FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Sledování telemetrických dat robotů pomocí holografických brýlí
Student:	Pavel Chytrý
Vedoucí:	Ing. Miroslav Skrbek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Počítačové inženýrství
Katedra:	Katedra číslicového návrhu
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Seznamte se s moderními holografickými brýlemi pro rozšířenou realitu, zejména pak možnostmi přenosu dat do brýlí přes wifi a zobrazování údajů v nich. Navrhněte a implementujte programové vybavení pro propojení holografických brýlí a robotů tak, aby se telemetrická data z robotů zobrazovala v zorném poli pozorovatele a bylo tak možné současně pozorovat pohyb robotů a jejich data. Uvažujte i možnost identifikace jednotlivých robotů a přiřazování dat k jednotlivým robotům. Vytvořte demo aplikaci, kde ukážete funkčnost vašeho programového vybavení. Rozsah práce upřesněte po dohodě s vedoucím práce.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Hana Kubátová, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 19. prosince 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Sledování telemetrických dat robotů pomocí holografických brýlí

Pavel Chytrý

Katedra číslicového návrhu
Vedoucí práce: Ing. Miroslav Skrbek, PhD.

15. května 2019

Poděkování

Tímto bych chtěl poděkovat mému vedoucímu panu Skrbkovi za vedení a podnětné připomínky, které napomohly k dokončení této bakalářské práce. Také děkuji Petrovi Guľovi za asistenci s rozchozením testovacích robotů a mé rodině, za kontrolu a podporu počas psaní práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 15. května 2019 .....

Odkaz na tuto práci

Chytrý, Pavel. *Sledování telemetrických dat robotů pomocí holografických brýlí.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Tato práce se zaměřuje na praktické využití holografických brýlí HoloLens první generace a vývoj aplikace schopné zobrazit telemetrická data (např. výstupy z čidel, PWM signály do motorů a serv, nebo i jakékoliv proměnné kódu) z připojených robotů tak, že je možné zároveň sledovat roboty a i jejich data. HoloLens k funkci nepotřebují žádné kabelové spojení, čímž zaručují absolutní volnost pohybu uživatelů.

Aplikace je na doporučení firmy Microsoft Corporation vyvíjena v herním enginu Unity 3D, dále exportována do vývojového prostředí Visual Studio a následně nahrána jako UWP C# aplikace do HoloLens. Komunikace mezi HoloLens a roboty (auta Sunfounder Smart Video Car a Hexapodu) je zprostředkována pomocí Wi-Fi spojení s možností budoucího rozšíření o technologii Bluetooth. Práce klade minimální požadavky na roboty, u kterých vyžaduje jen podporu TCP/IP socketů a serializace JSON.

Přijatá data se zobrazují na plavoucích panelech sledujících vytisknutý marker, který je fyzicky připevněný k cílenému robotovi. Panely se zobrazují ve dvou konfiguracích. Jeden panel uprostřed obsahující až tři datové záznamy, nebo dva panely symetricky rozložené okolo markeru, kde je každý schopnen zobrazit až pět záznamů.

Klíčová slova HoloLens, Rozšířená realita, Robot, Vizualizace dat, Unity 3D, UWP C#, Raspberry Pi

Abstract

This work aims to develop a practical application for holographic smartglasses HoloLens. Final program is going to visualize telemetric data, such as sensory data, PWM signals, code variables etc., from connected robots in a way that user can see the physical robot and its data simultaneously. HoloLens are fully untethered device, thus not impeding the ability to move of its users.

Application is on recommendation from Microsoft developed in game engine Unity 3D, then exported into Visual Studio and lastly installed as UWP C# onto HoloLens. Communication is based on Wi-Fi technology with possible extension to Bluetooth in subsequent development. Connected robots (Sunfounder Smart Video Car and Hexapod in demo) are not constrained in their programming language as long as they support TCP/IP sockets and JSON serialization.

Aquired data is presented on floating panels tracking real-world marker placed on targeted robot coming in two configurations. Default configuration shows up to three entries and the second up to five on both side panels.

Keywords HoloLens, Augmented reality, Robot, Data vizualization, Unity 3D, UWP C#, Raspberry Pi

Obsah

Seznam obrázků

Úvod

Technologie rozšířené reality je známá již několik desítek let s primárním vývojem v armádním sektoru. Avšak za posledních pár let se společně s virtuální techologií dostává mezi běžné spotřebitele. Rozšířená realita na rozdíl od virtuální nepřekrývá zrakový vjem uživatele, nýbrž jej rozšiřuje za použití hologramů. Tyto hologramy mohou být čistě vizuální (např. prezentace nového modelu auta, rozmístění nábytku v místnosti, ...), nebo spíše informačního charakteru (např. vizualizace zdrojů elektromagnetického záření, překrytí tepelné mapy pro snažší orientaci v hořících budovách a v neposlední řadě vypsání dat).

Díky velmi rychlému vývoji v oblasti mobilních technologií se na trh dostává stále více zařízení podporujících rozšířenou realitu. Jedním z těchto zařízení je i Microsoft HoloLens. HoloLens patří mezi první kompletně soběstačná AR (augmented reality) zařízení, tj. k chodu nepotřebují žádné externí senzory, kabelové připojení ani napájení, veškeré potřebné senzory a výpočetní výkon uživatel nosí přímo na hlavě.

Možnost zobrazit informační data a zároveň nepřekrýt realitu by mohla značně ulehčit život studentům a vývojářům, kteří vyvíjejí programy pro různé roboty. Celkem často se stává, že student/vývojář potřebuje sledovat a hlídat robota a nemá možnost zjistit, jaká data robot dostává z jeho senzorů, nebo který kód v daný moment běží. Tato práce se snaží alespoň z části tento problém vyřešit.

Kapitola 1

Cíl práce

Cílem této práce je vytvoření programového vybavení pro AR brýle HoloLens, které má za úkol přenášet data z robotů do brýlí a následně dané údaje vhodně zobrazit. Uživatel si bude moci vybrat jaká data a kolik jich chce zobrazit. Data se budou zobrazovat na plavoucích panelech lokalizovaných blízko skutečných robotů.

Dílčí cíle tedy jsou:

- Seznámení se s vývojovým prostředím pro HoloLens
- Nalezení vhodných způsobů identifikace robotů
- Návrh struktury komunikace mezi HoloLens a roboty
- Vytvoření grafických prvků pro vizualizaci dat
- Implementace aplikace pro vizualizaci dat z robotů

Kapitola 2

Seznámení s Hardwarem

V této části představím hardware, který byl použit k vývoji a testování funkčnosti výsledné aplikace. Nejprve se zblízka podíváme na HoloLens, které jsou naším primárním zaměřením a dále se zaměříme na roboty použité pro předvedení aplikace.

2.1 HoloLens

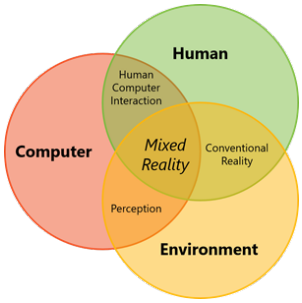
Obrázek 2.1: Microsoft HoloLens [1]

HoloLens jsou brýle pro rozšířenou a virtuální realitu vyvíjené společností Microsoft Corporation. První verze pro vývojáře byla vydána v roce 2016 s poměrně vysokou cenou $3 000. Druhá generace HoloLens by měla být dostupná koncem roku 2019 [5].

Brýle běží na platformě Windows Mixed Reality, která je součástí systému Windows 10. I když je procesor v HoloLens 64bitový, samotný systém běží jen na 32 bitech. Tato platforma nám umožňuje spustit většinu UWP C# aplikací, které můžeme získat na Windows Store, nebo vlastním vývojem [6].

2.1.1 Mixed Reality

Na tomto obrázku 2.1 si můžeme povšimnout zásadního rozdílu oproti čistě VR zařízením: displej HoloLens je průhledný a uživatel tedy může pozorovat reálný svět smíšený s virtuálním. HoloLens tedy kombinuje výstupy ze tří zdrojů: lidské příkazy (např. klávesnice, myš, dotek, hlas, ...), rozpoznávání prostředí (pozice uživale ve světě, tvary objektů v okolí, detekce světelných zdrojů, ...) a samotný počítačový kód. Tuto kombinaci Microsoft označuje jako Mixed Reality [2] (obr. 2.1).



Obrázek 2.2: Mixed Reality [2]

K splnutí reality a virtuálního světa je zapotřebí přesně detekovat pohyb uživatele. Pohyb nestačí snímat jen v souřadnicích x/y/z (dopředu, do strany, nahoru a dolů), ale je nutné detekovat i rotaci kolem všech tří os. Tuto volnost pohybu označujeme jako 6DoF (6 degrees of freedom). Všechna zařízení běžící na Windows Mixed Reality podporují 6DoF bez použití externích senzorů [7].

2.1.2 Hardwarové vybavení HoloLens

- CPU: Intel Atom 1.04 GHz
- HPU: HoloLens Graphics
- Inerciální měřící jednotka
- Hloubková kamera

- 4 kamery rozpoznávající okolí
- Kamera s vysokým rozlišením
- Wi-Fi 802.11ac

Procesor v HoloLens je na dnešní poměry celkem slabý. Proto k zpracování vstupů ze senzorů okolí HoloLens využívá speciálního HPU čipu, čímž přesune veškerou výpočetní zátěž z procesoru a nechá ho nevytížený pro hladký běh aplikací.

HPU zpracovává vstup ze čtyř bočních kamer a inerciální měřící jednotky pro lokalizaci v prostoru, rozpoznávání hlasu pomocí čtyř mikrofonů a nakonec rozpoznávání gest a tvarů pomocí kamery schopné rozlišit hloubku obrazu.

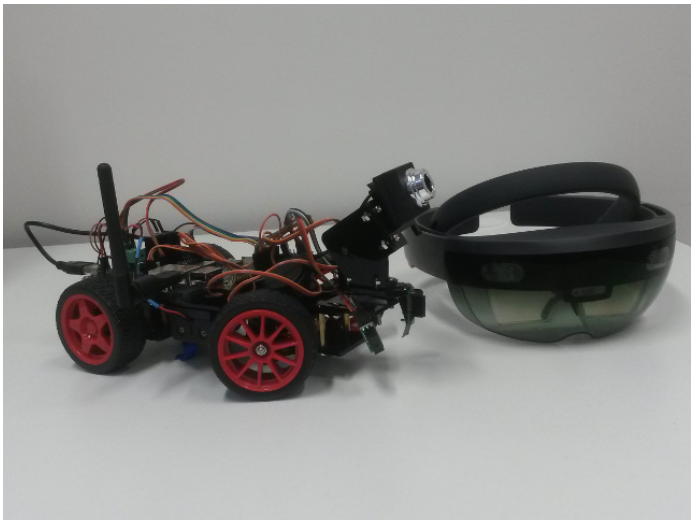
Jelikož jsou HoloLens jen pasivně chlazené, a procesor běží na velmi nízké frekvenci, tak při přehřívání násilně ukončí všechny běžící aplikace. Dále je značně omezené zorné pole holografického displeje (přibližně 30° na šířku a 17° na výšku). Na druhou stranu je to přijatelný kompromis, protože veškerý výpočetní hardware je přímo zabudovaný do brýlí a není potřeba výkonného počítače jako u ostatních VR zařízení.

2.2 Raspberry Pi

Raspberry Pi (RPi) je miniaturní počítač vyvinutý firmou Raspberry Pi Foundation. V této práci byla použitá verze 3 Model B, která je vybavená 1.2 GHz procesorem Cortex-A53, 1 GB operační paměti, dále pro nás důležitý Wi-Fi konektor a 40 GPIO pinů. Na obou RPi obsažených v testovaných robotech byl nainstalovaný systém raspbian podporující programování v jazyce Python2.7.

2.2.1 Sunfounder Smart Video Car Kit

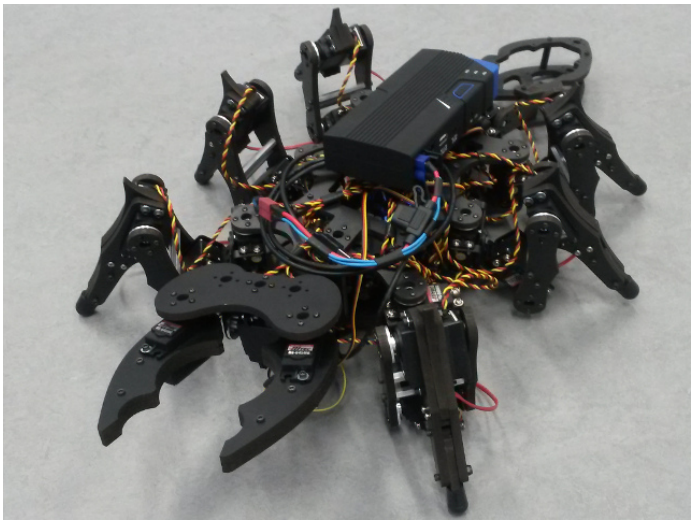
Smart Video Car Kit (viz. obr. 2.3)je stavebnice vydaná firmou Sunfounder založená na použití RPi jako řídící jednotky. Kromě RPi dále obsahuje servo controller PCA9685. RPi a controller nám umožní číst PWM data pro tři obsažená serva (dvě nastavující kameru a jedno pro řízení auta), PWM pro rychlost motorů a z GPIO portů získáme informace o směru jízdy (dopředu/dozadu). Webkamera přimontovaná na autě nebyla použitá, ale mohla by sloužit jako další podnět k rozšíření této práce do budoucna pro implementaci přenosu fotek a videa přímo do HoloLens.



Obrázek 2.3: Sunfounder Smart Video Car [1]

2.2.2 Hexapod

Hexapod (viz. obr. 2.4) je šestinohý robot od firmy Lynxmotion. Chůzi a API rozhraní pro čtení dat ze senzorů zpracovala v bakalářské práci Jitka Seménková. Díky tomu můžeme z Hexapoda získat informace o doteku na jeho šesti čidlech umístěných na koncích nohou.



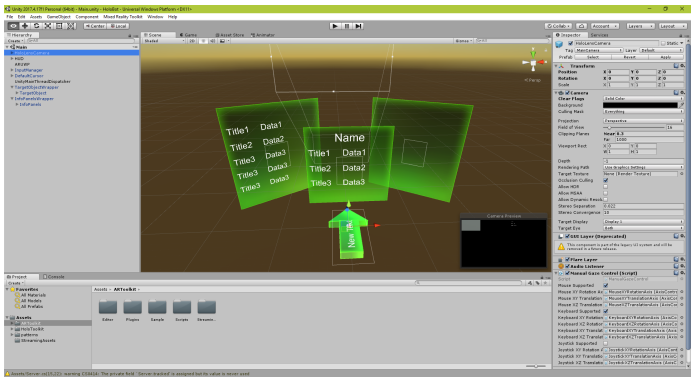
Obrázek 2.4: Hexapod [1]

Kapitola 3

Vývojové prostředí

V této sekci budou popsány technologie použité k programování HoloLens. Pro HoloLens můžeme programovat a následně nainstalovat vlastní UWP aplikace, pro které je potřeba povolit developerský režim na HoloLens i na zdrojovém počítači. Počítač také musí běžet na Windows 10. Pro vytváření grafiky potřebujeme další vývojové prostředí. U HoloLens máme možnost použít buď existující grafické enginy Unity 3D a Unreal Engine, a nebo implementovat vlastní engine založený na DirectX 11 a C++. Microsoft doporučuje Unity 3D, podporuje ho s různými návody a knihovnami, dále také existuje několik desítek projektů pro HoloLens psaných v tomto programu a jelikož jeho fóra jsou celkem rozsáhlá, bylo rozhodnutí pro Unity 3D vcelku jednoznačné.

3.1 Unity 3D



Obrázek 3.1: Unity GUI

Unity 3D je engine sloužící především k vývoji počítačových her. Zaměřený je primárně na 3D aplikace, nicméně v něm můžeme tvořit i 2D aplikace.

Díky podpoře desítek různých platforem a programů pro 3D grafiku patří k nejpoužívanějším enginům dneška. Uživatelé mají dále přístup do databáze placených i neplacených 3D modelů, animací, skriptů, atd. Podporované programovací jazyky jsou C# a JavaScript.

3.1.1 Mono

Unity pro kompilaci kódu nepoužívá standartní C# framework .NET, ale jeho open-source verzi Mono. Mono je na rozdíl od .NET multiplatformní nástroj a můžeme ho propojit i s ostatnímy jazyky například Java a Python [8]. HoloLens nicméně toto prostředí nepodporuje a nám nezbývá nic jiného než Mono použít jen pro testování v Unity playeru. Jelikož to samé platí i obráceně, tak kód obsahující .NET framework musí být při skriptování v Unity obalen bloky #ifdef.

3.1.2 Základní bloky Unity

Jelikož je program založený na Unity, tak se musíme seznámit s hlavnímy stavebnímy bloky: GameObject, Component a MonoBehaviour.

Každý objekt, který přidáme do Unity, se stane GameObjectem. GameObject sám o sobě funguje jako složka obsahující Componenty a nebo další GameObjecty, ale sama o sobě nemá žádnou fuknčnost. Každý vytvořený GameObject automaticky obsahuje komponentu Transform udávájící jeho pozici, rotaci a škálu. Ve výchozím nastavení se Transform váže relativně vůči nadřazenému GameObjectu.

Component je funkční prvek GameObjectu, který nám umožní přiřazený GameObject animovat, pouštět skripty, přiřadit HitBoxy apod.

MonoBehaviour je třída, ze které dědí většina použitých skriptů. Tyto skripty přiřadíme k nějakému GameObjectu jako Componentu a následně v nich můžeme interagovat s ostatnímy GameObjecty a Componentami ve hře. Pro náš vývoj nás nejvíce zajímají dvě zděděné metody: Start() a Update(). Start se volá jednou při vytváření GameObjectu a Update při každém snímku hry.

Aby naše aplikace běžela plynule, tak nechceme volat výpočetně náročné metody v Update, a proto by bylo ideální tyto metody pustit na vedlejších vláknech. Z důvodu programování pro HoloLens ale nemůžeme použít vícevláknové programování podporované v Unity a musíme se spokojit s třídou System.Threading.Task.

Na obrázku 3.1 vidíme GUI Unity. Vlevo je seznam všech GameObjectů v právě otevřené scéně, uprostřed pohled na hru, vpravo takzvaný inspector, ve kterém můžeme přidávat Componenty a měnit jejich vlastnosti, a dole prohlížeč se všemi assety aktuálního projektu.

3.1.3 Unity XR

Aplikace v Unity mohou být specifikovány pro takzvaný XR režim. XR je pojem zahrnující VR (virtual reality), AR (augmented reality) a MR (mixed reality) aplikace. Po zapnutí tohoto nastavení se změní několik důležitých položek. Pozice kamery se změní na podporu HMD (head-mounted display) zařízení. Kamera také začne renderovat dvakrát pro, každé oko zvlášť, a nastaví se pohledová a projekční matice v závislosti na zorném poli displeje, rotaci hlavy a pohybu v místnosti [9].

3.1.4 MRTK

MixedRealityToolKit [10] je open-source Unity knihovna vydaná Microsoftem usnadňující vývoj Mixed Reality aplikací. Projekt se snaží o snížení vstupních bariér programování a velká část komponent se dá použít jednoduchým drag-and-drop způsobem do již existujícího projektu. Ke knihovně existuje rozsáhlá dokumentace a řada návodů názorně předvádějící funkčnost některých prvků. Mezi významné části patří skripty pro rozpoznávání gest, kurzory pro vizualizaci pohledu a různé grafické prvky včetně materiálů.

3.2 UWP C#

UWP je open-source API od firmy Microsoft. Jeden z jeho hlavních cílů je jednoduchá přenositelnost aplikací mezi všemi zařízeními běžícími na Windows 10. UWP nabízí škálovatelné uživatelské rozhraní, které bude vypadat dobře na většině displejů. Uživatelé mohou snadno hledat a instalovat nové UWP aplikace přes zabudovaný Microsoft Store. V této práci budeme muset zdrojový Unity projekt zkompilovat do UWP rozhraní a dále upravovat ve Visual Studiu, odkud můžeme nahrát výslednou aplikaci do HoloLens. Tato dvojitá kompilace je celkově zdlouhavá a programátor HoloLens by se jí měl snažit co nejvíce vyhnout. Po nahrání je aplikace přímo nainstalovaná na HoloLens a pro opětovné spuštění (bez změny zdrojového kódu) není potřeba znova kompilovat.

Kapitola 4

Analýza

V této části se budeme zabývat rozborem již existujících aplikací, knihoven nebo výzkumných prací, které jsou tématicky podobné a nebo mohou být prospěšné pro dokončení této práce.

4.1 Existující obdobná řešení

HoloLens byly představeny v roce 2015 s krátkou demonstrací jejich schopností, které můžeme shlédnout na následujícím videu[1]. K demu bohužel nebyl vydán žádný kód ani dokumentace a následující rešerše se zabývala nalezením aplikací a částí kódu, které by šli použít k vytvoření podobné aplikace.

Od doby prezentace zmíněného dema vzniklo velké množství projektů zaměřených na komunikaci s roboty, primárně na jejich ovládání. Prvním z nich je EV3ControllAR [11]. Tento projekt propojil LEGO Mindstorms EV3 s HoloLens přes knihovnu MonoBrick. Uživatel mohl posouvat robotické rameno pomocí gest a hlasových příkazů.

Marcus Rieker napsal v roce 2018 práci s velmi podobným zaměřením [12]. Jeho práce taktéž používá Rasberry Pi pro řízení koncových robotů ale běžícím na systému Windows 10 IoT, který umožnil nainstalovat UWP aplikaci i na RPi, a tudíž použít stejné API jako na HoloLens. Komunikace probíhala přes TCP/IP sockety, kde RPi mělo roli serveru.

Co se týče grafické stránky, Periodic Table of Elements [13] byl shledán jako ideální vzor pro zobrazení dat uživateli. V této aplikaci uživatel klikne na krabičku s chemickým prvkem a zobrazí se mu animace jádra atomu společně s dvěmi panely obsahující vlastnosti atomu a krátký popis.

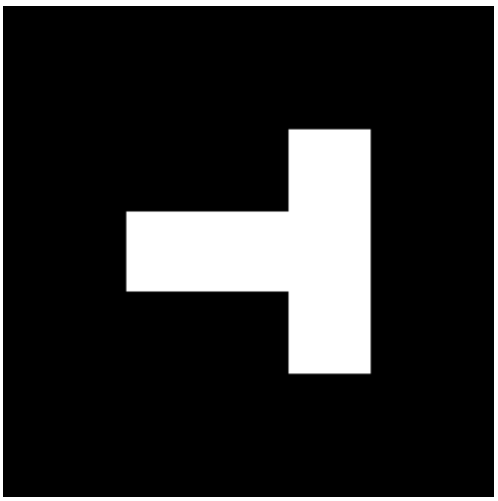
[1] https://www.youtube.com/watch?v=R_YixHJsLU4

4.2 Lokalizace objektů

V prezentačním demu bylo ukázáno, že HoloLens jsou schopné detekovat pozici různých objektů. Tato vlastnost se hodí, chceme-li zobrazit data blízko skutečných robotů bez nutnosti ručního mapování virtuálního světa na reálný (např. přetažením kalibračních bodů tak, že se oba pohledy překryjí). V následující části tedy popíši nalezené možnosti jak lze lokalizaci udělat, ať už s pomocí detekce reálných objektů a tvarů, anebo použitím vytisknutých značek (markerů) a obrázků.

4.2.1 HoloLensARToolKit

HoloLensARToolKit [14] je nadstavbou nad knihovnou ARToolKit [3], která byla přepsána tak, aby běžela na HoloLens. Knihovna je založená na detekci speciálních 2D markerů viz. obr. 4.1, které se vyznačují černým čtvercovým rámečkem vyplněným černobílým obrázkem. Implementace pro HoloLens podporuje jen markery obsahující tabulku 3x3 čtverců viz. obr. 4.1.



Obrázek 4.1: Marker pro HoloLensARToolKit [3]

Toto řešení běží na HoloLens bez jakékoliv třetí strany nebo nutnosti připojení k internetu, veškerá detekce probíhá přímo na HoloLens. Záporná strana této implementace je omezenost markerů, které jsme schopni detekovat, potřeba markery vytisknout a fyzicky přilepit na robota, a zhoršená detekce, pokuď je marker pod úhlem vůči HoloLens.

4.2.2 Vuforia

Vuforia [15] na rozdíl od HoloLensARToolKitu nabízí podstatně více možností. Kromě detekci 2D markerů (VuMarks viz. obr. 4.2) je schopná detekovat

i normální obrázky, dále 3D modely objektů vytvořených například v programu SCAD a v neposlední řadě i rozpoznávání objektů s pomocí umělé inteligence běžící na serverech Vuforie.

Obrázek 4.2: Vzorové markery VuMarks [4]

Ke všem možnostem je potřeba účet na serverech Vuforie, který je v základní verzi zadarmo, ale je omezený v maximálním počtu markerů nebo obrázků (100) a počtem detekcí za pomocí cloudu (1 000krát za měsíc) [16]. Tato omezení nijak neovlivní pár studentů, pro větší skupiny by již byla potřeba placená licence.

4.2.3 Microsoft Azure

Microsoft Azure je cloudová služba společnosti Microsoft, která lze využít i pro HoloLens. Azure přináší mnohem více než jen detekci objektů, například rozpoznávání tváří a překlad jazyků [17]. Všechny tyto služby běží na cloudových serverech a je k nim zapotřebí účet. Neplacená verze má podobná omezení jako Vuforia, i když značně vyšší počet detekcí za měsíc (až 10 000krát [18]).

4.2.4 Výběr řešení

K lokalizaci robotů jsem se rozhodl použít projekt HoloLensARToolKit. Hlavní body, které ovlivnily, můj výběr byly:

- Není nutné mít založený jakýkoliv účet a případně řešit i poplatky s ním spojené
- K funkčnosti není zapotřebí připojení na cloudovou službu

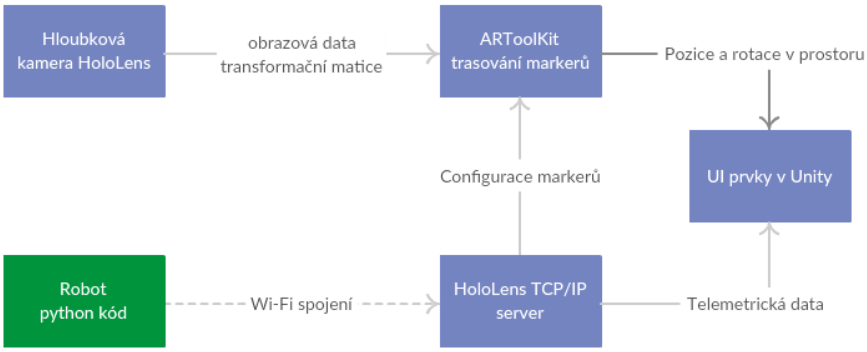
- Zdrojové kódy jsou volně dostupné
- Knihovna obsahuje okolo třiceti markerů, které budou naprosto dostačovat a pro inicializaci v programu stačí znát jen jejich id, typ a fyzickou velikost

Kapitola 5

Návrh

V této sekci se budeme zabývat návrhem struktury programu. Program se skládá ze dvou nezávislých částí. Serverovou část běžící na HoloLens, která má dále na starosti i samotnou vizualizaci přijatých dat a trasování markerů, a kód běžící na robotech, který má za úkol číst data ze senzorů a odpovídat na příkazy HoloLens.

5.1 Blokové schéma programu HoloLens



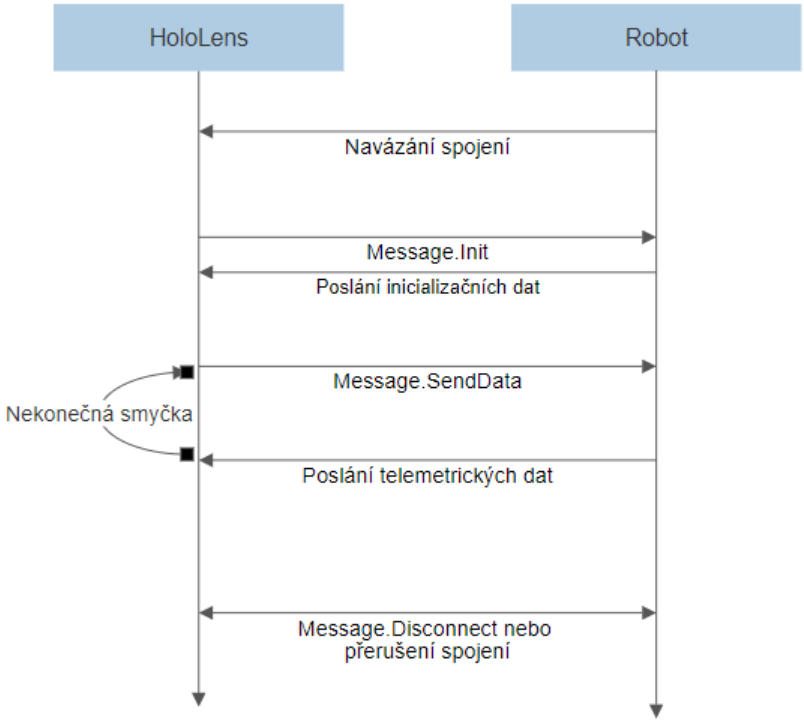
Obrázek 5.1: Blokové schéma programu

Na obrázku 5.1 můžeme vidět blokové schéma celého návrhu. První důležitou součástí je server. K implementaci serveru máme možnost použít TCP/IP, nebo UDP protokol. K realizaci jsem se rozhodl použít TCP/IP protokol z důvodu zajištění spolehlivosti spojení (výhodné protože používáme Wi-Fi) a zachování pořadí packetů. Taktéž díky celkově malému množství posílaných dat můžeme protokol UDP kompletně zamítnout. Server tedy bude čekat na spojení od robotů. Po připojení robot pošle inicializační data, zejména vlast-

nosti zvolených markerů. Tato data zpracuje ARToolKit a za použití vestavěné hloubkové kamery začne lokalizovat zvolený marker. Výsledky trasování a přijatá data budou posílány do Unity GameObjectu, u kterého nastavíme jeho pozici a text ve vytvořených panelech. Můžeme si povšimnout, že ARToolKit a třída komunikující s robotem mění vlastnosti u jednoho GameObjectu zároveň. Toto ovšem nezpůsobí žádné problémy, změněné vlastnosti jsou vzájemně odlišné a nehrozí přepisování sdílených dat.

5.2 Návrh komunikace

Obrázek 5.2 znázorňuje schéma komunikace HoloLens s roboty.

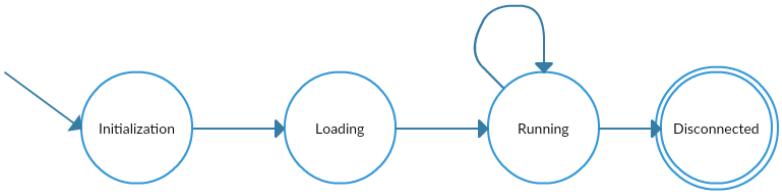


Obrázek 5.2: Komunikační diagram

Kromě navázání spojení budou HoloLens řídit veškerou komunikaci a robot bude jen odpovídat na přijaté dotazy. Tato komunikace nebude probíhat ve třídě serveru, nýbrž každý připojený robot dostane svoji instanci komunikační třídy. Všechny zprávy ze serveru budou definované ve zvláštní třídě. Ihned po vytvoření spojení HoloLens pošle dotaz na specifikaci markeru a ostatních konfiguračních dat. Pro specifikaci markeru použijeme serializovaný objekt. Poté přejdeme do hlavní nekonečné smyčky. Podle nastavené obnovovací frekvence

budou HoloLens posílat dotazy na data, která robot pošle v jiném serializovaném objektu. Tento objekt bude mít předepsanou očekávanou strukturu a bude na robotovi tuto strukturu dodržet. K ukončení této nekonečné smyčky může dojít několika způsoby:

- Uživatel ukončí spojení za použití tlačítka nebo hlasového povelu
- Robot pošle špatná data
- Robot ukončí spojení a nebo dojde k výpadku (ukončení a výpadek spojení vypadá z pohledu HoloLens stejně)



Obrázek 5.3: Stavový diagram

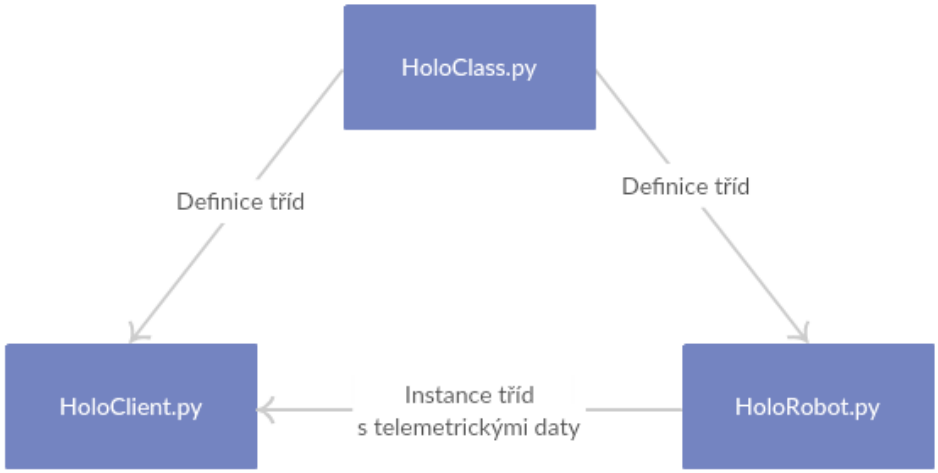
Zprávy, které HoloLens posílá, se řídí jednoduchým stavovým diagramem 5.3. Každé spojení je řízeno nezávisle na ostatních. Ihned po připojení přejdeme to stavu Initialization. Zde proběhne konfigurace vnitřních proměnných programu. Po přijetí veškerých konfiguračních dat program přejde do stavu Loading. V tomto stavu proběhne registrace nového markeru do controlleru ARToolKitu, taktéž vytvoříme nové instance GameObjectů. V této části bude program čekat neblokujícím voláním na dokončení inicializace GameObjectů a nebude posílat ani přijímat žádná data od robotů. Následný stav Running je nejdůležitější z pohledu uživatele. Veškerý pohyb vizualizačních panelů a vypisování dat bude probíhat zde. Poslední stav Disconnected slouží k vyčištění použité paměti a zrušení socketu.

5.3 Klientská strana robotů

Návrh komunikace se snažil, aby klientská strana robotů nebyla omezena programovacím jazykem. Po robotech je požadováno vytvoření TCP/IP socketů a navázání spojení na server. Následně bude robot poslouchat v čekací smyčce na příkazy od HoloLens. Po dekódování příkazu robot ihned zareaguje. Reakcí může být poslání určitých dat, serializovaných objektů (které odpovídají předepsanému formátu), nebo odpojení od serveru. Robot bude mít možnost kdykoliv přerušit spojení (hlavní důsledek je, že náhodný pád ze strany robota neshodí server).

K předvedení aplikace máme k dispozici dva roboty, oba schopné spustit kód v pythonu. Proto by bylo vhodné vytvořit rozhraní, kde pro každého

robota změníme implementaci pár metod a zbytek kódu bude sdílený. Návrh takového rozhraní vidíme na obrázku 5.4.



Obrázek 5.4: Schéma python souborů

Soubor HoloClass.py bude obsahovat definice tříd a zpráv ve stejném formátu, jako jejich vzor na HoloLens. Také bude obsahovat rozhraní pro práci s těmito třídami, zejména jejich konstruktory, metody na přidání a aktualizaci dat a metody pro jejich serializaci. Kód specifický pro každého robota bude uložený v souboru HoloRobot.py, který poskytne rozhraní pro sdílení dat z instancí tříd definovaných v HoloClass.py. Na oba soubory se bude odkazovat HoloClient.py, který bude mít na starosti řízení síťové komunikace a přenos dat.

Kapitola 6

Implementace

Implementace celé aplikace vspočívala ve střídavém programování v Unity a Visual Studiu. Z tohoto důvodu rozdělím popis implementace na několik částí. V první části se podíváme na hierarchickou strukturu GameObjectů v Unity, jejich parametry, vzájemné vazby a k nim přiřazené komponenty, a poté se podrobněji podíváme na důležité C# třídy, které už nutně nesouvisí s Unity. Na závěr krátce popíši program běžící na robotech.

6.1 Unity GameObjecty

Obrázek 6.1: Hierarchická struktura GameObjectů

Nejvyšším objektem Unity jsou takzvané scény. Scény v Unity můžeme chápat jako samostatné levely. Vzhledem k rozsahu této práce stačila jediná scéna Main.unity umístěná v adresáři Assets/Scenes. Po jejím načtení se vlevo zobrazí seznam všech použitých GameObjectů zobrazených na obrázku 6.1. Tyto GameObjecty se aktivují při spuštění aplikace a budou běžet až do jejího ukončení.

HoloLensCamera

Každá scéna v Unity obsahuje minimálně jeden objekt kamery. Kamera slouží jako pozorovací okno do naší hry. Většina kamer je umístěna na hlavě aktivního hráče, nicméně je můžeme použít i pro pohledy z ptačí perspektivy a v některých případech byly použíté jako imitace zrcadel. HoloLensCamera je takzvaný prefab (zabalený GameObject do souboru obsahující všechny jeho komponenty) obsažený v knihovně MRTK a je již přednastavený pro použití s HoloLens. V inspektoru byla změněná jediná vlastnost a to 'Clipping Planes Near' distance. Z původních 0.85 (doporučených Microsoftem) jsem ji snížil na 0.3. Toto číslo přibližně odpovídá vzdálenosti v metrech, kde hologramy bližší tomuto číslu se přestanou vykreslovat. Při testování markerů držených v ruce bylo 85 cm příliš daleko. Kamera je propojená se zabudovanou lokalizací HoloLens a pohyb uživatele se projeví změnou hodnot 'Transform' – pozice v x/y/z souřadnicích odpovídající vzdálenosti v metrech (Position) a rotací ve stupních okolo os x/y/z (Rotation).

HUD

HUD (heads-up display) je založený na komponentně Canvas. Canvas a dva přiložené skripty Canvas Scaler a Graphic Raycaster slouží k vykreslování UI GameObjectů a každý UI GameObject musí být podřízený nějakému Canvasu. HUD je umístěný v konstantní vzdálenosti 100 metrů od HoloLensCamery ve směru jejího pohledu. Této funkčnosti bylo dosaženo nastavením vlastnosti 'Render Mode' na 'Screen Space – Camera', přetažením GameObjectu kamery do řádku 'Camera' a nastavením 'Plane Distance' na 100. HUD obsahuje tři podřízené UI texty vypisující snímkové frekvence trasování, renderování a kamery. StatusText vypisuje počet připojených robotů a dá se použít k výpisu proměnných při ladění programu. Poslední složka WebCam při zapnutí zobrazuje zpětnou vazbu hloubkové kamery HoloLens (tedy to, co vidí HoloLens a ne GameObject HoloLensCamera).

ARUWP

ARUWP obsahuje hlavní výpočetní logiku celého programu. Implementuje dva skripty z knihovny HoloLensARToolKit a to ARUWPController a ARUWPVideo. Oproti originální verzi ARUWPControlleru nenastavuji v řádku 'Target Object' referenci na aktivní GameObject v scéně, nýbrž odkaz na prefab TargetObjectWrapper. Druhá změna je přidání vlastnosti 'Target Panel', kam přiřadím referenci na prefab InfoPanelsWrapper. Oba prefaby budou

popsány níže. U skriptu ARUWPVideo možnost 'Enable Video Preview' povolí vykreslování zpětné vazby kamery. Přesnější popis ostatních parametrů najdeme na stránce HoloLensARToolKitu[2]. Třetí skript Server jak napovídá název obsahuje implementaci TCP/IP serveru. V inspektoru můžeme změnit jeho IP adresu a port. Adresa se musí shodovat s IP adresou Wi-Fi adaptéru. HoloLens při použití na síti FIT-1048 mají statickou adresu 10.10.48.192. Port je volitelný, ale funkčnost byla testována jen na portu 9559.

InputManager

Další z prefabů získaných z knihovny MRTK. InputManager přináší schopnost reagovat na gesta a také stabilizuje pohledový kurzor, který s ním provážeme přetažením použitého GameObjectu DefaultCursor do řádku 'Cursor' v komponentě 'Simple Single Pointer Selector'.

DefaultCursor

Poslední z použitých prefabů MRTK. DefaultCursor imituje výchozí systémový kurzor na HoloLens reagující na objekty, které mají přiřazený vlastní collider. Kurzor je animovaný a mění se při používání gesta 'Tap' pro snažší obsluhu programu.

UnityMainThreadDispatcher

UnityMainThreadDispatcher [19] obsahuje skript o stejném názvu, který lze díky tomu odkudkoliv použít. Jeho hlavní účel je plánování volání Unity API metod, které nejdou volat z vedlejších vláken.

TargetObjectWrapper

Po detekci markeru dostaneme z třídy ARUWPMarker informace o jeho pozici a rotaci. Tyto údaje se přidělí GameObjectu TargetObjectWrapper. K vizualizaci pozice používám 3D model šipky označený jako Arrow a naškálovaný na 0.06 ve všech souřadnicích, což odpovídá skutečné velikosti přibližně 6 cm. Nad šipkou je umístěný UI text pod názvem IP, kde vypisuji IP adresu připojeného robota. V nadřazeném GameObjectu TargetObject se nachází komponenta Box Collider. Box Collider je nahrubo namapovaný na model šipky, na který můžeme následně klikat. Zpracování eventu kliknutí má na starosti skript TargetButton získaný z aplikace Periodic Table of Elements. TargetButton události přeposílá skriptu TargetObject, který reaguje změnou materiálu šipky pro zvýraznění a eventy dále komunikuje se skriptem PanelController, který je součástí prefabu InfoPanelsWrapper a popíši ho níže.

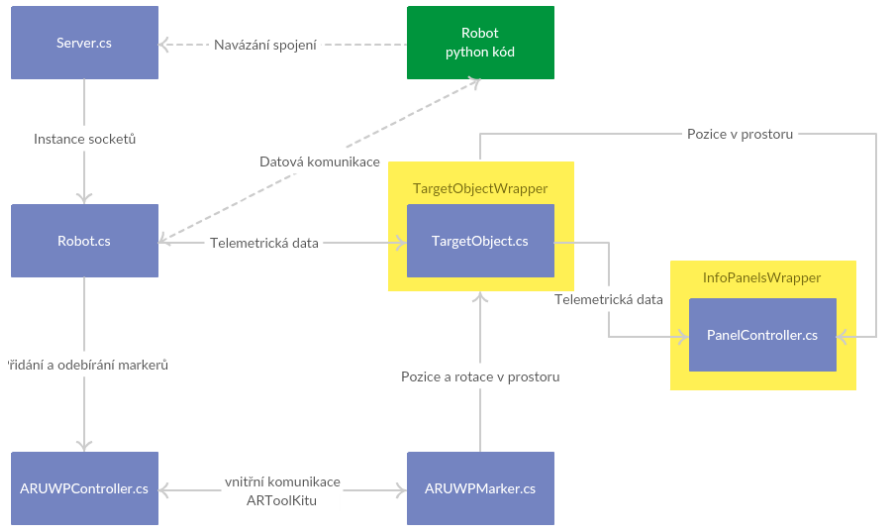
InfoPanelsWrapper

Původně součást TargetObjectu, InfoPanelsWrapper jsem vytvořil jako vedlejší GameObject během testování, kdy se ukázalo, že díky nepřesnostem

[2]http://longqian.me/tag/hololens-artoolkit/ (použitá verze 0.2).

v trasování markeru (zaokrouhlovací chyby, změna světelných podmínek, relativní úhel markeru vůči HoloLens atd.) všechny grafické prvky vibrovaly. Pro šipku znázorňující polohu markeru to nebyl problém, ale text na vedlejších panelech se stal téměř nečitelným. InfoPanelsWrapper tento problém řeší s metodou Vector3.SmoothDamp a plynule následuje TargetObjectWrapper. Pro lepší čitelnost textu se všechny panely otáčí směrem na kameru s pomocí metody transform.LookAt a mění velikost v závislosti na vzdálenosti uživatele od markeru, aby zabíraly konstantní úhlovou velikost. Ve výchozím stavu se zobrazí PanelMiddle jako pozadí textu a CanvasMiddle se jménem robota a třemi řádky telemetrických dat. Po kliknutí dojde k animaci, kdy panel uprostřed zmizí a jeho místo nahradí dva panely PanelRight a PanelLeft s odpovídajícími canvasy. Animaci řídí komponenta Animator s controllerem InfoPanels.controller a vlastnosti animace jsou uložené v souboru PanelInfo.anim. InfoPanelsWrapper ani TargetObjectWrapper nejsou ve hře aktivní, slouží jen k vývoji programu, a všechny změny se musí uložit do jejich prefab verze použitím tlačítka Apply.

6.2 Popis C# tříd



Obrázek 6.2: Blokové schéma datového toku mezi třídami

Implementace C# kódu zhruba odpovídá návrhu i když s malými odlišnostmi. Obrázek 6.2 se snaží trochu lépe vysvětlit závislosti mezi hlavními třídami a Unity GameObjecty. Server.cs poslouchá na specifikované ip adrese a portu a řízení každého spojení je předáno nové instanci třídy Robot.cs. Robot.cs

po přijetí konfiguračních dat zaregistruje nový ARUWPMarker.cs v třídě ARUWPController.cs. Po celou dobu aktivního spojení Robot.cs získává telemetrická data a předává je ke zpracování do třídy TargetObject.cs, která je následně přepošle do třídy PanelController.cs. Výsledky trasování (pozice a rotace markerů) z ARUWPMarker.cs nastavím jako transformační vlastnosti GameObjectu TargetObjectWrapper, které k vlastní potřebě čte PanelController.cs Pro shrnutí: každé spojení s robotem vytvoří nové instance tříd Robot.cs, ARUWPMarker.cs, TargetObject.cs v nové instanci TargetObjectWrapper.prefab a PanelController.cs v nové instanci InfoPanelsWrapper.prefab. Všechny vyjmenované instance jsou vlastní každému spojení a po ukončení jsou také smazány.

6.2.1 Specifikace datového přenosu

6.2.1.1 JSON

K zajištění správné funkčnosti aplikace bylo navrženo několik tříd, které lze serializovat pomocí JSON. Tyto třídy jsou neměnné a je na programátorovi robotů, aby tento vzor dodržel. Díky serializaci program není závislý na určitém programovacím jazyce a dokáže komunikovat se všemi jazyky, které technologii JSON podporují. Důležitou poznámkou je, že JsonUtility použitý na serverové straně rozlišuje malá a velká písmena u všech proměnných a názvů tříd. Všechny serializované třídy v C# musí obsahovat direktivu [System.Serializable] a jsou k vidění níže. Vzorový výpis JSON kódu, který HoloLens dokáží deserializovat je přiložený v souboru json.txt

```
public class MarkerInfo {
   public int MarkerId = -1;
   public int MarkerWidth = -1;
}

public class RobotDataEntry {
   public string Name;
   public string Data;
   public string Unit;
}

public class RobotInfo {
   public string Name;
   public RobotDataEntry[] MiddlePanel;
   public RobotDataEntry[] LeftPanel;
   public RobotDataEntry[] RightPanel;
}
```

6.2.1.2 Message

Třída Message specifikuje tvar zpráv, které budou HoloLens vysílat. Roboti mohou tuto třídu implementovat napřímo, nebo porovnávat definované stringy s přijatými daty.

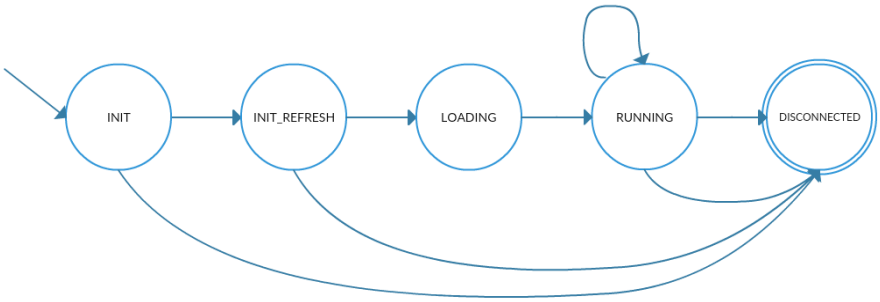
```
public static class Message {
   public const string SEND_MARKER_INFO = "SEND_MARKER_INFO";
   public const string SEND_DATA = "SEND_DATA";
   public const string DISCONNECT = "DISCONNECT";
   public const string SEND_REFRESH_RATE = "SEND_REFRESH_RATE";
}
```

6.2.1.3 Server

TCP/IP server uložený v souboru Server.cs je založený na implementaci třídy StreamSocketListener. StreamSocketListener zaregistruji na specifikovanou IP adresu a port voláním funkce listener.BindEndpointAsync. Každé nové připojení spoustí asynchronní metodu _receiver, ve které vytvořím novou instanci Robot.cs a předám jí její vlastní socket a odkaz na ARUWPController.cs. Task r.CommLoop následně řídí veškerou komunikaci mezi robotem a HoloLens. Níže vidíme vypsané důležité části kódu.

```
public class Server : MonoBehaviour {
   StreamSocketListener listener = null;
   private void Start() {
      listener = new StreamSocketListener();
      listener.ConnectionReceived += _receiver;
      listener.Control.KeepAlive = true;
      Listen();
   }
   async void Listen() {
      await listener.BindEndpointAsync(new Windows.Networking
         .HostName(hostName), serverPort);
   }
   async void _receiver(StreamSocketListener sender,
         StreamSocketListenerConnectionReceivedEventArgs args)
   {
      Robot r = new Robot(args.Socket, controller);
      await r.CommLoop();
   }
}
```

Výše zmíněný Task CommLoop v souboru Robot.cs přijímá a posílá data s pomocí tříd DataReader (napojený na socket.InputStream, instance reader) a DataWriter (napojený na socket.OutputStream, instance writer). Komunikace se řídí stavovým diagramem 6.3, který vychází z diagramu 5.3. V každém stavu kromě LOADING HoloLens pošle příkaz přes writer.WriteString a await writer.StoreAsync. Poté čeká na odpověď neblokujícím voláním await reader.LoadAsync(1024). 1024 specifikuje očekávaný počet bytů v bufferu. Nastavením vlastnosti reader.InputStreamOptions na InputStreamOptions.Partial zamezíme čekání na všech 1024 bytů a buffer přečteme pokaždé, když obsahuje alespoň jeden byte. Návratová hodnota 0 funkce LoadAsync značí přerušení spojení. Přijatá data se dekódují pomocí JsonUtility.FromJson<T>a zkontroluje se jejich validita (vyplněné hodnoty, rozsah polí).



Obrázek 6.3: Implementovaný stavový diagram

6.2.2 Implementace knihovny HoloLensARToolKit

Kód knihovny HoloLensARToolKit obsahuje pět tříd a změny byly potřeba jen u dvou: ARUWPController a ARUWPMarker. V třídě ARUWPMarker proběhla jen velmi malá změna a to přidání GameObjectu pro panely určené pro zobrazení dat. Z praktického hlediska bychom chtěli přidávat markery k trasování až když se k HoloLens připojí robot, místo inicializace při spuštění aplikace. Proto jsem do třídy ARUWPController přidal několik metod:

AddMarker se volá po připojení robota ze třídy Robot.cs poté, co nám pošle specifikace použitého markeru. Unity brání přístupu k Unity API odjinud (AddComponent a Instantiate) než z hlavního vlákna, a proto tato metoda musí být volána přes třídu UnityMainThreadDispatcher.cs. Metoda vytvoří novou komponentu ARUWPMarker s konfigurací podle vstupního parametru MarkerInfo a naklonuje dva GameObjecty trackingTarget (TargetObjectWrapper prefab) a infoPanels (InfoPanelsWrapper prefab), které k nově vzniklé komponentě přiřadí. GameObjecty vytvořené metodou Instantiate v tuto chvíli neprojdou inicializací, a proto je druhým vstupním parametrem callback metoda, která se uloží k pozdějšímu zavolání.

```
public IEnumerator AddMarker(MarkerInfo info,
                        Action<int, TargetObject> callback) {
   ARUWPMarker marker = gameObject.AddComponent<ARUWPMarker>();
   marker.type = ARUWPMarker.MarkerType.single_barcode;
   marker.singleBarcodeID = info.MarkerId;
   marker.singleWidth = info.MarkerWidth;
   marker.target = Instantiate(trackingTarget);
   marker.panels = Instantiate(infoPanels);
   unaddedMarkers.Add(new Tuple<ARUWPMarker,
                       Action<int, TargetObject>>(marker,
                                                  callback));
   yield return null;
}
```

AddMarkers se volá z metody Update pokaždé, když máme ve frontě unaddedMarkers nové instance ARUWPMarker, které ještě nebyly inicializované. Pro přiřazení ARUWPMarkerů do ARUWPControlleru potřebujeme dočasně přerušit celé trasování pomocí StopFrameReaderAsyncTask. První metoda volaná v cyklu propojí GameObjecty panels a target vytvořené metodou AddMarker mezi sebou. Druhá metoda je na pochopení složitější. Zde dojde k volání uložené callback metody. Jejím vstupem je id přiřazeného markeru a odkaz na instanci třídy TargetObject obsažené v GameObjectu target. Id získáme voláním funkce AddMarker, která současně zaregistruje marker k trasování. Po doběhnutí obnovíme běh trasování metodou StartFrameReaderAsynctask.

```
private async void AddMarkers() {
   await videoManager.StopFrameReaderAsyncTask();
   foreach (var m in unaddedMarkers) {
      m.Item1.target.GetComponentInChildren<TargetObject>()
         .SetPanels(m.Item1.panels);
      m.Item2(m.Item1.AddMarker(),
              m.Item1.target
                  .GetComponentInChildren<TargetObject>());
   }
   unaddedMarkers.Clear();
   await videoManager.StartFrameReaderAsyncTask();
}
```

Po odpojení robota z třídy Robot.cs zavoláme **RemoveMarker**. Odstranění markerů funguje podobným způsobem jako jejich registrace, jen obráceně.

Nejprve přerušíme trasování, odstraníme markery a následně naplánujeme smazání samotných GameObjectů voláním DestroyMarker přes UnityMainThreadDispatcher na hlavní vlákno.

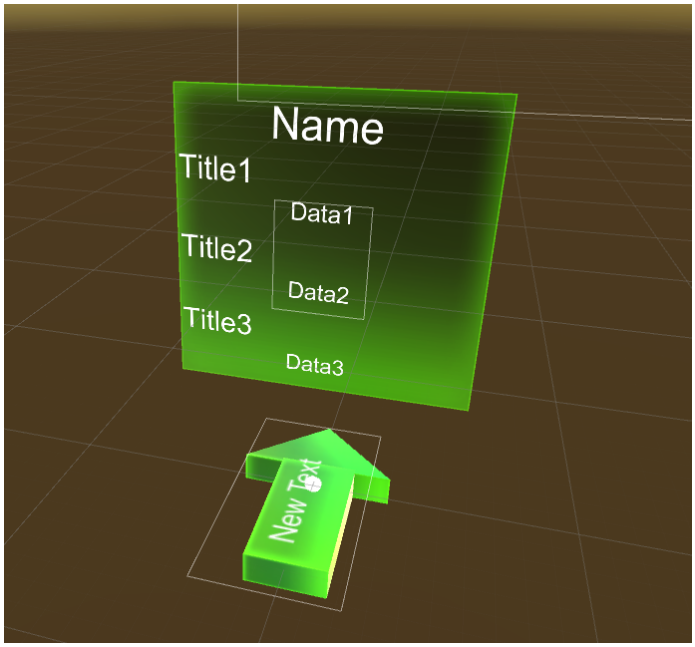
```
public async void RemoveMarker(int markerId) {
   await videoManager.StopFrameReaderAsyncTask();
   ARUWPMarker marker = markers[markerId];
   marker.RemoveMarker();
   UnityMainThreadDispatcher.Instance()
         .Enqueue(DestroyMarker(marker));
   await videoManager.StartFrameReaderAsyncTask();
}
```

Na závěr metoda **DestroyMarker** mazající nepoužívané GameObjecty.

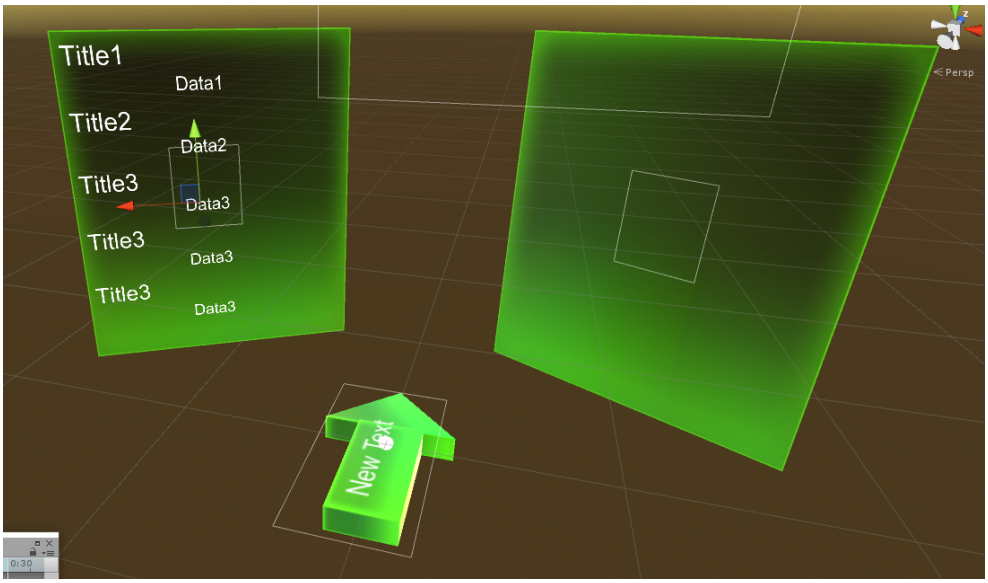
```
private IEnumerator DestroyMarker(ARUWPMarker marker) {
   Destroy(marker.target);
   Destroy(marker.panels);
   Destroy(marker);
   yield return null;
}
```

6.3 Grafické rozhraní

Návrh grafického rozhraní vycházel z aplikace Periodic Table of Elements. Kromě 3D šipky s textem IP adresy v objektu TargetObjectWrapper jsou všechny prvky umístěné v InfoPanelsWrapper. GUI tvoří tři canvasy a tři prvky Plane, umístěné do tří info panelů. Za povšimnutí stojí, že Canvas a Plane jsou na stejné úrovni v hierarchické struktuře, tutíž nesdílejí vlastnosti Transform a jejich umístění v prostoru musí být nastaveno ručně. Důvodem k jejich oddělení byla snaha pozadí škálovat podle počtu řádků přijatých dat. Tento záměr se nepodařilo uskutečnit, jelikož animování přepisovalo hodnotu Scale a v čase programování jsem neznal způsob, jak animaci zparametrizovat. Text na prostředním panelu je staticky vytvořený v Unity, avšak text na postranních panelech se generuje dynamicky při prvním přijetí dat metodou CreatePanels v PanelController. Všechny nastavené vlastnosti textu jsou ve stejné třídě uložené jako statické proměnné. Pro představu výsledného vzhledu v Unity slouží neaktivní UI prvky v CanvasLeft pojmenované Side_TitleX a Side_DataX (X je číslo v rozsahu 1 až 5). Na obrázku 6.4 vidíme výchozí vzhled a na obrázku 6.5 je vzhled po kliknutí na model šipky.



Obrázek 6.4: Prostřední UI panel



Obrázek 6.5: Postranní panely se vzorovým textem

6.4 Python klient

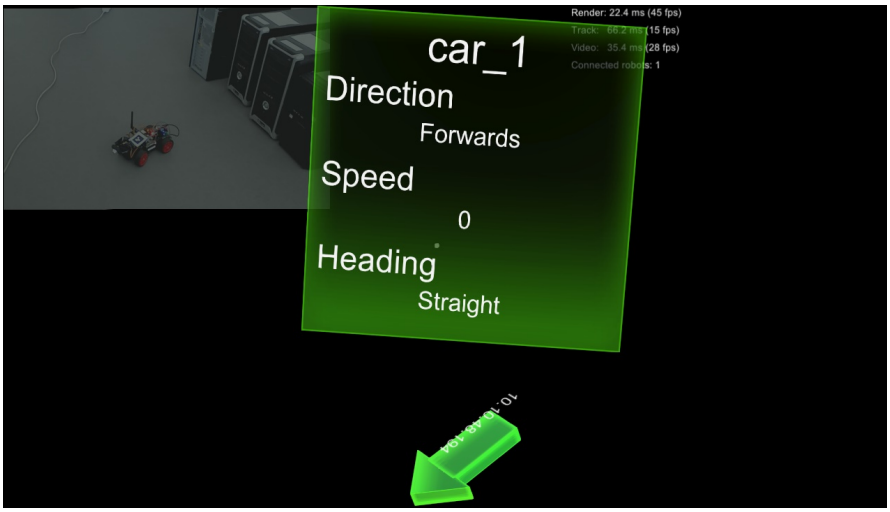
Implementace klientů v pythonu vychází z jeho návrhu.

V souboru **holoclass.py** najdeme definice tříd MarkerInfo, RobotDataEntry a RobotInfo. Konstruktor třídy MarkerInfo bere na vstupu id markeru a šírku markeru v jednotkách mm. Metoda toJSON vrátí jeho serializovanou podobu. Konstruktor RobotDataEntry má tři vstupní parametry. Name je označení datového záznamu typu string, Data jsou samotná telemetrická data typu string, anebo integer a unit je jednotka dat, také typu string, anebo integer. Metodou SetData lze aktualizovat datový záznam. Třída RobotInfo obsahuje všechny datové záznamy uspořádané do tří polí. Do každého pole můžeme přidat záznam typu RobotDataEntry pomocí metod AddToMidle, AddToLeft nebo AddToRight. Každé pole odpovídá jednomu panelu zobrazenému v HoloLens. Název robota můžeme změnit metodou SetName a metoda toJSON vrátí serializovaný tvar celé třídy. Proměnná command obsahuje seznam všech příkazů odpovídající definici v třídě Message.cs.

Holoclient.py řídí síťovou komunikaci. Nejprve proběhne vytvoření socketu příkazem socket.socket(socket.AF_INET, socket.SOCK_STREAM), poté připojení k serveru příkazem socket.connect na specifikovanou IP a port. Čtení zpráv probíhá na začátku každé iterace nekonečného cyklu přes s.recv a přijatý string porovnáváme s definovanými zprávami ze souboru holoclass.py. Podle přijatého příkazu pošleme serializovaný objekt metodou s.sendall anebo dojde k ukončení nekonečného cyklu a zrušení socketu.

Holorobot.py je předpřipravený kód pro použití programátorem robotů. Na začátku kódu specifikujeme pět parametrů: HOST (IP adresa HoloLens), PORT (použitý port na HoloLens), REFRESHRATE (doba, jak často chceme obnovovat data v milisekundách), MARKERID (ID použítého markeru), MARKERSIZE (velikost vytisknutého markeru v milimetrech, vzdálenost mezi černými okraji) a NAME (označení robota). Na vstupu metody CreateDataStruct dostaneme prázný objekt typu RobotInfo a změníme zde jeho sktrukturu použitím metod SetName, AddToMiddle, apod. Metoda OnSendData se volá pokaždé po přijmutí zprávy SEND_DATA a je vhodná pro aktualizaci dat pomocí metody SetData. Vzorová implementace holorobot.py pro Sunfounder Smart Video Car a Hexapoda je umístěná v souborech holocar.py a holohexa.py respektivě. Pro jejich použití musíme změnit řádek 'import holorobot as robot' v souboru holoclient.py na vybraný soubor.

6.5 Výsledek



Obrázek 6.6: Obrázek z pohledu HoloLens ve výsledné demo aplikaci

Na obrázku 6.6 můžeme vidět demonstraci hotové aplikace. Fotka je pořízená z HoloLens přes Device Portal, kde lze nahrávat a fotit hologramy překryté s reálným světem. Implementovaná knihovna ARToolKit ale používá vestavěnou web cameru a HoloLens zakazuje přístup ke kameře ze dvou různých procesů. Zobrazená šipka blízko kopíruje pozici markeru na autě, který vidíme v náhledu vlevo nahoře. Detekce markerů funguje stabilně na prvním testovacím markerů, u později vytisknuté sady může detekce chvíli trvat. Chyba je v samotných markerech, které jsou světlejší oproti prvnímu. Frekvence obnovování dat z robotů (jak často HoloLens pošle příkaz Message.SEND_DATA) je 4krát za sekundu, vyšší frekvence zahltily Wi-Fi síť a celkový čas odezvy se mnohonásobně zvýšil. Občas se stává, že se aplikace ukončí. Není to způsobeno pádem programu, který v debuggeru zobrazí exit code = 0, nýbrž špatnou detekcí gesta Bloom, které vypne momentálně spuštěnou aplikaci, anebo přehřátím HoloLens.

Závěr

Hlavním cílem této bakalářské práce bylo navrhnout a implementovat aplikaci pro holografické brýle HoloLens, která by umožnila sledovat telemetrická data robotů. Aplikace byla primárně vyvíjena v Unity 3D a testována s autem Sunfounder Smart Video Car a Hexapodem, kde na obou robotech běžel kód v pythonu. Programování bylo obecně zdlouhavé, hlavně z důvodu dvojité kompilace (Unity → Visual Studio → HoloLens). K obeznámení s programováním HoloLens v Unity byly použity stránky Microsoftu s tutoriály a vzorovými projekty.

Po analýze různých řešení lokalizace robotů byla vybrána knihovna ARToolKit. Lokalizace je založená na detekci vytisknutých markerů a na vypočtené pozici je zobrazený hologram s přijatými daty. Pro přenos dat byla navržená datová struktura využívající serializaci JSON. Během práce bylo upravováno grafické rozhraní na základě zpětné vazby od vedoucího, ale všechny připomínky se nepodařilo včas implementovat.

Budoucí vývoj by se mohl zaměřit na vylepšení grafického vzhledu aplikace, který je její nejslabší stránkou. Komunikace by se mohla rozšířit o více stavů a přidání chybových hláškek. Další možné rozšíření je přidání tlačítek, nebo hlasových povelů pro ovládání robotů a implementace přenosu fotek a videa přímo do HoloLens.

Literatura

[1] Skrbek, M.: HoloLens. online obrázek. Dostupné z: https://livs.fit.cvut.cz/web/doku.php?id=fit:students:start

[2] Microsoft: Mixed Reality Spectrum. online obrázek. Dostupné z: https://docs.microsoft.com/en-us/windows/mixed-reality/mixed-reality

[3] ARToolKit: ARToolKit v5.3.2. software, 2017-11-06. Dostupné z: https://github.com/artoolkit/artoolkit5

[4] Vuforia: VuMark examples. online obrázek. Dostupné z: https://library.vuforia.com/articles/Training/VuMark

[5] Sherr, I.; Stein, S.: *Microsoft's HoloLens 2 announced*. Dostupné z: https://www.cnet.com/news/microsoft-hololens-2-announced-for-3500-available-to-preorder-now-ships-later-this-year/

[6] Rubino, D.: *Microsoft HoloLens full specs*. Dostupné z: https://www.windowscentral.com/microsoft-hololens-processor-storage-and-ram

[7] Newman, J.; Chacos, B.: *HTC Vive vs. Oculus Rift vs. Windows Mixed Reality*. Dostupné z: https://www.pcworld.com/article/3223202/htc-vive-vs-oculus-rift-vs-windows-mixed-reality.html

[8] MonoProject: *About Mono*. Dostupné z: https://www.mono-project.com/docs/about-mono/

[9] Unity3D: *VR Overview*. Dostupné z: https://docs.unity3d.com/Manual/VROverview.html

[10] Microsoft: MRTK. software. Dostupné z: https://microsoft.github.io/MixedRealityToolkit-Unity/README.html

[11] Reguieg, K.: *EV3ControllAR - Control Robots via the Microsoft HoloLens*, 2017-06-23. Dostupné z: https://artcom.github.io/control-robots-via-hololens/

[12] Rieker, M.: *Using the Microsoft HoloLens as a Robotic Controller*, 2018-05-04. Dostupné z: https://repository.uwyo.edu/honors_theses_17-18/63/

[13] Microsoft: MRDesignLabsUnityPeriodicTable. software, 2017-11-17. Dostupné z: https://github.com/Microsoft/MRDesignLabs_Unity_PeriodicTable

[14] Qian, L.: HoloLensARToolKit v0.2. software, 2019-01-01. Dostupné z: https://github.com/qian256/HoloLensARToolKit

[15] Vuforia: *Developing Vuforia Engine Apps for HoloLens*. Dostupné z: https://library.vuforia.com/articles/Training/Developing-Vuforia-Apps-for-HoloLens

[16] Vuforia: *Vuforia Licence Manager*. Dostupné z: https://library.vuforia.com/articles/Training/Vuforia-License-Manager

[17] Microsoft: *MR and Azure tutorials*. Dostupné z: https://docs.microsoft.com/cs-cz/windows/mixed-reality/mr-azure-301

[18] Microsoft: *Limits and quotas*. Dostupné z: https://docs.microsoft.com/cs-cz/azure/cognitive-services/custom-vision-service/limits-and-quotas

[19] PimDeWitte: UnityMainThreadDispatcher. software, 2018-08-23. Dostupné z: https://github.com/PimDeWitte/UnityMainThreadDispatcher

Příloha **A**

Seznam použitých zkratek

6DoF Six Degrees of Freedom

API Application Programming Interface

AR Augmented Reality, Rozšířená Realita

CPU Central Processing Unit

GUI Graphical User Interface

HPU Holographic Processing Unit

JSON JavaScript Object Notation

MR Mixed Reality, Smíšená Realita

MRTK Mixed Reality ToolKit

RPi Raspberry Pi

TCP/IP Transmission Control Protocol/Internet Protocol

UDP User Datagram Protocol

UI User Interface

UWP Universal Windows Platform

VR Virtual Reality, Virtuální Realita

XR Spojení AR, MR a VR

Příloha B

Uživatelská příručka

Aplikace je v době psaní práce nainstalovaná na HoloLens pod názvem HoloBot. Po jejím spuštění čeká na připojení od robotů, ke kterému můžeme použít přiložený soubor holoclient.py. Kód v pythonu spustíme příkazem 'python2.7 holoclient.py'. Změnou řádku 'import holorobot as robot' na holocar nebo holohexa specifikujeme použitého robota. Oba skripty se musí spustit pouze na jejich přidělených robotech (Sunfounder Smart Video Car pro holocar.py a Hexapod pro holohexa.py). Holocar.py potřebuje k funkčnosti přiložený soubor PCA9685.py, dostupný na githubu Sunfounder[3] a holohexa.py potřebuje hexapod.py, který je součástí bakalářské práce Jitky Seménkové. Uživatel také může implementovat holorobot.py pro jiného robota.

Pro další vývoj aplikace je zapotřebí vývojové prostředí Unity 3D a Visual Studio. Použitá verze Unity byla 2017.4.19f1 a 2017.4.17f1. Program by měl podporovat i novější LTS verze Unity a při jejich instalaci je nutné zatrhnout kolonku 'Windows Store .NET Scripting Backend'. Pro export projektu do Visual Studia je zapotřebí změnit několik nastavení. V záložce File → Build Settings nastavíme platformu na 'Universal Windows Platform' pomocí tlačítka 'Switch Platform'. Dále změníme 'Target Device' na HoloLens a zaškrtneme položku 'Unity C# Projects'. Před kompilací zkontrolujeme, že v záložce Edit → Project Settings → Player je v části Other Settings → Scripting Backend zvoleno .NET, v části XR Settings zaškrtnuto 'Virtual Reality Supported' a v listu 'Virtual Reality SDKs' je záznam 'Windows Mixed Reality'. Poté můžeme projekt zkompilovat v File → Build Settings → Build (vhodné je výstup uložit do nové složky např. App, protože v domovské složce Unity projektu už existuje soubor HoloBot.sln, který by byl přepsaný).

Po dokončení kompilace se otevře prohlížeč ve složce projektu a ve Visual Studiu 2017 otevřeme soubor ./App/HoloBot.sln (v labu 1054 otevřeme pravým tlačítkem → Open with → 2. možnost Microsoft Visual Studio, první nefunguje). Pro instalaci aplikace na HoloLens zvolíme Configuration → Release,

[3] https://github.com/sunfounder/Sunfounder_Smart_Video_Car_Kit_for_RaspberryPi

Platform → x86 a Deploy na Remote Machine s adresou 10.10.48.192. HoloLens v tomto stavu musí být zapnuté a přihlášené na uživatele skrbek.

Aplikaci je možné ovládat gestem Tap, se kterým můžeme klikat na šipky, a gestem Bloom dojde k jejímu ukončení. Návod na použití gest je na stránkách Microsoftu[4].

[4]https://support.microsoft.com/en-us/help/12644/hololens-use-gestures

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src` ..... adresář se zdrojovými kódy
  - `unity` ..... adresář Unity projektu
  - `robots` ..... adresář s kódy pro roboty
  - `doc` ..... dokumentace kódu
  - `unity.zip` ..... zkomprimovaný soubor Unity projektu
- `text` ..... adresář se zdrojovými soubory práce
  - `bp_hololens.tex` ..... zdrojový text práce ve formátu LaTeX
- `bp_hololens.pdf` ..... text práce ve formátu PDF