VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PROMÍTANÁ ROZŠÍŘENÁ REALITA PRO ROBOTICKÉ PRACOVIŠTĚ

SPATIAL AUGMENTED REALITY FOR ROBOTIC WORKPLACE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAKUB ŠTROF**
AUTHOR

VEDOUCÍ PRÁCE **Ing. ZDENĚK MATERNA, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

153682

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Štrof Jakub**
Program:	Informační technologie
Název:	**Promítaná rozšířená realita pro robotické pracoviště**
Kategorie:	Uživatelská rozhraní
Akademický rok:	2023/24

Zadání:

1. Proveďte rešerši využití projekce pro robotická pracoviště.
2. Seznamte se se systémem ARCOR2 umožňujícím programování robotů v rozšířené realitě a možnostmi jeho uživatelského rozhraní AREditor.
3. Navrhněte promítané uživatelské rozhraní, které bude vhodně doplňovat AREditor.
4. Navržené řešení implementujte.
5. Proveďte uživatelské testování.
6. Zdrojové kódy a dokumentaci publikujte na GitHubu.
7. Vytvořte video prezentující vaši práci, její cíle a výsledky.

Literatura:

- MATERNA Zdeněk, et al. Interactive Spatial Augmented Reality in Collaborative Robot Programming: User Experience Evaluation. In: *RO-MAN 2018 - 27th IEEE International Symposium on Robot and Human Interactive Communication*. NanJing: Institute of Electrical and Electronics Engineers, 2018, s. 80-87. ISBN 978-1-5386-7980-7.
- Mengoni, Maura, et al. "Spatial Augmented Reality: An application for human work in smart manufacturing environment." *Procedia Manufacturing* 17 (2018): 476-483.
- Avalle, Giancarlo, et al. "An augmented reality system to support fault visualization in industrial robotic tasks." *IEEE Access* 7 (2019): 132343-132359.

Při obhajobě semestrální části projektu je požadováno:

- Body 1-3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Materna Zdeněk, Ing., Ph.D.**
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2023
Termín pro odevzdání:	9.5.2024
Datum schválení:	9.11.2023

Abstrakt

Cílem této práce je vytvoření promítaného uživatelského rozhraní, které vhodně doplní aplikaci pro programování robotů AREditor a kalibrace Kinectu a projektoru pomocí existující kalibrační metody. Pro kalibraci jsem použil metodu procam-calibration, která využívá promítání Grayových kódů na šachovnici pro výpočet kalibračních parametrů. Pomocí ní mohou být virtuální objekty umisťovány na správné místo v reálném prostoru. Uživatelské rozhraní jsem implementoval v herním engine Unity. Testováním rozhraní bylo zjištěno, že uživateli pomáhá lépe pochopit vztah mezi reálným a virtuálním prostorem při umisťování virtuálních objektů do robotického pracoviště. Také zlepšuje povědomí uživatele o pohybech robota při spolupráci.

Abstract

The focus of this work is to create projected user interface, which will complement AREditor, an application for robotic programming and also to calibrate Kinect and projector using existing calibration method. For calibration, I used the procam-calibration method, which uses projected Gray codes onto a chessboard to calculate calibration parameters. Thanks to it, virtual objects can be placed correctly into the real space. User interface was implemented using game engine Unity. By testing the interface, it was found that it helps the user to better grasp the relation between real and virtual space when placing virtual objects into the robotic workplace. It also raises user's awareness of the robot's movements when cooperating with it.

Klíčová slova

rozšířená realita, uživatelské rozhraní, robotické pracoviště, Unity, projekce, kalibrace, AREditor, Kinect, testování, vizuální programování

Keywords

augmented reality, user interface, robotic workplace, Unity, projection, calibration, AREditor, Kinect, testing, visual programming

Citace

ŠTROF, Jakub. *Promítaná rozšířená realita pro robotické pracoviště*. Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Zdeněk Materna, Ph.D.

Promítaná rozšířená realita pro robotické pracoviště

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Zdeňka Materny, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jakub Štrof
7. května 2024

Poděkování

Rád bych poděkoval panu vedoucímu Ing. Zdeňku Maternovi, Ph.D., za ochotu, odbornou pomoc při konzultacích i mimo ně a za čas, který mi při vedení mé práce věnoval. Dále bych chtěl také poděkovat všem studentům, kteří mi pomohli otestovat mou práci.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Existují různé metody pro programování kolaborativních robotů, přičemž většina z nich vyžaduje psaní kódu. Avšak existuje i možnost vizuálního programování, které místo kódu využívá grafické rozhraní. Je vhodné i pro uživatele s téměř nulovými zkušenostmi s programováním robotů, jelikož nevyžaduje znalost programovacího jazyka – místo textových příkazů může uživatel tvořit program robota pomocí grafických prvků.

Vizuální programování využívá i mobilní aplikace pro programování kolaborativních robotů AREditor. Ta je součástí systému ARCOR2, který slouží k zjednodušenému programování kolaborativních robotů v rozšířené realitě. Rozšířená realita pomáhá uživateli lépe vnímat vztah mezi body v naprogramované scéně a v reálném prostoru. Při používání AREditoru je však nutné držet v rukou tablet, což znemožňuje například sledování chodu programu a zároveň manuální spolupráci s kolaborativním robotem.

Cílem této práce je tedy návrh a implementace uživatelského rozhraní, které bude vhodně doplňovat AREditor při programování i běhu již vytvořeného programu robota. Pro toto rozhraní bude využita promítaná rozšířená realita. Ta bude vytvořena pomocí projektoru, umístěného nad robotickým pracovištěm, díky kterému mohou být uživateli zobrazovány informace bez nutnosti držení mobilního zařízení. Promítané rozhraní umožní zobrazení prvků programu přímo na plochu pracoviště, takže uživateli pomůže vidět, kde přesně se nacházejí na této ploše . Využít rozhraní lze i při spuštění programu, kdy uživatele informuje o stavu běžícího programu nebo ho varuje před neočekávanými pohyby robota.

Kapitola 2 popisuje rozšířenou realitu, kalibraci kamery a projektoru a výběr možných kalibračních metod. Kapitola 3 se věnuje rozboru systému ARCOR2, jehož součástí je i rozhraní AREditor. Cíl práce, návrh uživatelského rozhraní a výběr kalibrační metody je představen v kapitole 4 a jejich implementace v kapitole 5. Testování vytvořeného rozhraní je popsáno v kapitole 6.

Kapitola 2

Rešerše problematiky

Tato kapitola pojednává o problematice rozšířené reality a jejího využití a následně o promítané rozšířené realitě a jejím využití pro robotická pracoviště. Další část kapitoly se zabývá metodami pro kalibraci kamery a projektoru.

2.1 Rozšířená realita

Rozšířená realita [3] představuje fyzickou realitu, do které jsou přidány také digitální prvky. Je implementována několika způsoby, z nichž nejpoužívanější jsou náhlavní displeje (head-mounted display), handheld zařízení nebo projektory [5]. Nevýhody handheld zařízení, jako například mobilních telefonů, je nutnost stále je držet při používání a také jejich omezené zorné pole a poměrně malá velikost dispeje [15]. Náhlavní displeje [11] mají výhodu volnosti rukou uživatele, problémy však může působit jejich váha a nepohodlí. Promítaná rozšířená realita [22] se vyhýbá problémům s nepohodlností nebo nutností mít zařízení v rukou, ale má nevýhody například v podobě horší viditelnosti obrazu a nemožnosti zobrazení virtuálních objektů v 3D prostoru.

V průmyslu může rozšířená realita pomáhat zefektivnit údržbu strojů, dále může být využita v logistice, prototypování a také při školení zaměstnanců [2]. Obchody s nábytkem mohou pomocí rozšířené reality zobrazit nábytek přímo ve vašem pokoji a její využití je rozšířené i na sociálních sítích, na kterých je uživatelé mohou využít do svých příspěvků. Rozšířená realita je také používána vývojáři v herním průmyslu, kde je pravděpodobně nejznámějším příkladem mobilní hra Pokémon GO, tu lze vidět na obrázku 2.1.

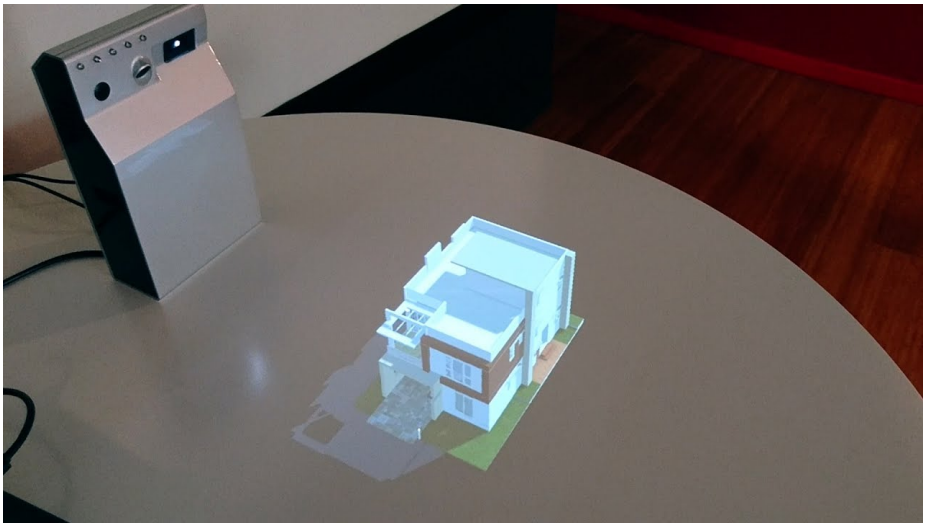
2.2 Promítaná rozšířená realita

Promítaná rozšířená realita [8] využívá projektoru a kamery, aby dokázala zobrazit přidanou digitální vrstvu do reálného prostředí. Dá se použít mnoha způsoby a jedním z nich je, stejně jako u ostatních typů rozšířené reality, průmysl. Může být využita pro zobrazení návodu sloužícího k sestavení nějakého celku pro pracovníka nebo i běžného uživatele, může různými barvami odlišit bezpečné a nebezpečné zóny pracoviště nebo zjednodušit programování kolaborativního robota. Využití nachází i v nemocničním prostředí, kde může promítat nezbytné informace pro chirurga při provádění operace přímo do operačního pole [17]. Výhody promítané rozšířené reality dobře ukazuje i článek [4], který porovnává tři různé způsoby zobrazení pro navádění uživatele při opravě zařízení. Uživatelé byli s promítanou rozšířenou realitou schopni v průměru rychleji splnit daný úkol a také z jejich pohledu snižo-

Obrázek 2.1: Příkladem využití rozšířené reality na mobilních zařízeních může být populární hra Pokémon GO. Převzato z [21].

vala pracovní vytížení. I v architektuře lze promítanou rozšířenou realitu využít, například pro zobrazení 3D modelů budov, jak ukazuje obrázek 2.2.

Avšak způsob využití, který je pro mou práci nezbytný, je pro robotická pracoviště. Zde slouží nejčastěji jako způsob komunikace robota s člověkem. Robot pomocí promítání může sdělovat informace okolí, ať už chybové hlášky, varování, nadcházející akce robota nebo pokyny o zacházení pro uživatele. Promítání jako způsob komunikace používá práce [7], ve které je projektor umístěný přímo na pojízdném robotovi a ukazuje přes něj směr nadcházejících pohybů pomocí šipek. Podobný princip je otestován i v práci [6], jejíž autoři navrhli způsob komunikace automaticky řízených vozíků s lidmi, a to na vozíku umístěným projektorem, který opět promítá na zem trasu vozíku. Výsledky uživatelského testování obou těchto prací dokázaly, že promítání trasy a dalších kroků robota zvyšuje důvěru uživatelů vůči němu a zjednodušuje jejich spolupráci. Práce [10] používá promítanou rozšířenou realitu pro vizuální programování robotického ramene pro jednoduché úkoly, jako je přesun předmětů na pracovišti. Při uživatelském testování byla prokázána efektivnost vizuálního programování – účastníci testování dokázali zadaný úkol splnit v průměru dvakrát rychleji než odborník používající teach pendant, standardní rozhraní používané pro programování robotů. Avšak rozhraní, které pro programování robota používá pouze promítanou realitu a rozpoznávání objektů, je složité na implementaci. V článku [16] je navrženo využití dvou projektorů pro zobrazování dat na robotickém pracovišti, jednoho stacionárního a druhého mobilního, namontovaného přímo na manipulátoru robota. Navrhuje využití promítaného rozhraní například pro zobrazení bodů, ve kterých bude robot provádět operace nebo také pro vizualizaci toho, kam se robot při dalším kroku bude hýbat, což může zabránit třeba střetnutí s uživatelem. Práce [18] navrhuje použití interaktivní promítané rozšířené reality pro programování kolaborativního robota. Snaží se pomocí vysoké úrovně abstrakce zjednodušit programování robotů, aby jej zvládl i neodborný uživatel. Také míří na snížení rozptýlení pozornosti uživatele tím, že jeho jedinými vstupy mohou být buď dotyková plocha pracovního stolu nebo robotická ramena. Uživatel se tak může soustředit pouze na pracoviště bez nutnosti interakce mimo něj. Článek však také zmiňuje, že při velkém počtu promítaných virtuálních objektů může být rozhraní matoucí.



Obrázek 2.2: Příkladem využití promítané rozšířené reality pro architekturu může být zařízení HoloLamp. Převzato z videa https://www.youtube.com/watch?v=3gu01JbC7VY.

Z výsledků těchto prací jsem usoudil, že v promítaném rozhraní pro robotická pracoviště je vhodné mít indikaci směru nebo cílového místa pohybu robota a také body, ve kterých robot bude provádět naprogramované operace. Vhodná je i nějaká forma varování uživatele, například před neočekávanými pohyby robota nebo při jeho selhání. Rozhraní by se ale mělo vyvarovat zobrazování příliš velkého počtu grafických prvků.

2.3 Kalibrace kamery a projektoru

Při přidávání virtuálních objektů do reality pomocí promítání je nutné, aby projektor promítal všechny tyto objekty na správné místo. Přesného promítání však není možné docílit bez zkalibrovaného projektoru, který je obtížné zkalibrovat bez kamery. Kvůli tomu je potřeba, aby systém obsahoval i kameru. Tu je však nutné společně s projektorem také zkalibrovat. Tato kapitola se tedy bude věnovat nejdříve způsobu kalibrace kamery a po něm způsobu kalibrace projektoru.

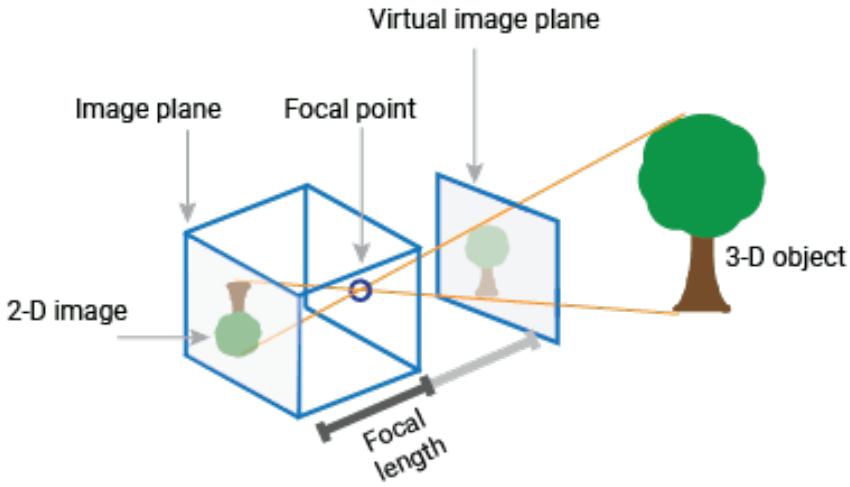
2.3.1 Kalibrace kamery

Kalibrace kamery, jak ji popisuje článek [19], je odhad parametrů kamery pomocí výpočtů. Tyto parametry následně mohou být použity pro opravu zkreslení, zjištění rozměrů reálného objektu ve snímku nebo pro zjištění polohy kamery ve scéně.

Pro modelování kamery se používá dírkový model, jehož princip je ilustrovaný na obrázku 2.3. Ten reprezentuje jednoduchou kameru bez objektivu, pouze s malým otvorem, kterým prochází světlo na zadní stranu kamery. Tam vytvoří převrácený obraz scény před kamerou.

Parametry tohoto modelu jsou určeny maticí nazývanou matice kamery 2.1. Tato matice mapuje 3D scénu reálného světa na obrazovou rovinu kamery. Počítá se pomocí vnějších a vnitřních parametrů kamery, přičemž vnější reprezentují pozici a rotaci kamery v reálném světě a vnitřní parametry optický střed kamery, ohniskové vzdálenosti a parametr zkosení.

$$P = K[Rt] \tag{2.1}$$



Obrázek 2.3: Princip dírkového modelu. Převzato z [19].

kde P je matice kamery, K je matice s vnitřními parametry, R je matice rotace a t je translační vektor.

Při výpočtu se nejprve převedou body z reálného prostoru na souřadnice kamery pomocí vnějších parametrů. Poté se použijí vnitřní parametry pro namapování souřadnic kamery na obrazovou rovinu.

Vnitřní parametry jsou definovány maticí:

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

kde f_x a f_y jsou ohniskové vzdálenosti v pixelech, c_x a c_y představují optický střed kamery a s představuje koeficient zkosení.

Vnější parametry [24] obsahují pozici a orientaci kamery v reálném prostoru. Pozici reprezentuje translační vektor 2.3 a orientaci matice rotace 2.4.

$$T = \begin{bmatrix} T_x & T_y & T_z \end{bmatrix}^T \tag{2.3}$$

$$R = \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix} \tag{2.4}$$

2.3.2 Kalibrace projektoru

Kalibraci projektoru [14, 9] nelze provést stejným způsobem jako kalibraci kamery, jelikož má opačný princip kamery. Ta vytváří 2D obraz 3D prostoru, jehož projekce dopadá na její obrazovou rovinu. U projektoru je 2D obraz promítán do 3D prostoru. Díky tomu lze projektor modelovat inverzním dírkovým modelem. Abychom mohli však zjistit vztah mezi promítanými 2D body a korespondujícími 3D body v reálném prostoru, potřebujeme k tomu kameru.

Je tedy třeba nalézt vhodnou metodu pro kalibraci celého systému, kamery i projektoru, abychom mohli odhadnout jejich pozice a orientace v prostoru.

2.3.3 Metody kalibrace

Pro kalibraci kamery a projektoru jsem neimplementoval svou vlastní metodu protože není v časových možnostech této práce implementovat kalibraci a také navrhnout a implementovat i uživatelské rozhraní. Vybíral jsem tedy z existujících metod, jejichž implementace je volně dostupná. Není však možné pro mou práci vybrat kteroukoliv z nich, protože každá z nich má své výhody i nevýhody a ty je třeba zvážit pro konkrétní aplikaci této práce.

První kalibrační metoda se nazývá procam-calibration[1]. Ta je implementována v jazyce Python a využívá knihovnu OpenCV[2], která poskytuje metodu pro kalibrování stereo systému – systému se dvěma kamerami. Pro výpočet kalibračních parametrů používá algoritmus, navržený v článku [20]. Ten používá promítání Grayových kódů na šachovnici a lokální homografie ke zlepšení přesnosti odhadu pod 1 pixel. Také nevyžaduje zkalibrovanou kameru. Autoři tohoto článku však kalibrují systém pomocí několika snímků šachovnice v jedné pozici. Autor kalibrace procam-calibration navrhuje, že pro jiné využití než je v článku, je vhodné pořídit alespoň 5 sad snímků šachovnice a to pokaždé v jiné pozici. Přesnost kalibrace by jinak mohla být velice nízká.

Další kalibrační metodou je ofxCvCameraProjectorCalibration[3]. Ta opět používá knihovnu OpenCV pro výpočet kalibračních parametrů, ovšem je implementována v jazyce C++ jako doplněk pro software OpenFrameworks. Prvním krokem je zkalibrování kamery samotné. Dále metoda vypočítá vnitřní parametry projektoru a použije metodu pro kalibraci stereo systému z knihovny OpenCV na odhad vnějších parametrů. Tato metoda opět pro kalibraci využívá šachovnici, avšak není nutné pořizovat jednotlivé snímky v různých pozicích. Snímání a výpočet parametrů probíhá v reálném čase, uživatel nejprve musí staticky držet šachovnici a poté ji naklánět a pohybovat s ní před kamerou a projektorem, metoda při tom dynamicky spočítá jejich vnější parametry.

Třetí kalibrační metodou je ProjectorCameraCalibration[4]. Ta je popsána i v článku [23]. Implementována je v jazyce C++ a opět používá knihovnu OpenCV. Nepoužívá však šachovnici jako předchozí metody, místo ní uživatel před kamerou manipuluje s deskou s náhodnými vzory teček. Jako předchozí metoda snímá v reálném čase a automaticky získá snímek pro kalibraci v momentě, kdy je deska nehybná. Po získání předem daného počtu snímků je kalibrace ukončena. Postup při kalibraci je vysvětlen ve videu[5] autorů této kalibrační metody.

[1] https://github.com/kamino410/procam-calibration
[2] https://opencv.org
[3] https://github.com/cyrildiagne/ofxCvCameraProjectorCalibration
[4] https://github.com/limingyangpro/ProjectorCameraCalibration
[5] https://www.youtube.com/watch?v=NpdOrKHBH_w

Kapitola 3

ARCOR2

Tato kapitola se zabývá systémem ARCOR2 [13] (Augmented Reality Collaborative Robot), který slouží pro programování kolaborativních robotů v rozšířené realitě. Představím v ní robotické pracoviště, na kterém bude probíhat návrh a implementace mé práce, dále terminologii, používanou v systému ARCOR2, jeho architekturu, způsob jakým komunikuje a jeho grafické uživatelské rozhraní, AREditor.

Informace pro tuto kapitolu jsou čerpány z repozitáře systému ARCOR2[1].

3.1 Robotické pracoviště

Pro návrh, implementaci i testování mého rozhraní je třeba systém s kamerou, projektorem, robotem a serverem s běžící instancí ARServeru. Vysvětlení toho, co je to ARServer, se nachází v kapitole 3.3. Využívám tedy robotické pracoviště (obrázek 3.1), které mi bylo zpřístupněno po registraci zadání práce. Vybavení pracoviště:

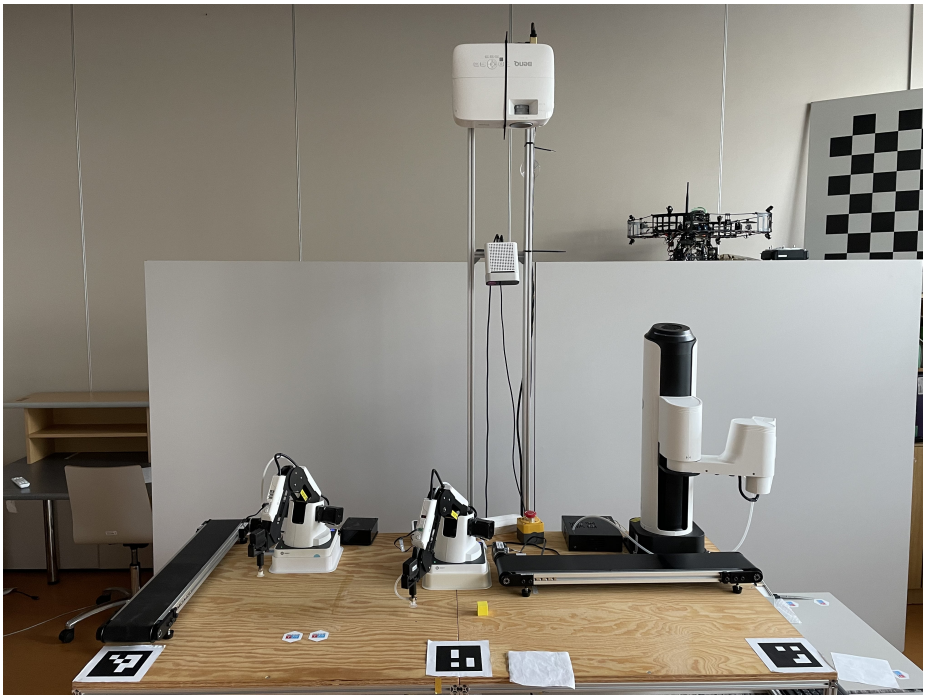
- Projektor BenQ MH733
- Azure Kinect DK
- 2× Robotické rameno Dobot Magician
- Průmyslový robot Dobot M1
- 2× Pás Dobot Conveyor Belt

3.2 Terminologie

Systém ARCOR2 má svou vlastní terminologii, kterou je pro pochopení a práci s ním nutné objasnit. Termíny budou pro vysvětlení uvedeny v originálním znění, dále v této technické zprávě však budou překládány.

Object type *Typ objektu*, plugin, který reprezentuje určitý typ reálného objektu a umožňuje integraci s ním. Například konkrétní typ robota nebo virtuální objekt jako je cloudové API. Plugin je napsán v jazyce Python a může využívat i několikanásobnou dědičnost tak, aby rozšířil nebo sdílel funkcionalitu. V systému již existují některé vestavěné třídy, reprezentující například robota nebo kameru a jejich API. Object type

[1] https://github.com/robofit/arcor2/wiki



Obrázek 3.1: Robotické pracoviště, na kterém probíhá implementace mé práce.

metody, jež mají speciální anotaci, se stávají akcemi, které se mohou přiřadit akčním bodům.

Action object *Akční objekt*, instance typu objektu v pracovišti, definovaná unikátním ID, jménem, pózou (volitelná) a parametry (např. API URI, sériový port, atd.). Reprezentují nějaký reálný objekt, zařízení nebo robota na pracovišti. Mohou však také reprezentovat nějaký virtuální objekt, kolizní zónu nebo takovou zónu, do které robot nemůže.

Scene *Scéna*, soubor několika akčních objektů, reprezentující pracoviště, jeho objekty a prostorové vztahy.

Action point *Akční bod*, v prostoru upevněný bod, který může obsahovat orientace, konfigurace kloubů robota a akce. Jeho pozice a orientace dohromady mohou tvořit pózu, využitelnou například jako parametr pro akci robota. Akční bod k sobě může mít přiřazeny akce.

Action *Akce*, reprezentuje jeden krok v programu robota. Každá akce je přiřazena právě jednomu akčnímu bodu. Jaké akce je možné přidat, specifikuje akční objekt, ke kterému akce přísluší. Například robotické rameno umožňuje zvednout, přesunout, položit, atd.

Project *Projekt*, soubor akčních bodů, který může obsahovat definici logiky (průběhu programu).

Execution package *Spustitelný balíček*, samostatný spustitelný snapshot projektu, může být vytvořen za účelem testu vytvořeného úkolu pro robota nebo za účelem uvedení projektu do provozního prostředí.

Main script *Hlavní skript*, obsahuje logiku projektu. Ta může být definovaná v grafickém prostředí nebo manuálně napsána s pomocí souboru vygenerovaných tříd, poskytujících přístup k datům projektu, jako jsou třeba akční body.

Pose *Póza*, používá se pro určení pozice a orientace objektu v systému ARCOR2. Pozici reprezentuje 3D vektor a orientaci 4D kvaternion[2].

3.3 Architektura systému

Framework ARCOR2 je rozdělen na backend a frontend, přičemž backend tvoří soubor nezávislých služeb a frontend uživatelská rozhraní. Hlavní službou systému je ARServer, který slouží jako centrální bod pro uživatelská rozhraní a zprostředkovává komunikaci s ostatními službami. Server je navržen tak, aby dokázal obsluhovat více připojených klientů (uživatelských rozhraní) najednou. Architektura celého systému je ilustrována na obrázku 3.2.

Pokud je na server připojený již jeden uživatel přes rozhraní a připojí se další, tak je mu otevřen stejný projekt jako prvnímu. Všechny připojená rozhraní tedy zobrazují stejný projekt a každé z nich jej může změnit. V systému je však implementován zamykací mechanismus, který brání více uživatelům najednou manipulovat se stejným prvkem v projektu. Kromě ARServeru jsou v systému další služby:

Project Service Poskytuje trvalé úložiště pro data, relevantní pro pracoviště: scény, projekty, typy objektů, modely, atd.

Scene Service Služba, používaná pro případy, ve kterých je implementace založena na ROS. Je zodpovědná za správu kolizních objektů.

Build Service Slouží k vytváření samostatných balíčků z projektů. Jejich logika může být definována buď v prostředí rozšířené reality nebo prostřednictvím souboru v JSON formátu.

Execution Service Spravuje spustitelné balíčky, vytvořené službou Build Service. Její nejdůležitější funkce spočívá v zasílání událostí, týkajících se stavu spuštěného balíčku (například jaká akce s jakými parametry je právě prováděna), ARServeru.

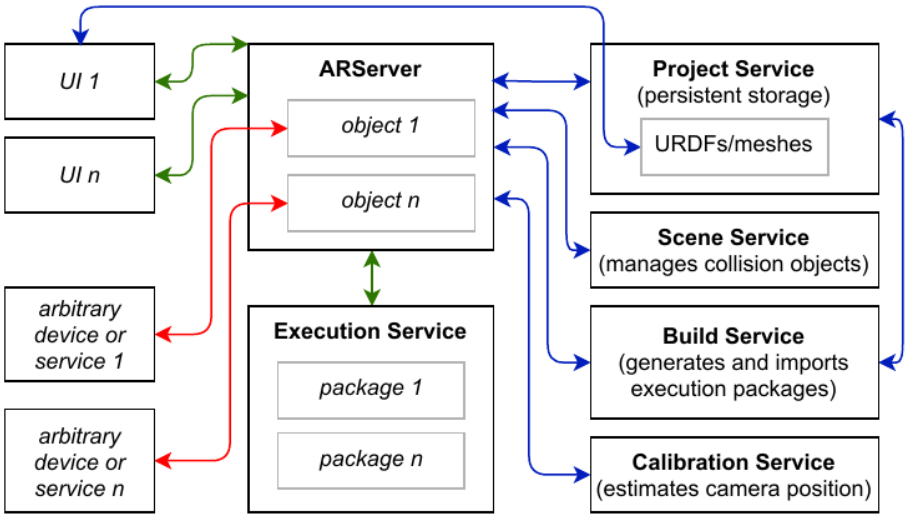
Calibration Service Poskytuje funkci pro odhad pozice a orientace kamery, která je založena na detekci ArUco značek. Také poskytuje metodu pro úpravu pozice a orientace robota pomocí RGBD kamery.

3.4 ARServer API

Komunikace mezi ARServerem a rozhraními probíhá přes protokol WebSocket, ten umožňuje obousměrnou komunikaci a je založený na zasílání událostí (events) a RPC. RPC se dělí na žádost a odpověď, přičemž žádost klienta na server může být buď odmítnuta nebo přijata. Události jsou používány službami pro upozornění klientů o asynchronních událostech. Mohou signalizovat několik typů změn:

- ADD – Přidání např. akčního bodu do projektu

[2] https://www.allaboutcircuits.com/technical-articles/dont-get-lost-in-deep-space-understanding-quaternions/



Obrázek 3.2: Architektura systému ARCOR2. Převzato z https://github.com/robofit/arcor2/wiki.

- UPDATE – Proběhnutí nějaké změny
- UPDATE_BASE – Změna, např. jména projektu, ne však jeho objektů
- REMOVE – Odstranění nějakého objektu

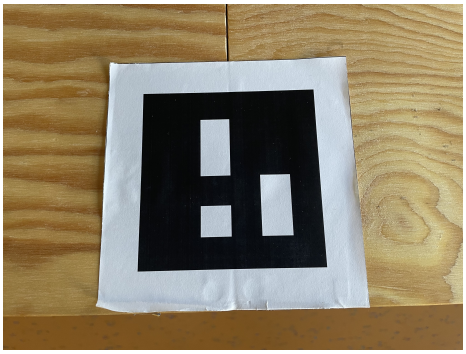
ARServer má pro připojení všech uživatelských rozhraní otevřen specifický port na jeho IP adrese. Při připojení klienta na server by mu od něj měla být zaslána jedna z těchto událostí:

- ShowMainScreen – V momentě, kdy není otevřena scéna, projekt ani není spuštěný balíček
- OpenScene – Když je otevřena scéna
- OpenProject – Když je otevřen projekt
- PackageState – Když je spuštěn balíček

Pokud chce uživatel v rozhraní zobrazit projekt, musí rozhraní získat data z právě otevřeného projektu pomocí události OpenProject. Ta obsahuje informace o všech objektech, které jsou v daném projektu obsaženy. Při změnách v projektu však server nezasílá znovu všechna data, ale pouze informace o těch objektech, kterých se změna týkala. Klient si tedy musí sám uchovávat aktuální stav otevřeného projektu a měnit jej na základě přicházejících událostí.

3.5 Uživatelské rozhraní AREditor

Uživatelské rozhraní AREditor je součástí systému ARCOR2. Jedná se o aplikaci na mobilní zařízení, přes kterou uživatel interaguje s robotem. Snaží se zjednodušit programování kolaborativních robotů, aby jej zvládl i neodborný uživatel, pomocí vizuálního programování a rozšířené reality.



Obrázek 3.3: ArUco značka, sloužící ke kalibraci.

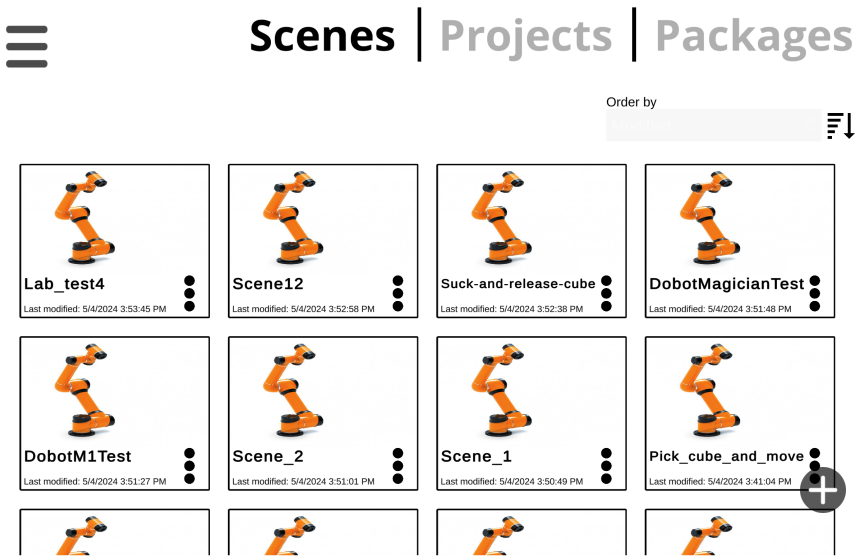
Obrázek 3.4: Automatická kalibrace, vyzývající uživatele k namíření tabletu na kalibrační značku.

3.5.1 Kalibrace

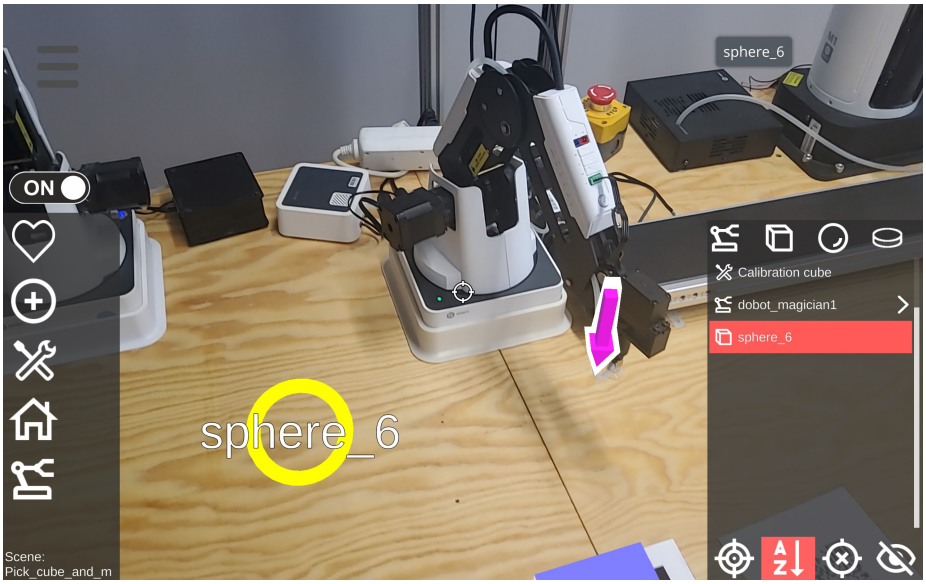
Jelikož jsou body v AREditoru vztaženy k reálnému místu v prostoru, je potřeba aplikaci před tvořením scény nebo projektu zkalibrovat. Kalibrace je automaticky spuštěna při otevření editoru scény nebo projektu. Uživatel je animací vyzván k hýbání s tabletem, dokud nedojde k zachycení ArUco značky (obrázek 3.3) na pracovní ploše a nedojde k automatické kalibraci. Ta anotuje střed značky jako střed scény (se souřadnicemi $(0, 0, 0)$) a zobrazí na této značce kalibrační kostku. Pokud jsou ve scéně nebo projektu definovány nějaké objekty, jsou po zkalibrování zobrazeny. Ukázka automatické kalibrace je na obrázku 3.4.

3.5.2 Hlavní obrazovka

První obrazovkou v aplikaci je hlavní obrazovka (obrázek 3.5), zobrazující seznam všech vytvořených scén, projektů a balíčků. Ty v ní může uživatel také vytvořit, upravit nebo odstranit.



Obrázek 3.5: Hlavní obrazovka AREditoru.



Obrázek 3.6: Editor scény s přidanými akčními objekty.

3.5.3 Editor scény

Další obrazovkou aplikace je editor scény (obrázek 3.6). V něm uživatel specifikuje akční objekty, které ve scéně chce mít. Akční objekty mohou být transformovány – lze nastavit jejich pozici, orientaci a u virtuálních kolizních objektů i velikost. Virtuální kolizní objekty jsou speciální typ akčních objektů, které reprezentují kolizní zónu pro robota. Momentálně existují v AREditoru tři druhy kolizních objektů – koule, kostka a válec. Všechny akční objekty, i ty kolizní, mají v aplikaci žlutou barvu. Objekty, které reprezentují zařízení, ale mají na rozdíl od těch virtuálních kolizních svůj předdefinovaný model s tvarem, kopírujícím korespondující reálné zařízení. Z vytvořené scény lze pomocí tlačítka vygenerovat projekt. Ten bude obsahovat všechny objekty, které obsahovala i korespondující scéna a umožní navíc definovat akce pro robota.

3.5.4 Editor projektu

V editoru projektu lze definovat projekt, vytvořený ze scény. V projektu uživatel může přidávat akční body (obrázek 3.7) a k nim akce, ty poté propojovat a tím definovat logiku a chod programu. Akční body v projektu reprezentují místa na pracovišti, na kterých má robot provést nějakou akci. Samy o sobě mají pouze pozici, lze jim však přiřadit orientaci nebo konfigurace kloubů robota pro akce. Jejich pozici lze při přidávání do projektu nastavit v menu pro manipulaci objektů. Pokud chce uživatel přidat akční bod co nejpřesněji, lze využít robota pro přidání akčního bodu na konec jeho ramene. V editoru projektu jsou reprezentovány fialovými body v prostoru.

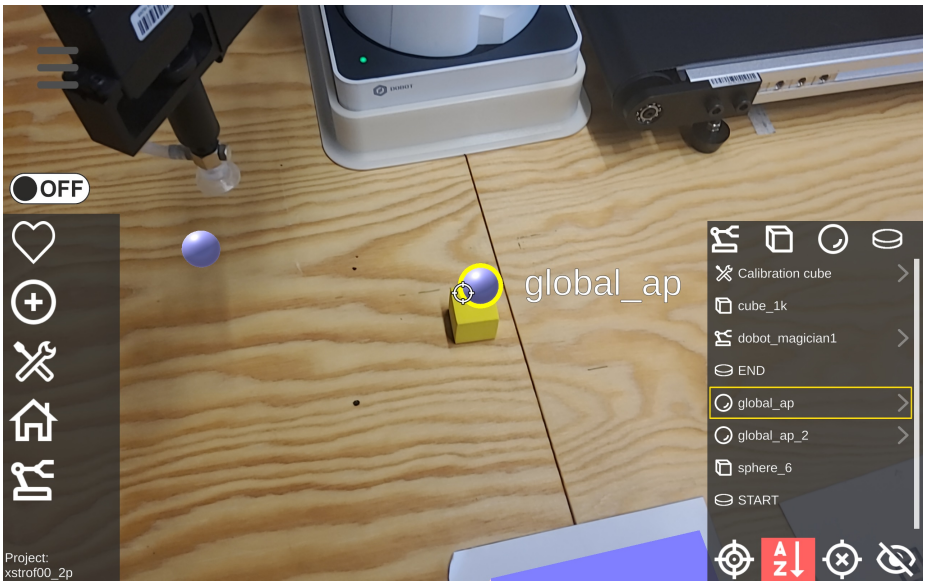
Akce robota (obrázek 3.8), jako například zvednutí, přesun nebo uchycení nějakého předmětu, se přiřazují akčním bodům. V AREditoru jsou znázorněny žlutými šipkami. Jejich typy jsou specifikovány konkrétním akčním objektem, každý může mít jiné. Jeden akční bod může obsahovat několik akcí.

Po tom, co uživatel přidá všechny akce, ze kterých se má skládat program, je pro jeho spuštění nutné ještě specifikovat chod programu. Ten je dán tím, v jakém pořadí uživatel akce propojí. Tyto propojení jsou reprezentovány modrými čarami, spojujícími akce. Lze je vidět na obrázku 3.8. Po propojení lze vytvořený program spustit přímo v editoru projektu. Druhou možností je vytvoření samostatného balíčku, který lze kdykoliv pustit z hlavní obrazovky.

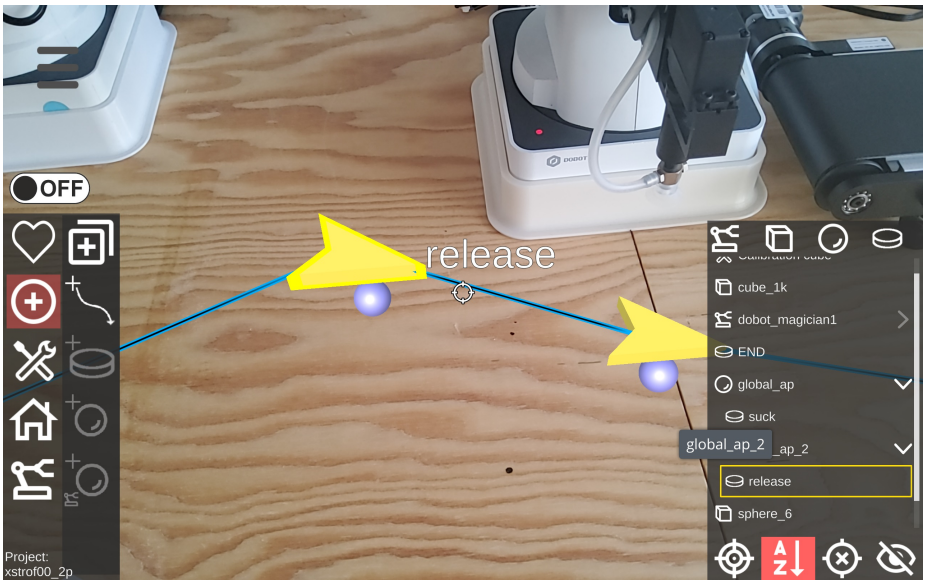
3.5.5 Spuštění programu

V obrazovce pro spouštění programu (obrázek 3.9) je zobrazena stejná virtuální scéna, jako v editoru projektu, ze kterého tento program vznikl. Jsou v ní tedy všechny prvky scény i projektu, ale navíc je zvýrazněna akce, kterou robot zrovna vykonává.

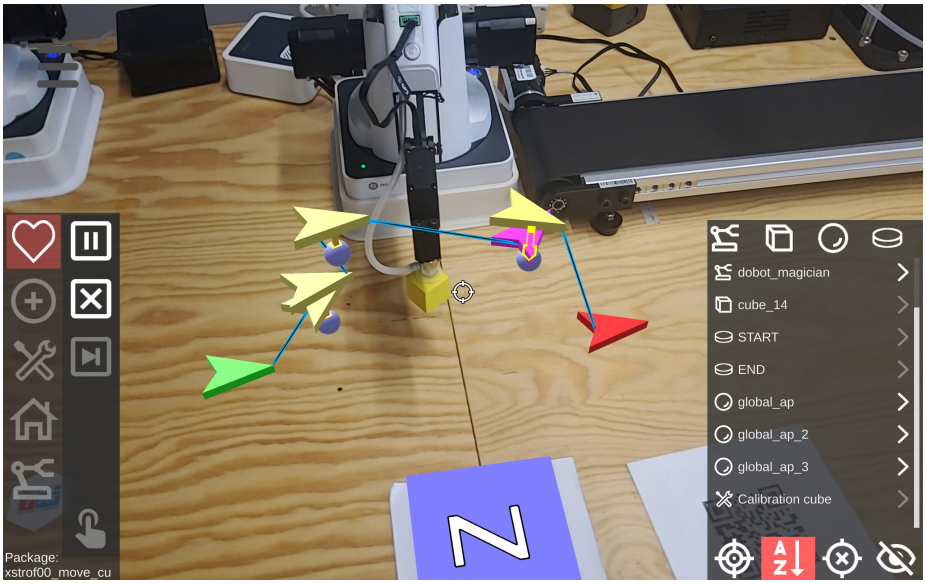
V menu této obrazovky jsou prvky pro ovládání běhu programu. Ty jej umožňují spustit, pozastavit nebo zastavit. Při zastavení se obrazovka vrátí zpět do předchozího stavu, tedy buď na seznam vytvořených balíčků nebo do editoru.



Obrázek 3.7: Akční body v editoru projektu.



Obrázek 3.8: Akce `release`, příslušící akčnímu bodu.



Obrázek 3.9: Obrazovka běhu programu.

Kapitola 4

Návrh uživatelského rozhraní

Tato kapitola je zaměřena na návrh uživatelského rozhraní, jak po stránce jeho vzhledu, tak po stránce jeho funkce a také na výběr metody pro kalibraci.

4.1 Cíl práce

Cílem této práce je vhodně doplnit aplikaci AREditor promítaným rozhraním. Toto rozhraní by mělo být připojeno k ARServeru a přijímat od něj události o změnách a vybrané změny vizualizovat uživateli. Rozhraní nebude interaktivní, bude sloužit pouze pro zobrazování, bez možnosti vstupu od uživatele.

Prvním krokem při používání bude kalibrace kamery a projektoru. Tu provede uživatel a data, která pomocí kalibrační metody získá, budou využita pro vytvoření virtuálního systému kopírujícího reálný vztah mezi kamerou, projektorem a jejich parametry. Druhým krokem bude spuštění aplikace, která se připojí na ARServer a zobrazí uživateli obrazovku, momentálně otevřenou v AREditoru.

4.2 Vzhled a funkce

Jelikož hlavním cílem práce je doplnění AREditoru, je třeba myslet na to, co je vhodné uživateli promítat v případě, že zrovna používá tablet na programování nebo i v případě, že tablet odloží a sleduje pouze pracoviště. Při programování by rozhraní mohl uživatel využít pro kontrolu umístění nebo rozměrů virtuálních objektů přímo na pracovišti. Přesnost promítání je však omezená kvalitou kalibrace, což je také třeba brát v potaz při používání a uživatele na to upozornit. Rozhraní však také lze využít, pokud uživatel tablet nepoužívá a sleduje pouze pracoviště. To může být případ pracovníka, který spolupracuje s robotem na splnění úkolu nebo také případ více lidí, kteří nemají možnost sledovat obrazovku tabletu. V prvním případě jde většinou o již spuštěný program robota, kde by byla vhodná nějaká forma varování uživatele před pohyby robota, jak jsem zjistil při průzkumu existujících řešení v kapitole 2.2. Pro tento i druhý případ s více lidmi u pracoviště by také bylo vhodné zobrazovat stav systému, ve kterém se nachází.

Uživatelské rozhraní tedy bude uživateli zobrazovat několik různých informací. První informací je, v jakém stavu se systém momentálně nachází. Může jít o hlavní obrazovku, některý z editorů nebo spuštěný program. Další informace se budou odvíjet právě od typu zobrazené virtuální scény. Pokud se bude uživatel v systému nacházet například na hlavní obrazovce, tak není nutné promítat žádné další informace. V editoru by ale bylo vhodné

zobrazit například pozice virtuálních objektů, které uživatel přidal přes AREditor a při spuštěném programu informaci o tom, která akce se zrovna vykonává.

Na všech obrazovkách bude tedy promítnuta informace o tom, která z nich je zrovna otevřena. Hlavní obrazovka nebude obsahovat žádné další prvky. Na ostatních budou zobrazeny následující prvky:

Editor scény Akční objekty

Editor projektu Akční objekty, akční body a propojení jejich akcí

Spuštěný program Akční objekty, akční body a barevně odlišené propojení akcí, které zrovna probíhají

Obrázek 4.1 ukazuje návrh rozhraní s otevřenou obrazovkou editoru scény, obrázek 4.2 pak návrh editoru projektu a obrázek 4.3 návrh pro stav, kdy je vykonáván program se zvýrazněným propojením z právě probíhající akce.

Vzhled jednotlivých prvků programu jsem navrhl podle rozhraní AREditoru, aby uživatel měl co nejintuitivnější práci s oběma rozhraními najednou:

Akční objekty Budou reprezentovány dvěma jednoduchými tvary, obdélníkem nebo kruhem a žlutou barvou.

Akční body Budou reprezentovány malými fialovými kruhy.

Propojení Znázorněny šipkami, které budou propojovat akční body. Budou však sloužit k znázornění chodu programu, který je stanoven propojeními mezi akcemi, které jsou přiřazeny daným akčním bodům.

Stav systému V dolním pravém rohu bude zobrazen text o tom, v jakém stavu se systém nachází. Barva textu se bude lišit podle jeho druhu.

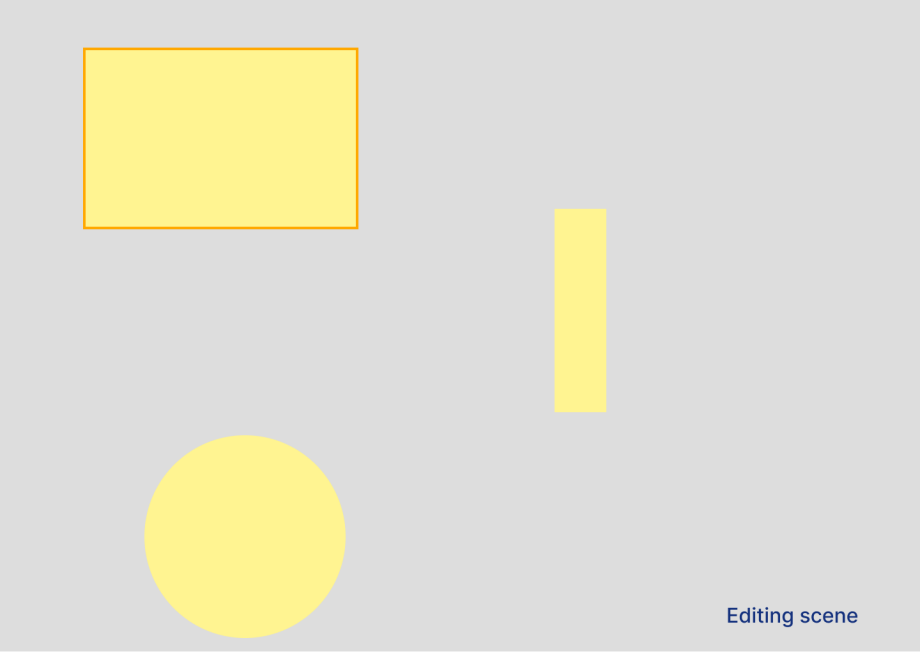
Návrh toho, jak by rozhraní mohlo vypadat na reálném robotickém pracovišti, je předveden na obrázku 4.4. Ukazuje běh programu, ve kterém je virtuální kolizní objekt, tři akční body, šipky ukazující směr průchodu programu a také text „Running“, který oznamuje, že je spuštěn program.

Pro implementaci mého rozhraní jsem se rozhodl použít herní engine Unity[1], který se často používá právě pro vytváření her. Avšak je vhodný i pro vytváření aplikací v rozšířené realitě, jako příklad může sloužit AREditor. Unity pro vytváření virtuální scény používá grafické prostředí, ale umožňuje také programování pomocí skriptů v jazyce C#, které mohou ovlivňovat aplikaci i za jejího běhu. Jeden z hlavních důvodů pro výběr Unity je právě využití těchto skriptů z AREditoru. Část implementace, která se například stará o komunikaci s ARServerem, mohu pouze převzít a pozměnit a není tím pádem nutné implementovat vše znovu.

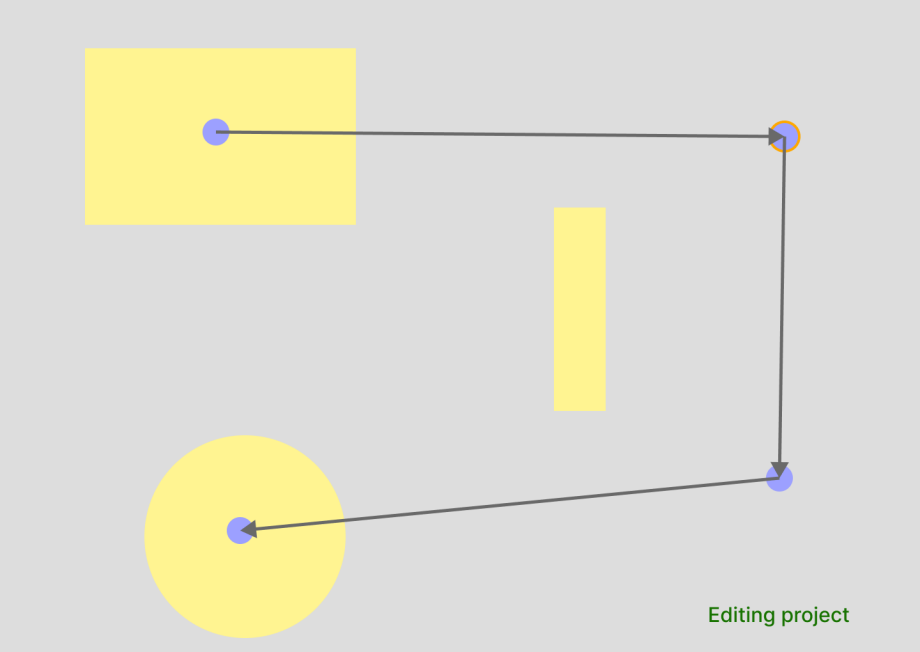
4.3 Kalibrační metoda

Metodu pro zkalibrování projektoru a kamery jsem vybíral z metod, popsaných v kapitole 2.3.3. První kalibrační metoda ProjectorCameraCalibration řeší kalibraci kamery a projektoru hlavně na větší vzdálenosti (více než 250 cm). Také v článku [23] její autor zmiňuje, že pro projektory s větším průměrem čočky má metoda větší chybu odhadu, to se

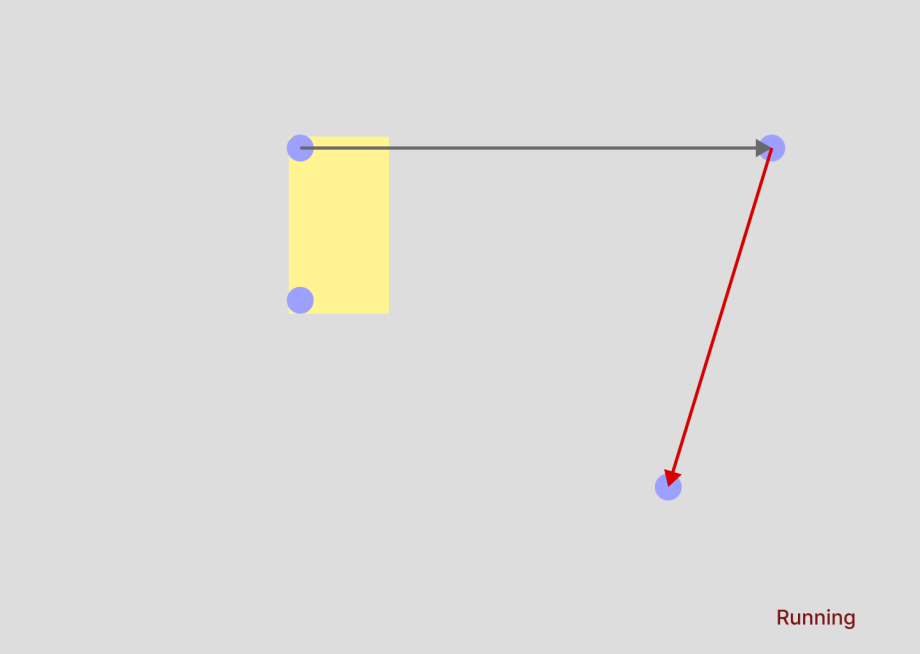
[1] https://unity.com



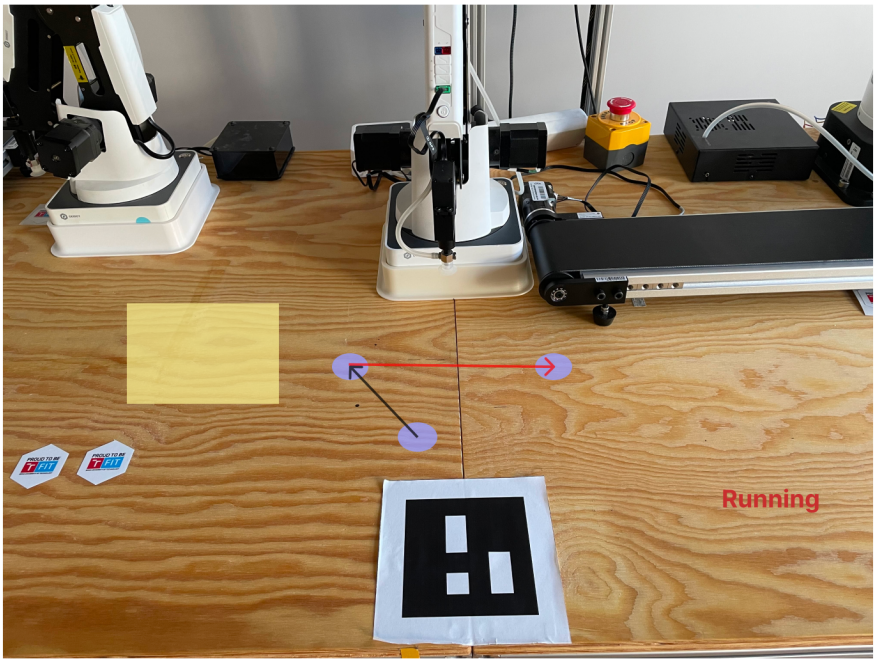
Obrázek 4.1: Návrh rozhraní při úpravě scény.



Obrázek 4.2: Návrh rozhraní při úpravě projektu.



Obrázek 4.3: Návrh rozhraní při běhu programu.



Obrázek 4.4: Návrh toho, jak by mohlo vypadat rozhraní při běhu programu v realitě.

však dá řešit kalibrací na větší vzdálenost. To ale může být problém, jelikož vzdálenost projektoru od stolu na pracovišti činí přibližně 130 cm.

Druhou metodou ve výběru byla ofxCvCameraProjectorCalibration, která funguje na podobném principu jako ProjectorCameraCalibration, avšak využívá šachovnici. Kvůli nedostatku dokumentace k této metodě – jako dokumentace slouží pouze README soubor v repozitáři nebo video na YouTube – a její implementaci jako doplňku do software OpenFrameworks[2], jsem se rozhodl ji nevyužít.

Rozhodl jsem se tedy využít metodu procam-calibration. Má jasně vysvětlený způsob používání, nevyžaduje žádnou další aplikaci pro své spouštění a její autor další informace, možnosti využití a rady k používání publikuje ve veřejném blogu. Tato metoda pro kalibraci používá šachovnici a promítané Grayovy kódy, se kterými je třeba získat několik sad fotografií a poté pomocí nich získat kalibrační parametry systému. Metoda je napsána v jazyce Python a pro kalibraci používá knihovnu OpenCV. Pro její použití je však třeba doimplementovat skript, který nasnímá fotografie pro kalibraci a uloží je do složek po sadách. Metoda vytvoří soubor ve formátu XML, ze kterého potom budu kalibrační parametry ukládat do tříd pomocí skriptu a nastavovat s nimi pozici a orientaci virtuálních objektů, reprezentujících kameru s projektorem.

[2] https://openframeworks.cc/about/

Kapitola 5

Implementace

V této kapitole bude nejprve vysvětlen princip kalibrace, dále postup použití vybrané kalibrační metody z kapitoly 4.3 a poté kalibrace Kinectu pomocí kalibrační služby. Dále se kapitola bude zabývat tím, jak budou výsledky kalibrace použity pro vytvoření virtuálního protějšku systému. Poslední část se bude zabývat implementací uživatelského rozhraní, navrženého v kapitole 4.2.

Implementace probíhala s využitím GitHubu, kde je celý repozitář[1] této práce i s dokumentací zveřejněn.

5.1 Kalibrace

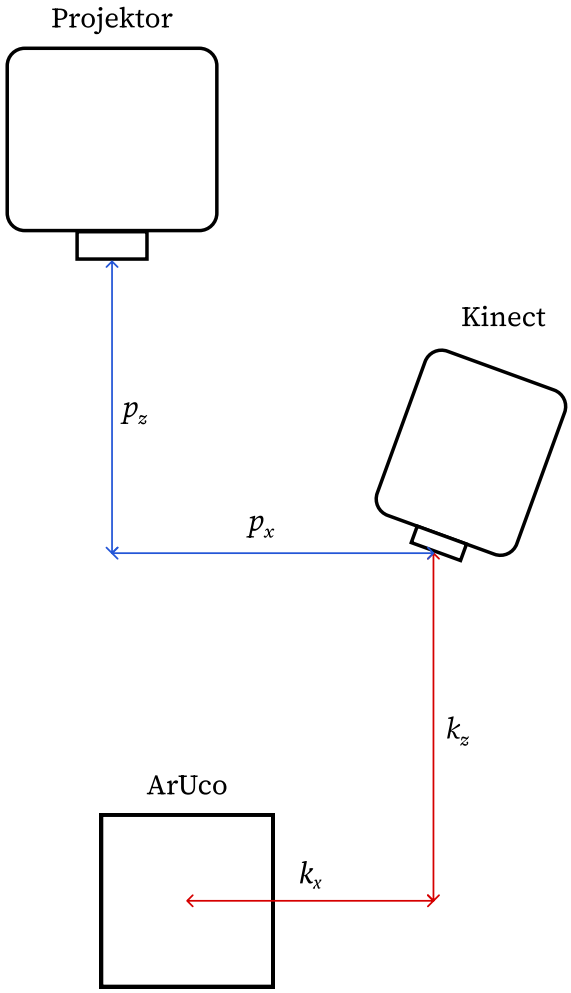
Pro promítání virtuální scény na robotické pracoviště je nutné mít společný bod s uživatelským rozhraním AREditor, podle kterého se bude promítané rozhraní orientovat při zobrazování objektů. AREditor jako bod se souřadnicemi $(0, 0, 0)$ používá prostřední ArUco značku. Od tohoto bodu se počítají pozice všech objektů při zobrazování v jeho prostředí. Můj systém je tedy třeba také zkalibrovat vůči této značce, abych při získávání souřadnic od serveru mohl pozice promítaných objektů správně nastavit. Kalibrace celého systému by se měla provádět pouze poprvé před jeho používáním a poté jen pokaždé, pokud dojde ke změně polohy projektoru nebo pracovního stolu.

Zvolená kalibrační metoda procam-calibration vypočítává pouze vnitřní a vnější parametry kamery a projektoru, což znamená, že pomocí ní získám vztah mezi Kinectem a projektorem v prostoru – jejich pozici a orientaci vůči sobě. Abych tedy znal vztah projektoru vůči ArUco značce, je třeba zjistit také pozici a orientaci Kinectu vůči ní. Ilustrace tohoto problému je na obrázku 5.1.

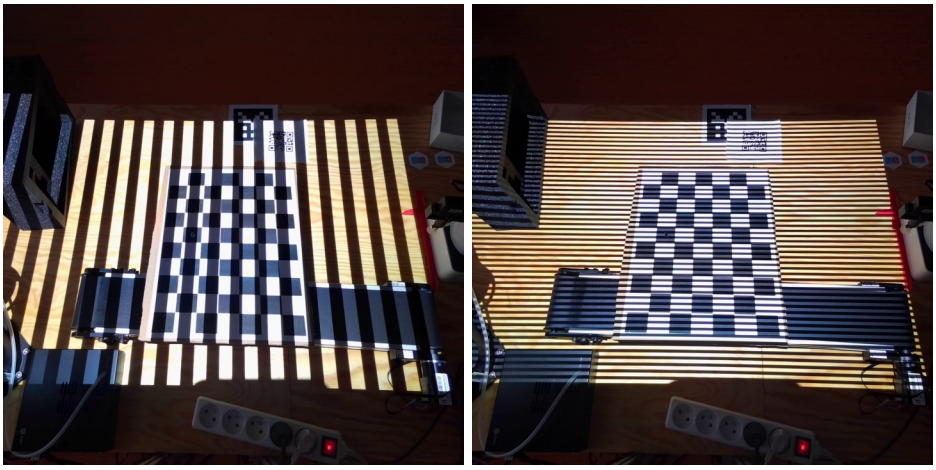
Prvním krokem je tedy kalibrace systému, tvořeného Kinectem a projektorem, pomocí metody procam-calibration a druhým krokem zjištění polohy Kinectu vůči ArUco značce. Metoda pro kalibraci Kinectu vyžaduje na vstupu jeho vnitřní parametry a ty lze získat dvojím způsobem. Prvním způsobem je metoda jeho API, která po zavolání vrací pevně nastavené vnitřní parametry. Druhým a pravděpodobně lepším způsobem je provedení procam-calibration jako první, z ní poté můžu získat a použít vypočítané vnitřní parametry Kinectu.

Pro uložení kalibračních parametrů jsem vytvořil dvě třídy, `KinectCalibrationData` a `ProjectorCalibrationData`, které budou později využity při aplikaci parametrů na virtuální protějšky Kinectu a projektoru. Do těchto tříd je uložen obsah výstupů obou kalibrací.

[1] https://github.com/xstrof00/arcor2_sar



Obrázek 5.1: Vztah mezi projektorem, Kinectem a ArUco značkou. Pomocí metody procam-calibration jsou nejdříve zjištěny hodnoty p_z a p_x, reprezentující vzdálenost projektoru od Kinectu na osách Z a X. Pomocí kalibrace samotného Kinectu jsou vypočítány hodnoty k_z a k_x, ty určují vzdálenost Kinectu od ArUco značky na osách Z a X. Osa Y je pro jednoduchost ilustrace vynechána, ale při kalibraci musí být započítána také.



Obrázek 5.2: Snímky šachovnice s Grayovými kódy, použité pro kalibraci pomocí metody procam-calibration.

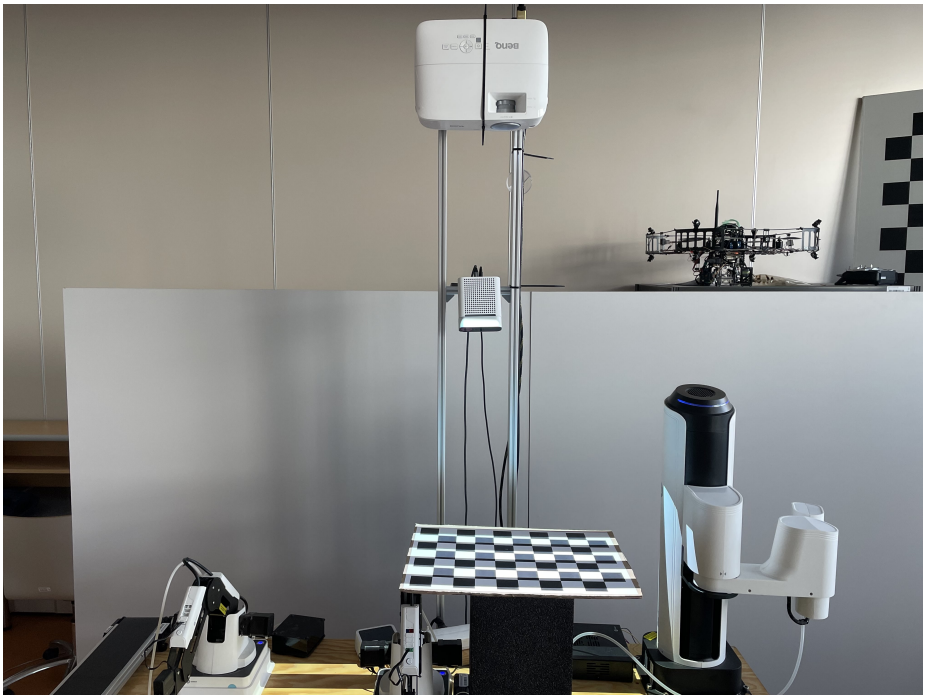
5.1.1 Kalibrace systému pomocí procam-calibration

Kalibrační metoda procam-calibration obsahuje dva skripty v jazyce Python. První skript generuje sadu Grayových kódů, které se uloží pro jejich pozdější promítnutí na šachovnici. Druhý skript implementuje samotný algoritmus pro kalibraci, který zpracovává snímky šachovnice s promítnutými Grayovými kódy a z nich vypočítává kalibrační parametry. Metoda však neobsahuje skript pro pořízení snímků, které mají pro kalibraci posloužit, je třeba jej tedy naimplementovat. Následující text obsahuje informace převzaté z návodu na kalibraci z repozitáře autora této metody[2].

Po spuštění prvního skriptu `gen_graycode_imgs` se vygeneruje sada Grayových kódů. Jejich počet je proměnný, protože záleží na vstupní proměnné `graycode_step`, která určuje velikost bitů (v pixelech) ve vzoru generovaných Grayových kódů. Standardně je velikost jeden pixel, avšak pokud se při snímání objeví moaré efekt[3], je třeba hodnotu proměnné zvyšovat. Dalším vstupem tohoto skriptu je rozlišení projektoru.

Druhým krokem je vytvoření vlastního skriptu, který použije Grayovy kódy a pomocí projektoru je postupně zobrazí na šachovnici na pracovišti. Při promítnutí každého z nich je třeba jej zachytit Kinectem a uložit do složky. Tím se vytvoří jedna sada snímků šachovnice se všemi Grayovými kódy. Příklad toho, jak mohou takové snímky vypadat, se nachází na obrázku 5.2. Skript, který jsem pro tento úkol vytvořil, jsem nazval `project_graycode`. Jako vstup bere dvě hodnoty, výšku a šířku rozlišení projektoru. Je rovněž napsán v jazyce Python. Pro to, abych mohl pomocí tohoto skriptu promítnout vzory na pracoviště, je nutné, aby projektor sloužil jako jeho displej. Skript tedy musí být spouštěn přímo na serveru, ke kterému je tento projektor připojený. Při kalibraci v laboratoři je tedy třeba se na běžící server připojit pomocí příkazu ssh a skript spouštět přímo na něm. Pro zobrazování využívám knihovnu Pygame, sloužící pro vytváření videoher. Ve skriptu nejprve vytvořím proměnnou `displaySurface`, které přiřadím displej a tomu nastavím rozlišení podle používaného projektoru ze vstupů od uživatele. Dále ve smyčce načítám obrázek ze složky s vygenerovanými Grayovými kódy a ten zobrazím ve spuštěném okně `displaySurface`.

[2] https://github.com/kamino410/procam-calibration
[3] https://w.wiki/9zTT



Obrázek 5.3: Kalibrování systému v laboratoři pomocí promítaných Grayových kódů.

Pro zachycení promítnutého vzoru na šachovnici však potřebuji Kinect. Ten umožňuje pořizovat snímky přes metodu jeho API. Pro využití tohoto API je však třeba nejdříve spustit službu Kinectu na serveru. Po spuštění služby je možné provolávat metody, jako právě například pořízení snímku.

Posledním krokem je použití skriptu `calibrate`, naimplementovaného od autora této metody. Ten použije sady pořízených snímků šachovnice pro odhad kalibračních parametrů. Tyto parametry kromě vypsání na výstup uloží i do souboru ve formátu XML pro pozdější použití.

Pro kalibraci mnou používaného systému v laboratoři je vhodná šachovnice na papíře o velikosti A3. S menší šachovnicí metoda vyžaduje více sad snímků a také menší vzdálenost šachovnice od Kinectu při pořizování. Pokud by byla šachovnice větší, bylo by obtížné ji celou pokrýt promítaným vzorem. Ukázku toho, jak probíhala kalibrace v laboratoři, lze vidět na obrázku 5.3.

Pro poměrně přesný odhad vnějších kalibračních parametrů je třeba alespoň 6 sad snímků šachovnice v různých pozicích. Odchylka odhadu translačního vektoru, určujícího pozici projektoru vůči Kinectu, se na všech osách zpravidla pohybuje v rozsahu maximálně 4 cm. Tato odchylka však stačí na to, aby i přesnost projekce měla podobně velkou odchylku. Tím pádem může být při prvním používání nezkalibrovaného systému nutné, provést celý proces kalibrace vícekrát, jelikož nemusí pokaždé odhadnout kalibrační parametry správně. Je také možné, že špatná přesnost projekce je způsobena odchylkou ve výpočtu rotační matice.

Kalibrační metodu procam-calibration jsem testoval před použitím v laboratoři také s menším projektorem a kamerou v jiném prostředí. Otestoval jsem její funkčnost při různých pozicích a orientacích obou zařízení vůči sobě. Kalibrace na menší vzdálenosti fungovala lépe než na větší, to však bylo pravděpodobně dáno malým rozlišením kamery. U této

kamery byl také problém s intenzitou okolního světla a nastavením jasu projektoru. Ty bylo nutné nastavit tak, ať metoda nemá i s nízkým rozlišením kamery problém rozpoznat jednotlivá pole šachovnice a také bity promítaného Grayového kódu. Systém v laboratoři však díky vyššímu rozlišení tímto problémem tolik netrpí.

5.1.2 Kalibrace Kinectu pomocí Calibration service

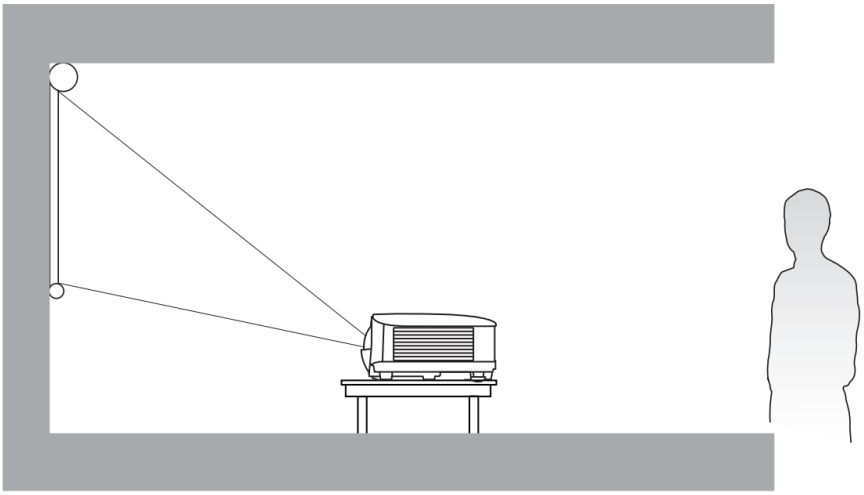
Druhým krokem je kalibrace Kinectu, která má odhadnout jeho pozici a orientaci vůči prostřední ArUco značce. Pro tuto kalibraci jsem naimplementoval skript `get_camera_pose` v jazyce Python. Tento skript použije Kinect pro pořízení snímku pracovní plochy. Snímek poté předá metodě kalibrační služby, zmíněné v kapitole 3.3, která slouží právě pro kalibraci Kinectu a vrací vnější kalibrační parametry a kvalitu kalibrace ve formátu JSON. Kvalita kalibrace záleží na kvalitě pořízeného snímku, ale i na počtu nalezených ArUco značek v něm. Na stole se nacházejí tři a pro nejlepší možné výsledky kalibrace je vhodné, aby se nacházely ve snímku všechny. U této kalibrační metody je nutné, aby měly snímky, sloužící pro výpočet vnitřních parametrů Kinectu, stejné rozlišení jako snímek použitý touto kalibrační metodou. Důvod je ten, že vypočítané vnitřní parametry jsou předávány na vstup metody kalibrační služby a obsahují například optický střed kamery, daný proměnnými c_x a c_y, jejichž jednotky jsou pixely. Při různých rozlišeních pro každou metodu by tedy vzniklo zkreslení při výpočtu pozice a orientace Kinectu.

Přesnost této kalibrační metody při výpočtu pozice se při mých kalibračních pokusech pohybovala v rozmezí maximálně 2 cm. To, spolu s odchylkou kalibrační metody procam-calibration, zmíněnou v kapitole 5.1.1, způsobuje nepřesnost projekce.

5.2 Virtuální reprezentace systému v Unity

Data z kalibračních metod je třeba nějakým způsobem aplikovat. Jednou z možností je vytvořit virtuální protějšek reálného systému v herním enginu Unity. Jako první je vytvořen Canvas, reprezentující plochu pracoviště, na kterou se bude promítat. Canvas je plocha, která se využívá jako místo pro všechny prvky vytvářeného uživatelského rozhraní. Na tuto plochu tedy budu po spuštění umisťovat herní objekty, reprezentující akční objekty, akční body a další. Canvas ale může být nastaven třemi různými způsoby renderování. Je třeba použít nastavení „World Space“, které Canvas umístí jako pevnou plochu do prostoru hry. Díky tomuto nastavení bude plocha bude nehybná bez ohledu na to, kam směřuje kamera, která ji sleduje. Dalšími herními objekty ve scéně jsou dvě kamery, reprezentující Kinect a projektor, ty mají názvy `Kinect` a `Projector`.

Canvas je v Unity scéně nastaven tak, aby jeho střed ležel v bodu $(0, 0, 0)$. Jeho střed tedy reprezentuje střed ArUco značky, vůči které byl zkalibrován systém. Dále je nastavena pozice a orientace `Kinectu`. Tyto hodnoty jsou přečteny z třídy `KinectCalibrationData` a aplikovány skriptem `TransformKinect` při spuštění hry. Tím se poloha Kinectu nastaví tak, jako je i v reálném prostoru. Stejný postup je aplikován i ve skriptu `TransformProjector` pro projektor s třídou `ProjectorCalibrationData`. Jak zmiňuje autor použité kalibrační metody ve svém článku [12], nestačí však aplikovat vnější parametry projektoru. Pokud aplikujeme jen vnější parametry, tak může zorné pole projektoru zabírat špatnou část Canvasu – úhel náklonu jeho komponenty Camera nemusí odpovídat reálnému projektoru. Nemusí to být pravidlem, avšak platí to pro projektor v laboratoři, který má nastaven režim promítání „Stolek vpředu“, kdy je úhel promítání v ose Y projektoru větší než 0°, viz obrázek 5.4. Jak jsem zjistil při kalibraci, tento úhel promítání není započítán ve vnějších



Obrázek 5.4: Nastavení projektoru „Stolek vpředu“, které zvětší úhel promítání na ose Y. Převzato z uživatelského manuálu projektoru BenQ MH733 [1].

parametrech, konkrétně v rotační matici. Úhel je třeba v Unity vytvořit pomocí vnitřních parametrů. Z nich je vypočítána projekční matice, která se aplikuje na komponentu Camera projektoru. Tato matice byla vytvořena a aplikována pomocí článku [12].

Vytvořený virtuální systém lze vidět na obrázku 5.5.

5.3 Uživatelské rozhraní

Uživatelské rozhraní je implementováno v herním enginu Unity. Byla pro něj vytvořena scéna, sestávající ze tří herních objektů, `Kinect`, `Projector` a `Canvas`. Poloha těchto objektů byla nastavena podle těch reálných a `Projector` slouží jako hlavní kamera, sledující plochu `Canvasu`, na kterém se za běhu aplikace budou vykreslovat všechny virtuální objekty, které mají být uživateli zobrazeny. Při implementaci vzhledu rozhraní jsem se řídil návrhem z kapitoly 4.2.

5.3.1 Prefaby

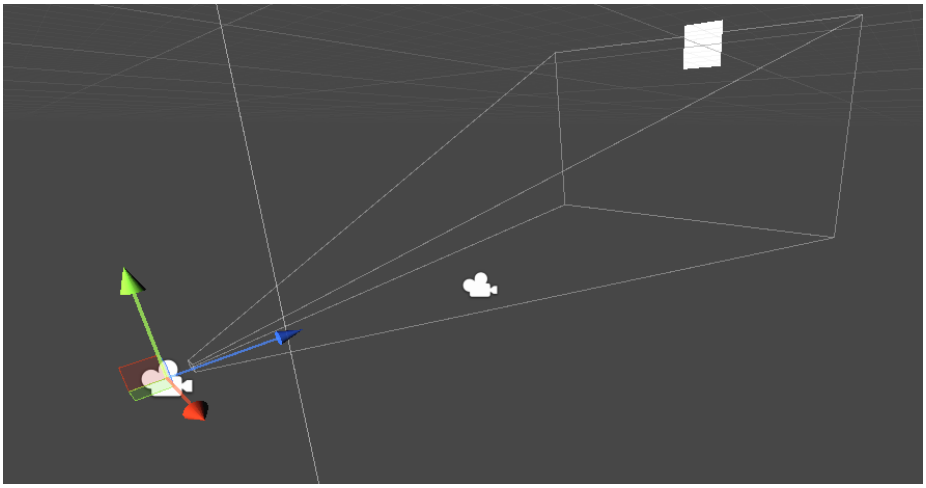
Jelikož je některé herní objekty v Unity nutné přidávat do scény až za chodu aplikace, je třeba je mít předem vytvořené. Pro tento účel v Unity existují Prefaby[4]. Jsou to předem vytvořené a nakonfigurované herní objekty, sloužící jako šablona objektu, který se dá přidat do scény za běhu. Prefaby v mé aplikaci využívám pro vytváření všech prvků rozhraní. Jediné objekty, které jsou ve scéně vytvořeny předem, jsou instance Kinectu, projektoru, zobrazovací plochy a prostřední ArUco značky, ta je však při spuštění aplikace nastavena jako neviditelná, jelikož slouží pouze pro kontrolu přesnosti kalibrace.

V mé aplikaci existuje 13 různých prefabů:

ActionTextRun Zobrazuje název akce při spuštěném programu.

ActionPlace Tvoří jej červený kruh a vyznačuje na pracovišti místo, na kterém se akce provádí při spuštěném programu.

[4] `https://docs.unity3d.com/Manual/Prefabs.html`



Obrázek 5.5: Virtuální reprezentace Kinectu, projektoru a pracovní plochy v Unity. Na tyto kamery již byly aplikovány výsledky kalibrace, měly by tudíž odpovídat reálnému systému. Bílý čtverec uprostřed plochy znázorňuje prostřední ArUco značku. Herní objekt `Projector` a jeho zorné pole je zvýrazněno.

ActionPoint Tvoří jej fialový malý kruh a je zobrazován v editoru projektu na místě akčního bodu v AREditoru.

ActionText Zobrazuje název akce pod akčním bodem v editoru projektu.

Cube Žlutý čtverec, reprezentující stejnojmenný virtuální kolizní objekt.

Cylinder Žlutý kruh, reprezentující stejnojmenný virtuální kolizní objekt.

DobotMagician Zelený čtverec, který představuje robotické rameno Dobot Magician.

DobotM1 Modrý obdélník, který představuje průmyslového robota Dobot M1.

RobotRange Kružnice stejné barvy jako robot, kterému náleží. Znázorňuje dosah robotického ramene. Poloměr je pevně dán v kódu rozhraní podle parametrů výrobce tohoto robota.

OpenedInstanceNameText Text, který při otevřené scéně, projektu nebo balíčku zobrazuje jeho název.

SmallInfoText Text, který slouží pro zobrazování různých informací pro uživatele.

Sphere Žlutý kruh, reprezentující stejnojmenný virtuální kolizní objekt.

StateInfoText Zobrazuje stav systému, například jestli je otevřená scéna, projekt, apod.

Všechny prefaby jsou při vytvoření jejich instance potomkem `Canvasu`. Jedinou výjimku tvoří Prefaby, které jsou vytvořeny jako potomek jiného, již existujícího objektu, ty jsou pak jeho potomkem i v herní scéně.

5.3.2 Nastavení pózy a velikosti objektů

Herní objekty v Unity získávají svou pozici a orientaci ze zpráv o změnách, které přicházejí z ARServeru. Není však možné aplikovat všechna získaná data v takovém formátu, v jakém přicházejí.

V AREditoru je pozice reprezentována třídou `Position`, která má údaje o posunu na všech třech osách. V mém rozhraní jsou však herní objekty na Canvasu ve 2D, což znamená, že nedává smysl na ně aplikovat posun na ose Z. Také je třeba jejich posun vůči ArUco značce invertovat.

Orientace je reprezentována třídou `Orientation`. V mém rozhraní je aplikována pouze na ose Z, jelikož by nedávalo smysl otáčet 2D objekty okolo jiné osy. Současné zobrazení objektů je tím pádem limitováno na jednoduché případy a nepodporuje například rotaci objektů kolem os X a Y.

Při nastavování pózy je nutné převést třídy `Position` a `Orientation` na typy, které používá Unity, což je `Vector3` pro pozici a `Quaternion` pro rotaci. Pro aplikaci pozice a orientace jsem vytvořil tyto dvě funkce:

```
private Vector3 AREditorToSARPosition(Position position)
{
   Vector3 convertedPosition = new Vector3();
   convertedPosition.Set(-1 * 10 * (float)position.X,
   -1 * 10 *(float)position.Y, 0.0f);

   return convertedPosition;
}

private Quaternion AREditorToSAROrientation(Orientation orientation)
{
    Quaternion convertedOrientation = new Quaternion();
    convertedOrientation.Set(0f, 0f, (float)orientation.Z,
    (float)orientation.W);

    return convertedOrientation;
}
```

U virtuálních kolizních objektů lze v AREditoru také měnit jejich velikost. Pro tento účel však nelze vytvořit pouze jednu funkci, jelikož každý objekt má svůj typ. Každý typ tedy musí mít svou funkci pro aplikování změny velikosti.

5.3.3 Obsah obrazovky

Obsah obrazovky závisí na aktuálním stavu systému. Návrh toho, jaké prvky budou zobrazovány pro jednotlivé stavy, jsem provedl v kapitole 4.2. Avšak při implementaci nastalo několik změn tohoto návrhu, proto byly některé prvky přidány a jiné odstraněny. Mimo to je však v rozhraní na všech obrazovkách, dle návrhu, zobrazován v pravém dolním rohu text o tom, v jakém stavu se systém nachází. A pokud je otevřena scéna, projekt nebo balíček, tak je zobrazován i jejich název.

Finální verze implementace tedy vypadá pro jednotlivé stavy systému takto:

Hlavní obrazovka

Je na ní zobrazována pouze informace o tom, že je rozhraní v režimu „stand-by“.

Editor scény

Jsou zobrazovány virtuální kolizní objekty pomocí Prefabů `Cube`, `Sphere` a `Cylinder`. Dále byl však přidán Prefab robota Dobot Magician i Dobot M1. Pomocí nich může uživatel zkontrolovat pozici přidaného robota i v reálném prostoru. Okolo robotů je také zobrazována kružnice, znázorňující jejich dosah. Pomocí ní může uživatel lehce zjistit, do jaké vzdálenosti od robota je možné přidávat například akční body. Vzhled rozhraní při otevřeném editoru scény lze vidět na obrázku 5.6.

Editor projektu

Jsou v něm zobrazovány všechny prvky scény, ze které byl vytvořen. Dále jsou dle návrhu zobrazovány akční body. Nejsou však propojeny šipkami, znázorňujícími propojení jejich akcí, protože při větším množství by mohly být matoucí a také by bylo obtížné srozumitelně zobrazit propojení více akcí jediného akčního bodu, kvůli 2D rozhraní. Místo toho jsem se rozhodl zobrazovat jména akcí pod akční bod, kterému náleží. To lze vidět na obrázku 5.7.

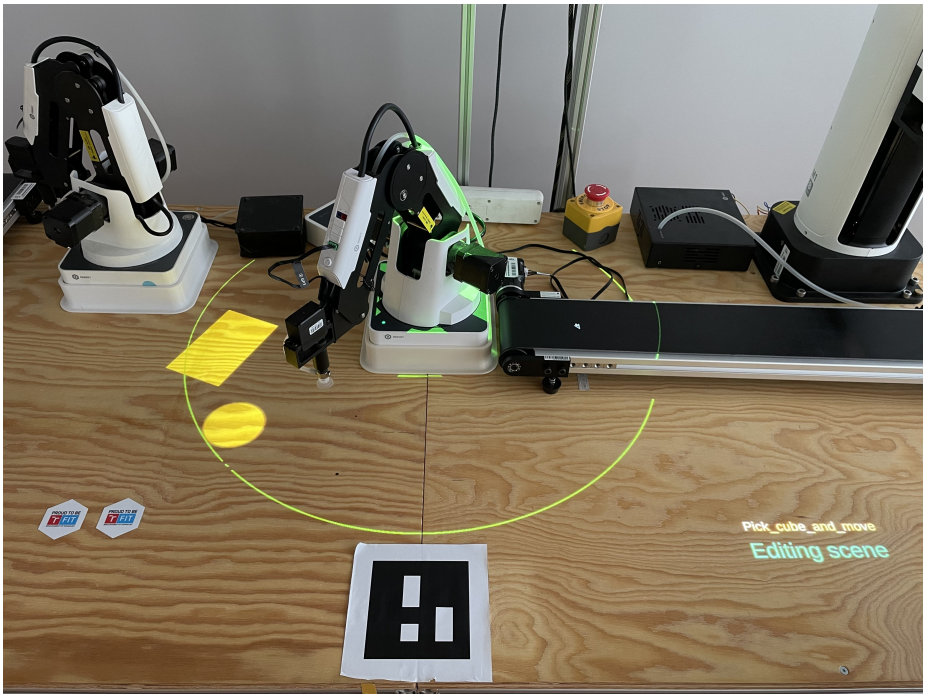
Běh programu

Výsledná implementace se liší od návrhu, jelikož jsem si uvědomil, že při běhu programu pravděpodobně není nutné zobrazovat všechny prvky scény a projektu. Místo toho rozhraní zobrazuje oblast a také jméno akce, kterou robot momentálně provádí. Dále je promítán text, který při zastavení programu uživateli oznamuje, ať vyčká na dokončení akce. Zobrazování této informace mi přišlo vhodné, jelikož při pozastavení či zastavení programu v AREditoru se program nezastaví ihned, ale nejdříve je dokončena aktuální akce. Tuto situaci lze vidět na obrázku 5.8.

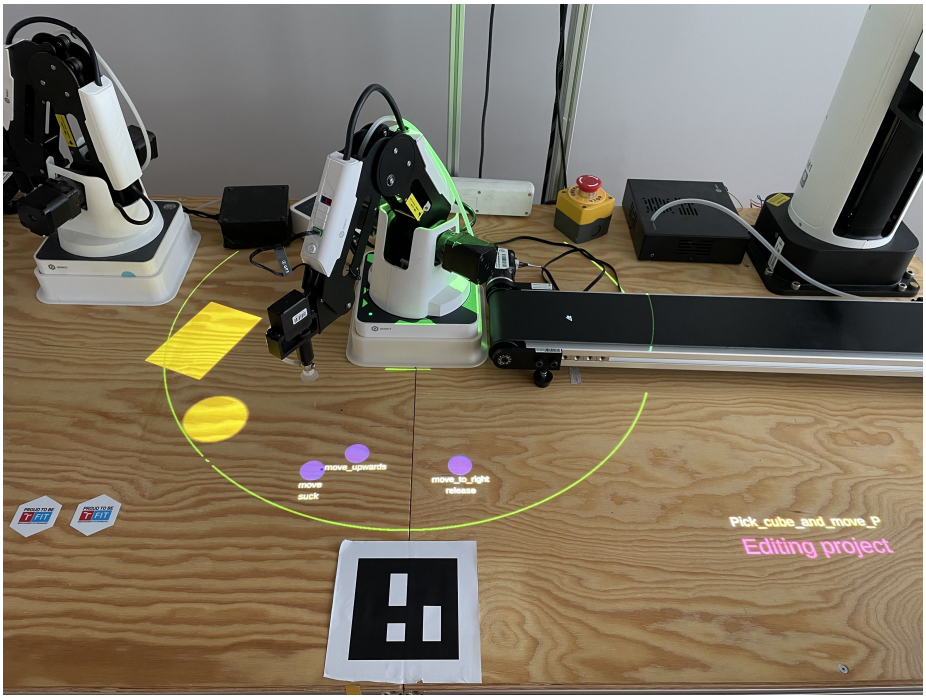
5.3.4 Komunikace s ARServerem a správa rozhraní

Aby rozhraní mohlo přijímat a zobrazovat události, jako přidávání, změnu nebo odstranění objektů, otevření scény, spuštění balíčku a podobně, je nutné, aby byla spuštěná aplikace připojena na ARServer. Ten má pro všechna rozhraní otevřen port pro komunikaci pomocí protokolu WebSocket. ARServer pak všem připojeným rozhraním zasílá události o změnách v systému, jak bylo popsáno v kapitole 3.4.

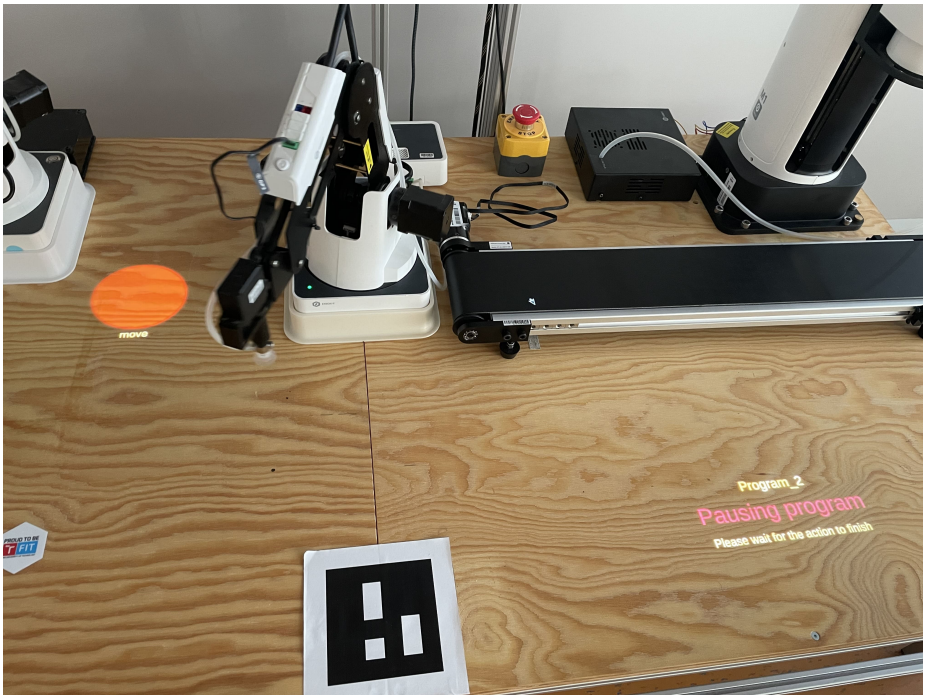
Pro komunikaci s ARServerem v AREditoru slouží třída `WebsocketManager`. Ta obsahuje logiku pro přijímání událostí od serveru a umožňuje je dále zpracovat. Pro zjednodušení implementace jsem tuto třídu převzal z AREditoru a upravil pro potřeby mého rozhraní. Při spuštění aplikace se jako první provolá funkce `ConnectToServer()`, která se pokusí připojit na zadanou adresu a port. Po úspěšném připojení čeká na přicházející zprávy, které rozlišuje podle typu. Zpráva může mít jeden z několika typů, jako příklad může sloužit událost „ShowMainScreen“, která značí, že byla otevřena hlavní obrazovka. Pro každou událost je ve třídě `WebsocketManager` funkce, která ji nějakým způsobem zpracovává. Některé z těchto funkcí jsem upravil tak, aby volaly funkce z mnou vytvořené třídy (také inspirované z řešení AREditoru), `GameManager`.



Obrázek 5.6: Promítané rozhraní při otevřeném editoru scény – dva virtuální kolizní objekty, `cube` a `sphere`, Dobot Magician a jeho znázorněný dosah a také informace o obrazovce v pravém dolním rohu.



Obrázek 5.7: Promítané rozhraní při otevřeném editoru projektu. Ten je vytvořen ze scény z obrázku 5.6. Navíc v něm jsou přidány tři akční body s akcemi, jejichž názvy jsou pod body vypsány.



Obrázek 5.8: Promítané rozhraní při spuštěném programu. Zobrazuje oblast a název akce, která je momentálně vykonávána. Program byl však pozastaven, proto je v pravém dolním rohu uživateli sděleno, že má počkat na dokončení akce.

`GameManager` se stará o správu celého rozhraní v Unity. Jsou v něm vytvářeny instance všech Prefabů, které jsou dále upravovány a odstraňovány z rozhraní podle typů událostí. Obsahuje v sobě tři seznamy:

`actionPoints` Udržuje v sobě všechny akční body projektu.

`sceneObjects` Udržuje v sobě všechny objekty scény.

`objectTypes` Udržuje v sobě všechny typy objektů.

Tyto seznamy slouží pro správu daných objektů při otevřené scéně, projektu nebo balíčku. Pokud se systém přepne do jiného stavu, jsou tyto seznamy upraveny nebo smazány. Například pokud se uživatel vrátí na hlavní obrazovku, tak jsou seznamy smazány a při otevření scény se naplní seznam `sceneObjects`. S tím souvisí i logika vytváření objektů rozhraní za běhu.

O změně stavu systému je rozhraní informováno pomocí těchto zpráv:

ShowMainScreen Je otevřena hlavní obrazovka.

OpenScene Je otevřen editor scény.

OpenProject Je otevřen editor projektu.

PackageInfo Zpráva, nesoucí informace o scéně a projektu při otevření balíčku.

PackageState Změna stavu spuštěného programu, může být `Running`, `Pausing`, `Paused`, `Stopping` nebo `Stopped`. Ty určují, jestli je zrovna program spuštěn, pozastaven nebo zastaven.

SceneClosed Scéna byla uzavřena.

ProjectClosed Projekt byl uzavřen.

Pokud rozhraní obdrží událost „OpenScene“, „OpenProject“ nebo „PackageInfo“, tak od ní získá data s popisem scény, projektu nebo balíčku a jaké prvky obsahují. Tato data jsou zasílána ve formátu JSON spolu s těmito událostmi a ve třídě `GameManager` jsou pak zpracována. Třída v sobě obsahuje funkce `SpawnSceneInGame()` a `SpawnProjectInGame()`, které projdou každý prvek scény nebo projektu a pokud mají být zobrazeny, tak je vytvořena instance jejich herního objektu z příslušného Prefabu. Tím jsou přidány do rozhraní. Při přidání do scény jsou však také doplněny do příslušného seznamu prvků, například `actionPoints`. Většina těchto prvků také obsahuje svou pozici a orientaci, které jsou pomocí funkcí, popsaných v kapitole 5.3.2, aplikovány na jejich herní objekty na `Canvasu`. Každému hernímu objektu je také přiděleno ID prvku, ze kterého byl vytvořen, jako jeho jméno.

Při získání události typu „SceneClosed“, „ProjectClosed“ nebo „PackageState“ se změnou stavu na `Stopped` provede aplikace smazání všech prvků příslušného seznamu a také odstraní všechny prvky rozhraní s tagem „Prefab“. Tag v Unity umožňuje odlišovat různé typy herních objektů a „Prefab“ je tag, přidělován všem Prefabům objektů scény, akčním bodům a jejich akcím. To zajistí, že v rozhraní například po zavření projektu zůstane pouze text s informací o stavu systému.

V případě, že už je scéna či projekt otevřen a je v AREditoru provedena změna, nezasílá server opět informace o všech prvcích, ale pouze o prvku, jehož se změna týká. V tomto případě se prvek nejprve nalezne v seznamu podle jeho ID a je upraven nebo odstraněn. Poté se upraví i v rozhraní.

Je třeba zvlášť zmínit implementaci úpravy rozměrů virtuálních kolizních objektů. Při úpravě jejich pozice nebo orientace není problém, jelikož data s pózou těchto objektů přicházejí spolu s událostí o změně jejich akčního objektu. Implementace změny jejich rozměrů je však složitější, vyžaduje čtení události „ChangedObjectTypes“. Jelikož je akční objekt, jako například virtuální kolizní objekt, instancí typu objektu, přichází při jeho změnách souběžně události „ChangedObjectTypes“ a „SceneObjectChanged“. Změna pozice přichází v události, týkající se objektů scény, zatímco změna rozměrů v události o změně typu objektu. V mém rozhraní jsou však kolizní objekty ve scéně reprezentovány pouze seznamem `sceneObjects` a ty v sobě nemají informaci o velikosti kolizních objektů. Vytvořil jsem proto druhý seznam, `objectTypes`, do kterého jsou při otevření scény načteny všechny typy objektů v systému pomocí funkce `GetObjectTypes()` z třídy `WebsocketManager`. Když tedy přijde událost o změně typu objektu, je podle jeho atributu `Type` nalezen příslušný akční objekt v seznamu a podle něj pak herní objekt, jehož rozměry jsou upraveny podle dat příchozí události.

Kapitola 6

Testování uživatelského rozhraní

V této kapitole je popsán návrh a průběh testování promítaného rozhraní. Testování proběhlo v laboratoři, ve které byla celá práce navržena i implementována. Byla však otestována pouze funkčnost samotného rozhraní. Kalibrace Kinectu a projektoru byla z testování vynechána, jelikož by ji běžný uživatel rozhraní, jako například obsluha robotického pracoviště, neměl provádět. Také by nemělo být třeba ji provádět často. Dá se totiž předpokládat, že se nebude mnohokrát měnit poloha projektoru vůči pracovišti.

6.1 Pilotní testování

Před navržením testování bylo provedeno pilotní testování rozhraní s jedním testovacím subjektem. Tento subjekt neměl žádné předchozí zkušenosti s aplikací AREditor, musel jsem jej tedy nejdříve do používání uvést. Při tomto pilotním testování jsem si pouze zapisoval poznámky subjektu a úspěšnost jeho kroků, jelikož pro něj nebyl vytvořen žádný dotazník ani jakákoliv míra úspěšnosti testu.

Subjekt měl za úkol vytvořit scénu, projekt a balíček, který poté spustil. Při tom mohl používat promítané rozhraní dle jeho uvážení a dávat mi zpětnou vazbu na užitečnost informací, promítaných v mém rozhraní.

Vytváření scény Vytvoření scény s jediným robotem proběhlo bez problémů, subjekt neměl připomínky. Promítané rozhraní používal pro zarovnání virtuálního modelu robota s jeho reálným protějškem.

Vytváření projektu Vyskytly se nějaké problémy, ty však byly způsobeny neznalostí aplikace AREditor. Subjekt promítané rozhraní využíval v malé míře, většinu času sledoval obrazovku tabletu. Nebyl si vědom toho, že kružnice okolo robota znázorňuje jeho dosah, myslel si, že jde pouze o nějaké zvýraznění jeho pozice. Také mu přišly zbytečné názvy akcí pod akčními body.

Spuštění programu Při spuštění programu se vyskytla chyba, protože jeden z akčních bodů byl umístěn mimo dosah robota. Šlo však o dosah na ose Z, spíše než na ose X, promítaná kružnice dosahu tomu tedy nemohla předejít. Po opravení projektu a opětovném spuštění programu se již spustil bez problému. Subjekt sledoval pouze promítané rozhraní bez tabletu a ocenil na něm promítání oblasti, na které robot aktuálně pracuje. Subjekt po zkušenosti se špatně umístěným akčním bodem navrhl kromě dosahu zobrazovat také varovnou hlášku, pokud uživatel umístí akční bod mimo dosah.

Z výsledků tohoto testování jsem vyvodil, že by bylo vhodné otestovat promítané rozhraní na subjektech, kteří mají předchozí zkušenost s AREditorem i na subjektech bez zkušenosti. Subjekt s předchozí zkušeností bude testovat převážně mé rozhraní, spíše než AREditor, což je ostatně i mým cílem. Je však důležitý i pohled subjektů bez znalosti programování robotů, jelikož mohou odhalit nedostatky, kterých bych si díky znalosti práce s aplikací AREditor nemusel všimnout. Jako například návrh subjektu se zobrazováním varovné hlášky.

6.2 Návrh

Testování bylo navrženo po pilotním testu. Jak jsem již zmínil, nebude testována kalibrace systému, ale pouze promítané rozhraní. Cílem má být ověření toho, zda je promítané rozhraní vhodným doplňkem k programování robota v AREditoru, ale také k následnému spuštění programu a spolupráci s robotem.

Vytvořil jsem tedy dva testovací scénáře. První má jako cíl otestovat využitelnost promítaného rozhraní při vizuálním programování v aplikaci AREditor. Druhý scénář má otestovat, jestli promítání na oblasti, kam se robotické rameno bude hýbat, ulehčí uživateli manuální práci v blízkosti tohoto robota.

Po dokončení těchto scénářů vyplní subjekt dotazník týkající se testování a bude dotázán na další případné poznámky nebo nápady k testování.

6.2.1 První testovací scénář

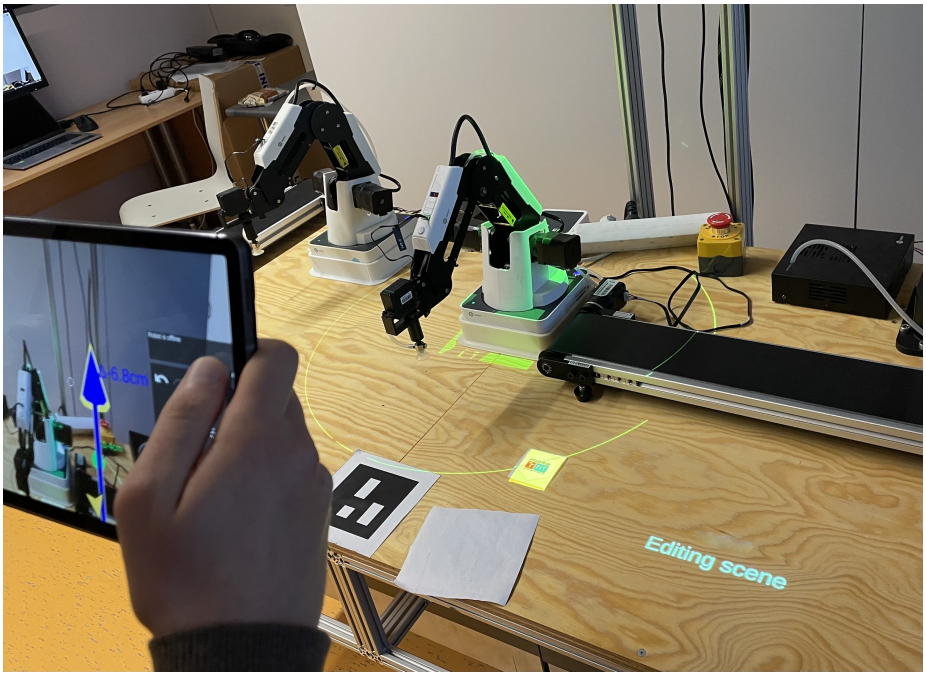
Testuje užitečnost promítaného rozhraní při programování robota v AREditoru. Testovací subjekty při něm nebudou nuceny ke sledování tohoto rozhraní, mohou jej využít dle vlastního uvážení.

Jako první musí subjekt vytvořit novou scénu, do které přidá robotické rameno Dobot Magician a jeden virtuální kolizní objekt typu „Cube“. Kolizní objekt musí být umístěn a zmenšen na předem nachystanou značku na pracovišti tak, aby přibližně kopíroval její tvar. Z této scény poté vytvoří projekt. V tomto projektu dostanou za úkol vytvořit jednoduchý program pro robota. Úkol sestává ze 4 akcí robota: dojet ramenem na pěnovou kostku, nasát ji a dopravit na jiné místo, kde ji následně pustí. Průběh prvního scénáře lze vidět na obrázku 6.1.

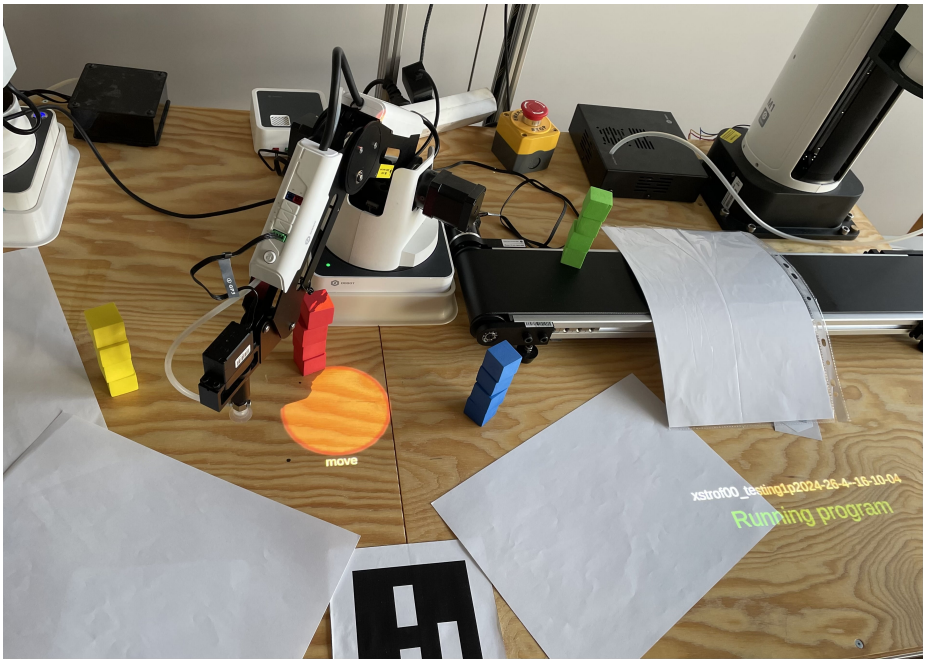
6.2.2 Druhý testovací scénář

Tento scénář slouží pro zjištění, zda je promítané rozhraní využitelné i při práci na robotickém pracovišti, pokud zrovna uživatel nezná chod programu, například spolupráci pracovníka a robota ve firmě. Subjekt tedy nebude seznámen s programem robota, který bude spuštěn a bude muset provádět jednoduchou práci vedle robota.

Úkolem subjektu bude v blízkosti robota postavit čtyři „věže“ z kostek, zatímco robot bude na začátku spuštěn a bude provádět pouze pohyby po pracovišti. Robot však při pohybu nesmí postavené „věže“ shodit, subjekt je má dovoleno přesouvat. Nesmí je však stavět mimo dosah robota. Tento test proběhne dvakrát. Při prvním běhu bude vypnuto promítané rozhraní, zatímco při druhém běhu bude zapnuto, aby bylo zjištěno, zda je splnění úkolu jednodušší se zapnutým promítáním. Splnění druhého scénáře je vidět na obrázku 6.2.



Obrázek 6.1: Průběh prvního testovacího scénáře. Subjekt přidává do scény virtuální kolizní objekt. Byl rozmazán název scény kvůli výskytu příjmení subjektu.



Obrázek 6.2: Průběh druhého testovacího scénáře. Pomocí papírů jsou vymezeny hranice toho, kam může účastník testu pokládat kostky.

6.2.3 Dotazník

Po skončení druhého scénáře vyplní subjekt jednoduchý dotazník. Ten se bude ptát na jeho názor k oběma scénářům dohromady, přičemž dostane ke každé otázce výběr z odpovědí na škále. Otázky v dotazníku budou následující:

- Myslíte si, že je promítané rozhraní vhodné doplnění k vizuálnímu programování?
- Jak často jste využívali promítané rozhraní při vytváření scény?
- Jak často jste využívali promítané rozhraní při vytváření projektu?
- Obtěžovalo vás promítání při jakémkoliv úkolu?
- Bylo splnění druhého úkolu jednodušší s promítaným rozhraním?

6.3 Průběh a výsledky

Testování proběhlo se čtyřmi subjekty. Z nich dva měli zkušenost s aplikací AREditor a dva neměli žádnou předchozí zkušenost, jeden z nich však měl znalost programování robota Dobot Magician pomocí kódu, viz tabulka 6.1.

Označení subjektu	Znalost AREditoru	Znalost programování robotů
1	Ne	Ne
2	Ano	Ano
3	Ne	Ano
4	Částečná	Částečná

Tabulka 6.1: Testovací subjekty

6.3.1 Seznámení subjektů s testováním

Jako první byl každý subjekt obeznámen s tématem mé práce. Těm, kteří neznali vizuální programování, jsem také vysvětlil způsob vytváření scény, projektu a spouštění programu přímo v AREditoru. Po vysvětlení jsem je seznámil s průběhem testování. Jako první jsem je obeznámil s testovacím scénářem, který se týká programování scény a projektu. Poté již mohly subjekty samy začít programovat a při tom libovolně používat i promítané rozhraní. Pokud kvůli neznalosti aplikace subjekt nevěděl jak postupovat, tak jsem mu vysvětlil, jak daný krok provést. Po úspěšném splnění prvního scénáře jsem je seznámil s průběhem druhého scénáře. Po provedení druhého scénáře vyplnily subjekty zmíněný dotazník.

6.3.2 První scénář

Většina subjektů při prvním testovacím scénáři ocenila zobrazení pozice robota i virtuálního kolizního objektu přímo na pracovišti. Subjekt 2 nepoznal, že kružnice okolo robota znázorňuje jeho dosah, i přes jeho znalost tohoto robotického ramene. Subjekt 3 se příliš nevěnoval promítanému rozhraní, to však podle něj bylo způsobeno nutností se soustředit na programování v AREditoru. Subjekt 3 si také nebyl jistý, jestli se řídit promítaným nebo mobilním rozhraním při nastavování pozice robota. To bylo dáno i tím, že obě rozhraní nejsou úplně přesná, takže ukazují pozici s odchylkou.

6.3.3 Druhý scénář

U druhého testovacího scénáře jsem kromě poznámek zapisoval i počty shozených věží. Ty lze vidět v tabulce 6.2:

Subjekt	Shozené věže (bez rozhraní)	Shozené věže (s rozhraním)
1	2	2
2	2	0
3	0	1
4	3	4

Tabulka 6.2: Počty shozených věží ve druhém scénáři

I přes to, že u dvou subjektů byl počet robotem shozených věží větší s promítaných rozhraním, všichni účastníci jednoznačně sdělili, že s promítaným rozhraním bylo splnění úkolu jednodušší. Tyto nejednoznačné výsledky mohou také být dány malým počtem věží a rozlohou pracoviště, které mohli využívat nebo malým počtem účastníků. Někteří účastníci také stavění věží s promítaným rozhraním pojali trochu více odvážně a schválně umisťovali věže například na okraj vyznačené oblasti pohybu robota, což také ovlivnilo výsledky. Takovému chování jsem však mohl předejít větším zdůrazněním toho, že se musí vyhnout shození jakékoliv věže.

Všechny subjekty navrhly přidání více prvků pro zobrazení pohybu robota pro varování uživatele. Oba subjekty, 1 i 2, navrhly promítání místa provádění aktuální, ale i následující akce, aby uživatel znal pohyb robota ještě dříve. Subjekty 3 a 4 zmínily, že by bylo vhodné promítat kromě místa akce i trasu, po které se tam robotické rameno dostane. To by jim mohlo pomoci při odhadování, zda rameno zasáhne věž při přesunu.

6.3.4 Výsledky

Po splnění obou scénářů byly subjekty požádány o vyplnění dotazníku, představeného v kapitole 6.2.3. Odpovědi na dotazník jsou v tabulce 6.3.

Všechny subjekty na první otázku odpověděly „Ano“, jediný subjekt 3 tak neodpověděl a také zmínil, že brýle pro rozšířenou realitu jsou podle něj vhodnější doplněk k vizuálnímu programování. Z odpovědí na dotazník také vyplývá, že většina subjektů využívala rozhraní při vytváření scény, avšak při editaci projektu už tolik ne. Jediný subjekt 4 byl názoru, že mu promítané rozhraní velmi pomohlo i při přidávání akčních bodů a akcí. Na čtvrtou otázku všechny subjekty odpověděly negativně, rozhraní jim nijak nepřekáželo ani nepůsobilo rušivě, například velkým počtem prvků. A jak jsem již zmínil v předchozí kapitole, všechny subjekty se shodly na užitečnosti mého rozhraní při vykonávání druhého scénáře. Subjekt 4 při vyplňování dotazníku také ocenil promítání informace o stavu systému, mohlo by podle něj být využito při více uživatelích u jednoho pracoviště.

6.4 Zhodnocení a plány do budoucna

Vzhledem k tomu, že subjekty neměly žádné větší výhrady k mému rozhraní a většina z nich rozhraní ocenila při programování i při práci u robotického ramene, považuji výsledky testování za uspokojivé. Pouze někdy měly subjekty výhrady k některým prvkům rozhraní, jako například příliš velký rozměr kruhu, zvýrazňujícího oblast akce při běžícím programu.

Otázka	Subjekt 1	Subjekt 2	Subjekt 3	Subjekt 4
Myslíte si, že je promítané rozhraní vhodné doplnění k vizuálnímu programování?	Ano	Ano	Spíše ano	Ano
Jak často jste využívali promítané rozhraní při vytváření scény?	Občas	Občas	Občas	Velmi často
Jak často jste využívali promítané rozhraní při vytváření projektu?	Skoro nikdy	Skoro nikdy	Skoro nikdy	Velmi často
Obtěžovalo vás promítání při jakémkoliv úkolu?	Ne	Ne	Ne	Ne
Bylo splnění druhého úkolu jednodušší s promítaným rozhraním?	Ano	Ano	Ano	Ano

Tabulka 6.3: Odpovědi na dotazník.

Nejméně využito bylo rozhraní při editaci projektu. Při editaci scény bylo využito více, avšak největší využití nachází pravděpodobně při spuštěném programu. Text pro zobrazování stavu také nebyl skoro využit, avšak ten má sloužit spíše pro více uživatelů najednou, jak zmínil i subjekt 4.

Bylo by vhodné zaměřit se do budoucna spíše na využití rozhraní pro spuštěný program než pro samotné programování. Při programování bude uživatel spíše sledovat obrazovku tabletu než přímo pracoviště a aktuální stav implementace promítaného rozhraní pro kontrolu naprogramované scény je dostačující.

Také není dostatečná míra znázornění směru pohybu robota při běhu, jak zmínily některé subjekty u druhého scénáře. Bylo by tedy užitečné doplnit promítání trasy robota. To by však vyžadovalo složitější implementaci, jelikož není možné od ARServeru získat přesnou trasu, po které se robot bude přesouvat. Musela by se tedy pravděpodobně počítat. Jednodušší, avšak neúplné řešení by mohlo být přidání rovné šipky, propojující aktuální a následující akci. To by však nesplnilo účel, který by splnila trasa, ale pomohlo by uživateli mít větší obezřetnost při pohybu robota. Samotná oblast akce může být lehce přehlédnuta v nepřehledném pracovišti nebo za špatných světelných podmínek.

Určitě by bylo užitečné implementovat i varovnou hlášku, že uživatel přidal akční bod mimo dosah, jak zmínil subjekt při pilotním testování. Doplnilo by se tím zobrazení samotného dosahu o další varovný prvek rozhraní. Jako poměrně užitečný nápad od tohoto subjektu mi přišlo i promítání na místo, nad kterým se momentálně nachází konec ramene robota. Uživatel by pak mohl při manuálním pohybu s ramenem vědět, nad kterým bodem pracoviště se přesně nachází. Avšak to by vyžadovalo přesnější kalibraci projektoru.

Kapitola 7

Závěr

Cílem této práce bylo navrhnout a implementovat uživatelské rozhraní pomocí promítané rozšířené reality, které má vhodným způsobem doplnit aplikaci AREditor při programování i běhu vytvořeného programu robota. Tento cíl se mi podařilo splnit.

Jako první jsem provedl rešerši prací, které využívají projekci pro robotická pracoviště a také rešerši metod pro kalibraci kamery a projektoru. Dále jsem se seznámil se systémem ARCOR2 a s jeho uživatelským rozhraním AREditor, nejdříve pomocí jejich dokumentace, později i prakticky. Po seznámení s nimi jsem zvážil možnosti využití projekce a na základě toho provedl návrh promítaného uživatelského rozhraní. Následně jsem vybral použitelnou kalibrační metodu, kterou jsem vhodně upravil a použil pro kalibraci Kinectu a projektoru. Dále jsem podle návrhu naimplementoval promítané uživatelské rozhraní v herním engine Unity, které komunikuje se serverem systému ARCOR2 a zobrazuje uživateli užitečné prvky programu, vytvořeného v rozhraní AREditor. Na závěr bylo toto uživatelské rozhraní ověřeno ve dvou testovacích scénářích, přičemž každý z nich se zaměřoval na jiné využití tohoto rozhraní. Zhodnotil jsem výsledky testování pro zjištění jeho výhod i nevýhod a na základě toho navrhl možná vylepšení do budoucna. Zdrojové kódy a dokumentaci jsem publikoval na GitHub a vytvořil jsem video, prezentující moji práci.

Podařilo se mi tedy navrhnout a implementovat promítané uživatelské rozhraní. To poskytuje uživateli, který používá aplikaci AREditor pro vizuální programování, alternativní vizualizaci programu přímo v pracovišti. Díky tomu se může uživatel lépe orientovat při umisťování objektů programu, sledováním jejich pozice v reálném prostoru. Rozhraní může také pomoci alespoň částečně vizualizovat vytvářený program pro uživatele, kteří nesledují dění v aplikaci AREditor. Lze jej využít i pro zlepšení spolupráce robota a uživatele, který nezná chod jeho programu. Nevýhodou rozhraní je však nepřesnost projekce a ne úplně přesná vizualizace virtuálních kolizních objektů. Při testování byl také zjištěn nedostatek prvků, varujících uživatele před pohybem robota při vykonávání akce.

Do budoucna by tedy bylo vhodné zaměřit se více na vývoj rozhraní pro využití při spuštěném programu, například přidáním více informací o dění na pracovišti, třeba další ukazatel plánovaného pohybu robota. Dále by bylo vhodné zlepšení přesnosti projekce. Jednou z možností by mohla být aplikace zjištěných koeficientů zkreslení projektoru. Druhou možností je výběr a použití jiné kalibrační metody. Dalším zlepšením rozhraní by mohla být také pozice informací o stavu systému nastavená automaticky pomocí Kinectu. Ten by na pracovišti pomocí snímku mohl nalézt vhodné místo bez překážek, které by jinak aktuální, pevně umístěný text, mohly zkreslit.

Literatura

[1] *MX731/MW732/MH733 Digital Projector User Manual*. BenQ Corporation, 2017. 12 s.

[2] Aleger Global. *Oblasti použití rozšířené reality: Od údržby k novému zážitku z nakupování* [online]. [cit. 2024-04-01]. Dostupné z: https://alegerglobal.com/cs/rozsirena-realita/oblasti-pouziti/.

[3] Arena, F., Collotta, M., Pau, G. a Termine, F. An Overview of Augmented Reality. *Computers*. 2022, sv. 11, č. 2. DOI: 10.3390/computers11020028. ISSN 2073-431X. Dostupné z: https://www.mdpi.com/2073-431X/11/2/28.

[4] Aschenbrenner, D., Rojkov, M., Leutert, F., Verlinden, J., Lukosch, S. et al. Comparing Different Augmented Reality Support Applications for Cooperative Repair of an Industrial Robot. In: *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*. 2018, s. 69–74. DOI: 10.1109/ISMAR-Adjunct.2018.00036.

[5] Carmigniani, J. a Furht, B. Augmented Reality: An Overview. In: Furht, B., ed. *Handbook of Augmented Reality*. New York, NY: Springer New York, 2011, s. 3–46. ISBN 978-1-4614-0064-6. Dostupné z: https://doi.org/10.1007/978-1-4614-0064-6_1.

[6] Chadalavada, R. T., Andreasson, H., Krug, R. a Lilienthal, A. J. That's on my mind! robot to human intention communication through on-board projection on shared floor space. In: *2015 European Conference on Mobile Robots (ECMR)*. 2015, s. 1–6. DOI: 10.1109/ECMR.2015.7403771.

[7] Coovert, M. D., Lee, T., Shindev, I. a Sun, Y. Spatial augmented reality as a method for a mobile robot to communicate intended movement. *Computers in Human Behavior*. 2014, sv. 34, s. 241–248. DOI: https://doi.org/10.1016/j.chb.2014.02.001. ISSN 0747-5632. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0747563214000612.

[8] Dumbre, S. Projection-based AR. [online]. 2023, [cit. 2024-04-01]. Dostupné z: https://medium.com/@sakshi.dumbre31/projection-based-ar-220240721883.

[9] Falcao, G., Hurtos, N. a Massich, J. Plane-based calibration of a projector-camera system. *VIBOT master*. 2008, sv. 9, č. 1, s. 1–12, [cit. 2024-04-23].

[10] Gong, L. L., Ong, S. K. a Nee, A. Y. C. Projection-based Augmented Reality Interface for Robot Grasping Tasks. In: *Proceedings of the 2019 4th International*

Conference on Robotics, Control and Automation. New York, NY, USA: Association for Computing Machinery, 2019, s. 100–104. ICRCA 2019. DOI: 10.1145/3351180.3351204. ISBN 9781450371834. Dostupné z: https://doi.org/10.1145/3351180.3351204.

[11] Jin, Y., Ma, M. a Zhu, Y. A comparison of natural user interface and graphical user interface for narrative in HMD-based augmented reality. *Multimedia tools and applications*. Springer. 2022, sv. 81, č. 4, s. 5795–5826.

[12] Kamino. *Use camera parameters obtained with OpenCV in Unity*. 2018 [cit. 2024-04-29]. Dostupné z: https://kamino.hatenablog.com/entry/unity-import-opencv-camera-params.

[13] Kapinus, M., Materna, Z., Bambušek, D., Beran, V. a Smrž, P. *ARCOR2: Framework for Collaborative End-User Management of Industrial Robotic Workplaces using Augmented Reality*. 2023.

[14] Kimura, M., Mochimaru, M. a Kanade, T. Projector Calibration using Arbitrary Planes and Calibrated Camera. In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. 2007, s. 1–2. DOI: 10.1109/CVPR.2007.383477.

[15] Kurkovsky, S., Koshy, R., Novak, V. a Szul, P. Current issues in handheld augmented reality. In: *2012 International Conference on Communications and Information Technology (ICCIT)*. 2012, s. 68–72. DOI: 10.1109/ICCITechnol.2012.6285844.

[16] Leutert, F., Herrmann, C. a Schilling, K. A Spatial Augmented Reality system for intuitive display of robotic data. In: *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. 2013, s. 179–180. DOI: 10.1109/HRI.2013.6483560.

[17] Mamone, V., Cutolo, F., Condino, S. a Ferrari, V. Projected Augmented Reality to Guide Manual Precision Tasks: An Alternative to Head Mounted Displays. *IEEE Transactions on Human-Machine Systems*. 2022, sv. 52, č. 4, s. 567–577. DOI: 10.1109/THMS.2021.3129715.

[18] Materna, Z., Kapinus, M., Beran, V., Smrž, P. a Zemčík, P. Interactive Spatial Augmented Reality in Collaborative Robot Programming: User Experience Evaluation. In: *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*. 2018, s. 80–87. DOI: 10.1109/ROMAN.2018.8525662.

[19] MathWorks. *What Is Camera Calibration?* [online]. [cit. 2024-04-10]. Dostupné z: https://www.mathworks.com/help/vision/ug/camera-calibration.html.

[20] Moreno, D. a Taubin, G. Simple, Accurate, and Robust Projector-Camera Calibration. In: *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission*. 2012, s. 464–471. DOI: 10.1109/3DIMPVT.2012.77.

[21] Padilla, M. *Pika-Who? How Pokémon Go Confused the Canadian Military* [online]. 2020 [cit. 2024-03-31]. Dostupné z: https://www.nytimes.com/2020/01/01/world/canada/pokemon-go-canada-military.html.

[22] Uva, A. E., Gattullo, M., Manghisi, V. M., Spagnulo, D., Cascella, G. L. et al. Evaluating the effectiveness of spatial augmented reality in smart manufacturing: a solution for manual working stations. *The International Journal of Advanced Manufacturing Technology*. Springer. 2018, sv. 94, s. 509–521.

[23] Yang, L., Normand, J.-M. a Moreau, G. Practical and Precise Projector-Camera Calibration. In: *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 2016, s. 63–70. DOI: 10.1109/ISMAR.2016.22.

[24] Zhang, Y.-J. Camera Calibration. In: *3-D Computer Vision: Principles, Algorithms and Applications*. Singapore: Springer Nature Singapore, 2023, s. 37–65. DOI: 10.1007/978-981-19-7580-6_2. ISBN 978-981-19-7580-6. Dostupné z: https://doi.org/10.1007/978-981-19-7580-6_2.