a nebo speciálně u Kinectu existuje RGBPointcloud případně RGBAPointcloud, kdy se jedná o strukturované nebo nestrukturované mračno bodů, kde je u každého bodu ještě uložena jeho barvená RGB respektive RGBA hodnota[Mic14].

3.4.2 Použité knihovny

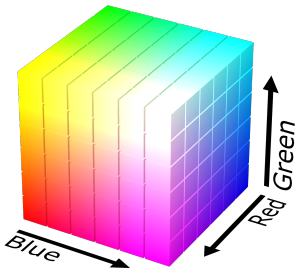
Pro zpracování samotných dat jsem použil dvě knihovny třetích stran a to konkrétně OpenCV a PCL - Point Cloud Library. OpenCV - Open Source Computer Vision Library je knihovna distribuovaná pod BSD licencí zajišťující základní přístup ke zdrojům obrazu a poskytující prostředky pro počítačové vidění. Má zabudované algoritmy pro rozpoznávání objektů a obličejů, sledování pohybu jak kamery, tak snímaných objektů, práci s mračny bodů a mnoho dalšího. Pro účely této práce knihovna poskytuje nástroje pro filtrování obrazu, maskování obrazu a funkce pro vizualizace. Co se filtrování týče, tak všechny výše zmíněné kamery poskytují barevnou informaci v RGB případně RGBA. RGB je aditivní model míchání barev založený na skládání třech základních barev - červené, zelené a modré. Každá barva z barevné palety je jednoznačně určena množstvím každé z těchto třech základních barev. V případě RGBA je k těmto třem hodnotám přidána ještě takzvaná α hodnota, která značí průhlednost.

3.4.3 Barevné modely

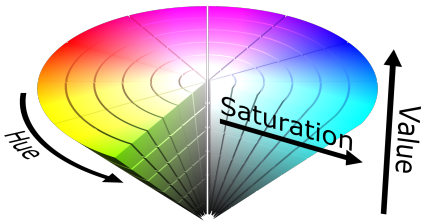
Barevný model RGB je aditivním barevným modelem, jehož barevný prostor si můžeme představit jako jednotkovou krychli, podobně jako na obrázku 3.10, kde každá kladná poloosa značí hodnotu jedné barevné složky. Objem krychle potom představuje celý barevný prostor, neboli všechny zobrazitelné barvy, čemuž se také říká barevný gamut. Jedná se ve své podstatě o jeden z nejjednodušších barevných modelů. Tím pádem je pro většinu uživatelů lehce pochopitelný, má však ale několik nevýhod. První z nich plyne z definice aditivního míchání barev, které funguje na míchání jednotlivých vlnových délek světla, a nikoliv barevných pigmentů. To znamená, že přidáváním každé barevné složky barvu postupně zesvětlujeme tak, až se dostaneme k barvě bílé. Oproti tomu, když jsou všechny barevné složky nulové, tak v podstatě žádné světlo nemáme a finální barva je tedy černá. Toto celé může být pro některé uživatele kontraintuitivní. Druhá nevýhoda se objevuje právě v počítačovém

vidění, kde se můžeme setkat s tím, že chceme vybrat pouze kupříkladu červené objekty. V tomto modelu však musíme vybírat intervaly na všech třech barevných poloosách, což může být náročné, zdlouhavé a nepřesné.

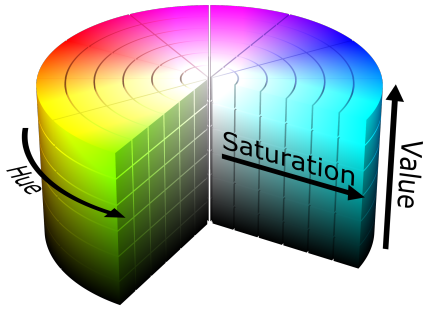
I z těchto důvodů byl vytvořen barevný model HSV - Hue Saturation Value neboli odstín, sytost a hodnota jasu, který je považován za nejpodobnější barevný model ke způsobu, jakým vnímají barvu lidé. Standardně je vizualizován jako kužel na obrázku 3.11 případně válec jako na obrázku 3.12. Odstín standardně nabývá hodnot 0-360, které značí stupně na kruhové podstavě tělesa a na této stupnici se vybírá základní odstín barvy. Další parametr je sytost barvy nabývající hodnotu <0,1> a je znázorňován jako vzdálenost od středu podstavy ve směru kolmém k výšce jehlanu. Jako poslední je hodnota jasu, která je znázorňována ve směru výšky jehlanu případně válce. Když se vrátím zpět ke srovnání tohoto barevného modelu s vnímáním barvy člověkem, je zde vidět podobnost v tom, že lidský mozek je také zvyklý barvu popisovat jako odstín společně s jeho sytostí, případně tmavostí, nikoliv jako míchání základních barev. V praxi tedy člověk standardně řekne, že objekt je třeba světle modrý a nikoliv, že je v něm většina modré složky a hodnoty složky červené a zelené jsou zanedbatelné.



Obrázek 3.10: Vizualizace RGB ve formě jednotkové krychle



Obrázek 3.11: Kuželovitá vizualizace HSV



Obrázek 3.12: Válcovitá vizualizace HSV

3.4.4 Práce s hloubkovými daty

Pro práci s hloubkovými daty jsem se rozhodl pro využití knihovny Point Cloud Library. Jedná se o open source projekt zaměřený na zpracovávání 2D/3D obrazu a mračen bodů. Její součástí jsou algoritmy pro filtrování, feature estimation (rozeznávání významných vlastností), rekonstrukce povrchů, registrace mračen bodů, fitování (prokládání), rozpoznávání objektů a segmentace. Příklady aplikací těchto algoritmů jsou filtrování nepřesností ze zašuměných dat, spojování mračen bodů do jednoho, segmentování relevantní částí scény nebo třeba pro rozpoznávání objektů na základě jejich geometrie [RC11]. Knihovna pro svou činnost zároveň využívá další knihovny třetích stran a to konkrétně knihovny Boost, Eigen, FLANN, Qt, QHull a OpenNI. Knihovna je v projektu používána na většinu operací s mračnem bodů. Data z kamery jsou získávána za pomocí grabberu založeného na [Sug16] a [Yus19].

Existuje množství různých algoritmů na fitování mračen bodů [MM15], ale pro tuto aplikaci je nezbytný RANSAC - Random sample consensus algoritmus, jehož pseudokód je uveden níže. Jedná se o iterativní algoritmus pro fitování objektů do množiny vstupních dat. Základní myšlenka algoritmu by se dala přirovnat k mnohem známějšímu algoritmu nejmenších čtverců, kde je snaha o minimalizaci součtu druhých mocnin vzdáleností vkládaného modelu od vstupních dat. Pro prokládání modelu mračnem bodů je ale metoda nejmenších čtverců nevhodná, protože každý bod z mračna, který není součástí fitovaného modelu může potenciálně výrazně narušit výsledek prokládání. Potřeba je tedy použít jiný algoritmus, kterým je právě RANSAC. Tento iterativní algoritmus v každé iteraci náhodně vybere předdefinované množství bodů z mračna a pokusí se jimi proložit hledaný model. Algoritmus spočítá úspěšnost tohoto proložení podle toho, kolik bodů celkem náleží takto proloženému objektu. V další iteraci vybere opět náhodnou podmnožinu bodů z celého mračna, opět proloží model a spočítá body, které modelu také náleží. Mezi iteracemi si algoritmus pamatuje pouze proložení, které bylo nejúspěšnější, tedy proložení, kterému náleželo největší množství bodů. Aby algoritmus měl do určité míry omezenou délku běhu, tak je ještě na začátku potřeba nadefinovat maximální počet iterací, které mají proběhnout. [Col17][Svo09]

```
while iterations < k do
    maybeInliers := n randomly selected values from data
    maybeModel := model parameters fitted to maybeInliers
    alsoInliers := empty set
    for every point in data not in maybeInliers do
        if point fits maybeModel with an error smaller than t
             add point to alsoInliers
    end for
    if the number of elements in alsoInliers is > d then
        betterModel := model parameters fitted to all points
            in maybeInliers and alsoInliers
        thisErr := a measure of how well betterModel fits
            these points
        if thisErr < bestErr then
            bestFit := betterModel
            bestErr := thisErr
        end if
    end if
iterations++
```

Kapitola 4

Implementace

4.1 Práce s 2D RGB obrazem

V první fázi jsem začal pracovat se standardní RGB kamerou, a to konkrétně webkamerou Logitech HD Webcam C270. Tato kamera pracuje na rozlišení 1280 x 720 pixelů. Hlavním účelem bylo vyzkoušení a otestování různých technik a metod zpracování obrazu, a to nejdříve na plošných datech, a to hlavně z důvodu, že práce s nimi je jednodušší a je možné o ní najít větší množství článků a informací. Zaměřil jsem se hlavně na získávání obrazu, možnosti jeho filtrování, práci s barevným prostorem HSV a na základní metody rozpoznávání objektů. Primární bylo si vyzkoušet různé možnosti a algoritmy pro práci s 2D obrazem, které by mohly být dále využity pro práci s hloubkovými daty. Pro výše zmíněné účely jsem si pro práci vybral knihovnu OpenCV a to také z důvodů jednoduché instalace, možnosti základního prototypování v jazyce Python a v neposlední řadě její jednoduchosti používání a přehledné dokumentaci. Co se samotné implementace týče, tak pro načítání obrazu z kamery jsem použil nativní rozhraní OpenCV.

4.1.1 Využití OpenCV

Pro účely této práce jsem se rozhodl pro rozpoznávání objektu podle barvy. Proto jsou data po načtení převedena z výchozího RGB prostoru do HSV pomocí `cvtColor(frame_BGR, frame_HSV, COLOR_BGR2HSV);`, kde frame_BGR je vstupní BGR matice a frame_HSV je výstupní HSV matice. Zajímavost knihovny OpenCV spočívá v tom, že platí konvence, kdy jsou barevné složky RGB prostoru uloženy po složkách v pořadí modrá, zelená, červená, tedy BGR. V rámci OpenCV je toto konzistentní, ale problém může nastat, kdyby bylo tato data potřeba používat zároveň v jiné knihovně. Následně jsem pro jednotlivé hledané barvy vytvořil masky, které z původní HSV matice odmaskují všechny pixely, které nespadají do intervalů hledaných HSV barev. `inRange(frame_HSV, Scalar(low_H, low_S, low_V), Scalar(high_H, high_S, high_V), frame_threshold)`, kde low_H,S,V jsou dolní meze hledané barvy v barevném prostoru HSV, high_H,S,V jsou meze horní a frame_treshold je výsledná maska[Hof17]. Následuje funkce `dialate()`, která nechá vymaskovanou část obrazu se trochu rozrůst, tím je schopná vyhladit

některé nepřesnosti v zaznamenaných barvách pixelů, čímž ulehčí práci následující funkci `findContours()`, která, jak název napovídá, nalezne kontury masky jednak pro vizualizaci a dále pro počítání velikosti vymaskovaného prostoru, čímž je možné získat velikosti objektů. Následovala už pouze vizualizace dat.

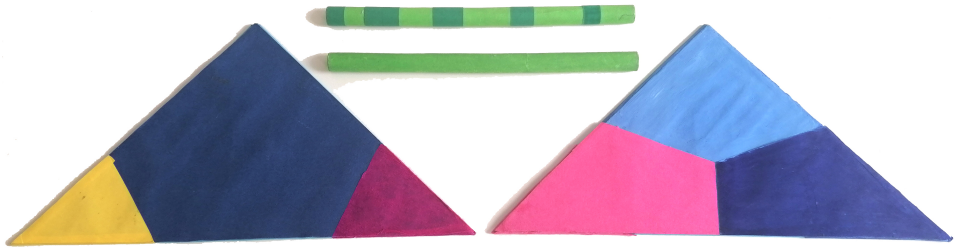
4.2 Práce s hloubkovými daty

4.2.1 Rozeznávání na základě barvy

Než se dostanu k samotné implementaci, tak bych rád popsal celkové uvažování stojící za zvolenou metodou rozpoznávání objektů. Jedná se o virtuálně realitní aplikaci, kdy uživatel má standardně na hlavě nasazený headset, v němž je zobrazována kompletně počítačem renderovaná scéna. Díky tomu mohou mít sledované předměty z vizuálního hlediska v podstatě jakoukoliv podobu. Jednou možností by bylo použití systému určitých značek. Tento přístup se mi ale zdál zbytečně složitý, a proto jsme přistoupili k rozlišování pouze na základě barvy. V první fázi se jednalo pouze o jednobarevné objekty. Z takto získaných dat lze u jednoduchých objektů získat jejich polohu a do určité míry i rotaci. Pokud je však předmět nějakým způsobem souměrný, tak zde dochází pouze k hádání správné rotace. Proto bylo potřeba najít způsob rozlišení jednotlivých stran předmětů. Za tímto účelem vzniklo, jak je vidět na obrázku 4.1, více verzí jednotlivých objektů. U tužky bylo řešení poměrně jednoduché. Otáčení podél její hlavní osy nebylo potřeba řešit, jelikož to u tohoto tvaru zkrátka není pozorovatelné. Úplně nahoře je ukázána dvojbarevná verze tužky, která ale nakonec také použita nebyla, neboť se jedná pouze o prototyp a u modelu, který je renderován, není poznat, kde je jeho horní a dolní stana.

U trojúhelníku je ale situace jiná. Zde je velmi dobře vidět, jak je přesně natočený. Jeho celkovou rotaci nakonec skládám ze dvou zdrojů. Prvním z nich je výstup z RANSAC algoritmu, který vrací normálový vektor k ploše trojúhelníku. Bylo zapotřebí i zjistit rotaci trojúhelníku podél tohoto vektoru. Z tohoto důvodu jsou oba testované trojúhelníky trojbarevné. První byl testovaný ten vpravo. Zde jsem pro určení jeho orientace používal vzájemnou polohu všech třech barevných částí. Vyskytl se zde ale problém, kdy za určitých světelných podmínek bylo problematické rozpoznat jednu modrou barvu od druhé. Zároveň, i když byly barvy dobře rozpoznány, tak rotace nemusela být přesně zjištěna. K tomu docházelo hlavně z důvodu, že těžiště jednotlivých barevných částí jsou poměrně blízko u sebe. Stačila pak i malá nepřesnost v měření a výsledná rotace vyšla výrazně jinak. K vyřešení obou zmíněných problémů jsem vytvořil nový, upravený model. Po dlouhém testování za různých světelných podmínek jsem nakonec vybral barevnou kombinaci žlutá, tmavě modrá a fialová. Tímto krokem jsem se zbavil problémů s rozlišováním jednotlivých barev. Druhý nedostatek jsem alespoň částečně opravil jiným vzájemným rozmístěním žluté a fialové části. Jejich těžiště jsou nyní dál

od sebe. Jelikož rotaci trojúhelníka vypočítávám odečtením jednoho těžiště od druhého, je nyní získávání této informace jednodušší a přesnější. To, jak přesné toto rozeznávání je, budu popisovat v kapitole 5.



Obrázek 4.1: Fotografie sledovaných objektů

4.2.2 Implementace rozpoznávání

Pro tuto část projektu jsem se rozhodoval mezi Kinectem v2 a BlasterX Senz3D a nakonec vybral Kinect. Hlavním důvodem tohoto rozhodnutí byla dle mého názoru lepší integrace zařízení s mnou používanými knihovnami. V druhé řadě jsem usoudil, že obnovovací frekvence 30 snímků za sekundu bude dostačující. Obzvlášť také proto, že většina vývoje probíhala na notebooku. Kvůli jeho omezenému výkonu bylo jasné, že výpočty a operace s daty budou pomalejší a výsledek tedy nebude možné vracet třicetkrát za sekundu. Pro představu se jednalo v první fázi o dvoujádrový, čtyřvláknový procesor Intel i5-5200U běžící na frekvenci 2,70 GHz. Později byl nahrazen za stroj s čtyřjádrovým, osmivláknovým procesorem Intel i5-8265U, který v plné zátěži pracuje na taktu jen okolo 2 GHz. Později jsem i Kinect v2 nahradil Azure Kinectem. Dále tedy budu popisovat implementaci použitou právě pro toto zařízení.

Pro práci s mračnem bodů jsem si vybral PCL - Point Cloud Library. Co se týče komunikace s kamerou, tak PCL standardně využívá knihovnu OpenNI2 a to konkrétně OpenNI grabber, který ale v kombinaci s PCL Kinect přímo nepodporuje. Bylo tedy potřeba najít jinou cestu. Nejdříve jsem se snažil napsat si vlastní grabber, později jsem ale narazil na práci Tsukasa Sugiura, který v současné době spolupracuje na vývoji PCL. Vycházel jsem tedy z článku na jeho blogu, kde přesně popisuje, jak načítat data z Kinectu do datových struktur využívaných v PCL [Sug16]. Kód načítá data z Kinectu v podobě organizovaného mračna bodů do `pcl::PointCloud<pcl::PointXYZRGBA>::Ptr cloud;`. Jedná se o mračno PointXYZRGBA, kde, jak už název napovídá, při načítání dochází k zarovnání dat z RGB kamery na jednotlivé prvky mračna bodů. Každý bod obsahuje informaci jak o své pozici, tak i o své barvě a průhlednosti. Než ale vůbec dojde k získání dat z kamery, je jí potřeba nainicializovat. O to se stará funkce `void pcl::KinectAzureDKGrabber::setupDevice(...)`, která za své para-

metry bere id incializovaného zařízení - tedy Kinectu, hodnotu módu snímání hloubkové kamery, barevný formát barevné kamery a rozlišení barevné kamery. Následně jsou ve funkci `void pcl::KinectAzureDKGrabber: :start()` nastaveny parametry kamery, kterými jsou délka expozice, vyvážení bílé, jas, kontrast, saturace, ostrost a gain. S touto konfigurací může být pak kamera spuštěna. Zároveň bych rád podotkl, že takovéto možnosti nastavení barevného obrazu má pouze Azure Kinect, což byl jeden z rozhodujících důvodů, proč většina testování probíhala právě na něm. Posledním krokem inicializace je spuštění komunikace mezi touto částí aplikace a aplikací běžící v Unity, která s daty získanými z PCL vykresluje finální scénu. Tento účel plní funkce `int initNetworking()` v souboru networking.cpp, jejíž návratová hodnota je 0 při úspěšné inicializaci a 1 při nějaké chybě.

4.2.3 Zpracování dat

Nyní se dostáváme do fáze práce se samotnými daty. Zpočátku je nutné omezit množství dat, se kterými se pracuje. Data jsou nejdříve profiltrována podle jejich `z` souřadnice, tedy vzdálenosti od kamery. Zde jsou zahozeny všechny body, které jsou od kamery dále než 90 cm a to z důvodu, aby se nadbytečně nemusela zpracovávat data mimo předpokládanou oblast výskytu sledovaných objektů. V PCL sice pro toto existuje pass throught filter `pcl::PassThrough<pcl::PointXYZRGBA>`, ale pro zvýšení výkonu aplikace jeho práci provádím ručně přímo při načítání dat [CM16]. Dále podobně jako při práci v OpenCV by mohlo dojít k převedení celého RGB mračna bodů do HSV mračna bodů. Je zde však jeden problém, neboť knihovna umí efektivně převézt bod typu XYZRGBA do formátu XYZHSV, ale už není možnost, jak body převést zpět do formátu XYZRGBA, se kterým pracují další algoritmy. Sice v knihovně existuje `pcl::PackedRGBComparison< PointT >` a `pcl::PackedHSIComparison< PointT >` , ale pro přehlednost jsem se rozhodl filtrování napsat sám. Jedná se o jednoduchý průchod získaným mračnem bodů, kde je u každého bodu zjištěna jeho HSV hodnota. Podle toho, zda spadá do hledaného intervalu, je odpovídající bod vykopírován do nového, barevně příslušného mračna bodů. Nevýhodou však je paměťová náročnost, kdy vytváříme další datové struktury, kam ukládáme duplicitní data. Toto by se dalo vyřešit použitím pole odkazů do původního mračna, ale je možné, že by to naopak ztížilo další operace, které na takové řešení nejsou připravené.

4.2.4 Fitování objektů

Nyní je potřeba v jednotlivých barevných mračnech zjistit polohu předmětů, které hledáme. K tomu v této aplikaci využívám RANSAC algoritmus. V Point Cloud Library je předdefinováno několik základních tvarů, které lze pomocí tohoto algoritmu hledat. Vzhledem k vybraným objektům v této práci používám vyhledávání modelů `pcl::SACMODEL_CYLINDER` pro fitování tužky a `pcl::SACMODEL_PLANE` pro fitování barevných ploch trojúhelníku. Pro optimální fungování algoritmu je třeba vhodně nastavit

několik proměnných. Jejich hodnoty je možno nalézt v tabulce 4.1. Pro definitivní určení, zda mračno bodů odpovídá hledanému objektu, a které body přesně objektu náleží, je použit RANSAC algoritmus. Ten vrací mračno bodů, které odpovídá hledanému předmětu a zároveň se z něj dají získat parametry daného proložení objektem[Rus16]. Pro plochu jsou jejími parametry hodnoty Hessiánské normálové formy. Pro válec se jedná o estimaci jeho těžiště, směru jeho hlavní osy a poloměr podstavy. Estimace těžiště válce není, bohužel, moc přesná a u plochy žádná taková hodnota dokonce neexistuje. Proto jsem se rozhodl určovat pozici jednotlivých těles pomocí výpočtu těžiště bodů, které mu pravděpodobně náleží. K tomu slouží funkce `pcl::compute3DCentroid(const pcl::PointCloud< PointT > &cloud, Eigen::Matrix< Scalar, 4, 1 > &centroid)`, které je předáno mračno bodů a vektor, do kterého zapíše hodnoty souřadnic x,y,z v kterých leží těžiště.

typ objektu	tužka	tmavě modrá část trojúhelníku	žlutá a fialová část trojúhelníku
typ modelu (pcl::SACMODEL_)	CYLINDER	PLANE	PLANE
počet iterací	100	1000	1000
váha normálových vektorů	0.1	0.05	0.5
hranice vzdálenosti	0.015	0.002	0.002

Tabulka 4.1: Tabulka parametrů RANSAC algoritmu pro jednotlivé předměty

4.2.5 Komunikace s Unity

Jelikož neexistuje žádná rozumná možnost integrace Point Cloud Library do Unity, bylo potřeba také vyřešit komunikaci programu na rozpoznávání s Unity. Pravděpodobně je více možností, jak by se tento problém dal řešit, ale jako nejvýhodnější mi vyšlo nechat obě aplikace komunikovat přes TCP protokol. Neprovádím žádné složité navazování spojení. Jedná se čistě o klient-server komunikaci, kdy je odesílán jen jeden typ zprávy obsahující parametry objektu. Ty jsou před odesláním převedeny z floatu na int a po přijetí jsou opět převedeny zpět na desetinná čísla. První je id objektu, pro určení, o který objekt se jedná. Následuje trojice intů definujících polohu, poté 6 intů definujících rotaci. Tři, které definují forward vektor a tři up vektor. Pro válec jsou poslední tři hodnoty nulové, jelikož jeho rotace podél hlavní osy není nijak zjišťována.

4.2.6 Unity

Pro tvorbu samotné virtuálně realitní aplikace bylo jasnou volbou využití herního enginu Unity. Prvním důvodem volby byly moje předchozí zkušenosti s vývojem na této platformě a druhým, neméně důležitým důvodem, byla dobrá podpora SteamVR, HTC Vive a Leap Motionu. Stěžejní částí je skript

TCPTestServer.cs, který plní roli serveru pro komunikaci s částí aplikace využívající PCL. Jeho funkcí je tedy navázání spojení a přijímání parametrů jednotlivých předmětů, převod dat zpět na desetinná čísla a následně vhodná aplikace transformací na jednotlivé předměty. Součástí scény je několik objektů. Ty bych rozdělil do dvou skupin - aktivní a pasivní. Pasivní jsou objekty, kterými je manipulováno a to na základě dat z objektů aktivních. Pasivními jsou deska stolu, a modely trojúhelníku a tužky. Prvním aktivním objektem je Kinect rig, ten dostává data z jednoho z HTC Vive trackerů a tím je definována jeho poloha. Vůči této poloze jsou také renderovány sledované předměty. Druhým aktivním objektem je Leap Motion rig, který také svou polohu upravuje podle dat získaných z jednoho z Vive trackerů. Jelikož se počítá s připevněním Leap Motionu na náhlavní displej, tak je součástí tohoto rigu také hlavní virtuální kamera. Veškeré sledování a renderování uživatelových rukou je v režii oficiálního prefabu Leap Rig.

Kapitola 5
Testování

5.1 První fáze testování

První část testování jsem prováděl v dobře osvětlených prostorech a to na třech konkrétních předmětech, modrém plastovém kelímku, žlutém kartonovém kelímku a pro simulaci většího mračna bodů jsem používal modrou mikinu. V této části testování jsem se zaměřil hlavně na segmentaci hloubkových dat a to z pohledu rychlosti a přesnosti segmentace. K získávání hloubkových dat jsem v této části práce používal zatím ještě jen Kinect v2. První problém, se kterým jsem se při testování setkal, byl v případě plastového kelímku, který je vidět na obrázku 5.1. Ten je sice z velké většiny matný, ale některé jeho části jsou lesklé. Této vlastnosti jsem při barevném filtrování přisoudil i poněkud velké množství bodů, které ležely v jeho blízkosti, měly stejnou barvu jako kelímek, ale očividně nebyly jeho součástí, jak je vidět na obrázku 5.1. Jelikož oba další předměty byly takřka perfektně matné a podobné artefakty u nich vidět nebyly, usoudil jsem, že se jedná o nepřesnosti měření právě vlivem lesklého povrchu.

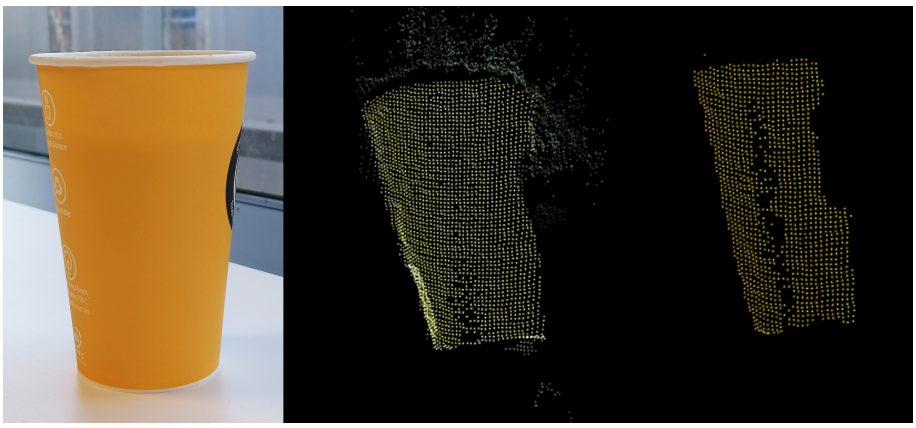
Druhou skutečností, která při testování nebyla zcela v pořádku a nebyl jsem jí schopen nijak vysvětlit, bylo množství vykreslených snímků za sekundu. I přes veškeré snahy o získávání omezeného množství dat z Kinectu a různých dalších optimalizací, nebylo možné dosáhnout 30 snímkům za sekundu. Tedy dostat se na obnovovací frekvenci, na které by měl Kinect scénu snímat. Během všech měření jsem nedosáhl více něž stabilních 22 snímků za vteřinu. Tímto se dostávám obecně k metodice měření. Ta byla prováděna při spuštění programu z Visual Studia 2017 a to v Release módu. Celý program běžel na počítači s procesorem Intel i5-8265U se čtyřmi jádry a 8 vlákny, který v dané zátěži pracoval zhruba na frekvenci 3,10 GHz. Procesoru sekundovala grafická karta Nvidia MX250. Některé testy jsem prováděl i na počítači s šestijádrovým procesorem Intel i5-8600k pracujícím na frekvenci 4,5 GHz, kde jsem sledoval nárůst výkonu zhruba o 20 %. Pro neúplnost těchto testů zde jejich přesné výsledky uvádět nebudu. Vykreslování celého mračna, tedy celých 217088 bodů, program zvládal zhruba o frekvenci 22 Hz. Při vykreslování mračna bodů, které odpovídalo hledanému předmětu, tedy nafitovanému žlutému kelímku, bylo možné dosáhnout obnovovací frekvence zhruba 17

Hz. Postup segmentace kelímku je možno vidět na obrázku 5.2. Tento velmi malý pokles výkonu při přidání RANSAC algoritmu přisuzuji velmi dobře profiltrovaným datům, nad kterými byl algoritmus schopný nalézt optimální proložení modelem velmi rychle.

velikost mračna bodů	60000	7000	<100
Barevný a vzdálenostní filtr	17fps	18fps	19fps
Barevný a vzdáleností filtr, výpočet těžiště	16fps	17fps	19fps

Tabulka 5.1: Hodnoty fps - množství vykreslených snímku za sekundu v závislosti na velikost vykreslovaného mračna a aplikovaných výpočtů

Obrázek 5.1: Plastový kelímek a velké množství chybně zaznamenané body

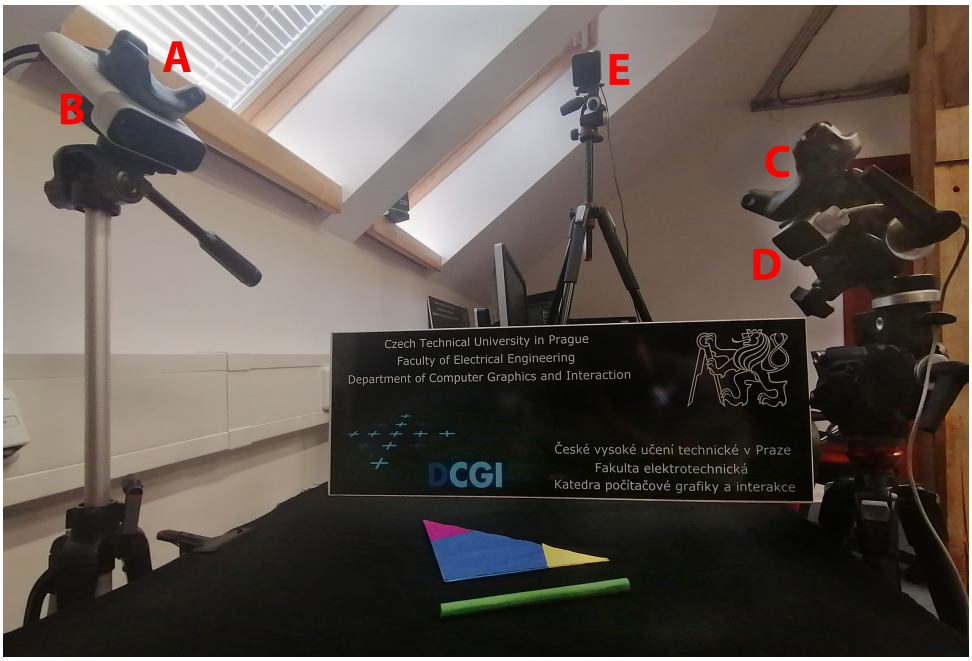


Obrázek 5.2: Papírový kelímek, profiltrované mračno bodů a mračno bodů, které které označil RANSAC jako náležící kelímku

5.2 Druhá fáze testování

5.2.1 Rozložení scény

Tato fáze již probíhala na konci celé práce a to tedy s finálním hardwarem a kompletní aplikací. Rozestavení zařízení v testovací scéně je vyobrazené na obrázku 5.3. Na fotografii je vidět pod písmenem A HTC Vive tracker přilepený na Azure Kinect (B). Pomocí toho trackeru je sledována přesná poloha a náklon Kinectu. S využitím tohoto údaje je kalibrována přesná poloha renderovaného trojúhelníku a tužky vůči stolu. V tomto stavu je Kinect zhruba 45 cm nad stolem a je nakloněn pod úhlem přibližně 45°. Naproti němu je v pravé části umístěn opět na stativu druhý Vive tracker (C), který zprostředkovává polohová data pro pod ním umístěný Leap Motion (D). Tato zařízení jsou umístěna 40 cm nad hranou stolu, což je pozice, ve které se předpokládá, že bude zhruba uživatelova hlava. Hlavně z důvodu jednoduššího vývoje a možnosti opakování stejného měření, je Leap Motion umístěn na stativu a ne na headsetu, který by měl uživatel standardně nasazený. V pozadí, u písmene E, je na stativu umístěna jedna ze základních stanic HTC Vive, která se stará o sledování HTC zařízení ve scéně, tedy trackerů, headsetu, případně ovladačů. Jelikož pro sledování je oficiálně doporučováno použití dvou základních stanic [Cor16], tak druhá je upevněna v podobné pozici, jen na opačné straně. Za povšimnutí stojí také deska stolu, na které je položena černá látka. Jelikož se jedná o matnou a barevně neutrální textilii, tak je na ní poměrně jednoduché rozlišovat jednotlivé barevné předměty.



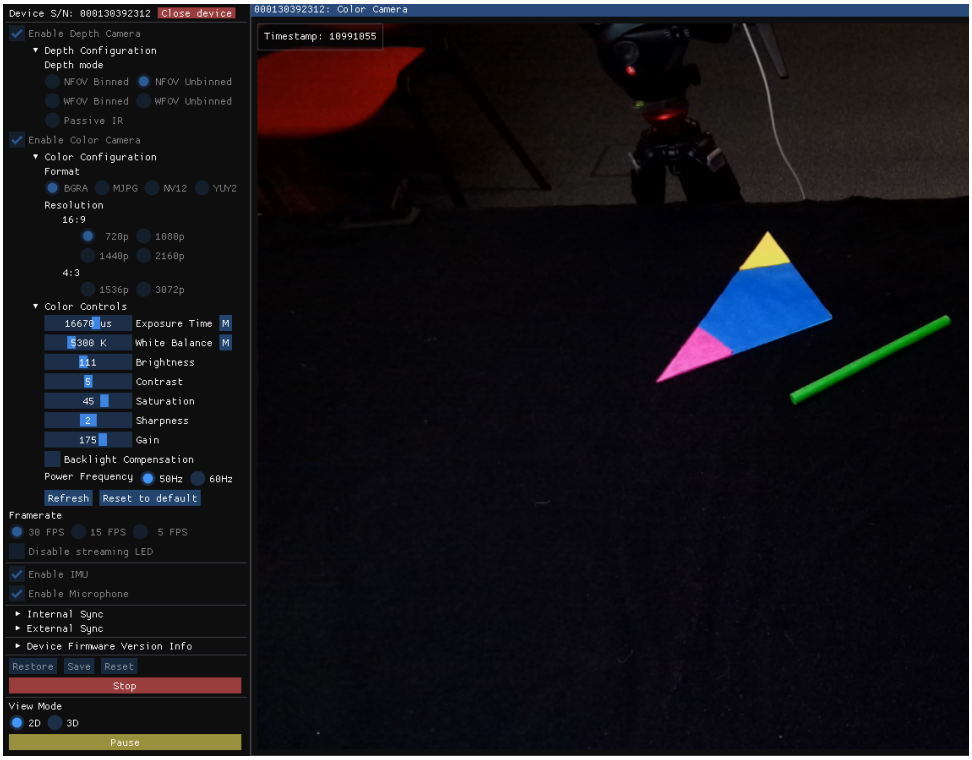
Obrázek 5.3: Fotografie testovacího rozložení.

5.2.2 Barevná kalibrace

Při každé změně světelných podmínek je vhodné zkalibrovat barevná data získávaná z hloubkové kamery. Za tímto účelem je vhodné spouštět aplikaci s parametry `drawTriangle` a `drawPen`. Spustí se sledování scény a otevřou se dvě okna, jedno z nich je konzole a druhé je OpenGL okno. V druhém jmenovaném by se do několika vteřin po spuštění měl začít vykreslovat profiltrovaný obraz z kamery. Pokud je na obraze vidět bez větších problémů celý trojúhelník a celá tužka a zároveň je vykresleno velmi malé množství jiných bodů, trojúhelníku ani tužce nenáležejících, je aplikace zkalibrovaná na dané světelné podmínky a je připravena k použití. Pokud jsou vykresleny nějaké další barevné plochy mimo předměty, je potřeba kameru zkalibrovat. Zde se jedná, bohužel, o jedinou část kalibrace, která je čistě manuální.

Pro zkalibrování je potřeba otevřít aplikaci Azure Kinect Viewer, která by se měla automaticky nainstalovat společně s instalací Azure Kinect Sensor SDK. Pokud je k počítači připojený Azure Kinect, tak po spuštění Vieweru bude v jeho levé horní části zelené tlačítko `Open device`. Po jeho stisknutí se níže objeví různá nastavení kamery. Nyní je důležitá pouze část `Color Controls`, po jejímž rozkliknutí se objeví několik nastavení barvy. Dalším krokem je stisknutí tlačítka `Start` níže, čímž se spustí záznam scény a v pravé části se objeví několik oken obsahujících výstupy z různých senzorů. Jediné podstatné okno je to s daty z barevné kamery, které je možné stisknutím políčka `+` v jeho pravé horní části maximalizovat. Screenshot z této aplikace je možné vidět na obrázku 5.4.

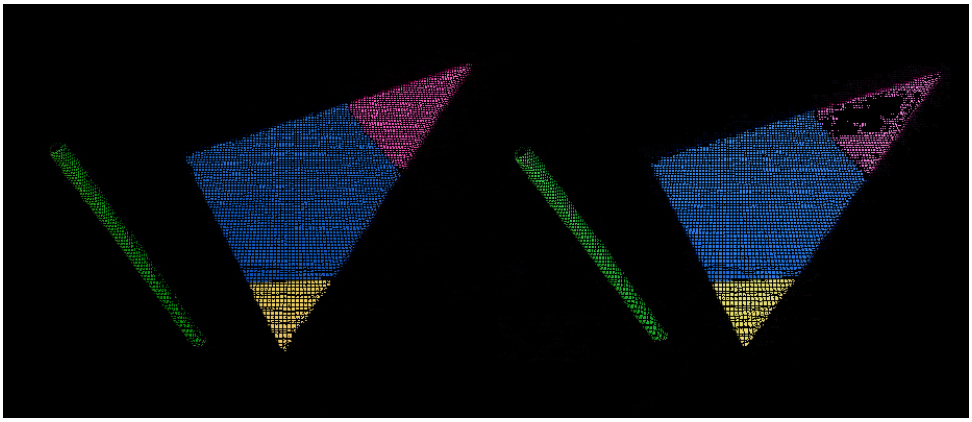
Nyní přichází na řadu samotná kalibrace, kdy je potřeba nastavit hodnoty jednotlivých parametrů, tak aby se obraz barevně co nejvíce podobal obrázku 5.4. Referenční hodnoty nastavení obrazu je možné nalézt v tabulce 5.2. Při testování jsem narazil na bug, kdy nebyla nebyla aplikována hodnota parametru gain. Z tohoto důvodu doporučuji tuto hodnotu nijak nenastavovat a při kalibraci jí nebrat v potaz. Tyto hodnoty odpovídají nastavení při kombinaci částečně zastíněného denního světa se zářivkou připevněnou nad Leap Motionem, která má výkon 68 W a teplotu světla 4000 K. Toto jsou zároveň defaultní data při spuštění aplikace bez jakýchkoliv parametrů. Po zkalibrování obrazu je potřeba vypnout Kinect Viewer nebo minimálně stisknout tlačítko `Close device` v levé vrchní části, jinak nebude možné ke Kinectu přistupovat z žádné další aplikace. Nyní už je možné spustit aplikaci samotnou a to s parametry nastavení barvy, které byly navoleny ve Vieweru. Spouštění aplikace s parametry je popsáno v uživatelské příručce. Na obrázku 5.5 je v levé části vidět, jak by měl vypadat výstup v okně při spuštění aplikace s parametry `drawTriangle` a `drawPen`. V pravé části je výstup v situaci, kdy je obraz moc světlý. Je zde vidět, že část fialové plochy trojúhelníku byla odsegmentována pryč.



Obrázek 5.4: Snímek obrazovky Azure Kinect Vieweru společně s vhodně nastavenými parametry barevné kamery.

Exposure Time	16670 μs
White Balance	5300 K
Brightness	111
Contrast	5
Saturation	45
Sharpness	2
Gain	175

Tabulka 5.2: Defaultní hodnoty nastavení barevné kamery Azure Kinectu



Obrázek 5.5: Screenshot objektů po barevné segmentaci

5.2.3 Testování přesnosti

Při tomto testování jsem se zaměřil na přesnost estimace polohy sledovaných předmětů. Postupně jsem spouštěl sledování a rozpoznávání nejdříve trojúhelníku a následně i tužky, abych mohl srovnat, jaké přesnosti estimace pozice jsem schopný dosáhnout. Pro snímání scény jsem používal Azure Kinect. Při testování jsem postupoval tak, že jsem na několik desítek vteřin spustil sledování, položil objekt na stůl před kameru, čímž jsem zajistil, že se nebude hýbat. Během celého měření jsem si ukládal data o poloze objektu. Po konci měření jsem vybral souvislou podmnožinu naměřených dat. Tato množina měla 70 hodnot, ze kterých jsem následně vypočítal průměrnou pozici objektu a následně i absolutní a směrodatnou odchylku. Výsledky je možné vidět v tabulce 5.3 níže. Výsledky dat měření polohy trojúhelníku je pak možné vidět v tabulce 5.4

Sledovaná hodnota	x v metrech	y v metrech	z v metrech	vzdálenost od kamery
Průměrná hodnota	-0,007172747	-0,017443849	0,622213671	0,729092843
Průměrná odchylka	0,005837863	0,00321572	0,000866698	0,00072779
Směrodatná odchylka	0,006953139	0,003905268	0,001163324	0,000958865

Tabulka 5.3: Měření polohy tužky a hodnoty odchylek těchto měření. Hodnoty měření v metrech

Z provedených měření je zřejmé, že určování pozice je výrazně přesnější u trojúhelníku, než u tužky. Na druhou stranu je potřeba říci, že se v obou případech směrodatné odchylky pohybují podél jednotlivých os v jednotkách milimetrů. Když ale vezmeme tato data v kontextu celé aplikace, tak nepřesnosti v řádech jednotek milimetrů jsou na předmětu o velikosti desítek centimetrů pouhým okem v podstatě nepozorovatelné. Podobně i vzdálenost

Sledovaná hodnota	x v metrech	y v metrech	z v metrech	vzdálenost od kamery
Průměrná hodnota	-0,02646968	0,03593025	0,59301144	0,70717566
Průměrná odchylka	0,000669079	0,000956322	0,000223613	0,000161254
Směrodatná odchylka	0,000906617	0,00131044	0,000301645	0,000211911

Tabulka 5.4: Měření polohy trojúhelníku a hodnoty odchylek těchto měření. Hodnoty měření v metrech

od kamery, vypočítaná pomocí Pythagorovy věty z prvních třech hodnot, byla velmi konzistentní. Směrodatná odchylka této hodnoty byla nakonec menší než jeden milimetr, tedy menší než směrodatné odchylky ostatních hodnot.

K estimaci polohy předmětů neodmyslitelně patří i estimace jejich rotace. Metodika měření stability výpočtů rotací byla podobná jako v případě měření polohy. Výsledky tohoto měření je možné vidět v tabulce 5.5. Z měření je patrné, že podobně jako v případě polohy, je přesnější estimace rotace, tedy výstup z RANSACU u trojúhelníku než u tužky.

Sledovaná rotace	rotace tužky	normálový vektor trojúhelníku	rotace trojúhelníku
Průměrná hodnota a směrodatná odchylka podél osy x v m	0,194232425 0,010289203	-0,00929358 0,002355535	0,00308906 0,001977605
Průměrná hodnota a směrodatná odchylka podél osy y v m	-0,667292932 0,004541805	-0,72859323 0,000781511	0,11523685 0,002016825
Průměrná hodnota a směrodatná odchylka podél osy z v m	0,718930425 0,002558207	-0,68487864 0,000816218	-0,12327615 0,000612367

Tabulka 5.5: Měření stability estimace jednotlivých rotací. Udávané hodnoty jsou jednotlivé složky směrových vektorů.

5.2.4 Testování rychlosti

V této části jsem se zaměřil na testování rychlosti aplikace. Jako jednotku jsem zvolil počet vypočítaných pozic objektů za jednotku času. Podobné měření jsem prováděl již v části 5.1. Tam jsem se ale soustředil na rychlost algoritmu v závislosti na množství vstupních dat a zároveň toto testování bylo ještě zatížené renderováním vstupních mračen bodů. Oproti tomu v tomto testu jsem se soustředil na rychlost zpracovávání dat RANSAC algoritmem a

to v závislosti na hledaném předmětu a výkonu počítače. Samotné měření nebylo nijak složité. Šlo v podstatě jen o spuštění aplikace po dobu 10 sekund a následně o spočítání množství měření, které stihl program vykonat. Testoval jsem postupně sledování každého jednoho předmětu zvlášť a následně obou předmětů zároveň. Výsledky testu je možné vidět v tabulce 5.6. Testování probíhalo na notebooku s čtyřjádrovým a osmivláknovým procesorem Intel i5-8265U, který v tomto zatížení pracoval na taktu 1,8 GHz. Pro srovnání, druhé testování jsem prováděl na stolním počítači osazeném šestijádrovým procesorem Intel i5-8600K pracujícím na frekvenci 4,3 GHz. Z výsledků měření je jednoznačně vidět nárůst výkonu při použití výkonnějšího hardwaru. U sledování pouze trojúhelníku případně obou předmětů najednou je nárůst výkonu zhruba o 120 %, zatímco nárůst výkonu pří sledování pouze tužky je 40%. Nárůst výkonu je pravděpodobně způsoben pouze zvýšením taktu procesoru. Odlišný počet jader v tomto případě pravděpodobně nehraje moc velkou roli a to hlavně z důvodu poměrně slabé paralelizace programu. Ve verzi 1.10.0 PCL knihovny byla přidána lepší paralelizace RANSAC algoritmu, ta ale není podporována ve Visual Studiu a migrace projektu do jiného vývojového prostředí by byla poměrně složitá. Z dat je však vidět, že samotný RANSAC algoritmus není nutně tou největší výkonnostní zátěží celého programu. Velký podíl na délce výpočtů má také zpracování celého mračna bodů a jeho převod do struktur používaných v PCL. Zde by stálo také za úvahu využití paralelizace. Otázkou zůstává, nakolik by tento přístup pomohl, protože s jednotlivými jednotkami dat se neprovádí žádné dlouhé operace, jen množství zpracovávaných dat je opravdu veliké. V neposlední řadě stojí za zmínku fakt, který z naměřených čísel není nutně zřejmý, a to, že na výkonnějším hardware byla aplikace mnohem lépe použitelná a to nejen díky zvýšenému počtu snímků za sekundu, ale i délka odezvy programu byla celkově mnohem kratší.

Sledovaný objekt	trojúhelník	tužka	oba
i5-8265U	3,9 Hz	7,2 Hz	7 Hz
i5-8600K	8,7 Hz	10,1 Hz	16 Hz

Tabulka 5.6: Počet vypočítaných estimací poloh pro jednotlivé objekty za jednu sekundu.

5.2.5 Testování sledování jednotlivých dvou předmětů

V této části jsem se rozhodl zaměřit hlavně na rychlost odezvy celé aplikace a na jevy s tím spojené. Nejdříve jsem otestoval, nakolik a jestli vůbec má pohybování předmětem vliv na přesnost estimace jeho polohy. Provedl jsem 10 měření, kdy jsem nejdříve nechal ležet objekt na jednom místě, následně s ním rychle pohnul o stanovenou vzdálenost, která činila 10 centimetrů. Následně jsem z naměřených pozičních dat zkoumal, zda nedochází k narušení výpočtů případně jestli nevzniká nějaká chyba.

Výsledkem tohoto zkoumání bylo, že v 7 případech z deseti měření byla okamžitě naměřena nová poloha předmětu, jejíž hodnota spadala do směrodatné odchylky měření polohy. Dvakrát došlo k zaznamenání polohy, která náležela trajektorii posunovaného předmětu. V jednom měření došlo k chybě, kdy byla naměřena poloha, která se zhruba o 4 centimetry neshodovala s trajektorií posunu. Toho mohlo být způsobeno buď zastíněním části objektu rukou, případně detekcí ruky, kterou jsem trojúhelníkem pohyboval, jakožto jeho součástí. Vypozoroval jsem tedy, že vlivem samotného posunu standardně nedochází k žádným nepřesnostem v měření.

Dále bych se rád zmínil o limitacích sledování a rozpoznávání, na které jsem při testování narazil. Problémem byly velmi světlé a naopak velmi tmavé barvy. Otázka světlých barev se vztahuje hlavně na situaci, kdy dojde k výraznému nasvícení rukou uživatele, případně k odrazu světla od nich ke Kinectu. V této chvíli se barva lidské kůže může velmi přibližovat barvě žluté plošky na trojúhelníku. Toto v důsledku může vést k chybě estimace rotace trojúhelníku okolo jeho normálového vektoru. Na druhé straně se objevuje problém v situaci, kdy jsou při používání aplikace horší světelné podmínky. V tu chvíli se na černé ploše objevuje barevný šum, který může být v závislosti na nastavení vyvážení bílé detekován buď jako fialová, nebo tmavě modrá plocha. To opět ovlivňuje detekci barevných ploch trojúhelníku, což vede k chybné estimaci jeho polohy. Tento problém se ale dá vyřešit rekalibrací barev kamery a speciálně pak opatrným výběrem hodnoty vyvážení bílé.

5.2.6 Uživatelské testování celé aplikace

V této finální části testování jsem se rozhodl otestovat všechny komponenty aplikace v plném rozsahu. Pro snímání scény jsem použil Azure Kinect, jehož poloha byla sledována pomocí jednoho z Vive trackerů. Leap Motion byl připevněn přímo na náhlavní displej HTC Vive. Tímto odpadla nutnost sledovat pozici Leap Motionu pomocí druhého z trackerů. Ten mohl být tím pádem použit na přesné zjišťování polohy desky stolu, na kterou byl položen. Jelikož testování probíhalo z technických důvodů pouze na ne příliš výkonném notebooku, tak v důsledku nešlo tolik o celkový zážitek z používání aplikace, jako o otestování spolupráce jednotlivých komponent.

Ze softwarového pohledu celkové propojení komponent proběhlo bez větších problémů. V podstatě jedinou problematickou částí byla transformace z pravotočivého souřadného systému používaného v Kinectu do levotočivého souřadného systému používaného v Unity. Zároveň bylo potřeba korektně převést a transformovat souřadnice z lokálního prostoru Kinectu do prostoru celé scény v Unity. Následně bylo potřeba nastavit vhodné odsazení mezi Vive trackerem a počátkem světa tak, jak ho zaznamenává Kinect. Jelikož se Leap Motion po připevnění na HTC Vive kalibruje sám, tak vzniklo korektní rozložení všech objektů.

Nyní už bylo potřeba jen, na základě testování, určit vhodnou metodu interakce rukou s předměty. Jako první možný způsob se ukázalo použití uchopení, které je připraveno jakožto součást Leap Motion SDK. Tato metoda funguje na základě detekce gesta uchopení předmětu, kdy je následně na předmět aplikována stejně rotace a translace jako na ruku předmět držící. Zpočátku bylo poněkud obtížné se ve virtuálním světě na předmět trefit, proto jsem implementoval zastavení pohybu předmětů po přiblížení ruky uživatele. Toto bohužel ale vedlo jen k navýšení rozdílu mezi polohou reálného a virtuálního předmětu. Na základě těchto pokusů jsem dospěl k závěru, že nejkonzistentnější výstup poskytovala metoda, kdy jsem Leap Motion použil pouze na vizualizaci rukou. Poloha předmětů je poté po celou dobu určována pouze z dat získaných z Kinectu. Ve výsledku se podařilo vytvořit aplikaci, kde je možné ve virtuální realitě manipulovat s objekty. Virtuální objekty svou polohou a velikostí odpovídají poloze a velikosti uživatelových rukou. V závěru testování jsem si ale ještě všiml situace, kde s rostoucí vzdáleností od středu stolu se postupně rozchází poloha rukou a virtuálních předmětů. Předpokládám, že tento jev je způsoben zkreslením dat z Kinectu vlivem zakřivení získaného obrazu. Pro vyřešení tohoto problému by ale byla potřeba mnohem hlubší analýza.

Kapitola 6

Závěr

6.1 Shrnutí postupu

Svůj postup při tvorbě obsahu této bakalářské práce bych shrnul do třech fází a to v souladu s tím, jak jsem postupně pracoval. V první fázi jsem se zaměřil vůbec na seznámení se s metodami počítačového vidění a zpracování obrazu. Pro práci nad reálnými daty jsem používal knihovnu OpenCV a pro jejich záznam webkameru od firmy Logitech. Co se samotného programování týče, tak jsem si vyzkoušel převod barevného videa do formátu HSV, a následně segmentaci těchto dat podle barvy. Z rozsegmentovaných barevných ploch jsem následně určoval jejich těžiště a tím pádem polohu odpovídajících barevných předmětů.

Následoval přesun od barevné kamery ke kameře hloubkové a k operacím s hloubkovými daty. Nejdříve jsem provedl rešerši dostupných technologií a metod pro práci s hloubkovými daty. Z té mi jako nejvýhodnější vzešlo používání hloubkových kamer řady Kinect od firmy Microsoft a pro zpracovávání získaných dat jsem si vybral knihovnu Point Cloud Library. V samotné programovací části jsem se snažil aplikovat podobné metody, které jsem využíval při práci jen s barevnými daty. V první řádě to bylo načítání dat z hloubkové kamery. Následovalo převedení barev do barevného prostoru HSV, kde byla data na základě právě HSV hodnoty každého bodu profiltrována. Odtud jsem dostal jednotlivá jednobarevná mračna bodů, která zhruba odpovídala sledovaným předmětům. Z těchto mračen jsem následně aplikací RANSAC algoritmu získal konkrétní pozici a rotaci jednotlivých předmětů.

Nyní se dostávám k fázi třetí a nejrozsáhlejší. Zde bylo potřeba získaná data z RANSAC algoritmu dostat do herního enginu a následně podle nich vyrenderovat příslušné objekty. Jako herní engine jsem si vybral Unity a to hlavně díky velmi dobré podpoře všech zařízení, která jsem při práci používal. V Unity bylo ke sledovaným předmětům přidáno ještě sledování samotného Kinectu, desky stolu a v neposlední řadě také Leap Motionu, kterého jsem využil pro sledování uživatelových rukou. Když jsem měl k dispozici veškerá tato data, tak bylo potřeba je všechna zkombinovat, aby seděly veškeré proporce a pozice jednotlivých objektů ve scéně. Posledním

úkolem bylo už jen vyřešení celkového ovládání aplikace. Zde jsem se, jak jsem již výše v 5.2.6 popisoval, rozhodl pro využití polohy rukou získané z Leap Motionu pouze jako vizuální vodítko, nikoliv jako přímé nástroje pro manipulaci s předměty.

6.2 Závěrečné zhodnocení

V závěru bych rád shrnul několik bodů celé práce, které se dle mého názoru podařily a několik, které by potřebovaly ještě další zkoumání a vývoj. V první řadě se mi podařilo ukázat, že je možné vcelku přesně a rychle estimovat pozici jednoduchého tělesa a to s využitím poměrně základních nástrojů počítačového vidění. Existují však i jiné možnosti jak obdobných výsledků dosáhnout, jako je třeba využití neuronových sítí pro rozpoznávání objektů. Toto by již ale bylo nad rámec této práce. Jako další úspěch bych považoval poměrně dobrou optimalizaci celé aplikace, kdy je možné jí spustit a používat i na poměrně slabém hardware, zejména pokud vezmeme v potaz, jak velké množství dat a z kolika různých zdrojů bylo získáváno. Toto mě vede k poslednímu poznatku, s využitím odpovídajícího úsilí, bylo možné spojit poměrně velké množství různých technologií a postupů za účelem dosažení kýženého cíle.

Na druhé straně je zde však několik bodů, které se z různých důvodů nepodařilo plně dosáhnout. Za jejich společného jmenovatele by šlo označit robustnost celé aplikace. První slabé místo je kalibrace barevného vstupu. Celá aplikace je poměrně závislá na přesném rozsegmentování obrazu na základě barvy, což předpokládá konstantní světelné podmínky. Ty jsou však pouze s využitím denního světla prakticky nedosažitelné. V důsledku toho bylo během testování aplikace v průběhu dne nutné ji několikrát kalibrovat. Jako druhou slabinu této aplikace bych označil nekonzistenci odchylky polohy fyzických a virtuálních předmětů v některých situacích. V tomto případě by ale objasnění tohoto jevu vyžadovalo hlubší analýzu.

Závěrem je tedy možné říci, že na základě zkoumání a srovnávání různých postupů a metod se mi podařilo vytvořit prototyp aplikace pro sledování reálných objektů ve virtuální realitě, kde je uživatel schopen s předměty manipulovat. Chování prototypu se podařilo popsat a ověřit na základě několika testů a provedených experimentů.

Příloha A

Literatura

[BJ00] Tobias Hollerer Bernd Jähne, *Computer vision and applications a guide for students and practitioners)*, Academic Press, 2000.

[CM16] Ming Li Carlos Moreno, *A comparative study of filtering methods for point clouds in real-time video streaming*, October 2016.

[Col17] Robert Collins, *Robust estimation : Ransac*, 2017.

[con18] Wikipedia contributors, *Kinetóza*, 2018.

[con19] ———, *Virtual reality*, 2019.

[con20a] ———, *Computer stereo vision*, 2020.

[con20b] ———, *Leap motion*, 2020.

[con20c] ———, *Raster graphics*, 2020.

[con20d] ———, *Structured light*, 2020.

[con20e] ———, *Triangulation (computer vision)*, 2020.

[Cor16] HTC Corporation, *Vive pre user guide*, 2016.

[Hof17] Janosch Hoffmann, *Development of a modular, turtlebot-based 2d/3d object recognition system*, November 2017, pp. 16–28.

[LRD16] Anna Lekova, D. Ryan, and Reggie Davidrajuh, *Fingers and gesture recognition with kinect v2 sensor*, Information Technologies and Control **14** (2016).

[Mic14] Microsoft, *Kinect for windows sdk*, 2014.

[MM15] Peter Plassmann Andrew Ware Marius Mikni, Ross Davies, *Near real-time point cloud processing using the pcl*, IEEE, september 2015.

[RC11] Radu Bogdan Rusu and Steve Cousins, *3D is here: Point Cloud Library (PCL)*, IEEE International Conference on Robotics and Automation (ICRA) (Shanghai, China), May 9-13 2011.

[Rus16] Radu B. Rusu, *Cylinder model segmentation*, 2016.

[Sug16] Tsukasa Sugiura, *Drawing point cloud retrieve from kinect v2 using point cloud library – part.1 without grabber*, 2016.

[Svo09] Tomáš Svoboda, *Ransac random sample consensus*, nov 2009.

[TS19] Phil Meadows Tetyana Sych, Brent Allen, *Azure kinect dk depth camera*, 2019.

[WRA14] Revan Zuhair Mansoor Wael R. Abdulmajeed, Ph.D, *Implementing kinect sensor for building 3d maps of indoor environments.*

[Yus19] Wu Yusen, *Kinect azure dk programming combat use opencv, pcl to operate and visualize kinect data*, 2019.

Příloha B

Struktura přiložených souborů

```
/
├── bin
│   ├── TCP listener
│   └── ObjectTracking
├── documents
│   └── uzivatelska_prirucka.pdf
├── imgs
├── src
│   ├── TCP listener
│   └── ObjectTracking
├── videos
│   ├── testTuzka.mp4
│   └── testTrojuhelnik.mp4
└── readme.txt
```