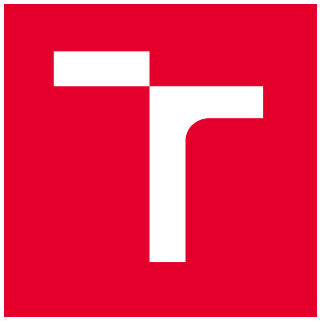


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

IMPLEMENTACE HRY DÁMA S VYUŽITÍM ROBOTICKÉHO MANIPULÁTORU

IMPLEMENTING THE CHECKERS GAME USING A ROBOTIC MANIPULATOR

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. David Lichosyt**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Tomáš Lázna**

SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Kybernetika, automatizace a měření**

Ústav automatizace a měřicí techniky

Student: Bc. David Lichosyt ***ID:*** 203279

Ročník: 2 ***Akademický rok:*** 2021/22

NÁZEV TÉMATU:

Implementace hry dáma s využitím robotického manipulátoru

POKYNY PRO VYPRACOVÁNÍ:

Cílem diplomové práce je implementace deskové hry Dáma s využitím výukového robotického manipulátoru FANUC a platformy ROS. V optimálním případě by měl být robot schopen autonomně hrát proti lidskému soupeři.

1. Proveďte rešerši existujících robotických systémů, na kterých byla implementovaná dáma nebo jiná společenská hra. Uveďte výhody a nevýhody těchto řešení.
2. Popiště možné způsoby ovládání ramene FANUC LR Mate (včetně gripperu) z platformy ROS, zvolený způsob implementujte.
3. Navrhněte koncepci celého systému, popište význam a funkci jednotlivých bloků a způsob jejich vzájemné interakce/komunikace.
4. Vytvořte fyzický model hry (šachovnici a figurky) s ohledem na kinematické parametry robotu.
5. Navrhněte a implementujte metodu pro detekci okamžitého stavu hry.
6. Proveďte rešerši existujících enginů pro hru dáma, jeden zvolte a implementujte jej do systému.
7. Navrhněte, jakým způsobem může lidský soupeř provést tah s ohledem na bezpečnost. Po dohodě s vedoucím zvolené řešení implementujte.
8. Vytvořte algoritmus pro kontrolu dodržování pravidel a navrhněte způsob signalizace jejich porušení.
9. Oživte celý systém a jeho funkčnost doložte videem. Diskutujte možná rozšíření.

DOPORUČENÁ LITERATURA:

[1] MATUSZEK, Cynthia, Brian MAYTON, Roberto AIMI, et al. Gambit: An autonomous chess-playing robotic system. In: 2011 IEEE International Conference on Robotics and Automation [online]. IEEE, 2011, 2011, s. 4291-4297 [cit. 2021-9-13]. DOI: 10.1109/ICRA.2011.5980528

Termín zadání: 7.2.2022 ***Termín odevzdání:*** 18.5.2022

Vedoucí práce: Ing. Tomáš Lázna

doc. Ing. Petr Fiedler, Ph.D.
předseda rady studijního programu

ABSTRAKT

Práce začíná krátkou rešerší již existujících robotických systémů umožňující hru dámy. V dalších částech popisuje možnosti ovládání manipulátoru Fanuc LR Mate 200iD 4S skrze kontrolér R-30iB Mate za pomoci platformy ROS na externím hardwaru. Následuje popis navržených a vyrobených fyzických modelů. Dále se řeší problematika detekce herního stavu v reálném čase, herní logiky a lidského zásahu s ohledem na bezpečnost. V závěru práce se pojednává o kontrole dodržování pravidel, obsluze systému a možných budoucích rozšířeních systému.

KLÍČOVÁ SLOVA

ROS, MoveIt, robotický manipulátor, stacionární robot, Fanuc, LR Mate 200iD/4S, R-30iB Mate, hra dáma, dáma

ABSTRACT

The thesis starts with a field research about existing robotic systems with implemented checkers or similar desk games. In its next parts are described posibilities of controling robotic manipulator Fanuc LR Mate 200iD 4S using controller R-30iB Mate from platform ROS located on external hardware. In few next pages of the thesis there are mentioned designed and used physical models. Following chapters describe real time game situation detection, game logic implementation and human input with safety in mind. At the end of the thesis there are mentioned rule checking algorithms, system operation instructions and possible future upgrades of the system.

KEYWORDS

ROS, MoveIt, robotic manipulator, stationary robot, Fanuc, LR Mate 200iD/4S, R-30iB Mate, checkers game, checkers

LICHOSYT, David. *Implementace hry dáma s využitím robotického manipulátoru*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2022, 82 s. Diplomová práce. Vedoucí práce: Ing. Tomáš Lázna

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. David Lichosyt
VUT ID autora:	203279
Typ práce:	Diplomová práce
Akademický rok:	2021/22
Téma závěrečné práce:	Implementace hry dáma s využitím robotického manipulátoru

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval mému vedoucímu diplomové práce panu Ing. Tomáši Láznovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Úvod

Diplomová práce práce navazuje na výsledky z práce bakalářské, které byly následně zdokonaleny v semestrálním projektu v rámci předmětu Robotika. Čtenář je s těmito základy v následujících dvou odstavcích úvodu seznámen, a je tak uveden do celé problematiky.

Závěrem předcházející bakalářské práce bylo dosaženo vytvoření softwarového balíčku v rámci platformy ROS Melodic Morenia, který uskutečňoval nejen jednoduchou komunikaci mezi jednotlivými uzly výsledného programu, ale i mezi nimi a kontrolérem manipulátoru. Celý systém byl schopen detekce herní plochy, herních kamenů, jejich rozlišení a rozřazení mezi oba hráče. Dále bylo dosaženo jednoduché vizualizace dat ze zmíněného zpracování obrazu. Pohyb figurek byl uskutečněn pomocí konzolového zadávání. Práce avšak nedisponovala správným modelem manipulátoru pro výpočet inverzní kinematické úlohy, a ani modulem pro ovládání gripperu [1].

V rámci projektu z předmětu Robotika byly v návaznosti na bakalářskou práci implementovány oba hlavní chybějící moduly, a to správný kinematický model manipulátoru a modul pro ovládání gripperu. Jako kinematický model manipulátoru byl použit model vytvořený tvůrcem driverů pro komunikaci mezi platformou ROS a kontroléry Fanuc. V tomto bodě se však vyskytl další problém v podobě nefunkčního modulu pro výpočet inverzní kinematické úlohy. Ten byl vyřešen nahrazením stávajícího MoveIt pluginu *IKFast* za alternativní balíček *trac_ik_kinematics_plugin*. Ovládání gripperu bylo zprovozněno za pomoci externího hardwaru v podobě Arduina Nano v3.0 a sériové UART komunikace [2].

Samotná práce je věnována teorii a postupu týkajícího se vývinu nejen softwarového řešení, ale i návrhu hardwarových součástí pro vytvoření autonomního robotického systému pro hru dámy proti lidskému protihráči. Dělí se na dvě hlavní části. První z nich jsou body zadání jedna až čtyři, kde je rozebrána příprava platformy pro samotnou hru. Je zde řešena zejména problematika nejen komunikace jak uvnitř, tak vně platforma ROS, ale i tvorby hardwarových součástí a modelů. V počátku tato část pojednává o rešerši existujících robotických zařízení, na kterých byla implementována dáma či obdobná hra dvou hráčů, a jejich stručný rozbor. Dále se čtenář dozvídá o možných způsobech komunikace mezi platformou ROS a kontrolérem, o různých možnostech ovládání robotického ramene, o volbě řešení a nakonec o postupu jeho implementace. Následující kapitola rozebírá koncepci celého systému, funkce jednotlivých bloků a různé typy užité komunikace mezi nimi. V závěru této části jsou čtvrtou kapitolou myšlenkové pochody za návrhem jednotlivých modelů a fyzických součástí, jejich významy a důvody zvolených designů.

Druhou částí se rozumí body zadání pět až devět, kde se práce dostává ke kon-

krétním řešením hlavních součástí systému, mezi které patří mimo jiné hlavně detekce herní plochy, herní algoritmus, nebo bezpečnost. V počátku je tedy čtenář pátou kapitolou seznámen s detekcí herní plochy v reálném čase, která byla od původní práce změněna od základů. Následně je proveden stručný rozbor již existujících herních algoritmů pro výběr nejvhodnějších tahů počítačem ovládaného hráče hry dáma, kde dále dojde ke zvolení a implementaci jednoho z nich. Další kapitolou je čtenáři přiblížena problematika zpracování lidského zásahu s ohledem na vysokou bezpečnost celého systému. Osmá kapitola pojednává o kontrole dodržování pravidel ze strany lidského protihráče, kde se řeší situace neplatných tahů a následná signalizace uživateli. V poslední kapitole je popsán návod k obsluze systému, který následně přechází v diskuzi návrhů možných budoucích zamýšlených funkcionalit.

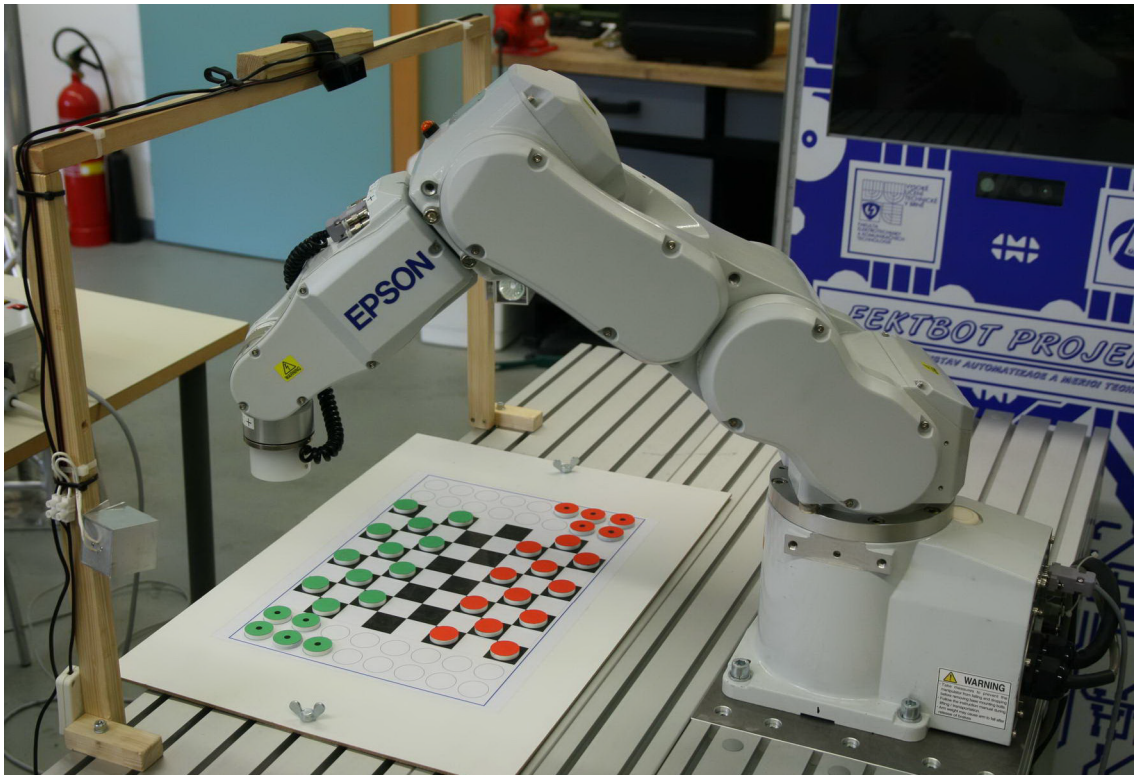
Cílem této práce je osvojit si především metody komunikace nejen v rámci platformy ROS, ale i mezi ní a kontrolérem manipulátoru, naučit se řádně pracovat s knihovnou openCV a vhodně použít její nástroje a využít znalosti nabyté v předchozích prací. Následně všechny tyto dovednosti spojit dohromady a vytvořit reprezentativní systém schopný utkat se s lidským protivníkem v partičce dámy. Hra proti manipulátoru by měla co nejvěrohodněji napodobovat běžnou hru proti lidskému hráči, v ideálním případě by měl lidský protihráč pouze interagovat s herními figurkami a vše ostatní by měl mít na starosti systém Robocheckers.

1 Rešerše robotických deskových her

V této kapitole jsou podrobněji rozebrány již existující robotické systémy, do kterých byla implementována dáma či podobná jiná desková hra. Následně jsou popsány výhody a nevýhody těchto užitých řešení.

1.1 Demonstrační úloha pro robotický manipulátor Epson

Nejprve bych rád zmínil diplomovou práci, která vznikla zde na ústavu automatizační a měřící techniky v roce 2012. Podobně jako tato práce využívá pro pohyb herních kamenů po šachovnici šestiosý robotický manipulátor. Bylo užito robotického ramene výrobce Epson z řady C3 a řídící jednotky RC180. Pro úchop herních kamenů byl robot vybaven přídržným elektromagnetem. Názorná ukázka výsledku práce je ukázána na obrázku 1.1.



Obr. 1.1: Dáma v podání manipulátoru Epson. [3]

Komunikace mezi počítačem a řídící jednotkou je založená na bázi soketového přístupu. Knihovna pro samotné řízení manipulátoru byla vytvořena v prostředí

jazyka C#. Herní plocha je rovněž snímána kamerou, kde potom dochází k detekci a rozpoznávání jednotlivých figurek.

Celá tato podkapitola čerpá ze zdroje [3]

Detekce herní plochy

Nejprve dochází k detekci herního rámu okolo celé herní plochy za pomoci Houghovy transformace. Následně probíhá kolineární transformace, která odstraní perspektivní zkreslení vzniklé zvoleným umístěním kamery. Transformovaný obraz je předložen do mapovací funkce, kde dojde k vytvoření dvou nových obrazů, přičemž na každém z nich dojde ke zvýraznění barev figurek jednoho z hráčů.

Herní algoritmus

Pro vyhodnocování nejvhodnějších tahů ze strany počítače je užito algoritmu minimax s alfa-beta ořezáváním.

Tento algoritmus nejprve přiřadí hodnotu všem možným výsledkům sekvence tahů do určité hloubky (tím se rozumí do určitého počtu po sobě jdoucích tahů) na základě jednoduchých pravidel. Čím je tato hodnota vyšší, tím je tah pro počítačového protivníka výhodnější a naopak. Následně dojde k přiřazení hodnoty každému větvení podle nejnižší hodnoty mezi možnými výsledky zmíněného větvení až k prvnímu tahu. Nakonec je vybrán tah dle nejvyšší hodnoty.

Při užití alfa-beta ořezávání dochází ke snížení časové náročnosti výpočtu běžného minimax algoritmu. Toho je docíleno za pomoci prozkoumávání pouze určitých větví a ořezání ostatních. Počítá se zde s parametry alfa, který nese hodnotu dosud nejnižšího hodnocení, a beta, který naopak uchovává hodnotu dosud nejvyšší. Každý z těchto parametrů představuje nejlepší možný výsledek tahové sekvence právě pro jednoho z hráčů.

Zásah do herní plochy

Jak již bylo zmíněno, herní kameny jsou přesouvány za pomoci šestiosého robotického manipulátoru, který je vybaven nástavcem v podobě přídržného elektromagnetu. Ten je navíc vybaven koncovým spínačem, který zajišťuje zpětnou vazbu v podobě informace, zdali byl herní kámen opravdu uchopen.

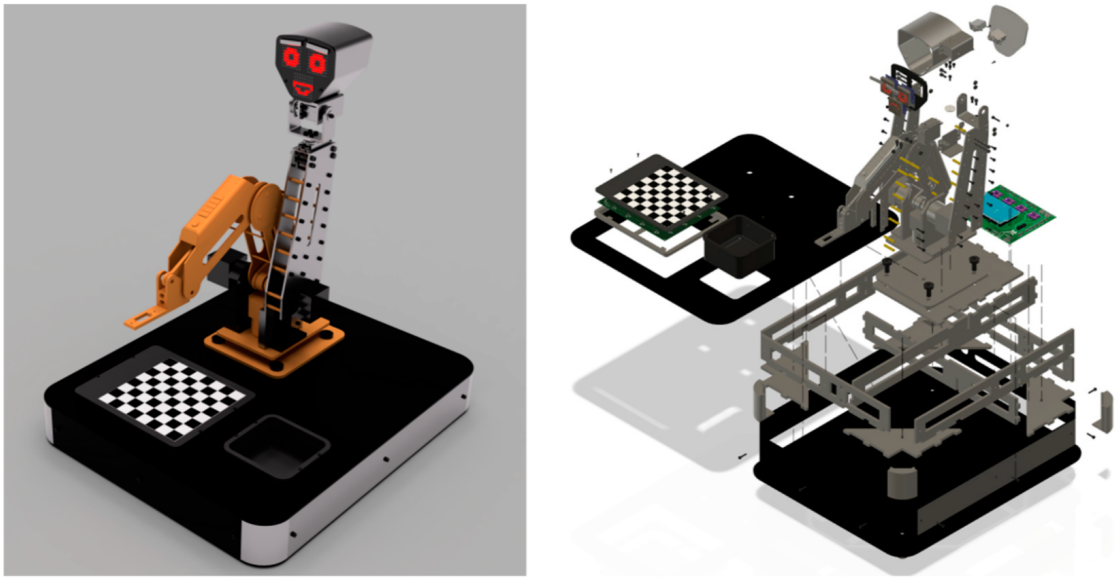
Zhodnocení

Způsob zpracování se jeví jako velmi kvalitní. Jejími silnými stránkami vůči této práci jsou například nízké profily herních kamenů usnadňující jednodušší detekci

herní plochy. Díky úchopového systému s použitím elektromagnetu nedochází ke kolizi nástavce manipulátoru s herním kamenem v případě, že se nachází dále od středu herního pole.

1.2 Interaktivní robot pro hru ruské dámy

V tomto návrhu řešení je opět užito robotického ramene, avšak místo průmyslově vyráběného šestiosého byl zvolen manipulátor tříosý. Zvolený gripper pro úchop a přesun herních kamenů je čistě pneumatický. Manipulátor je ovládán za pomoci mikrokontroléru typu Arduino. Komunikace s ním je zajištěna pomocí sériového rozhraní UART. Celý tento systém lze vidět na obrázku 1.2, kde část *a* představuje 3D model celého systému a *b* podrobněji rozebranou variantu. Rozměr šachovnice je 150x150 mm.



(a) **(b)**

Obr. 1.2: Interaktivní robot pro hru ruské dámy. [4]

Inovativní součástí je LED displej znázorňující aktuální "náladu„ robotického protivníka. Ten má až šest různých stavů. Například při porušení pravidel ze strany lidského hráče imituje hněv a upozorní jej zvukovým efektem. Nebo v momentě, kdy bere soupeřovi jeho figurku imituje výraz štěstí.

Zvolený design je mnohem jednodušší než v případě předešlého systému popsaném v kapitole 1.1. Celá tato podkapitola čerpá ze zdroje [4].

Detekce herní plochy

Pro zjištění aktuálního stavu na šachovnici je užito pole třiceti dvou Hallových sond, které jsou umístěny pod černými poli. Každá figurka má v sobě magnet, přičemž nezávisí na jeho natočení (polaritě). To kvůli tomu, že hráčská příslušnost a hodnost se určuje z předchozího stavu hry. Jedná se tedy o mnohem jednodušší řešení, než v případě užití kamery a zpracování obrazu.

Herní algoritmus

Pro hru bylo užito algoritmu na bázi AlphaZero. Jedná se o umělou inteligenci, která se zdokonaluje hraním hry sama proti sobě bez potřeby jakýchkoliv jiných dat týkajících se hry než samotných pravidel. Vé své podstatě představuje spojení neuronové sítě a Monte Carlo tree search algoritmu.

Neuronová síť f_θ je závislá na parametru θ a jejím vstupem je aktuální stav šachovnice s. Má přitom dva výstupy: spojitou hodnotu stavu šachovnice z perspektivy aktuálního hráče $v_\theta(s) \in [-1, 1]$ a vektor pravděpodobností pro všechny možné tahy $\vec{p}_\theta(s)$.

Při tréninku sítě po každé hře se sebou samou je počítána funkce proher (rovnice 1.1), kde $\vec{\pi}_t$ je vektor vah odpovídající předpokládané výhodnosti tahu ze stavu s_t, $z_t \in -1, 1$ představuje výsledek hry z perspektivy hráče ve stavu s_t.

$$l = \sum_t (v_\theta(s_t) - z_t)^2 - \vec{\pi}_t \cdot \log(\vec{p}_\theta(s_t)) \tag{1.1}$$

Výsledkem po naučení je neuronová síť, která přibližně odhaduje, které kroky vedou k vítězství a které k prohře.

Při známém stavu herní situace s odhadne neuronová síť vektor pravděpodobností $\vec{p}_\theta$. Tyto odhady je zapotřebí ve fázi učení co nejvíce vylepšit. K tomu slouží Monte Carlo tree search algoritmus. Při stromovém vyhledávání představuje každý uzel jedno platné rozložení herní plochy. Pokud je možno jedno takové rozložení změnit jedním tahem na druhé, existuje mezi jejich uzly spojnice. Počínaje prázdným stromem pouze s jedním hlavním uzlem (kořen) dochází k rozšiřování postupně uzel po uzlu až do uzlu konečného (list). V momentě, kdy je vytvořen nový uzel, jeho hodnota je stanovena z neuronové sítě.

Počítá se horní hranice důvěryhodnosti $U(s, a)$ (rovnice 1.2), kde:
$Q(s, a)$ je očekávaný zisk z provedeného tahu a ze stavu s,
$N(s, a)$ značí kolikrát byl tah a proveden ze stavu s v průběhu simulací,
$P(s, \cdot) = \vec{p}_\theta(s)$ je počáteční odhad provedení tahu ze stavu s podle neuronové sítě.

$$U(s, a) = Q(s, a) + c_{puct} \cdot P(s, a) \cdot \frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)} \tag{1.2}$$

Princip AlphaZero algoritmu byl popsán ze zdroje [5].

Zásah do herní plochy

Přesun herních kamenů je zajištěn za pomoci pneumatické pumpy, která při kontaktu gripperu s figurkou vytvoří podtlak. Jedná se o velmi jednoduchou metodu úchopu.

Zhodnocení

Jedná se o nejjednodušší systém ze všech rozebíraných řešení v rámci této kapitoly. Mezi výhody může patřit například vysoká spolehlivost při detekci figurek na herní ploše a naprostá nezávislost na světelných podmínkách. Na druhou stranu nelze určit přesné souřadnice všech figurek v rámci kartézského souřadnicového systému. To ovšem není až takový problém kvůli zvolené metodě přemísťování herních kamenů, která není závislá na přesné poloze, protože při vychýlení figurky od středu pole nemůže dojít ke kolizi s gripperem manipulátoru.

Jako další z výhod bych uvedl využití sofistikovanějšího softwaru pro volbu vhodných tahů ze strany počítačem řízeného hráče. Při správném naučení dokáže dosahovat velmi slibných výsledků.

Inovativní systém pro zobrazování "emocí„ robota je taktéž velice originální idea a tudíž jistou výhodou.

Jedna z nevýhod by mohla být například ne příliš vysoká spolehlivost zařízení pro úchop.

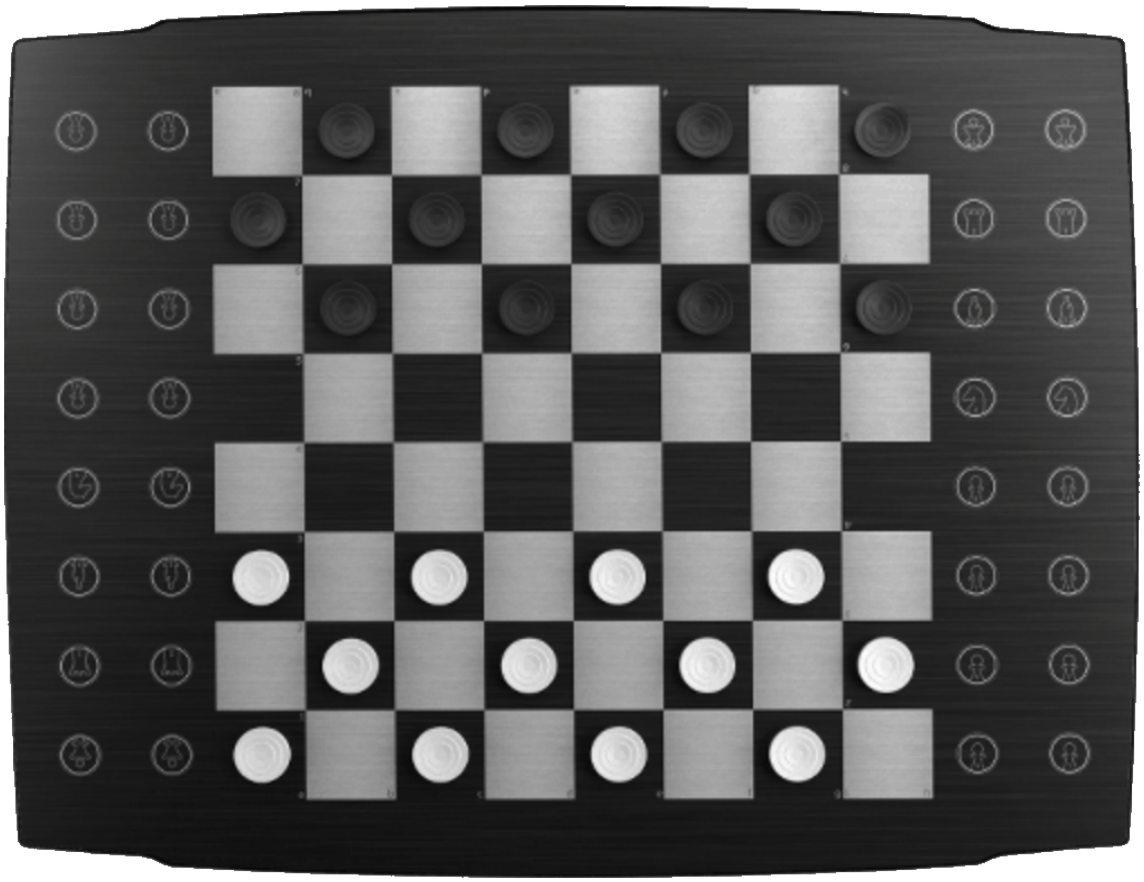
1.3 Square off Swap

Square off je profesionální výrobce na trhu autonomních deskových her. Přestože nejsou jejich výrobky dostatečně po technické stránce zdokumentované, stojí rozhodně za zmínku. V této podkapitole je popsán přesněji jejich výrobek s názvem Swap. Jedná se o šachovnici umožňující hrát hry jako šachy, dáma, connect 4 a halma. Na obrázku 1.3 je ukázána v režimu pro hru dáma.

Podkapitola čerpá ze informace převážně ze zdrojů [6] a [7].

Detekce herní plochy

Pro uskutečnění tahu je nutné vybranou figurkou zatlačit mírně do startovního pole a následně to stejné do pole cílového. Z toho vyplývá, že podobně jako v kapitole 1.2 bylo užito systému, který vyhodnocuje situaci na herní ploše z předchozího rozložení. Po uskutečnění tahu tedy rozezná o jaký tah se jednalo a upraví svůj vnitřní stav herní plochy dle paměti stavu předchozího a tahu právě provedeného.



Obr. 1.3: Square Off Swap šachovnice v režimu dáma. [6]

Zásah do herní plochy

Dle slov spoluzakladatele společnosti Square Off jménem Bhavya Gohil jsou všechny figurky magnetické a pod herní deskou se nachází robotický manipulátor. Ten vždy zajede pod zvolenou figurku, se kterou poté cestuje po hranách polí (aby nedocházelo ke střetům s ostatními figurkami na šachovnici) buď do cílové destinace, nebo při sebrání figurky do speciální pozice na boku šachovnice.

Zhodnocení

Jakožto sériově vyráběný cenově dostupný produkt je tento výrobek znamenité kvality i přesto, že jako chybu lze vytknout občasné kolize při autonomním přesunu figure.

Šachovnice lze propojit s mobilní aplikací za pomoci rozhraní Bluetooth, kde si následně člověk volí obtížnost, barvu a podobně. Výrobek disponuje i možností připojení k internetu a hrát tak s reálnými hráči online.

2 Ovládání robota a vnější komunikace

V této kapitole jsou zmíněny možné způsoby komunikace a ovládání manipulátoru typu Fanuc LR Mate 200iD/4S, čemuž následuje podrobnější rozbor vnější komunikace a zpracování dat na straně kontroléru. Pro správnou funkci programu je zapotřebí, aby na kontroléru manipulátoru byly příslušné programy ve stavu *Running*. Mezi nimi se nachází programy s názvy *ROS_TRAJ*, *ROS_STATE*, *ROBOIO* a *ROS_MOVESM*, které jsou volány hlavním programem ROS. Tyto programy následně komunikují s programy ovladače *fanuc_driver*, který je součástí programového celku uvnitř platformy ROS.

Programy *ROS_TRAJ*, *ROS_STATE* a *ROBOIO* jsou programy psané v jazyce KAREL, který je podobný jazyku Fortran, a jsou spuštěné na pozadí. V popředí je přitom spuštěn program *ROS_MOVESM*, který je typu TP (speciální programy psané na Tech Pendantu manipulátoru). Jedná se o primitivnější jazyk, který se skládá převážně z větvení a následných skoků na různé popisky dle výsledku porovnávání (*if, jump, label, apod.*). Samotný program je určen převážně pro pohyb manipulátoru.

2.1 Způsoby ovládání manipulátoru

Aby bylo možné manipulátor ovládat externě například z platformy ROS, je nejprve zapotřebí zvolit správnou formu komunikace. Jelikož společnost Fanuc dodává svá zařízení v co nejzákladnější podobě a jakékoliv, byť minimální, rozšíření je zapotřebí dokoupit, je nejlepší variantou zvolit komunikaci, kterou kontrolér výukové buňky již disponuje. Tímto odpadají možnosti jako například profinet, nebo modbus a jako nejvhodnější se jeví komunikace pomocí soketů a TCP/IP protokolu.

Pro komunikaci mezi platformou ROS a kontrolérem Fanuc R-30iB Mate již existuje open source ovladač, který je schopen výměny zpráv právě za pomoci soketů a TCP/IP protokolu. Tato varianta byla již užita v rámci bakalářské práce a byla vyhodnocena jako velmi vhodná.

2.2 Vnější komunikace

Vnější komunikací se rozumí komunikace mezi kontrolérem manipulátoru a platformou ROS, která je spuštěná na externím hardwaru s operačním systémem Ubuntu. Tato dvě zařízení musí být propojena pomocí ethernetového kabelu. Komunikace probíhá za pomoci TCP/IP protokolu. Komunikační kanály jsou celkem tři, přičemž všechny mají účastníka typu server na kontroléru a účastníka typu klient na straně

PC. Aby tato komunikace mohla bezproblémově probíhat, je nutností, aby byl kontrolér manipulátoru vybaven rozšířením *User Socket Messaging (R648)* [8].

2.2.1 Komunikační kanály a použité sokety

Jak již bylo zmíněno, jsou nastaveny celkem tři komunikační kanály, přičemž první dva jsou součástí ovladače *fanuc_driver* a třetí byl vytvořen v rámci této práce. U každého z nich je uvedeno označení užitého *Server Tagu* na kontroléru manipulátoru, k němu přiřazený port a stručný popis funkce, kterou plní.

Ovládání pohybu manipulátoru

- Server Tag: S4
- Port: 11000

Data jsou zasílána převážně z platformy ROS do kontroléru. Ta mohou obsahovat například údaje o následující poloze, která se po zpracování ukládá do pozičních registrů.

Informace o poloze manipulátoru

- Server Tag: S5
- Port: 11002

Data se odesílají směrem z kontroléru do platformy ROS. Obsahují informace ohledně aktuální pozice manipulátoru. Ty se následně využívají pro výpočty inverzní kinematické úlohy pro následující pohyb, nebo i pro vizualizaci aktuálních stavů.

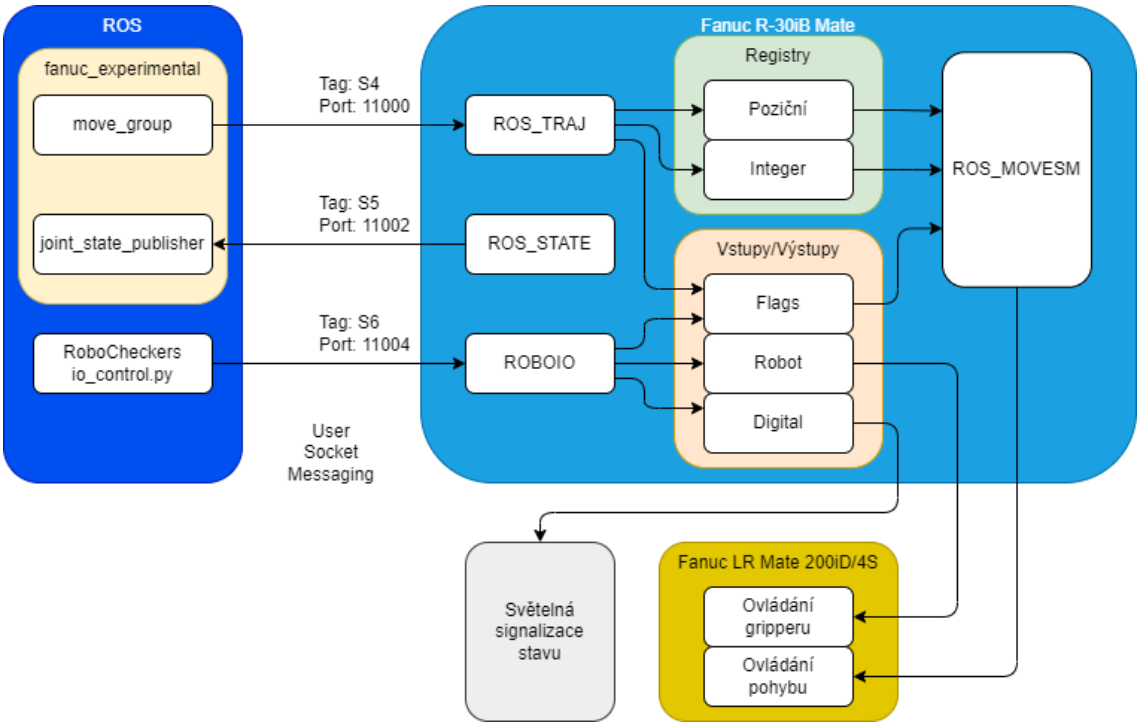
Ovládání IO kontroléru

- Server Tag: S6
- Port: 11004

V tomto případě je opět tok dat uskutečněn směrem z platformy ROS do kontroléru. Každá zpráva obsahuje údaje o typu nastavované periferie, adrese zvoleného registru vybrané periferie a hodnoty, na kterou se nastaví. Zatím je podporováno pouze nastavování binárních registrů periferií.

2.3 Implementovaný způsob ovládání

V této podkapitole je rozepsán způsob, jakým je manipulátor ovládán po přijetí různých typů zpráv. Popsány jsou i postupy, kterými jsou data z jednotlivých komunikačních kanálů zpracována na straně kontroléru. Veškeré toto dění je graficky znázorněno na obrázku 2.1



Obr. 2.1: Ovládání a vnitřní struktura kontroléru.

2.3.1 Ovládání pohybu

Výpočet inverzní kinematické úlohy probíhá uvnitř programových balíčků *fanuc* a *fanuc_experimental*. Proto tímto komunikačním kanálem přicházejí už vypočtené hodnoty natočení jednotlivých kloubů, které se uloží do určených pozičních registrů. Současně s nimi přijdou i data o plánované rychlosti pohybu v procentech, ty se uloží do integer registru.

V tomto okamžiku, pokud je flag *"1: f_msm_rdy"* programem *ROS_MOVESM* nastavený v logické jedničce, je hodnota flagu *"2: f_msm_drdy"* rovněž přenastavena na logickou jedničku. To má za následek splnění čekací podmínky v programu pro pohyb *ROS_MOVESM*. Ten si nejprve data přepíše z původních registrů do registrů, kde má přístup pouze on, aby nedošlo k přepsání dat za běhu programu. Následně vykoná pohyb, a pokud se nevyskytla žádná chyba, program opět zresetuje oba flagy a očekává další instrukce od programu *ROS_TRAJ*.

2.3.2 Odesílání informací o stavu

Odesílaní zpětné vazby je poněkud jednodušší. Program pouze ze systémových proměnných vyčte aktuální polohu manipulátoru, kterou následně pošle zpátky do platformy ROS.

2.3.3 Ovládání vstupů a výstupů

Vývoj softwaru pro ovládání vstupů a výstupů kontroléru manipulátoru zahrnoval nejen vytvoření programu na straně platformy ROS, ale bylo zapotřebí vytvořit programy v jazyce KAREL na straně kontroléru pro zpracování dat, zajistit vzájemnou komunikaci a vytvořit jednoduchý komunikační protokol. Oproti minulému řešení, v podobě užití vývojové desky Arduino Nano, tak bylo dosaženo spolehlivějších výsledků, lepší kompaktnosti celého systému a hlavně odstranění přebytečného hardwaru.

Zpočátku byl software vyvíjen jako balíček pouze pro ovládání gripperu. Po jeho dokončení a úspěšném otestování došlo k jeho vylepšení o možnost ovládání všech možných booleovských vstupů a výstupů (Například digitální IO, Robot IO, Flagy a podobně). Díky tomu bylo možné po výrobě a následném připojení oživit i informační LED panel informující uživatele a obsluhu o aktuálním stavu systému a hry.

Když je navázána komunikace z programu *io_control.py* uvnitř platformy ROS, program *ROBOIO* je schopen přijímat zprávy o změnách vstupů a výstupů. Zpráva ma zpravidla 6 znaků, přičemž první dva znaky definují typ periferie (například pro ovládání gripperu se používá *robot_output* a pro LED panel *digital_output*), další tři určují pořadí registru (jeho adresu) vybrané periferie a poslední znak určuje stav, do kterého se nastaví. Formát zprávy je graficky znázorněn na obrázku 2.2.

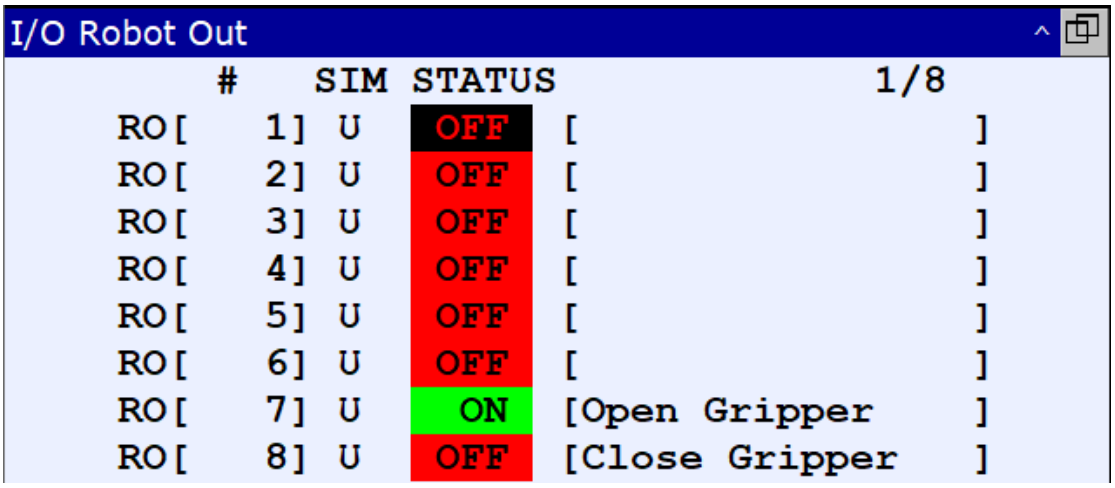
Popis	IO Typ [00-38]		Pořadí v paměti [000-999]			Stav [0-1]
Číslo znaku	0	1	2	3	4	5

Obr. 2.2: Formát zprávy pro ovládání IO.

Ovládání gripperu

Fyzický stav gripperu závisí na hodnotách výstupů typu *Robot* na adresách 7 a 8. Tyto dva výstupy jsou nastaveny jako komplementární. To má za následek, že v každém případě musí vždy mít opačnou hodnotu. V momentě, kdy upravíme jakýkoliv z nich, ten druhý se automaticky překlopí na opačnou hodnotu.

Jak lze vyčíst z obrázku 2.3, aktivní výstup č. 7 gripper rozevírá, přičemž výstup č. 8 gripper zavírá. Gripper je tedy ovládán zasláním zprávy, která ovládá *Robot Digital Output* č. 8.



Obr. 2.3: Výstupy typu *Robot*.

Při zaslání logické jedničky dochází k úchopu a naopak při zaslání logické nuly dochází k uvolnění figurky. Každý volený vstup a výstup je reprezentován číselným kódem. V našem případě to je *09*. Seznam všech možných vstupů a výstupů je sepsán v knihovně *kl_io_lib.py*.

Ovládání informačního LED panelu

Celá sekce ovládání informačního LED panelu čerpá ze zdroje [9].

V kontroléru jsou k dispozici celkem dva vstupně výstupní konektory, *CRMA58* a *CRMA59*, které jsou napojené na vnitřní sběrnice *CRMA15* a *CRMA16*. Pro napojení informačního panelu byl vybrán konektor *CRMA59*, protože obsahuje znatelně méně výstupů a žádné vstupy. Toto rozhodnutí bylo učiněno z důvodu lepší implementace budoucích zařízení komunikující s IO kontroléru právě do konektoru *CRMA58*, kde tak bude možné použít více pinů.

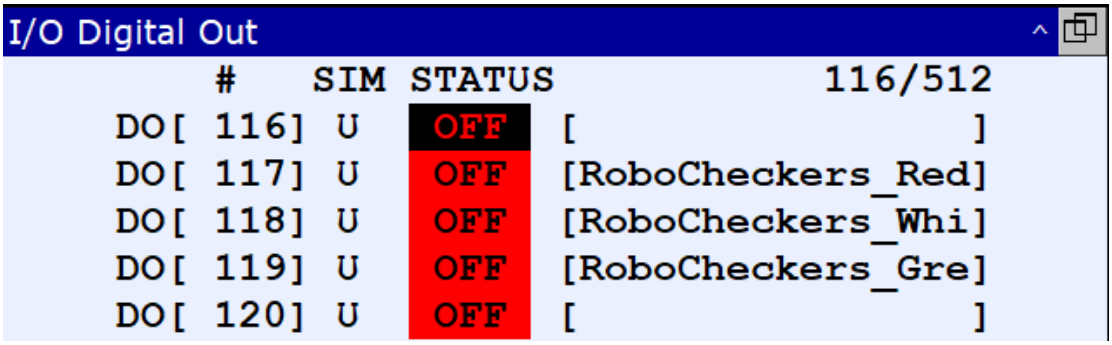
Dle diagramu konektoru *CRMA59* byly vybrány piny 26 až 28, které odpovídají digitálním výstupům na adresách 117 až 119. Jelikož je nutné digitální výstupy spínat vůči potenciálu 0V, byl vybrán ještě dodatečně pin 29. Na tyto piny byly připájeny vodiče, které vedou skrytě po kleci manipulátoru až do informačního panelu na jejím vrchu. Varianta kontroléru R-30iB Mate, kterou tato práce využívá, nemá v základě zapojené napájení digitálních výstupů a pouze je za pomoci tranzistorů spíná. Proto bylo zapotřebí zapojit dodatečně zdroj externího napájení 24V na piny 31 *DOSRC2* a 30 s potenciálem 0V. Přehled jednotlivých pinů konektorů *CRMA59* je vyobrazen v tabulce 2.1.

Po zapojení byly ještě jednotlivé digitální výstupy na teach pendantu okomentovány, aby bylo jasně zřetelné, že jsou používány. Jejich aktuální stav lze vyčíst

Tab. 2.1: Schéma pinů konektoru CRMA59. [9]

01	XHOLD			33	CMDENBL
02	RESET			34	FAULT
03	START	19	SDICOM3	35	BATALM
04	ENBL	20		36	BUSY
05	PNS1	21	DO120	37	
06	PNS2	22		38	
07	PNS3	23		39	
08	PNS4	24		40	
09		25		41	DO109
10		26	DO117	42	DO110
11		27	DO118	43	DO111
12		28	DO119	44	DO112
13		29	0V	45	DO113
14		30	0V	46	DO114
15		31	DOSRC2	47	DO115
16		32	DOSRC2	48	DO116
17	0V			49	24F
18	0V			50	24F

v podmenu *I/O Digital Out*, jak lze vidět na obrázku 2.4.



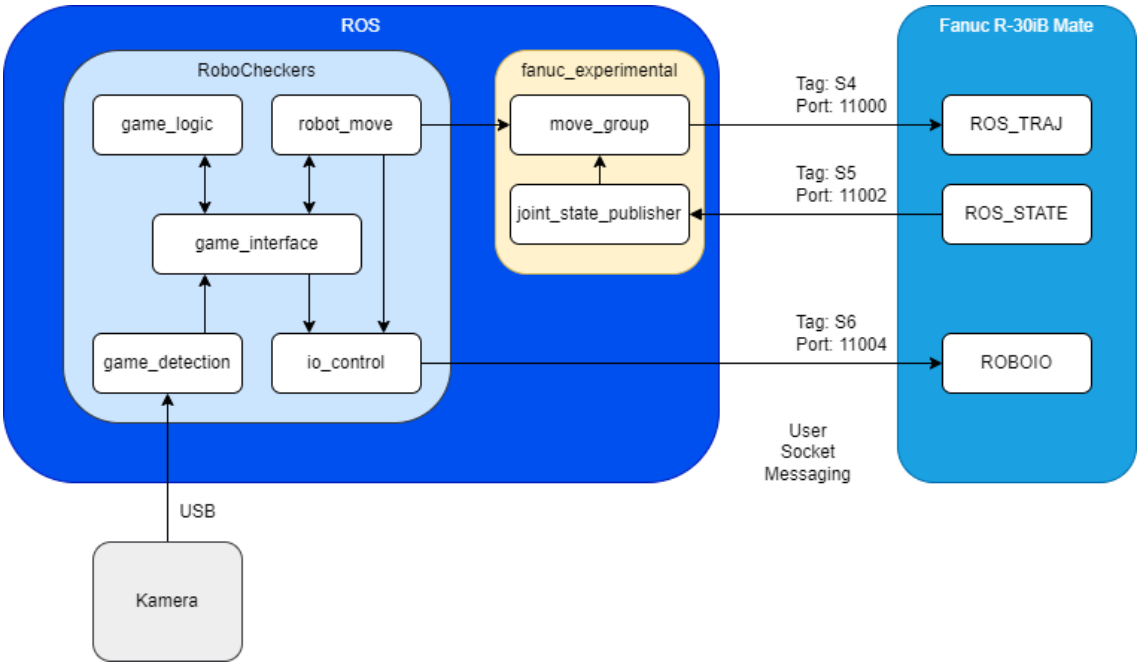
Obr. 2.4: Výstupy typu *Digital*.

3 Koncepce systému a vnitřní komunikace

Celý systém je ve výsledku rozdělen na dvě hlavní části. První z nich je umístěna na kontroléru manipulátoru. Ta již byla probrána podrobně v rámci předchozí kapitoly. Druhou z nich je celý systém ROS a všechny jeho součásti. Jejich vzájemná komunikace byla rovněž probrána v předchozí kapitole. Proto se teď budeme věnovat právě části druhé.

Celá kapitola čerpá značné množství informací z původní bakalářské práce [1].

3.1 Vnitřní struktura systému ROS



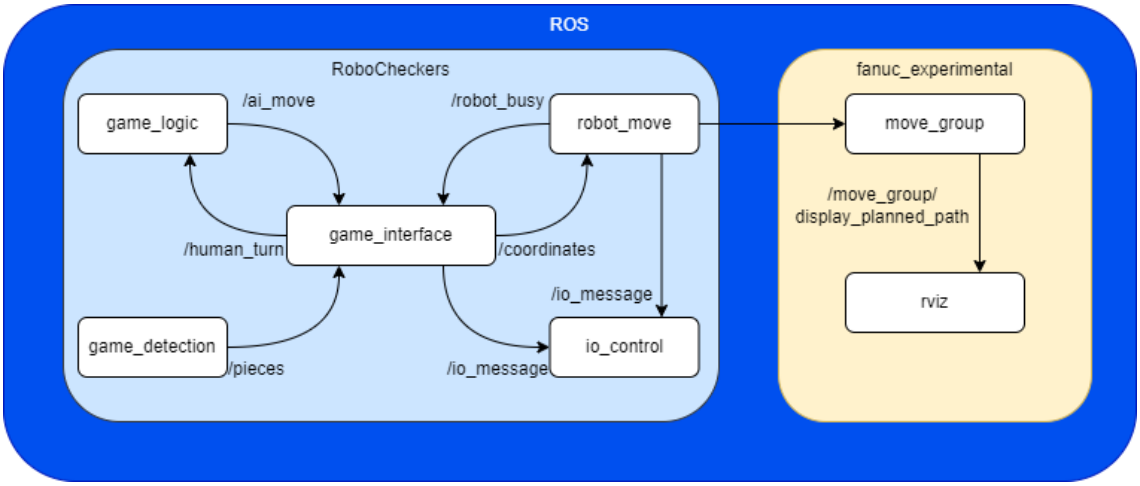
Obr. 3.1: Rozložení a komunikace jednotlivých uzlů.

Vnitřní struktura celého systému vychází z původního návrhu v rámci bakalářské práce. Ta je vša obohacena o zcela nové uzly *io_control* a *game_logic*. První z nich plní funkci ovládání všech booleovských vstupů a výstupů na manipulátoru, zatímco ten druhý plánuje tahy, které manipulátor následně provede. Uzel *game_detection* prošel největší proměnou a byl od základů předělán. Všechny jeho změny jsou dále důsledně rozebrány v kapitole 5. Další větší změnou prošel i node *game_interface*, kde byl doplněn kód pro zjišťování lidského zásahu z dat kamery a kontrolu pravidel. Dále byly všechny programy balíčku *Robocheckers* zpřehledněny, opraveny, vylepšeny a opakující se funkce byly přesunuty do knihoven.

Na obrázku 3.1 lze vidět strukturu jednotlivých uzlů a jejich propojení v rámci platformy ROS. Jednotlivé uzly spolu komunikují pomocí jasně specifikovaných zpráv skrze komunikační kanály známe jako *topic*.

Vnitřní komunikace

Komunikace uvnitř platformy ROS probíhá převážně skrze komunikační kanály známe jako *ROS Topics*, kde si předávají zprávy *ROS Messages*. Na obrázku 3.2 je znázorněna komunikace nejdůležitějších uzlů v rámci platformy ROS. Každý z těchto uzlů je znázorněn bílým obdélníkem a každý topic je vyobrazen jako šipka mezi uzly. Jedinou výjimkou v tomto schématu jsou uzly */robot_move* a *move_group*, protože komunikace mezi nimi probíhá v rámci programu *robot_move.py*, kde jsou oba inicializovány.



Obr. 3.2: Komunikace uvnitř platformy ROS.

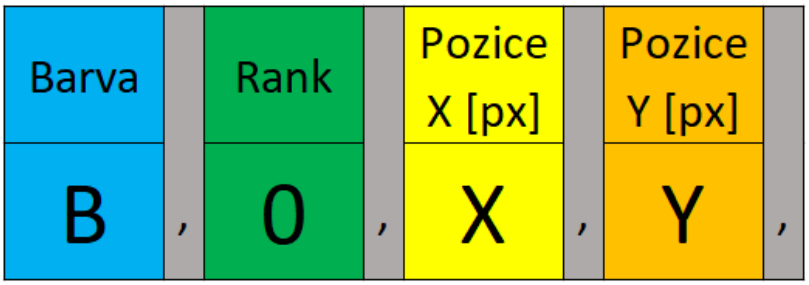
Detekce herní plochy

V tomto uzlu se zpracovává obraz z kamery a výsledkem jsou informace ohledně umístění jednotlivých herních kamenů, jejich hráčské příslušnosti a zdali se jedná o pěšce či dámu.

Veškeré zpracování obrazu je v rámci práce řešené za pomoci nástroje OpenCV. Před každým spuštěním by měla proběhnout kalibrace kamery vzhledem k umístění šachovnice, která určí pozici šachovnice v obrazu. Takto získané hodnoty se uloží do souboru ve formátu yaml. V rámci zpracování obrazu se tak obraz ořezává pomocí těchto hodnot. Ořez se dále normalizuje na velikost 512x512, jsou na něj aplikovány různé filtry a nakonec projde maskováním dle barev ve formátu HSV. V jednotlivých

obrazech dle barevných masek se nakonec uskuteční detekce pixelových seskupení (tzv. blobů) a dojde k nalezení jejich těžiště. Podrobnější popis funkce je rozebrán v kapitole 5.

Jakmile dojde k nalezení blobů všech tří hledaných barev, vyhodnotí se, které figurky představují pěšce a které dámy. To je stanoveno podle toho, zdali v blízkosti nalezené figurky byla nalezena i zelená rozlišovací ploška. V tomto okamžiku byla získána všechna důležitá data a dochází k sestavení zprávy a jejímu následnému odeslání do uzlu *game_interface* skrze topic */robocheckers/pieces*. Data jsou odesílána ve formátu textového řetězce za každého hráče zvlášť. Každý cyklus tedy dojde k odeslání dvou zpráv ve formátu, který je znázorněn na obrázku 3.3.



Obr. 3.3: Formát zprávy o nalezených figurkách.

Na začátku zprávy je znak popisující hráčskou příslušnost. *„B“* pro modrého hráče, *„O“* pro oranžového hráče. Mezi jednotlivými informacemi se vždy nachází dělící znak *„,“* (čárka). Dále se za každou nalezenou figurku daného hráče posílají celkem tři údaje. První z nich je hodnost (rank), která je vyjádřena Booleovskou hodnotou. Nula představuje pěšce a jednička dámu. Posledními hodnotami jsou umístění středu figurky v ořezaném obraze. Ty nabývají hodnot 0–255 a jsou popsány datovým typem float, protože se jedná o desetinná čísla (přestože pozici pixelu nelze vyjádřit desetinným číslem, pro vyjádření středu shluku je to zapotřebí).

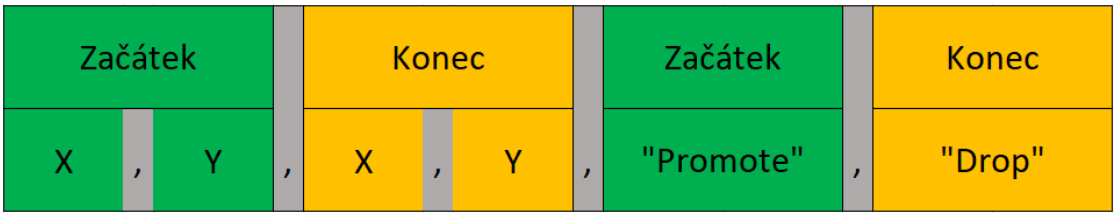
Výsledný tok dat je omezen pouze dobou trvání cyklu zpracování obrazu.

Herní rozhraní

Nejdůležitější ze všech uzlů, který řídí veškeré dění a zpracovává uživatelský zásah. Asynchronně přijímá zprávy z uzlu *game_detection*, které dále zpracovává až v momentě potvrzení tahu stiskem klávesy. Jako jediný komunikuje se všemi ostatními programy balíčku Robocheckers. Dochází v něm také ke kontrole platnosti a legitimnosti jednotlivých tahů, hlídání chyb, řízení informačního LED panelu, přepočtu pozice detekovaných figurek na reálné umístění v reálném světě a mnoho dalšího.

Při spuštění je uživatel dotázán, který z hráčů bude začínat. Má možnost poslat znak *„0“* pro svůj začátek, nebo pošle znak *„1“* pro zahájení hry počítačem řízeným hráčem. Tím je vygenerována speciální zpráva ve formátu *[(0, X)]*, kde *X* představuje zadanou hodnotu. Ta je odeslána po kanále */robocheckers/human_turn* do uzlu *game_logic*. V následujícím okamžiku ze stejného uzlu přijdou zpět informace po kanále */robocheckers/ai_move* o možnostech lidského protivníka a podle volby začínajícího hráče, s nimi může přijít i tah provedený počítačem řízeným hráčem.

Pokud dojde ke zpracování informace o provedeném tahu herní logikou, dojde k ověření, zdali se jedná o běžný tah, přeskakování figurek lidského hráče, povýšení pěšce za dámu, nebo jejich libovolné kombinace. Podle toho se pro příslušné figurky, se kterými se bude hýbat, vypočítají reálné souřadnice umístění v kartézském systému. Následně se vytvoří tahová sekvence, která se odešle po komunikačním kanále */robocheckers/coordinates* do dalšího uzlu jménem */robot_move*. Je opět ve formátu textového řetězce a její struktura je znázorněna na obrázku 3.4.



Obr. 3.4: Formát zprávy o pohybu.

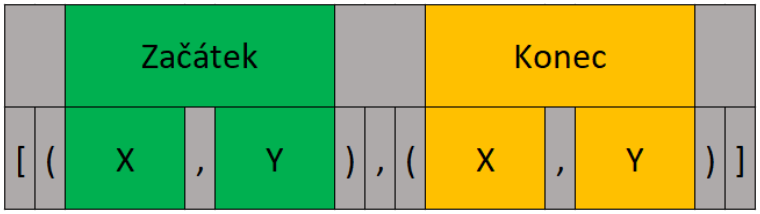
Jednotlivá data jsou opět oddělena čárkou. Každý jeden pohyb se skládá z dvojice začátek–konec. Každý člen této dvojice je buďto vyjádřen dvojicí souřadnic X a Y, nebo unikátním klíčovým slovem. Souřadnice jsou původem vyjádřeny datovým typem *float* a znázorňují pozici v milimetrech. Klíčová slova jsou datového typu *string* a jsou pevně definována. To znamená, že začátek pohybu může využít pouze slova *promote* a pro jeho konec lze užít pouze slova *drop*. I v tomto případě je komunikace oboustranná. V momentě odeslání dat o pohybu se čeká na přijetí zprávy po kanále */robocheckers/robot_busy* o dokončení pohybu. Dokud tato zpráva nepřijde, program nemůže z bezpečnostních důvodů přepnout na tah hráče.

Mezi těmito jednotlivými fázemi také dochází s uzlem */io_control* ke komunikaci po kanále */robocheckers/io_message*. Jedná se o ovládání informačního LED panelu, který informuje hráče o aktuálním stavu hry. Při spuštění proběhne krátká sekvence, která všechny kontrolky na dvě vteřiny zapne a následně vypne pro ověření poruchy indikátorů. Následně vždy na začátku tahu počítačem řízeného hráče dojde k rozsvícení bíle kontrolky do doby, dokud manipulátor neukončí celý svůj tah a ne-

vrátí se do své výchozí polohy. Následně dojde k rozsvícení zelené kontrolky, která dává uživateli najevo, že je na tahu lidský protihráč. Jeho tah končí v momentě, kdy provede platný tah a potvrdí jej stiskem klávesy *Enter*. Pokud provede neplatný tah, zelená kontrolka zůstává svítit, jelikož se stále jedná o jeho tah, a k tomu se rozbliká i červená kontrolka. Formát zprávy je shodný s formátem vnější komunikace pro ovládání vstupů a výstupů, který je ukázán na obrázku 2.2.

Herní logika

Nejnovější součást balíčku Robocheckers, který slouží k vyhodnocování nejvhodnějších tahů ze strany manipulátoru. Jedná se o upravený open source kód ze sítě *github.com* [18]. Byl rozšířen o knihovnu umožňující komunikaci se zbytkem celého systému. Konkrétně čeká na zprávy z uzlu *game_interface* o tahu lidského hráče. Může se však jednat i o speciální zprávy typu výběr začínajícího hráče. V momentě jejího přijetí dochází k jejímu zpracování, vyhodnocení a nakonec k vygenerování zprávy o provedeném tahu herní logikou. Celý algoritmus a jeho implementace do systému je podrobněji rozebrána v podkapitole 6.2.



Obr. 3.5: Formát zprávy o provedeném tahu manipulátoru.

Ta je odeslána po kanále */robocheckers/ai_move* zpět do uzlu *game_interface*. Skládá se ze dvou částí, které jsou odděleny speciálním znakem „|“. První část představuje tah, který herní logika provedla. Její formát je znázorněn na obrázku 3.5. Hodnoty X a Y jsou celá čísla v rozsahu 1–8, jedná se tedy o specifikaci políčka šachovnice.

Druhou částí zprávy je výčet možných tahů, které lidský protihráč během svého kola může provést. Její formát je list jednoho, či více možných tahů, které mají stejný zápis, jako znázornění na obrázku 3.5.

Existují však ještě unikátní případy, kdy místo provedeného tahu herní logikou přichází speciální zpráva, kdy zároveň místo výčtu možných tahů přichází prázdný list. Jedná se o zprávy informující například vítěze klání, nebo jestli při druhotném ověření validity lidského tahu nedošlo k chybě.

Plánování pohybů manipulátoru

V rámci inicializace uzlu se vytvoří instance třídy *MoveGroup*. Metody a vnitřní proměnné této třídy využívají funkce z knihovny *moveit_commander*. Ta umožňuje z parametrického serveru načítat data o užitém manipulátoru, zjišťovat a dále sdílet aktuální stav natočení jednotlivých kloubů, či samotné ovládání robotického manipulátoru. Slouží tedy ke komunikaci s kontrolérem. Jakmile se tato instance vytvoří, dojde k rozevření gripperu a manipulátor se přestaví do pozice na boku, aby nebránil kameře ve výhledu na herní plochu. V poslední řadě dojde v simulaci k vytvoření kolizních zdí, se kterými se následně počítá při výpočtech trajektorie. Ty se také zobrazí ve vizualizaci v systému *Rviz*. V tomto stavu vyčkává uzel */robot_move* na informace o pohybu z topicu */robocheckers/coordinates*.

V okamžiku, kdy dojde k obdržení pohybové sekvence, uvádí se manipulátor do chodu. Následně manipulátor provede sekvenci pohybů, a to přesně v tomto pořadí:

- Pohyb nad pole s figurkou (nebo nad zásobník na dámy)
- Pohyb směrem dolů
- Uchopení figurky gripperem
- Pohyb směrem nahoru
- Pohyb nad cílové pole (nebo nad odhazovací koš)
- Pohyb směrem dolů
- Uvolnění figurky gripperem
- Pohyb směrem nahoru
- Pohyb do výchozí pozice na straně

Pokud je v pohybové sekvenci více pohybů, k pohybu do výchozí pozice na boku dojde pouze po vykonání všech dílčích pohybů. V momentě ukončení této pohybové sekvence dochází opět k vyčkávání nových instrukcí z programu */game_interface*.

Ovládání vstupů a výstupů

Aktuálně jediné ovládané vstupy a výstupy jsou ty, které uskutečňují změnu stavu gripperu a informačního LED panelu. Program */io_control* očekává příchozí zprávy po kanále */robocheckers/io_message*. Ty mohou přijít pouze z uzlů */robot_move* a *game_interface*. Zprávy, které přicházejí jsou stejného formátu, jak bylo znázorněno na obrázku 2.2.

Při spuštění tohoto programu dojde nejprve k načtení IP adresy kontroléru manipulátoru z parametrického serveru v rámci platformy ROS. Následně dojde k inicializaci soketu typu klient a k navázání komunikace s kontrolérem manipulátoru. V momentě obdržení zprávy o změně stavu vstupu/výstupu proběhne ověření, zdali má zpráva platný formát. V případě kladného výsledku ověření se zpráva přepošle kontroléru. V opačném případě dojde k zahození zprávy.

3.2 Spouštění

Pro spuštění je zapotřebí zavolat dva příkazy. První z nich spouští programy nutné pro připojení ke kontroléru a zajišťují vnější komunikaci s ním, převážně se jedná o programy balíčků *fanuc* a *fanuc_experimental*, které dále odkazují na programy a funkce z nástavby *MoveIt!*. Druhý volá všechny programy balíčku *Robocheckers*. Oba příkazy jsou k vidění zde:

```
$ roslaunch robocheckers robot_socket_connect.launch
$ roslaunch robocheckers robocheckers.launch
```

Programy balíčku *Robocheckers* se napojí na programy, které byly spuštěny prvním příkazem. Jakmile jsou vnitřní i vnější komunikace navázány, je možné zahájit hru.

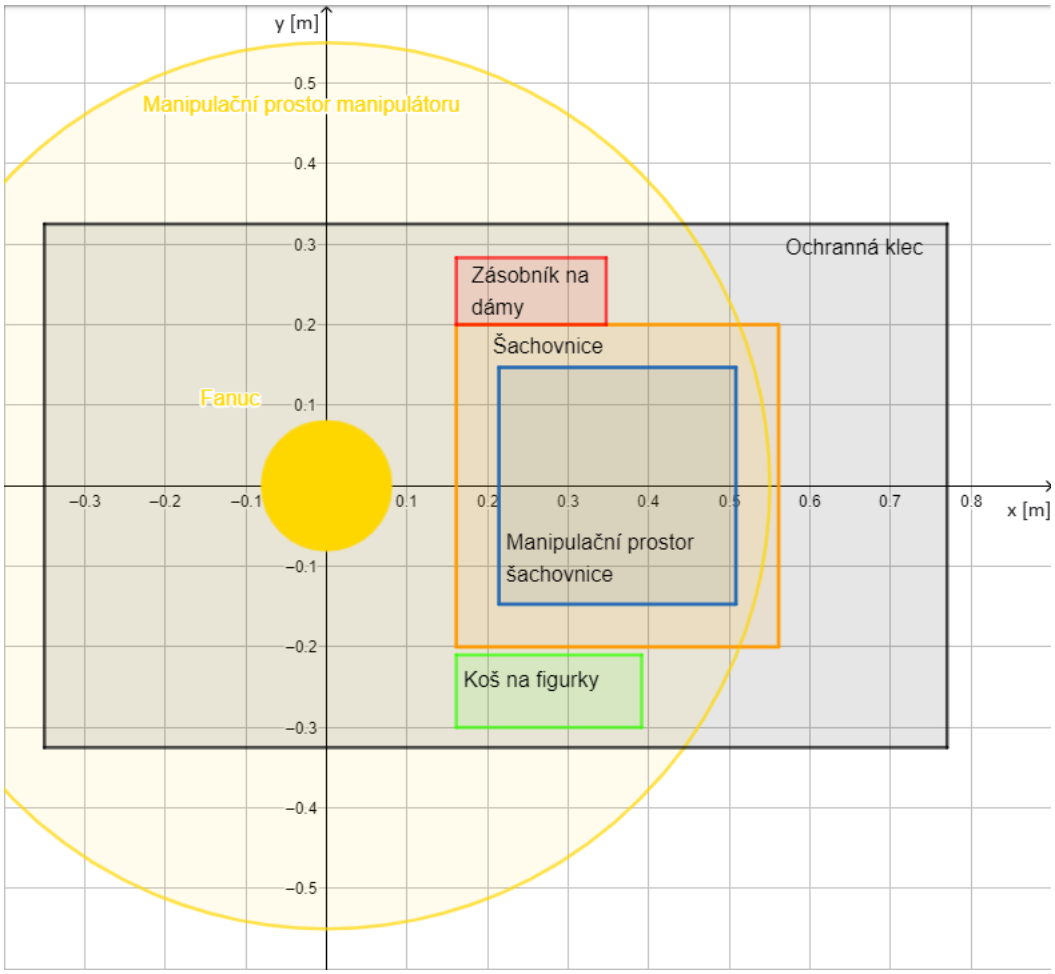
Pro jednoduché spouštění bylo vytvořeno hned několik spustitelných souborů, které si nejen mezi sebou předávají různé argumenty, ale dokáží je i nahrávat na parametrický server, kde je poté mohou využívat i další programy.

Při spuštění souboru *robot_socket_connect.launch* je nutné zadat parametr *robot_ip*. Dojde k načtení zadané IP adresy, názvů jednotlivých kloubů a URDF modelu manipulátoru na parametrický server. Následně se spouští další soubory již z balíčků *fanuc* a *moveit* pro ovládání manipulátoru, čtení aktuálních stavů a vizualizaci v prostředí Rviz.

Soubor *robocheckers.launch* na druhou stranu spouští pouze programy v rámci balíčku *robocheckers*. Jeho jediným povinným parametrem je ID použité kamery pro zpracování obrazu. Ten se rovněž nahrává na parametrický server. Následně se spustí všechny čtyři hlavní programy tohoto balíčku, a to *game_detection.py*, *game_interface.py*, *robot_move.py* a nakonec *io_control.py*.

4 Fyzické modely pro hru

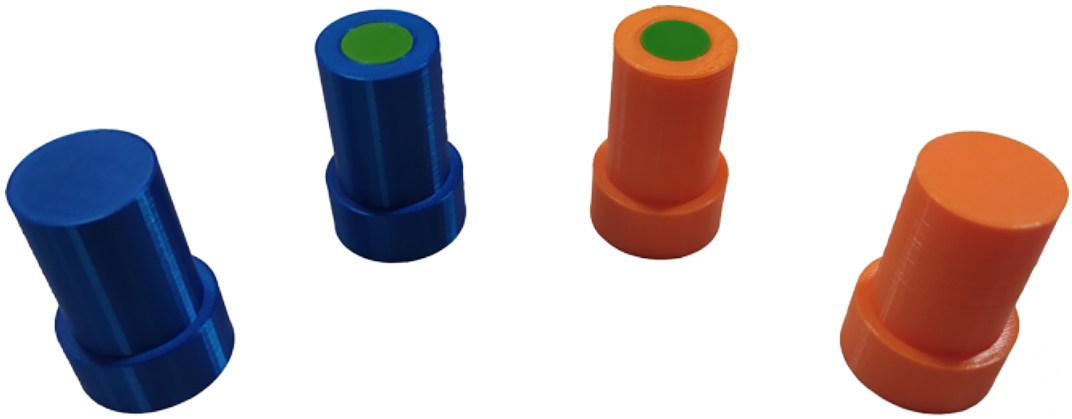
Mezi fyzické modely pro hru patří nejen šachovnice a figurky, ale i zásobník na dámy, koš na odebrané figurky, nebo dokonce informační panel. Všechny tyto součásti byly navrženy a umístěny do manipulačního prostoru manipulátoru s ohledem na jeho kinematické parametry. Jejich rozložení lze vidět na obrázku 4.1.



Obr. 4.1: Rozmístění součástí v manipulačním prostoru manipulátoru.

4.1 Figurky

Na rozdíl od posledního postupu v rámci předešlé bakalářské práce došlo ke změně barev figurek. Místo zelených a tmavě modrých s červenými značkami pro rozlišení hodností jsou nové figurky oranžové a světle modré se značkami zelenými. Nově vytištěné figurky lze vidět na obrázku 4.2.



Obr. 4.2: Figurky.

V rámci bakalářské práce byly vytištěny přesně čtyři kusy figurek, přesněji od každé barvy a hodnosti přesně jedna. V době tisku figurek do diplomové práce již však nebyl dostatek stejnobarevného filamentu k dispozici. Proto byly zvoleny nové dostatečně kontrastní barvy figurek tak, aby jejich barvy byly na kruhu HSV modelu co nejvzdálenější. Nakonec byla vhodně zvolena i barva rozlišovacích značek.

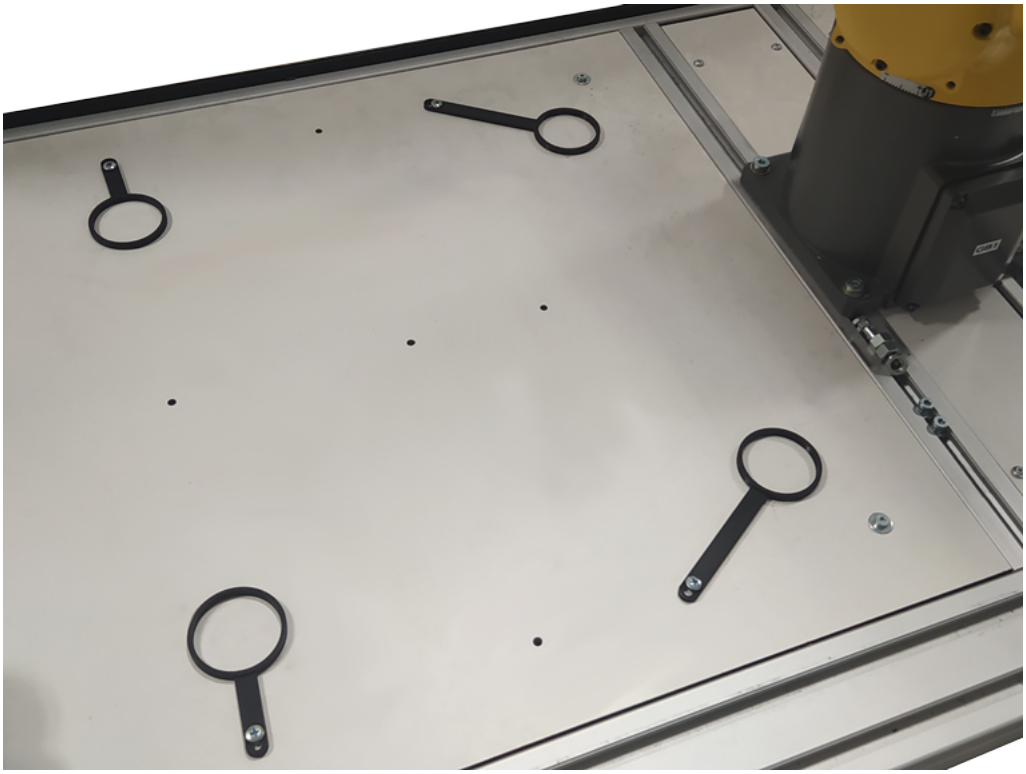
4.2 Šachovnice

Obr. 4.3: Šachovnice.

Šachovnice na druhou stranu neprošla téměř žádnou větší změnou. Rozměr zůstává 400x400 mm při výšce 20 mm + 40 mm nožičky. Rozměr jednotlivých polí je 32x32 mm. Jediným rozdílem je nové plátno, které nyní překrývá celý dřevěný

povrch desky z důvodu odstranění šumu při detekci oranžových figurek. Snímek šachovnice je k vidění na obrázku 4.3.

Nově byly navrženy a vytištěny držáčky, které drží šachovnici pevně na místě. Tím je docíleno, že není nutné kalibrovat pozice dvojice manipulátor–šachovnice. Její pozice byla zaznamenána do souboru *dimensions_data.yml* a tyto hodnoty již není nutné dále upravovat, pokud se s držáčky nebude hýbat. Pro jejich upevnění ke kleci manipulátoru bylo užito již vytvořených děr, aby nebylo nutné invazivním způsobem zasahovat do výukové buňky. Hotové, vytištěné a připevněné držáčky jsou k vidění na obrázku 4.4.



Obr. 4.4: Držáky šachovnice.

4.3 Odkladové pozice figurek

Také byly na herní plochu přidány zásobník na dámy a koš na vyřazené figurky pro počítačem řízeného hráče. Lidský hráč má dostatek prostoru na dámy i soupeřovy sebrané herní kameny dostatek místa, proto nebylo zapotřebí vytvářet určená místa i pro ně.

4.3.1 Zásobník na dámy

Po straně šachovnice je umístěn jeden zásobník schopen pojmout až šest dám. Rozměry tohoto zásobníku jsou 186 mm na délku, 83 mm na šířku a 60 mm na výšku. Zásobník je pevně přidělán ke stolu a je odebiratelný nezávisle na šachovnici. Zásobník byl speciálně navržen tak, aby při nájezdu na jednotlivé figurky nedocházelo ke kolizi s ostatními. Bylo také dbáno na to, aby byly dámy v dostatečné výšce a manipulátor tak nemusel při pokusu o úchop najíždět až pod úroveň šachovnice. To by mohlo v krajních případech mít za následek kolizi gripperu se šachovnicí. Vyrobený zásobník se čtyřmi dámami je k vidění na obrázku 4.5.

Obr. 4.5: Zásobník na dámy.

4.3.2 Koš na vyřazené figurky

Když počítačem řízený hráč přeskočí lidskému hráči figurku, nebo dojede vlastním pěšcem na konec šachovnice, je zapotřebí nějakou figurku z herní plochy odstranit. K tomu byl speciálně navržen koš na vyřazené figurky. Ten byl vyroben z kartonového papíru a má rozměry 240 mm na délku, 90 mm na šířku a 115 mm na výšku. Je dále vybaven zaoblenou plošinou, aby se vyřazené herní kameny neshromažďovaly na jednom místě, ale postupně zabraly celý koš. Na jedné z delších stran je také

průhledná fólie, aby i hráči menšího vzrůstu, či nižšího věku s pohodlím věděli, zdali se v koši nachází alespoň jedna vyřazená figurka.

4.4 Informační panel

Informační panel slouží k informování lidského hráče o aktuálním stavu hry. V panelu se nachází celkem tří pozice pro kontrolky. Ty jsou zelené, bílé a červené barvy. Zelená kontrolka symbolizuje lidského hráče, tedy pokud svítí, jedná se o jeho tah. Bílá signálka na druhou stranu představuje počítačem řízeného hráče. Z toho jasně vyplývá, že tyto dvě kontrolky nemohou v žádném platném případě svítit najednou. Barvou poslední kontrolky je červená. Ta informuje o problémech a chybách. Zatím je implementováno pouze její blikání společně se svitem zelené v případě chybného tahu lidského hráče.

Lidský hráč by do prostoru manipulátoru měl zasahovat jen a pouze tehdy, pokud svítí právě zelená kontrolka.

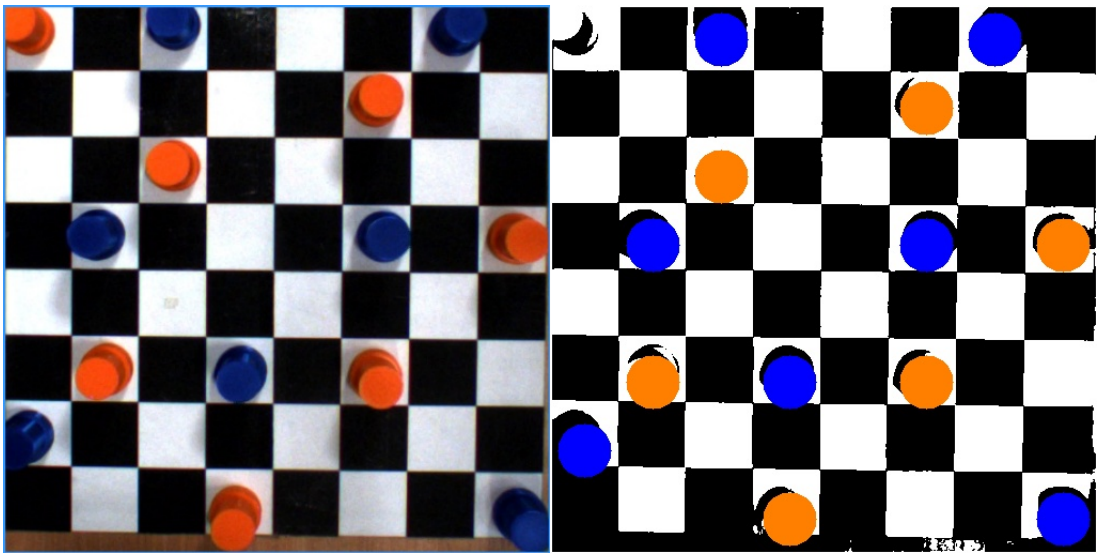
Mezi speciální případy patří například konec hry, kdy blikání příslušné kontrolky značí vítěze. Kupříkladu při výhře lidského hráče bliká zelená signálka.

Obr. 4.6: Informační panel.

Na obrázku 4.6 je ukázán hotový výrobek informačního panelu. Ten se skládá přesné ze čtyř částí. Rám, plná bočnice, bočnice na kontrolky a víko. Na jedné ze stran je také otvor rezervovaný pro vývodku M16x2,5. Kontrolky jsou propojeny s kontrolérem manipulátoru dvěma dvoužilovými kabely o průřezu $0,5\ \mathrm{mm}^2$. Jelikož se jedná o lankové kabely, které jsou uchyceny šroubovými svorkami, bylo nutné na jejich konce nakrimpovat kabelové dutinky.

5 Detekce herní plochy v reálném čase

Jak již bylo dříve zmíněno, program pro detekci aktuální situace na herní ploše byl změněn téměř od základu, vzhledem k výsledku bakalářské práce. Byla několikanásobně zvýšená nejen spolehlivost, ale také přesnost, s jakou jsou jednotlivé figurky nalezeny. V původní detekci nedocházelo k určení pozice v reálném světě, nýbrž pouze ke klasifikaci políčka, na kterém se nachází. Vizualizace výsledků starého snímání je zobrazena na obrázku 5.1.

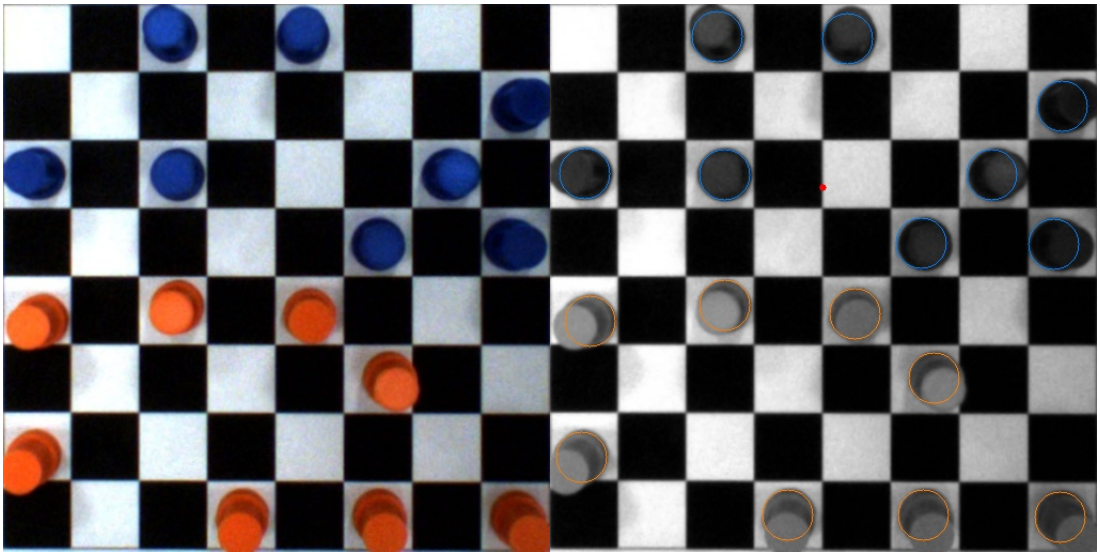


Obr. 5.1: Výsledek původního programu pro detekci herní plochy.

Z obrázku lze vyčíst, že už samotný ořez není ani zdaleka přesný, jak by bylo zapotřebí pro přesnou detekci. Ten byl určen dle dřevěného orámování kolem šachovnice, do kterého často zasahovaly figurky z perspektivy kamery. Ořez se počítal každý průběh smyčky detekce, což nemělo positivní vliv ani na rychlost průběhu nebo na přesnost. Mezi nežádoucí další vlivy na detekci, které nebyly odfiltrovány, patří kupříkladu mírné natočení kamery způsobené stylem jejího uchycení, nezkalibrovaný a tím deformovaný obraz, ořezání figurek zasahující mimo prostor ořezu a podobně.

Nová detekce herní plochy odstranila veškeré problémy, které vznikly v průběhu vývoje první verze. Ořez se provádí dle dat z kalibrační sekvence, kterou je potřeba spustit pokaždé, kdy je přemístěna kamera. Výpočet tedy proběhne jen jednou. Kalibrační data obsahují i údaj o správném natočení obrazu. Dalším problémem, který byl úspěšně odstraněn, byla špatná detekce krajních figurek. Ten byl odstraněn zvětšením ořezávané plochy o 16 pixelů na každou stranu. Následně po nalezení

všech figurek je obraz opět o tuto hodnotu oříznut. Veškeré tyto úpravy jsou prováděny na zkalibrovaných snímcích, díky čemuž je odfiltrován vliv zakřivení čočky objektivu na deformaci obrazu. Vizualizace výsledků nového snímání je zobrazena na obrázku 5.2.



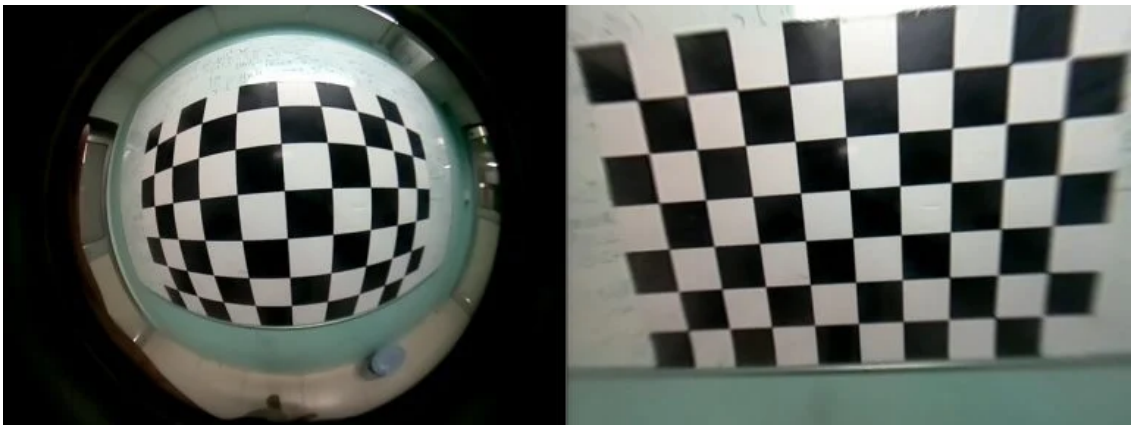
Obr. 5.2: Výsledek nového programu pro detekci herní plochy.

5.1 Kalibrace kamery

Jak bylo zmíněno v předchozích odstavcích, čočka objektivu má vliv na zakřivení a deformaci snímané plochy. Cílem této kalibrace je tedy toto zakřivení změřit a následně odstranit. Celá tato podkapitola čerpá ze zdrojů [10] a [11].

Každý kamerový objektiv má své vnitřní vlastnosti, mezi které patří například ohnisková vzdálenost, koeficient zkreslení a podobně. Knihovna OpenCV nabízí způsob, jak tyto parametry zjistit a následně použít na odstranění jakéhokoliv zkreslení, které způsobují. Na obrázku 5.3 je ukázka deformovaného obrazu (vlevo) a následně obrazu kalibrovaného (vpravo) za pomoci geometrické korekce z OpenCV knihovny.

Pro nalezení správné projekce 3D bodu v obraze je zapotřebí nejprve přepočítat pozici daného bodu ze souřadnicového systému světa do souřadnicového systému kamery pomocí matice rotace a vektoru translace. Následně lze za využití vnitřních parametrů kamery vypočítat projekci pozice 3D bodu v obraze pomocí rovnice 5.1



Obr. 5.3: Kalibrace zkresleného obrazu pomocí parametrů kamery. [10]

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = P \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{5.1}$$

X_w, Y_w, Z_w - souřadnice bodu v souřadnicovém systému světa
u', v', w' - představují pozici projekce v obraze
P - projekční matice

Přesné souřadnice projekce v obraze v pixelech se následně vypočítají podle rovnic 5.2.

$$u = \frac{u'}{w'},\ u = \frac{v''}{w'} \tag{5.2}$$

Projekční matice P je formátu 3x4 a skládá se celkem ze dvou částí. První z nich je matice vnitřních parametrů kamery K a druhou je matice vnějších parametrů $[R|t]$ sestávající z matice rotace R a vektoru translace t. Matice K má pod hlavní diagonálou samé nuly, označujeme ji tedy jako horní trojúhelníkový tvar (rovnice 5.3).

$$K = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.3}$$

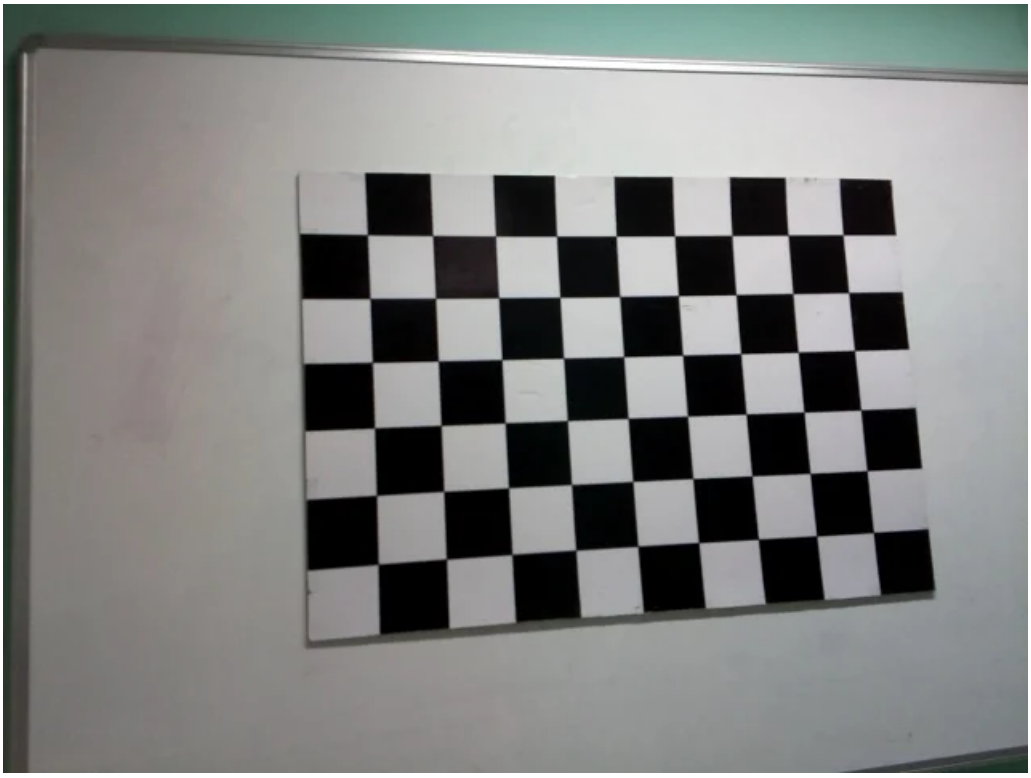
f_x a f_y - snímek hodnoty ohniskových vzdáleností pro oba rozměry (protože senzor kamery nemusí být čtvercový)
c_x a c_y - souřadnice optického středu kamery (ten nemusí souhlasit se středem souřadnicového systému obrazu)
γ - zkosení mezi osami (většinou má nulovou hodnotu)

5.1.1 Postup kalibrace kamery

Cílem kalibrace kamery je nalezení 3x3 matice vnitřních parametrů kamery K, 3x3 matici rotace R a 3x1 vector translace t za pomoci množiny známých 3D bodů a souřadnic jejich umístění v obraze. V momentě získání těchto hodnot lze kdykoliv každý další snímek, pořízený stejnou kamerou se stejným objektivem, zkalibrovat.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.4}$$

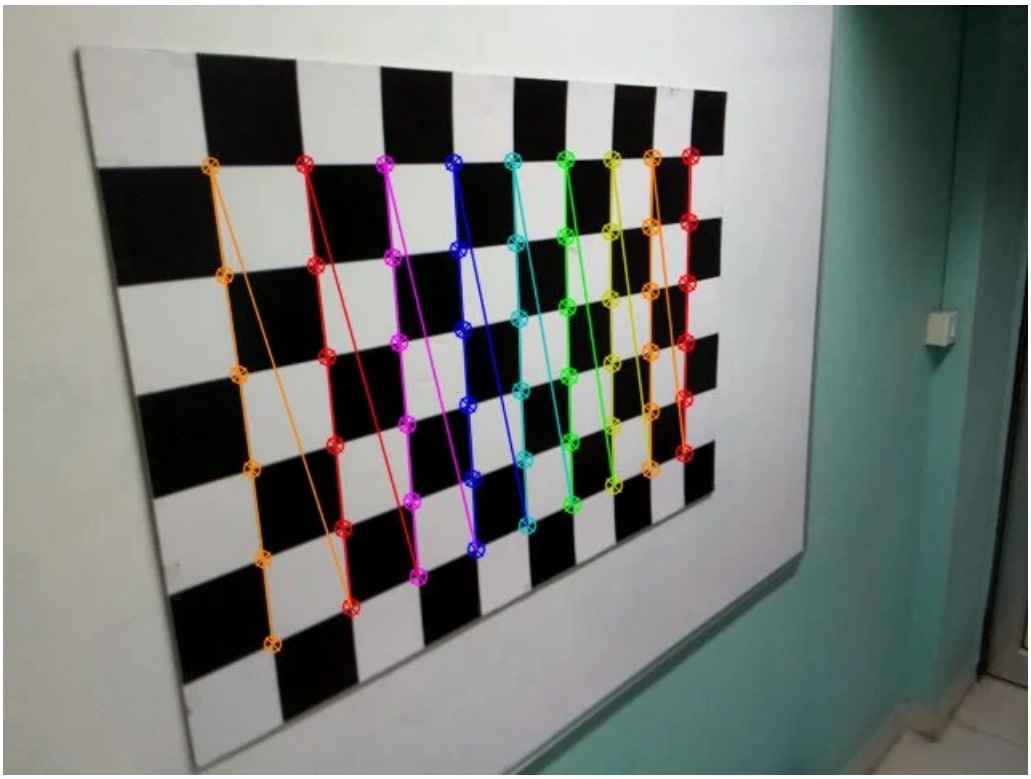
V knihovně OpenCV má matice vnitřních parametrů kamery K mírně odlišný tvar. Jak lze vyčíst z rovnice 5.4, neobsahuje koeficient γ.



Obr. 5.4: Příklad vhodné šachovnice ke kalibraci. [10]

Na obrázku 5.4 je šachovnice, připravená ke kalibraci kamery, připevněná ke zdi. Námi hledané 3D body jsou umístěny vždy na rohu čtyř sousedících políček (v tomto případě to je 54 bodů). Každý z těchto bodů má hodnoty v souřadnicovém systému reálného světa X_w (vodorovná osa), Y_w (svislá osa) a Z_W (osa kolmá na zeď za šachovnici, pro všechny body je tedy nulová). V průběhu kalibrace se počítají parametry kamery na základě známých pozic v reálném světě a jejich korespondujících pozicích v souřadnicovém systému kamery pro každý nalezený bod.

Je zapotřebí vytvořit co největší dataset snímků šachovnice. Čím různorodější orientace, vzdálenosti a pozice, ze kterým jsou snímky pořízeny, tím kvalitnější a přesnější data kalibrace. Mimo to je potřeba i znát rozměr jednotlivých políček šachovnice. Na každém snímku jsou následně detekovány body, z nichž se jeden stanový jako referenční (se souřadnicemi [0, 0]). Od něj se pomocí známe hodnoty rozměru políček spočítají reálné souřadnice ostatních. Vizualizace takto nalezených bodů je ukázána na obrázku 5.5.



Obr. 5.5: Vizualizace nalezených rohů na šachovnici. [10]

Pro jejich nalezení nabízí OpenCV funkci *findChessboardCorners*, která má následující syntaxi.

retval, corners = cv2.findChessboardCorners(image, patternSize, flags)

image - snímek šachovnice ve formátu osmibitového obrazu v odstínech šedi
patternSize - počet hledaných rohů, v pythonu se zadává jako tuple dvou hodnot (bodů_v_řádku, bodů_ve_sloupci)
flags - flagy upravující postup kalibrace
corners - výstupní list souřadnic detekovaných rohů
retval - True/False hodnota na základě úspěšného nalezení rohů

Jelikož každá dobrá kalibrace je o přesnosti, je více než vhodné použít funkci

cornerSubPix, která vypočítá přesnou pozici rohů v desetinné hodnotě mezi pixely. Ta pro každý nalezený bod hledá vhodnější umístěný v jeho nejbližším okolí. Jedná se o iterativní algoritmus, je tedy nutné specifikovat podmínku ukončení.

cv2.cornerSubPix(image, corners, winSize, zeroZone, criteria)

image - snímek šachovnice ve formátu osmibitového obrazu v odstínech šedi
corners - vstupní/výstupní list souřadnic detekovaných rohů (upraví se jeho interní hodnoty)
winSize - poloviční délka vyhledávacího okna
zeroZone - poloviční délka mrtvého regionu uvnitř vyhledávacího okna
criteria - ukončovací podmínky, jedná se o tuple tří hodnot (maska kritérií, maxCount, epsilon)

Posledním krokem je zpracovat všechny získané hodnoty funkcí *calibrateCamera*, jejímž výsledkem jsou kalibrační data kamery. Její syntaxe je následovná.

retval, cameraMatrix, distCoeffs, rvecs, tvecs = cv2.calibrateCamera(objectPoints, imagePoints, imageSize)

objectPoints - Vektor vektorů 3D bodů v reálném světě (většinou jsou jednotlivé vektory totožné, protože se šachovnice nemění)
imagePoints - Vektor vektorů 2D bodů v každém snímku (pro každý vektor připadá právě jeden 3D vektor)
imageSize - Velikost obrazu kamery
retval - Návratová hodnota True/False indikující úspěšnost průběhu
cameraMatrix - matice vnitřních parametrů kamery K
distCoeffs - Koeficient zakřivení čočky
rvecs - 3x1 vektor rotace r
tvecs - 3x1 vektor translace t

5.1.2 Provedení kalibrace a uložení dat

Nejprve je zapotřebí pořídit alespoň patnáct snímků šachovnice. Ty lze vytvořit dvěma způsoby. Buď je umístěna šachovnice na jedno místo a pohybuje se s kamerou, nebo lze upevnit kameru a následně pohybovat se šachovnicí. Takto pořízené snímky je následně nutné přemístit do adresáře */calibration/images* uvnitř balíčku Robocheckers.

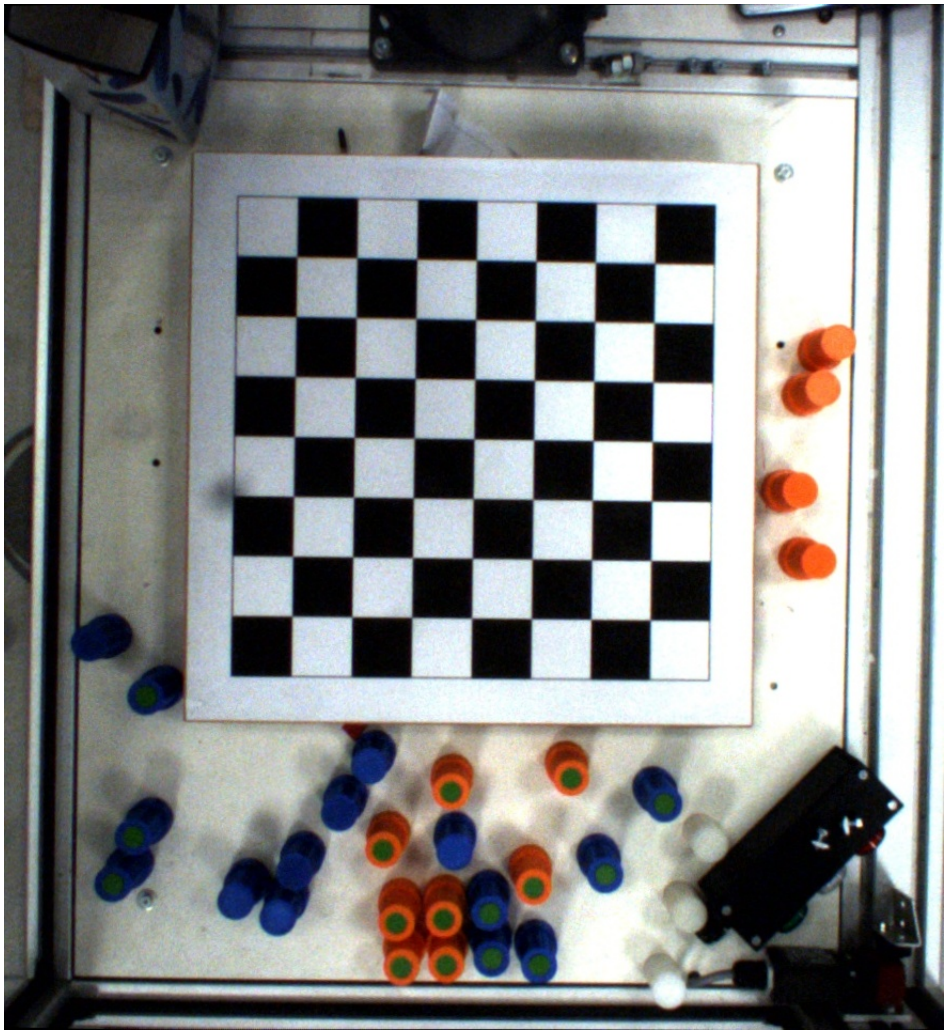
Nakonec stačí samotnou kalibraci spustit, přičemž jsou získané hodnoty uloženy do souboru *calib_data.yml* ve složce *calibration*. Hodnoty předchozí kalibrace zálohují do souboru *calib_data_backup.yml*. V průběhu programu detekce herní plochy jsou načteny a následně užity ke kalibraci kamery. Tuto kalibraci je nutno provést

pouze v momentě výměny kamery, nebo objektivu. Kalibrační sekvence se spouští následujícím příkazem.

```
$ roslaunch robocheckers calibrate_camera.launch
```

5.2 Kalibrace páru kamera–šachovnice

Poněkud jednodušší kalibrační sekvence, která má za účel detekovat umístění šachovnice v obraze a zjistit její vlastnosti. Tuto kalibrační sekvence je nutno provést pouze pokud bylo manipulováno s kamerou. Doporučuje se tento skript spouštět na začátku každého spuštění celého systému. Pro správný průběh je zapotřebí, aby kamera měla nerušený výhled na prázdnou šachovnici. V případě, že se na ni budou nacházet figurky, budou získaná data chybná.

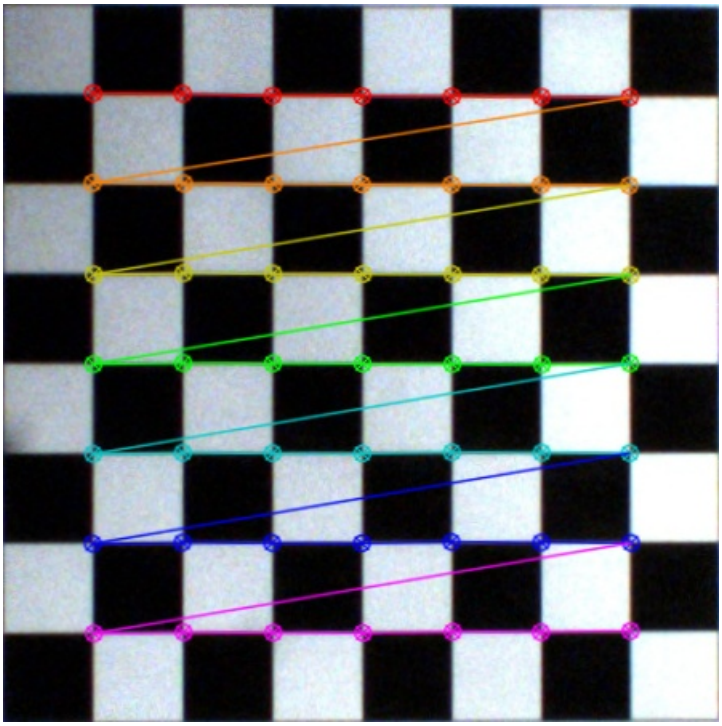


Obr. 5.6: Snímek zkalibrované kamery bez kalibrace páru kamera–šachovnice.

Nejprve je obraz zkalibrován dle hodnot získaných postupem v předchozí kapitole, aby se odfiltrovalo zkreslení. Výsledek této operace je ukázán na obrázku 5.6. Následně je zapotřebí zjistit, kde přesně se nachází šachovnice. K tomu je užito již známých OpenCV funkcí *findChessboardCorners* a *cornerSubPix*. Ze získaných bodů šachovnice je nyní potřeba získat čtyři rohové extrémy. Toho je docíleno následující sekvencí.

1. Vybrání náhodného bodu
2. Nalezení nejvzdálenějšího bodu od prvního náhodně vybraného (první roh)
3. Nalezení nejvzdálenějšího bodu od prvního rohu (druhý roh)
4. Nalezení matematického středu mezi prvními rohy
5. Výpočet vzdálenosti všech bodů od středu
6. Seřazení sestupně dle vzdálenosti a uložení prvních čtyř bodů

Nyní je ze všech vzdáleností mezi políčky v obou osách zvlášť vypočítaná průměrná délka políčka v pixelech. Dále jsou vypočteny hodnoty pozice středu šachovnice (střed mezi čtyřmi rohovými extrémy) a úhel natočení šachovnice z hodnot délek stran pomocí funkce arkus tangens.



Obr. 5.7: Snímek zkalibrované kamery s kalibrací páru kamera–šachovnice.

V tomto momentě máme všechny potřebné parametry pro výpočet hodnot kalibrace páru kamera–šachovnice. Nejprve je potřeba převést všechny čtyři rohy ze souřadného systému snímku na souřadný systém středu šachovnice. Následně dojde

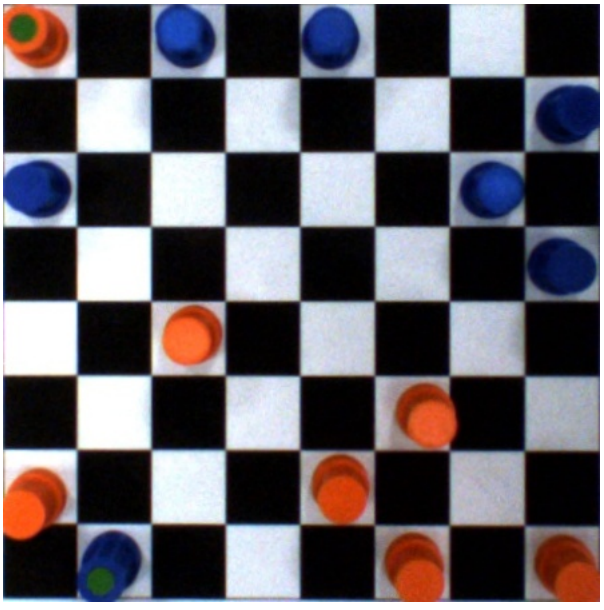
k orotování rohů kolem středu nového souřadnicového systému o vypočtený úhel natočení šachovnice. Dále je ke každé straně okna přičtena průměrná hodnota délky políčka, čímž je dosaženo velmi přesného ořezu herní plochy. Posledním krokem je výpočet hodnot odpovídajících původnímu souřadnému systému snímku. Snímek ořezaný pomocí těchto kalibračních dat je ukázán na obrázku 5.7.

Získaná data (Rohy šachovnice, střed šachovnice a úhel natočení šachovnice) jsou nakonec uložena do yaml souboru *chessboard_data.yml* v adresáři */calibration* uvnitř balíčku Robocheckers. Obdobně jako u kalibrace kamery dojde nejprve k záloze starých dat do souboru *chessboard_data_backup.yml*. Samotná sekvence získání kalibračních dat se spouští následujícím příkazem.

```
$ roslaunch robocheckers calibrate_board.launch
```

5.3 Zpracování obrazu

Detekce aktuálního stavu na herní ploše začíná jednoduššími příkazy, mezi které patří například načtení ID kamery z ROS parametrického serveru, načtení kalibračních dat z yaml souborů, vytvoření instancí tříd hráčů a tak podobně. Ořez obrázku však není vytvořen přesně podle dat kalibrace šachovnice, ale je k němu připočten offset. To je z důvodu, že figurky na kraji šachovnice mohou z perspektivy kamery sahat mimo ořez. Jinak by došlo i k ořezání částí figurek, jejichž přesná detekce by už nebyla možná.

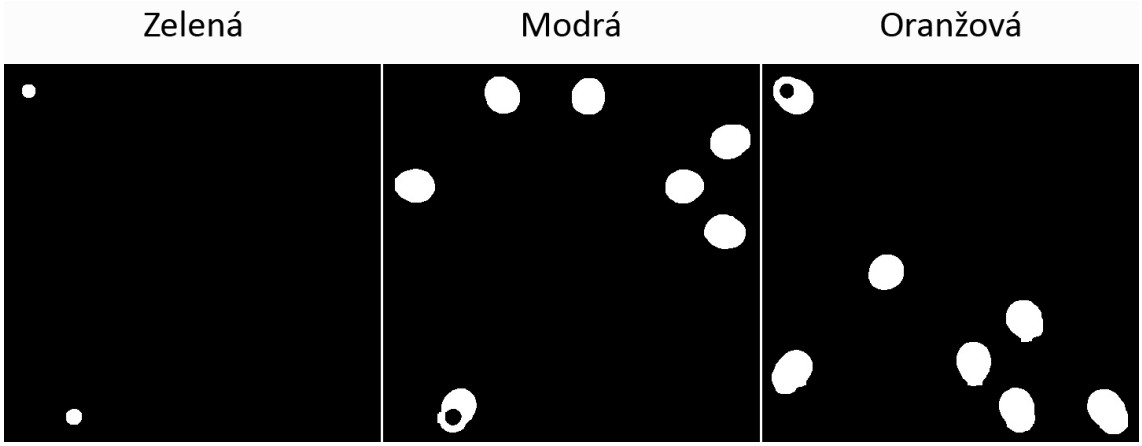


Obr. 5.8: Snímek po kalibraci před detekcí figurek. (Bez offsetu ořezu)

Jakmile je tato inicializační sekvence dokončena, program se dostává do smyčky a každý cyklus posílá dvě zprávy o pozicích a hodnostech jednotlivých figurek. Každá ze zpráv představuje právě jednoho hráče a výčet jeho figurek. K jejich sestavení je zapotřebí mnoho úprav snímku. Jejich postup bude přiblížen na ořezu ukázaném na obrázku 5.8.

5.3.1 Aplikace kalibračních dat, filtrace obrazu

Nejprve jsou aplikována kalibrační data na aktuální snímek. Tím je odfiltrováno zkreslení a dojde k ořezání nalezené šachovnice. V dalších kroku dochází k barevné filtraci dle HSV modelu. Tím jsou vytvořeny tři snímky, přičemž ve dvou jsou pixelové shluky figurek hráčů a ve třetím jsou pouze rozlišovací plošky. Tyto tři snímky lze vidět na obrázku 5.9. Jedná se o binární obrazy, které vycházejí z ořezu s offsetem, čehož si lze povšimnou při srovnání s obrázkem 5.8.



Obr. 5.9: Snímky po aplikaci všech tří barevných filtrů.

Na modrém a oranžovém snímku vychází najevo, že pixelové shluky dam v sobě mají po barevné filtraci díru. Proto je před jejich detekcí zapotřebí tyto díry zaplnit. Toho je dosaženo sloučením s výsledným obrazem filtrace rozlišovacích plošek. Jak lze vidět na obrázku 5.10, na obou snímcích dojde k přičtení plošek, které patří k figurkám ze snímku opačného. Tyto plošky jsou oproti pixelovým shlukům figurek tak malé, že je jejich odfiltrování jednoduché.

5.3.2 Detekce pixelových shluků (blobů)

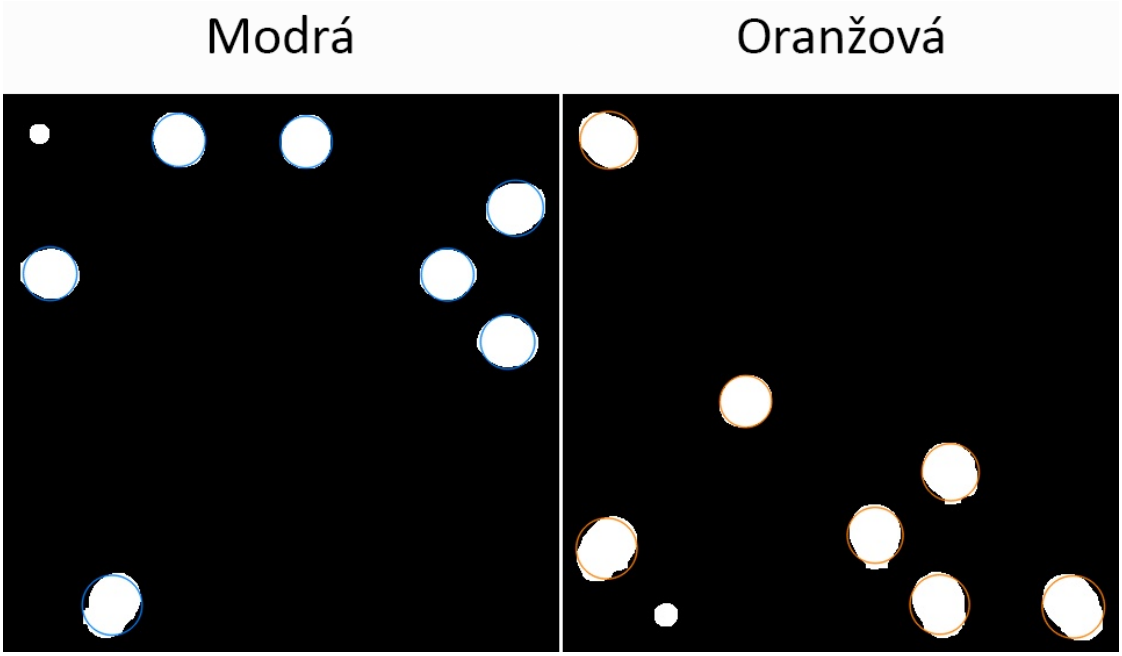
Dostáváme se k detekci jednotlivých jednotlivých blobů. Nejprve je potřeba stanovit parametry pro takovou detekci, čehož je dosaženo vytvořením proměnné struktury

cv2.SimpleBlobDetector_Params(). Značná část této kapitoly čerpá ze zdrojů [12] [13].

Následující členy této struktury je nutno nastavit pro správnou detekci.

minThreshold (float) - Počínající threshold
thresholdStep (float) - Krok navýšení thresholdu pro každý obraz
maxThreshold (float) - Konečný threshold
filterByArea (bool) - Aktivace filtru detekce dle velikosti shluku
minArea (float) - Nastavení filtru minimální velikosti shluku
filterByColor (bool) - Aktivace filtru detekce dle barvy
blobColor (float) - Nastavení barvy hledaných shluků
filterByInertia (bool) - Aktivace filtru detekce dle kulatosti (spektrum přímka–elipsa–kruh)

Nastavení thresholdů není příliš důležité, stejných výsledků by bylo dosaženo i při výchozích hodnotách. To je z důvodu, že je do metody detekce blobů zaslán již binární obraz. Hodnoty jsou nastavené pouze z důvodu rychlejšího průběhu programu. Nejdůležitějším parametrem je však kritérium velikosti shluku, díky kterému jsou ve sloučených snímcích odfiltrovány samostatné rozlišovací plošky. Podstatná je i aktivace filtrování dle barvy a následné přepnutí režimu na hledání bílých shluků (hodnota *filterByColor*=255). V poslední řadě je vhodné vypnout kritérium *Inertia ratio*, které popisuje poměr délek na sebe kolmých průměrů nalezeného shluku, neboť od sebe nepotřebujeme rozlišovat různé tvary.



Obr. 5.10: Aplikace funkce vyhledávání pixelových shluků.

Takto vytvořená struktura je zaslána konstruktoru při tvorbě instance třídy *cv2.SimpleBlobDetector*, na kterou je vzápětí aplikována metoda *detect*. Nejprve je vytvořeno několik binárních obrazů na základě zvolených thresholdů. V nich jsou dále jednotlivé shluky (oblasti sousedících bílých pixelů) seskupeny. Následně dojde k výpočtu středu pro každý blob. Pokud jsou dva sousedící středy blíže jak stanovená hodnota *minDistBetweenBlobs*, dojde k jejich sloučení. Posledním krokem je odhad přesnějšího umístění středu a velikosti poloměru každého seskupení. Tyto hodnoty jsou navráceny ve formátu listu instancí třídy *cv2.keypoints*. Jejich vizualizace je k vidění na obrázku 5.10.

5.3.3 Klasifikace nalezených shluků

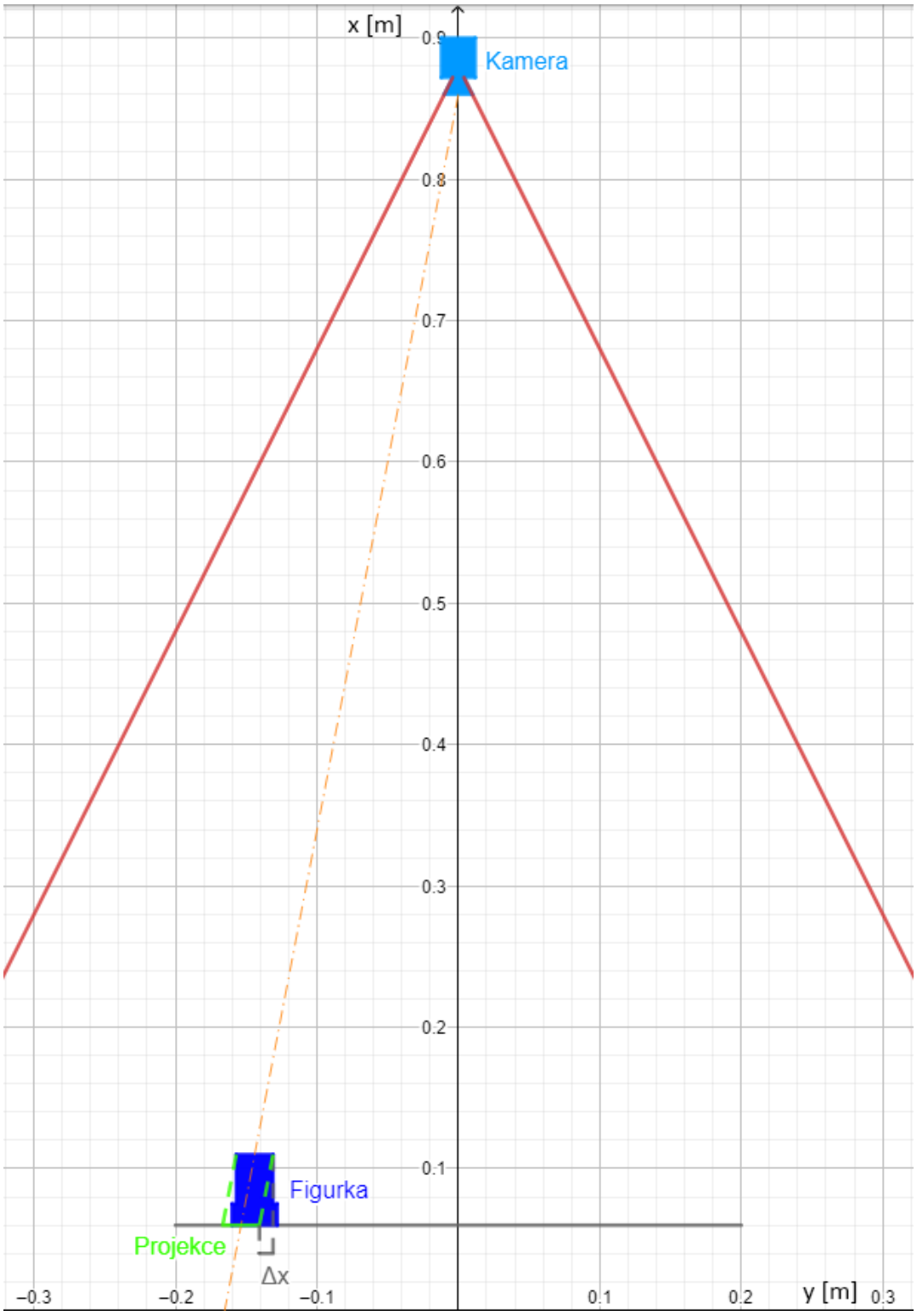
Výsledkem detekce pixelových shluků jsou tři listy blobů (pro každou barvu jeden). Ty je potřeba klasifikovat a získat tak seznam figurek, jejich hráčské příslušnosti a hodnosti. Pro každého z hráčů probíhá klasifikace zvlášť, přičemž je zapotřebí užít data nejen blobů daného hráče, ale i data blobů rozlišovacích plošek. Z dat o každém blobu (i hráče, i plošek) je nejprve extrahována jeho pozice, od které je následně odečten offset ořezu (výsledkem musí být pozice v přesném ořezu šachovnice).

$$d = \sqrt{(X_{piece} - X_{mark})^2 + (Y_{piece} - Y_{mark})^2} \tag{5.5}$$

Pro každou nalezenou figurku je zjištěno, zdali se jakákoliv z detekovaných rozlišovacích plošek nenachází v její bezprostřední blízkosti. Bere v potaz se vzdálenost 32 px, která vyjadřuje v pixelech velikost poloviny políčka (dle rovnice 5.5). Jelikož celý tento algoritmus probíhá uvnitř metody třídy *player*, výsledná data jsou ukládána každým průběhem smyčky ve formátu instancí třídy *piece* do atributu *player.pieces*.

5.3.4 Normalizace perspektivy

Posledním problémem přesné detekce figurek je jejich výška. Čím vyšší jsou snímané figurky, tím více jsou od sebe vzdálené plochy snímané části a šachovnice, čímž vzniká zkreslení detekované plochy o Δx (obrázek 5.11). Stejně tak roste velikost snímané boční plochy figurky. Tento problém byl hlavním důvodem změny algoritmu z detekce kruhů (pomocí Houghovy transformace) na detekci těžišť pixelových shluků. Tím, že není přesná pozice horního kruhu k dispozici, je komplikovanější sestavit rovnici na takový přepočet. Proto byly rovnice stanoveny experimentálně z naměřených dat.

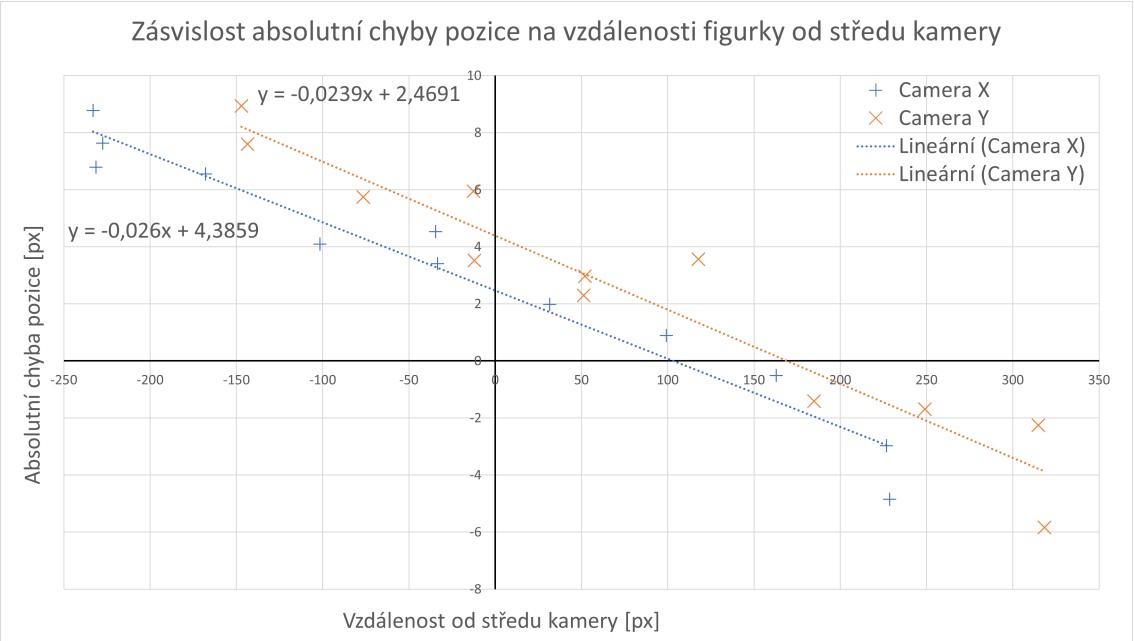


Obr. 5.11: Zkreslení pozice figurky perspektivou kamery.

Nejprve bylo manipulátorem na herní plochu rozmístěno několik figurek, čímž byla zjištěna jejich přesná poloha v jeho souřadném systému. Následně byla prove-

dena detekce jednotlivých figurek v souřadném systému ořezu kamery. Posledním chybějícím parametrem byla pozice středu kamery také v souřadném systému ořezu kamery.

Získané hodnoty byly vykresleny do grafu závislosti absolutní chyby na vzdálenosti figurky od středu kamery. Výsledek byl proložen přímkou a metodou nejmenších čtverců byly vypočteny rovnice přepočtu. Výsledný graf je k vidění na obrázku 5.12.



Obr. 5.12: Graf zkreslení pozice figurky perspektivou kamery.

Výsledkem proložení jsou tedy dvě rovnice. První je pro body v ose X (rovnice 5.6) a druhá v ose Y (rovnice 5.7).

$$X_{norm} = -0.026X_{raw} + 4.3859 \tag{5.6}$$

$$Y_{norm} = -0.0239Y_{raw} + 2.4691 \tag{5.7}$$

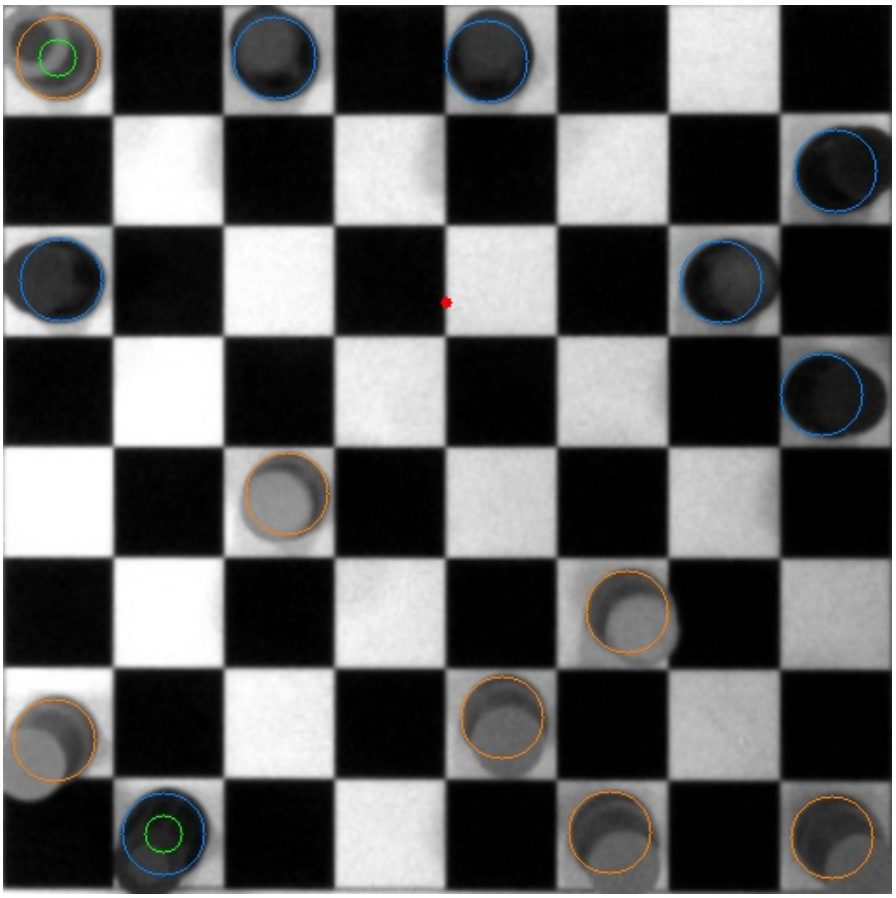
Jelikož v momentě měření a sestavování rovnic nebyla šachovnice umístěna přesně v souřadném systému manipulátoru, vypočtený offset není na rozdíl od činitele dostatečně přesný. Rovnice tak musely být ručně upraveny, aby dosahovaly žádoucích výsledků. Výsledky těchto úprav jsou uvedení níže v rovnicích 5.8 a 5.9.

$$X_{norm} = -0.026X_{raw} + 6.2 \tag{5.8}$$

$$Y_{norm} = -0.0239Y_{raw} + 4.4 \tag{5.9}$$

5.3.5 Výsledek detekce herní plochy

Výsledek celkové detekce herní plochy v reálném čase je graficky znázorněn na obrázku 5.13. Jak lze vidět, barevný kruh představující detekovanou pozici je vždy u paty figurky. Nepřesnosti celého algoritmu se pohybují okolo jednoho pixelu.



Obr. 5.13: Výsledek detekce herní plochy v reálném čase.

Celý výsledný skript pro detekci obrazu byl minimálně závislý na proměnlivosti světelných podmínek za předpokladu, že se hraje na bílých polích. Hra na černých polích nebyla dostatečně otestována, ale při všech pokusech dosahovala totožných výsledků.

5.3.6 Konečný přepočet na reálné souřadnice

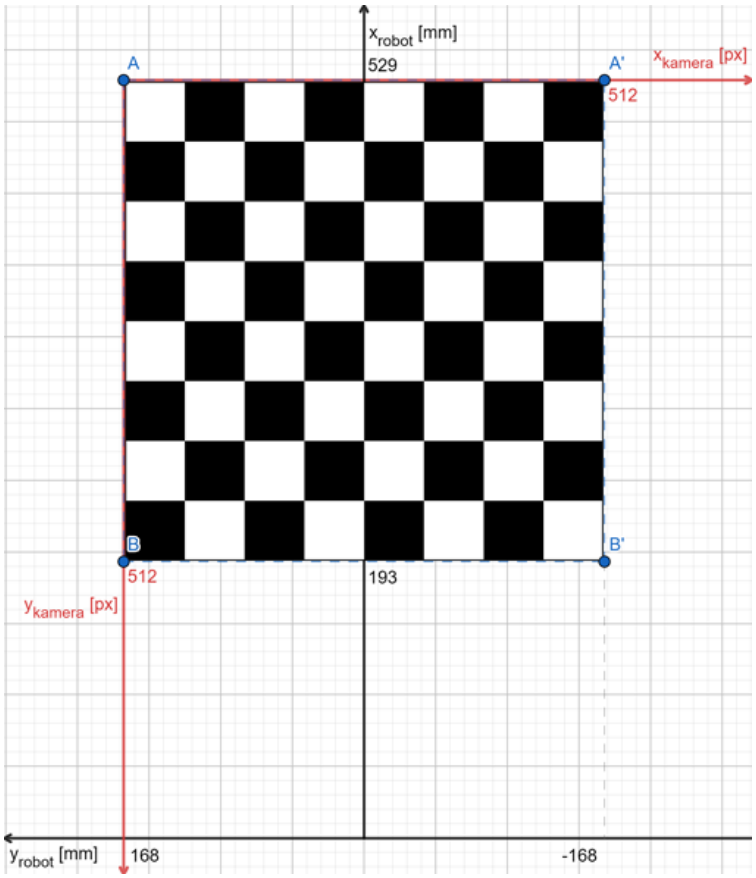
K přepočtu ze souřadnicového systému kamery na souřadný systém manipulátoru dochází až v uzlu *game_interface*. Hodnoty užité k přepočtu jsou definovány v souboru */config/dimension_data.yml*. Ty vyjadřují umístění šachovnice v souřadném systému manipulátoru v milimetrech. Rovnice výpočtu souřadnice x v souřadném systému manipulátoru je rovnice 5.10, rovnice výpočtu souřadnice y je rovnice 5.11.

$$x_{real} = y_{cam} \cdot coef + x_{offset} \tag{5.10}$$

$$y_{real} = x_{cam} \cdot coef + y_{offset} \tag{5.11}$$

x_{real}y y_{real} - Pozice v kartézském souřadném systému manipulátoru [mm]
x_{cam}, y_{cam} - Pozice v obraze kamery [px]
x_{offset}, y_{offset} - Pozice počátku souřadného systému kamery v souřadném systému manipulátoru [mm]
y_{coef} - Koeficient přepočtu z pixelů na milimetry, je vždy záporný [px $\cdot$ mm^{-1}]

Z rovnice je patrné, že pro výpočet souřadnic osy x manipulátoru jsou užity hodnoty na ose y snímku kamery a naopak. Také je v popisu zmíněno, že koeficient přepočtu je vždy záporný. To je z důvodu, že oba souřadné systémy nejsou souhlasně orientované. Tyto rovnice byly tak sestaveny dle obrázku 5.14.



Obr. 5.14: Vzájemný vztah souřadnicových systémů kamery a manipulátoru.

6 Výběr a implementace enginu hry dáma

Rešerše herních enginů již byla provedena v rámci semestrálního projektu předmětu Robotika. Obsahuje popis čtyř různých open source algoritmů v jazyce Python, které jsou zdarma k dispozici na síti *github.com*. Je sepsána stručně a velmi obecně, proto zde bude rozebrána podrobněji. [2]

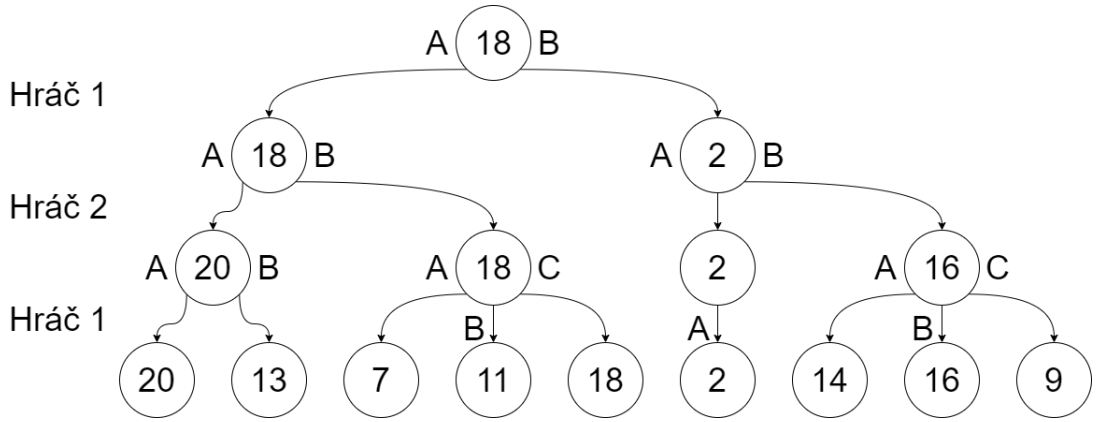
Dalším kritériem, které vyšlo najevo v průběhu implementace, je nutnost, aby byl daný engine napsaný v jazyce Python verzi 2. To je z důvodu, že byl celý systém od začátku tvořen na verzi ROSu jménem *Melodic*, která používá ve výchozím nastavení právě tuto verzi Pythonu.

6.1 Rešerše

Na rozdíl od předcházejícího projektu je zdejší rešerše koncipována odlišně. Nejsou zde rozepsány konkrétní open source zdrojové kódy, ale principy různých způsobů rozhodování. Ty jsou probrány velmi povrchově, jelikož se práce věnuje především platformě ROS a její komunikaci s manipulátorem Fanuc.

6.1.1 Minimax

Jedná se o nejpoužívanější vyhledávací strom pro jeho jednoduchost. Principiálně funguje tak, že má stanovenou určitou hloubku, do které bude vyhledávat. Tato hloubka značí kupříkladu počet tahů dopředu. Zjistí každý možný tah a další jeho návaznosti do stanovené hloubky, přičemž každému z nich přiřadí určitou hodnotu. Následně vybere větev, kde mají jeho tahy nejvyšší hodnoty a zároveň tahy protivníka ty nejnižší. Celá tato podkapitola čerpá ze zdroje [14].

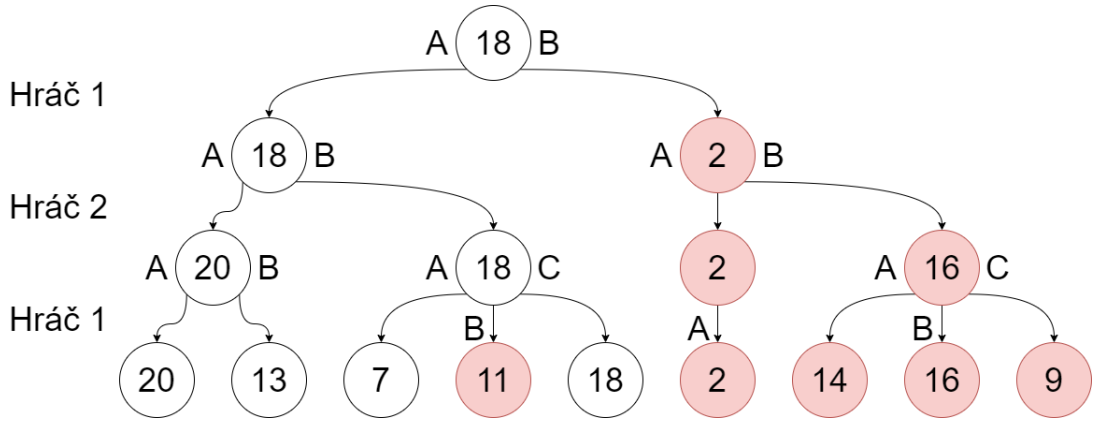


Obr. 6.1: Rozhodovací strom minimax algoritmu.

Hodnoty listů (konečná hodnota jedné větve) jsou stanoveny na začátku a reprezentují výslednou hodnotu herní situace po provedení sérii tahů vedoucí právě k této situace. Následně se rekurzivně plní jednotlivé uzly podle stanovených pravidel. Uveďme si příklad na obrázku 6.1. Hráč 1 se snaží dosáhnout na konci nejvyšší hodnoty a hráč 2 naopak té nejnižší. Hráč 1 tedy z nejnižší vrstvy pro každý rodičovský uzel vyšší vrstvy vybere vždy nejvyšší číslo. Předpokládá, že hráč 2 v dalším kroku vybere to nejnižší. Nakonec sám vybere to nejvyšší. Tím má zajištěno, že nejmenší hodnotou, kterou může získat, je právě 18. Tento algoritmus předpokládá, že každý z hráčů provede každé kolo pro sebe nejvýhodnější tah.

Alfa-Beta ořezávání

Přestože minimax algoritmus funguje spolehlivě, s rostoucí hloubkou roste exponenciálně i jeho výpočetní náročnost a doba. Některé výpočty však už od samého počátku budou nepodstatné. Pokud se však ihned zahodí, dojde ke značnému zkrácení doby výpočtu. Ukončení výpočtu určité větve nastane ve chvíli, kdy se prokáže, že již nemůže dojít do stavu, kdy by byl lepší než nejlepší předchozí. Z tohoto důvodu záleží na pořadí, v jakém se jednotlivé větve počítají. Celá tato podkapitola čerpá ze zdroje [14].



Obr. 6.2: Rozhodovací strom minimax algoritmu s alfa-beta ořezáváním.

Jak lze vidět na obrázku 6.2, červeně označené uzly představují ty, které se již dále nebudou počítat. Postupujeme dle konvence zprava doleva. První větev je vypočítaná vždy celá a každá následující podle toho, zdali je výhodnější. Výsledek bude vždy stejný jako u čistého minimax algoritmu. Avšak jeho rychlejší průběh není zaručen, jen velmi pravděpodobný.

6.1.2 Zpětnovazební učení (Reinforcement Learning)

Jedná se o typ strojového učení, ve kterém probíhá řešení otázek typu *„Co dělat?“* nebo *„Jak vyhodnotit situaci?“* za účelem získání co nejvyšší číselné odměny. Ve své podstatě se jedná o učení v uzavřené smyčce, protože počínání sítě ovlivní její vstupy v budoucnu. Učení navíc probíhá metodou pokus–omyl, takže síť musí sama zjišťovat, jaké akce vykonávat, a které z nich přinášejí největší odměny. V těch nejzajímavějších případech může počínání sítě ovlivnit nejen bezprostřední odměnu, ale také odměny v následujících situacích. Zpětnovazební učení je definováno třemi hlavními charakteristikami, mezi které patří následující. [15]

- Učení v uzavřené smyčce
- Absence jakýchkoliv instrukcí co dělat
- Následky různých činů mají dopady i po delší době

Ve standardním modelu zpětnovazebního učení je agent připojen do svého prostředí pomocí schopností vnímání a provedení určité akce. V každém kroku vzájemné interakce obdrží agent vstup i a informace o aktuálním stavu prostředí s. Následně zvolí akci a, aby vygeneroval patřičný výstup. Ten změní aktuální stav prostředí, přičemž změna tohoto stavu je agentu zpět zaslána v podobě skalární proměnné r *(reinforcement signal)*. Agentovo chování B by mělo vybírat takové akce, které z dlouhodobého hlediska zvyšují celkovou sumu příchozího signálu r. [16]

Q-učení (Q-learning)

Představuje formu zpětnovazebního učení bez modelu. Agent takového typu strojového učení disponuje schopností učit se optimálně reagovat v Markovově oblasti tím, že dostává zpětnou vazbou důsledky svých činů bez toho, aby si potřeboval tyto oblasti zmapovat. Agent tedy provede akci v určitém stavu prostředí, vyhodnotí její následky v podobě bezprostřední odměny (nebo penalizace) a odhadne hodnotu stavu, do kterého se tak dostane. Neustálým zkoušením všech možných akcí ve všech možných stavech se postupně učí, které postupy z dlouhodobého hlediska skýtají největší odměny. Jedná se stále o primitivní formu učení, která je ale schopna ovládat a řídit i sofistikovanější zařízení. Celá podkapitola čerpá ze zdroje [17].

Představme si agenta pohybujícího se uvnitř konečného světa, přičemž vybírá z konečného množství možných akcí každý krok. Svět je tvořen kontrolovaným Markovovým rozhodovacím procesem a agentem, který jej kontroluje. V kroku n je agentu znám stav světa $x_n(\in X)$ a může vybrat akci $a_n(\in A)$. Agent obdrží pravděpodobnostní odměnu r_n, jejíž průměrná hodnota $R_{x_n}(a_n)$ závisí pouze na stavu a akci dle rovnice 6.1.

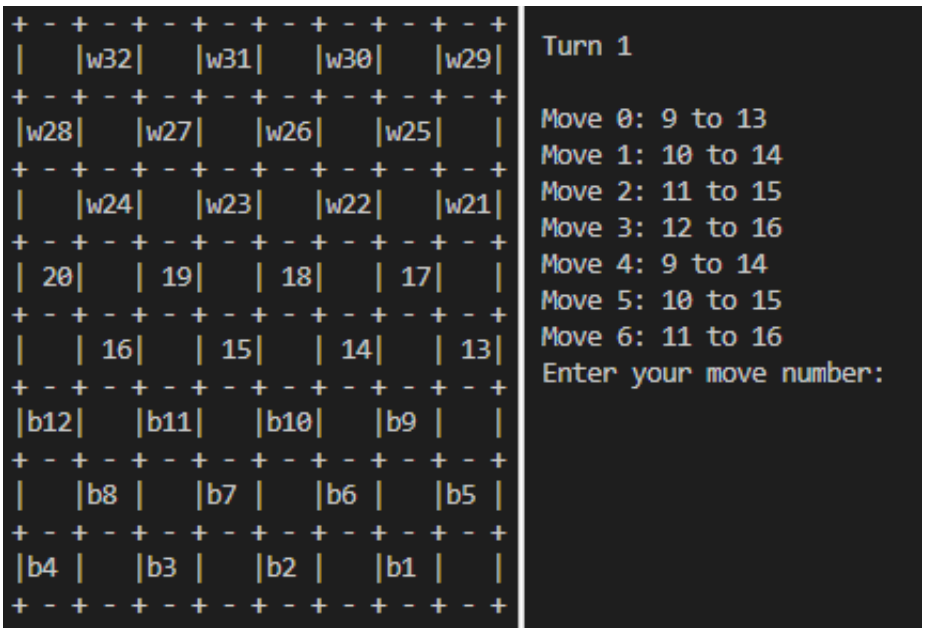
$$Prob\left[y_n = y|x_n, a_n\right] = P_{x_n y}(a_n) \tag{6.1}$$

Agent se snaží jednat tak, aby maximalizoval celkovou očekávanou znehodnocenou odměnu (discounted reward). znehodnocenou odměnou je na mysli suma jednotlivých odměn obdržených v rámci s kroků, tudíž má nižší hodnotu než bezprostřední odměna podle faktoru znehodnocení $\gamma^s (0 < \gamma < 1)$. Podle politiky π je hodnota stavu x stanovena rovnicí 6.2.

$$V^{\pi}(x) \equiv R_x(\pi(x)) + \gamma \sum_y P_{xy}[\pi(x)] V^{\pi}(y) \tag{6.2}$$

6.2 Implementace zvoleného enginu

Jednotlivé enginy v předchozí rešerši nebyly vybrány náhodou. Jedná se nejčastější používané algoritmy v open source projektech pro hru dámy. Nakonec byl vybrán balíček *almostimplemented/checkers*, který je postaven na základě prvního programu pro hru dámy od Arthura Samuela. Ten vycházel především z minimax algoritmu s alfa–beta ořezáváním [19]. Repositář je k dispozici ve zdroji [18].



Obr. 6.3: Původní uživatelské rozhraní zvoleného enginu.

Jedná se o jednodušší kód, který komunikuje s uživatelem za pomoci konzole a je napsán ve verzi Pythonu 2. Engine není příliš silným protivníkem, ale zároveň dokáže lidského hráče potrestat za jeho chyby. Jedná se tedy o ideální jednoduchý engine, jehož implementace bude snadná a hra proti němu nebude ani příliš obtížná,

ani příliš jednoduchá. Uživatelské rozhraní je tvořeno ze dvou částí. První z nich je vypsání aktuálního stavu hry, kde očíslovaná políčka reprezentují pole, na kterých se hraje a znak před jejími čísly představují figurku, která na nich stojí (b - černý pěšec, w - bílý pěšec, B - černá dáma, W - bílá dáma). Druhou částí je vypsání aktuálního kola a výčet možností, které lidský hráč může provést. Ten pak zadá do konzole pořadí svého vybraného tahu a řada přejde zpět na herní algoritmus. Tento způsob zadávání usnadňuje tvorbu budoucího algoritmu, protože je možné využít jeho informaci o možných tazích lidského hráče.

6.2.1 Oprava chyb zvoleného enginu

Užitý skript před samotnou implementací bylo zapotřebí opravit, protože obsahoval celkem dvě chyby. Obě se týkaly chybného vypisování validních lidských tahů v určitých momentech.

První z chyb bylo špatné vypisování možných tahů u vícenásobných přeskoků. Jejich zadávání totiž funguje postupně, tedy přeskok každé jednotlivé figurky ve vícenásobném přeskoku je hráči vypsán a uživatel jej následně potvrdí. Problém nastal v momentě, kdy před začátkem takového vícenásobného přeskoku měl hráč k dispozici i další možné přeskoky soupeřových figurek. V jednotlivých fázích vícenásobného přeskoku se totiž hráči vypisovaly všechny možné přeskoky, což logicky podle pravidel nedávalo žádný smysl. Chyba byla odstraněna editací funkce *get_move_strings* (která byla později přejmenována na *get_move_tuples*) tak, že pokud zjistila, že se nachází uprostřed vícenásobného přeskoku, navrátí pouze ty hodnoty, které se ho týkají.

Druhou z nich bylo chybné vypisování u některých přeskoků ve směru dopředu doleva. Hráči přicházela informace, že mohou skočit o jedno pole více doleva, než je podle pravidel možné. Tento problém byl obdobně jako ten předchozí odstraněn editací funkce *get_move_strings*.

6.2.2 Vytvoření uzlu herní logiky

Aby se dal herní engine implementovat do celého systému, bylo zapotřebí nejen upravit uživatelské rozhraní, ale i vytvořit komunikační knihovnu, která by řešila veškeré příchozí i odchozí zprávy. Takto byl vytvořen skript *robocheckers_link.py* a na straně původního kódu byl převážně předělán pouze skript *game.py*, který zprostředkovával komunikaci s uživatelem.

Nejprve z původního programu byly odstraněny možnosti hry více lidských hráčů, více počítačů (protože systém podporuje pouze hru člověka proti počítači) a možnost volby agenta. Zadávání informace o začínajícím hráči bylo zprostředkováno skrze vnitřní ROS komunikaci. Následně byla vytvořena knihovna na komunikaci

se zbytkem systému, která byla implementována do hlavního skriptu celého herního enginu, a tak vznikl nový uzel jménem *game_logic*.

Sekvence celé herní logiky je znázorněna na obrázku 6.4 (zobrazená sekvence popisuje případ, kdy je začínajícím hráčem právě člověk).

Obr. 6.4: Algoritmus uzlu herní logiky.

Pokud je jako začínající hráč zvolen počítač, po prvotní inicializaci dojde hned na zaslání této situace do agenta, odkud už hra pokračuje ve smyčce podle předcházejícího diagramu.

Ačkoliv kontrola platnosti lidského tahu probíhá každý cyklus, neměla by nastat situace, kdy dojde k vyhodnocení tahu jako neplatný. Ověřovaní validity uživate-

lových tahů probíhá v herním uzlu *game_interface*, odkud přicházejí již zaručeně platné tahy. Pro větší robustnost a spolehlivost kódu je však tato teoretická situace ošetřena z obou uzlů.

Přiřazení spouštěcího souboru

Značná část tohoto uzlu nebyla vytvořena v rámci této práce. Proto jsem se rozhodl jej nevkládat do balíčku *robocheckers*, ale byl vytvořen nový balíček s názvem *almostimplemented*. V balíčku *robocheckers* byl následně vytvořen spustitelný soubor *game_logic.launch*, který volá hlavní skript *game.py*, kde následně dojde k inicializaci uzlu a podobně. To je z důvodu, aby všechny uzly celého systému byly spustitelné právě z balíčku *robocheckers* za pomoci následujícího příkazu:

```
$ roslaunch robocheckers <název_uzlu>
```

7 Zásah lidského hráče a bezpečnost

V předešlých pracích bylo zvažováno, zdali zásah lidského hráče bude prováděn fyzicky, nebo z konzolové, či webové aplikace. V rámci této práce bylo definitivně rozhodnuto, že lidský hráč bude provádět tahy fyzickou interakcí s figurkami. S tím ovšem přichází otázky bezpečnosti. Kdy je bezpečné figurkou táhnout? Může nastat nějaká situace, kdy by došlo k poranění uživatele?

7.1 Bezpečnost

Aktuální bezpečnostní situace není dostačující pro obsluhu neškoleným uživatelem. Po dohodě s vedoucím práce nebyla bezpečnostní opatření implementována, protože je potřeba nejprve lépe danou problematiku rozebrat a až následně vybrat konkrétní vhodné řešení. Na přístupové straně manipulátoru byly demontovány dveře bezpečnostní klece a byla tak vyřazena jejich funkčnost. Proto se v oblasti manipulátoru mohou aktuálně pohybovat pouze řádně proškolení pracovníci.

Nejrychlejším možným řešením, jak uvést celý systém do bezpečného stavu i pro obyčejného laika s pouhým dohledem pověřené osoby je následující. Nejprve by bylo nutné zpět namontovat dveře bezpečnostní klece. Následně kdykoliv v průběhu programu (včetně tahu lidského hráče) dojde při otevření těchto dveří k překlopení režimu kontroléru robotického manipulátoru do režimu chyby. Tím je manipulátor bezpečně zastaven a nemůže se dále pohybovat. Zde však nastává mírná komplikace při potvrzování hráčem provedeného tahu. Před jeho provedením by musela řádně poučená dohlížející obsluha resetovat stav manipulátoru a následně opětovným stiskem zeleného tlačítka *Cycle Start* na předním panelu kontroléru manipulátoru obnovit chod pozastavených programů (předpokládá se, že kontrolér bude nastavený pro chod programů do režimu *auto*).

7.1.1 Instalace světelné závory

Dalším možným budoucím řešením je instalace takzvaných světelných závor. Ty se často používají v průmyslu v místech, kde obsluha například musí v určitý čas cyklu zasahovat do pracovní oblasti stroje. Jedná se o sérii páru laserových vysílačů a přijímačů, které jsou nasměrovány proti sobě. Tyto závory stejně jako všechny ostatní bezpečnostní prvky musí být napojeny na speciální bezpečnostní okruh. Jakmile dojde k přerušení jakéhokoliv páru vysílač–přijímač, dojde k okamžitému vyřazení všech akčních prvků (jako jsou například motory, písty a podobně) v okolí.

Výstup takové světelné závory by měl správně být napojen na safety okruh manipulátoru, což by mělo opět za následek potíž s přechody do chybového stavu. Avšak

je tady k dispozici ještě jedna možnost. Na již jednou užitém konektoru *CRMA59* je k dispozici vyvedený signál *01: XHOLD*. [9] Ten musí být neustále v logické jedničce, jinak manipulátor bude v režimu hold. Aktuálně je do této hodnoty přestaven softwarově. Odstraněním tohoto softwarového bypassu a následným připojením výstupu světelné závory právě na vstup *01: XHOLD* konektoru *CRMA59* by byl tento problém vyřešen. Po celou dobu přerušení jediného páru vysílač–přijímač světelné závory by došlo ke změně manipulátoru do stavu *HOLD*. V tomto stavu není manipulátoru umožněn jakýkoliv pohyb. Nakonec po potvrzení provedení tahu lidského protivníka by byly programy na straně manipulátoru opět uvedeny do provozu za pomoci signálu *START*.

7.2 Tahy figurkami

Tah lidského hráče je indikován svitem zelené kontrolky LED panelu. Během tohoto tahu je manipulátor nečinný a vyčkává lidského zásahu. Ten je prozatím zpracován tak, že hráč provede fyzicky na šachovnici tah a potvrdí jej v rámci konzolového rozhraní uzlu *game_interface* stiskem klávesy *Enter*. Provedený tah je následně zaslán do dvouvrstvého ověřovacího systému. Pokud je tah vyhodnocen jako platný, zhasne zelená kontrolka LED panelu a rozsvítí se bílá. Tím je jasně dáno najevo, že manipulátor je v pohybu, protože je na tahu zrovna počítačem ovládaný hráč.

7.2.1 Možné metody potvrzování tahů

Aktuální situace potvrzování provedených tahů není zcela ideální. Cílem je odstranit nutnost komunikace uživatele se systémem skrze počítač, na kterém je spuštěn. V ideálním případě by měl být systém naprosto soběstačně schopen rozeznat provedení lidského tahu, následně oznámit jeho přijetí libovolným zvukovým, či světelným signálem a následně se sám překlopit do tahu manipulátoru.

V následující sekci této podkapitoly jsou popsány možné vhodnější budoucí způsoby potvrzování tahů ze strany lidského protivníka.

Tlačítko připojené na vstupy kontroléru manipulátoru

Jednou z variant je připevnit tlačítko i s pouzdrem ke kleci manipulátoru. To následně připojit na digitální vstupy kontroléru robotického manipulátoru. Následovalo by řešení ze softwarové stránky tak, že by byl vytvořen program v jazyce KAREL, který by tento signál dále zpracovával. Nejprve by bylo nutné s příchodem tohoto signálu jednorázově vyresetovat chyby a obnovit chod programů. Jednorázově je myšleno to, že při dlouhém stisku se nebude reset provádět v nekonečné smyčce. Po jednom stisku se musí program překlopit do takového stavu, že provede reset a bude

vyčkávat, dokud digitální vstup odpovídající adrese zapojení tlačítka nebude zpět v nulové hodnotě. Tím by přestavení stavu manipulátoru do režimu fault již nebyl takový problém.

Další nutností by bylo upravit programy na ovládání vstupů a výstupů jak na straně kontroléru, tak na straně platformy ROS a jejich vzájemného komunikačního protokolu. Doposud totiž není podporováno čtení jakýchkoliv hodnot z kontroléru manipulátoru platformou ROS. Kontrolér by tedy neměl možnost, jak předat zbytku celého systému informace o potvrzení hráčova tahu.

Kontrola ukončení tahu dle informací ze světelné závory

V případě instalace a implementace světelné závory do systému, by světelná závora mohla sloužit jako prostředek k potvrzení tahů. Její aktuální výstup by se promítal do zvoleného flagu, ze kterého by následně uzel *game_interface* vyčítal jeho hodnotu za pomoci uzlu *io_control*. Program by tedy nevyčkával na vstup potvrzovací klávesy, ale na změnu hodnoty tohoto flagu. V momentě jeho změny stavu na logickou jedničku a zpět na logickou nulu (hráč vstoupí do pracovní oblasti manipulátoru, provede tah a následně jej opustí) by byla provedena kontrola platnosti takto provedeného tahu. Pokud by tah byl vyhodnocen jako platný, hráč by byl na tuto skutečnost upozorněn zvukovým, či světelným signálem a program by přešel do fáze tahu manipulátoru. V opačném případě by hráč byl informován o neplatnosti provedeného tahu světelným signálem a program by následně vyčkával na dvojí překlopení hodnoty vybraného flagu.

Tato varianta však opět požaduje vylepšení dosavadní komunikace mezi kontrolérem manipulátoru a platformou ROS o zasílání zpráv i v opačném směru, jinak není vyčítání hodnoty zvoleného flagu možné.

Periodická kontrola ukončení tahu

Poslední navrhovanou možností je periodicky kontrolovat změnu stavu šachovnice. Tato varianta je nejjednodušší na implementaci, neboť nevyžaduje pro svou funkčnost vyčítání hodnot registrů kontroléru platformou ROS. Její princip by spočíval v periodickém kontrolování změn stavu šachovnice co pevně stanovený časový interval. V případě, že nenastala žádná změna, by se program pouze vrátil na začátek smyčky kontroly provedení tahu. V případě neplatného tahu by došlo taktéž k navrácení na začátek smyčky a k tomu ke světelné signalizaci provedení chybného tahu. V posledním možném případě validního tahu by byl hráč upozorněn zvukovým, či světelným signálem, že byl tah přijat. Následně by proběhlo pozastavení programu, aby lidský hráč byl schopen na daný signál zareagovat a nakonec by došlo k překlopení stavu programu do tahu manipulátoru.

Pro tuto variantu vzniká hned několik menších problémů. Například pokud dojde k vyhodnocování tahu v době, kdy lidský hráč zasahuje do výhledu kamery, aktivuje se signalizace chybně provedeného tahu, což je zavádějící. Další nepříjemností může být vysoký časový interval, který bude nejen zpomalovat hru, ale program může i působit jako zamrzlý, což obyčejného laika může podněcovat ke zbytečným opakovaným zásahům do manipulačního prostoru robotu.

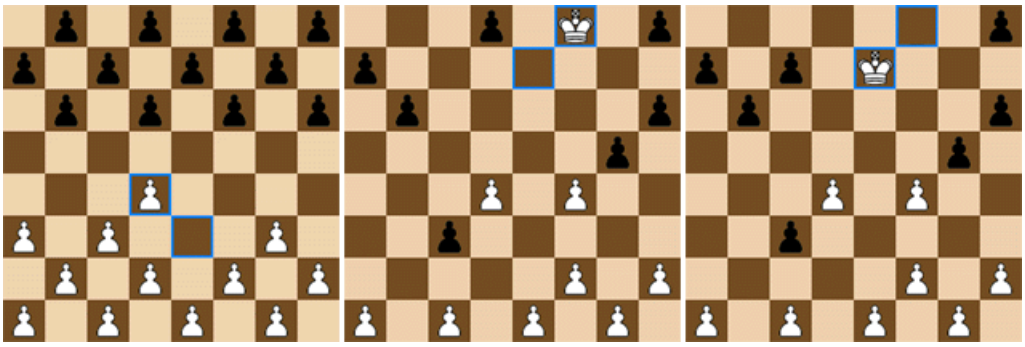
8 Kontrola dodržování pravidel

Jelikož ze strany lidského protivníka může docházet k provádění chybných tahů, či pokusů o podvod, je celý systém vybaven algoritmem na kontrolu pravidel. Ten funguje na bázi dvojího ověřování. Nejprve dojde k ověření v rámci uzlu herního rozhraní *game_interface*, kde probíhá hlavní část kontroly pravidel. Pokud ta proběhne úspěšně, následuje částečná urychlená kontrola i uvnitř uzlu *game_logic*.

8.1 Pravidla

Celý systém vychází z klasických pravidel americké dámy. Ta se hraje pouze na polích jedné barvy. Jednotliví hráči se na tahu střídají. Každý hráč začíná s dvanácti pěšci a cílem hry je zajmou všechny soupeřovy herní kameny. Pokud je na tahu hráč, který nemůže provést žádný pohyb podle pravidel, automaticky prohrává. Celá tato podkapitola čerpá ze zdroje [20].

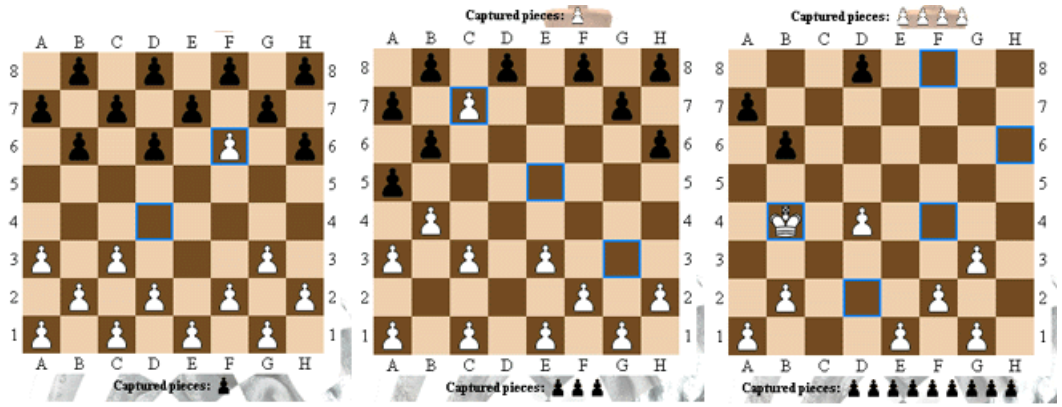
Jak je ukázáno na obrázku 8.1, pěšci se pohybují diagonálně pouze dopředu. Pokud pěšec přejde přes celé hrací pole až na poslední řadu, dojde k jeho povýšení na dámu (anglicky king - volně přeloženo). Dáma se stejně jako pěšec může pohybovat pouze o jedno pole (zde je největší rozdíl oproti české variantě pravidel) s tím rozdíle, že si může vybrat libovolný směr.



Obr. 8.1: Pravidla pohybu figurek (zleva doprava - pohyb pěšce, povýšení pěšce na dámu, pohyb dámy). [20]

Pokud se figurka libovolného hráče nachází na diagonálně sousedícím poli se soupeřovou figurkou za níž se ve stejném směru nachází volné pole, musí tuto soupeřovu figurku přeskokem zajmout a odstranit z herní plochy. Přeskoky se nesmí ignorovat. Pokud má hráč k dispozici více možných přeskoků, vybírá mezi nimi bez omezení. Pokud se figurka po prvním přeskoku dostane do další situace, kdy může zajmout další soupeřovou figurku, musí tak učinit znovu. Tento tah se nazývá vícenásobný

přeskok. Pokud je vícenásobný přeskok prováděn dámou, je možné v jeho průběhu měnit směry dílčích přeskoků. Všechny tyto typy přeskoků jsou znázorněny na obrázku 8.2.



Obr. 8.2: Pravidla přeskoku figurek (zleva doprava - přeskok, vícenásobný přeskok, vícenásobný přeskok dámou). [20]

8.2 Kontrola pravidel

Jak již bylo dříve zmíněno, kontrola pravidel probíhá ve dvou různých uzlech. Nejprve si rozebereme kontrolu probíhající v uzlu *game_interface*, protože se jedná o tu důležitější z nich. Zde před světelnou indikací hráčova tahu proběhne načtení informací o figurkách z uzlu *game_detection*. Následně je hráči signalizováno, že je na tahu. Jakmile ten provede a potvrdí svůj tah, dojde k porovnání nové situace na herní ploše s tou původní. Zapíši se všechny rozdíly a dochází k porovnání, které se skládá ze dvou částí.

První část je porovnání platnosti tahu vzhledem k pravidlům dámy. Nejprve je zjištěno, zdali lidský hráč odebral a přidal právě jednu figurku své vlastní barvy (přesun figurky představuje ve své podstatě odstranění figurky na jednom políčku a položení jiné figurky na políčko jiné). Následně je ověřena neplatná manipulace s dámami, jestli hráč například nepovýšil libovolnou figurku, aniž by dojel na konec herní plochy, a podobně. Dalším krokem je zjištění, jaký tah lidský hráč provedl. Ten lze klasifikovat do tří různých kategorií, které jsou jmenovitě *běžný tah*, *přeskok* a *vícenásobný přeskok*. Pokud je tah vyhodnocen jako libovolný typ přeskoku, dojde k zjištění, dali všechny přeskočené figurky byly ve validní trajektorii přeskoků. Pokud se jedná o obyčejný tah, je ověřeno, jestli byl proveden diagonálně a právě jen o jedno pole. Jestliže veškeré ověřování proběhlo úspěšně, program přichází do druhé části

ověřování, v opačném případě dochází k signalizaci porušení pravidel a program se vrací na začátek této smyčky, tedy do stavu, kdy vyčkává na lidský zásah.

Druhou fází je porovnání platnosti tahu vzhledem k návaznosti na předešlou situaci na herní ploše. Tato fáze kontroly probíhá přesně dvakrát. Nejprve v uzlu *game_interface* a následně v uzlu *game_logic*. Jak již bylo nastíněno v kapitole 6.2.2, uzel herní logiky generuje a následně zasílá list všech možných tahů, které lidský hráč v dané situaci může provést. Pokud tedy provedený tah projde první fází ověřování, je porovnán s možnostmi v tomto listu. Jestli dojde k nalezení shody, tah je odeslán do uzlu *game_logic*, kde dochází ke stejnému ověření se stejným listem. V tomto případě by situace, že by byl tah vyhodnocen jako neplatný až v uzlu herní logiky, nikdy neměla nastat, jedná se pouze preventivní o ošetření. Pokud k tomu přeci jen dojde, herní logika pošle uzlu herního rozhraní unikátní zprávu ve formátu "$[(-1,-1)]|[\ ]$". Ta uzel *game_interface* přesune zpět do stavu čekání na lidský zásah a tuto skutečnost hráči opět signalizuje.

8.2.1 Signalizace porušení pravidel

Pokud jakákoliv část kontroly pravidel popsané v předešlých odstavcích je vyhodnocená jako neplatná, je zapotřebí o této skutečnosti informovat lidského hráče. Toho je dosaženo za pomoci LED informačního panelu, kde zůstává svítit zelená kontrolka signalizující hráčův tah a zároveň se rozbliká červená kontrolka signalizující chybu. Tento stav je znázorněn na obrázku 8.3.

Obr. 8.3: Indikace porušení pravidel (zelená svítí, červená bliká).

Zároveň chyba, které se hráč dopustil, je blíže specifikována v konzolovém výpisu uzlu *game_interface*.

9 Obsluha systému a možná rozšíření

Systém je plně funkční a schopen předvedení názorné demonstrace. Tato kapitola je věnována převážně návodu k obsluze a následně se zde čtenář dočítá, jaká možná budoucí rozšíření je možné na tento systém navázat.

9.1 Konfigurační soubory

Zásah lidského hráče neprobíhá pouze na šachovnici. Některé informace je potřeba uživatelem zadat před spuštěním. Existují celkem tři konfigurační soubory, které má uživatel k dispozici. Všechny se nachází v adresáři *config* a všechny jsou vytvořeny ve formátu *YAML*.

Hlavní konfigurační soubor

Prvním z nich je hlavní konfigurační soubor *config.yaml*, jehož obsah vypadá následovně:

```
# Debug mode [True/False]
debug: False
# Camera ID [Integer]
camera_id: 0
# Robot IP address
robot_ip: 10.0.33.33
# Computer algorithm depth [Integer]
ai_depth: 5
# Color of the human pieces [Blue/Orange]
human_color: Blue
```

Lze zde zapnou debugovací mód, který aktivuje pokročilé vypisování chyb běhu programu. Tento mód je doporučeno zapnout pouze v případě spuštění jednotlivých uzlů zvlášť pro přesnější vypisování. Další z možností je *camera_id*. Jedná se o velice důležitou položku. Zadaná hodnota určuje vybranou kameru ze všech připojených podle jejich ID. Argument *robot_ip* určuje, na jaké IP adrese se bude platforma ROS snažit připojit ke kontroléru robotického manipulátoru. Další dvě položky jsou méně technické a týkají se spíše samotné hry. Prvním je nastavení hloubky, do které bude minimax algoritmus s alfa–beta ořezáváním počítat. S rostoucí hloubkou roste nejen obtížnost, ale také výpočetní náročnost a doba (na hloubce 7 mohl extrémně dlouhý výpočet trvat až 3 a půl minuty). Druhým z nich je barva figurek lidského protivníka (na velikosti písmen nezáleží), pokud je zadaná hodnota neplatná, hráči

jsou přiřazeny modré figurky. Všechny tyto proměnné jsou při spuštění souboru *load_config.launch* nahrány na parametrický server.

Konfigurační soubor souřadných systémů

Další dva konfigurační soubory již vyžadují pokročilejší znalost systému. Nejprve se pustíme do konfiguračního souboru, který má na starost přepočet pozic figurek z obrazu kamery na reálné umístění v souřadném systému manipulátoru, aby mohlo dojít k přesnému úchopu. Nejdůležitější je pro uživatele následující část:

```
pose_shift:
    coef: -0.65625
    offset_x: 527.523
    offset_y: 165.18
```

Pomocí těchto hodnot dochází k onomu přepočtu, jak již bylo zmíněno v podkapitole 5.3.6. Argument *coef* vyjadřuje poměr délky nasnímané šachovnice v pixelech vůči délky reálné v milimetrech. Obě hodnoty offsetů pak vyjadřují umístění počátku souřadného systému snímané šachovnice v souřadném systému robotického manipulátoru.

Konfigurační soubor statických pozic manipulátoru

Posledním konfiguračním souborem je seznam statických pozic manipulátoru. Jedná se o pozice, které se jednou nadefinují, přiřadí se k nim určitý klíč (název) a již se dále nemění. Každá pozice má následující formát:

```
default:
    position:
      x: 0.11
      y: -0.17
      z: 0.3
    orientation:
      x: 0.878
      y: -0.478
      z: 0.0
      w: 0.0
```

Úplně první slovo je klíč, tedy název, pod jakým je v programu tato statická pozice uložena. Následuje pozice v kartézském souřadnicovém systému a orientace ve formátu vektoru kvaternionu. Tyto hodnoty lze získat pouze za použití metody *get_current_pose()* instance třídy *moveit_commander.MoveGroupCommander()*.

Vyčítání hodnot z teach pendantu manipulátoru se nedoporučuje, neboť souřadné systémy v něm a v platformě ROS nemusí být souhlasné.

9.2 Kalibrační sekvence

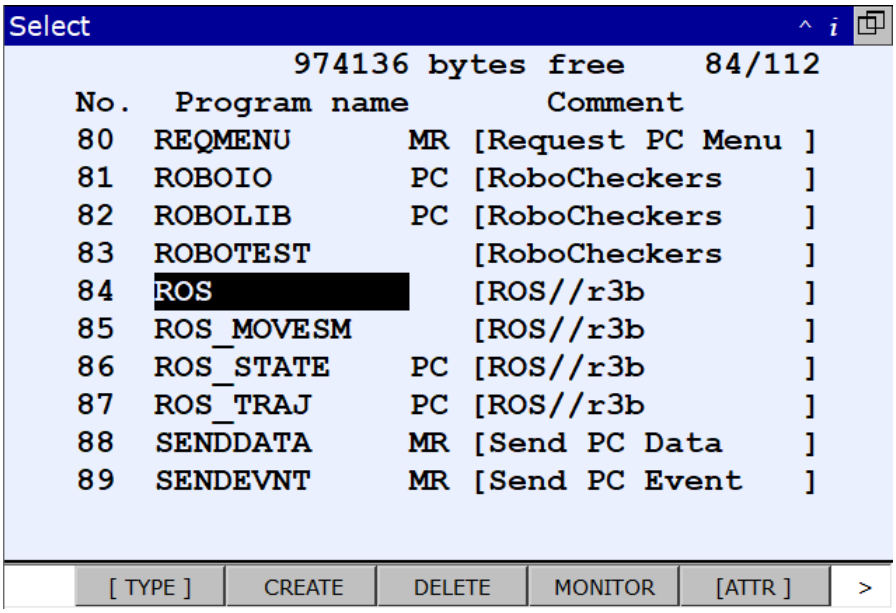
Aktuálně jsou v programu k dispozici pouze dvě kalibrační sekvence. První z nich je určena ke kalibraci kamery, zatímco ta druhá slouží ke kalibraci páru kamera–šachovnice. Ty byly již rozebrány v podkapitolách 5.1 a 5.2, zde je pouze zopakován způsob jejich spouštění. To probíhá dle následujících příkazů (první spouští kalibraci kamery, druhý kalibraci páru kamera–šachovnice):

```
$ roslaunch robocheckers calibrate_camera.launch
$ roslaunch robocheckers calibrate_board.launch
```

Výsledky kalibračních sekvencí jsou dále uloženy do souborů v adresáři *calibration*. Ty slouží pouze pro čtení a neměly by být za žádných okolností editovány běžným uživatelem.

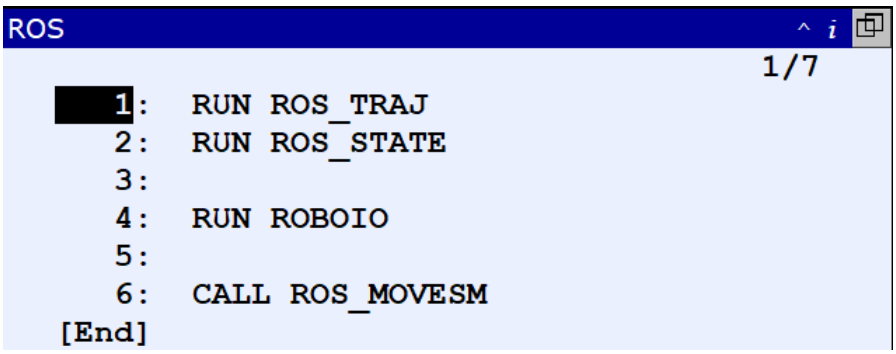
9.3 Spouštění systému

Jak již bylo dříve zmíněno, celý systém se skládá ze dvou částí. První z nich jsou programy běžící na kontroléru robotického manipulátoru a druhou je platforma ROS a její obsah.



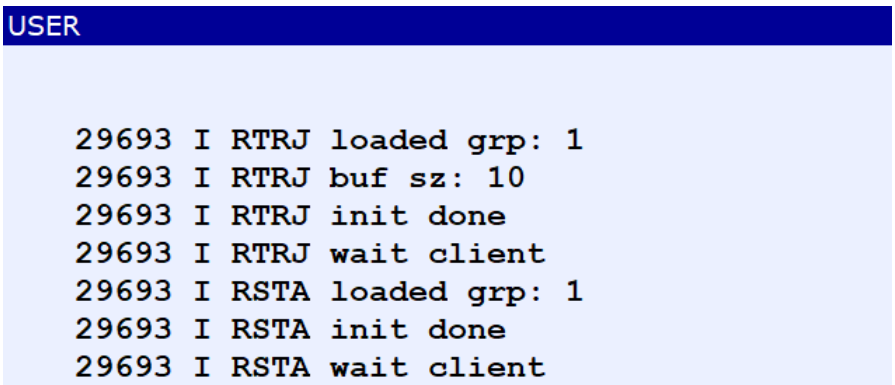
Obr. 9.1: Výběr programu na kontroléru manipulátoru.

Nejprve je nutné spustit programy na straně manipulátoru. Toho je dosaženo vybráním programu ROS, jak je znázorněno na obrázku 9.1, a následným stiskem tlačítka *Enter* na teach pendantu manipulátoru. Tím je program zvolen a také dojde k zobrazení jeho obsahu, který je k vidění na obrázku 9.2.



Obr. 9.2: Obsah programu ROS programu na kontroléru manipulátoru.

Jakmile je program zvolen, je nutno se ujistit, že je kontrolér v režimu *"AUTO"*. Nakonec stačí jen stisknout zelené tlačítko *Cycle Start* na předním panelu kontroléru, čímž dojde ke spuštění programů. Na obrazovce se zobrazí informace o očekávání připojení, jak je znázorněno na obrázku 9.3.



Obr. 9.3: Výpis programu ROS programu na kontroléru manipulátoru.

V tomto momentě už stačí inicializovat platformu ROS, spustit její programy a ty se automaticky napojí na programy kontroléru. Toho lze docílit zavoláním dvou následujících příkazů (jak již bylo zmíněno v podkapitole 3.2):

```
$ roslaunch robocheckers robot_socket_connect.launch
$ roslaunch robocheckers robocheckers.launch
```

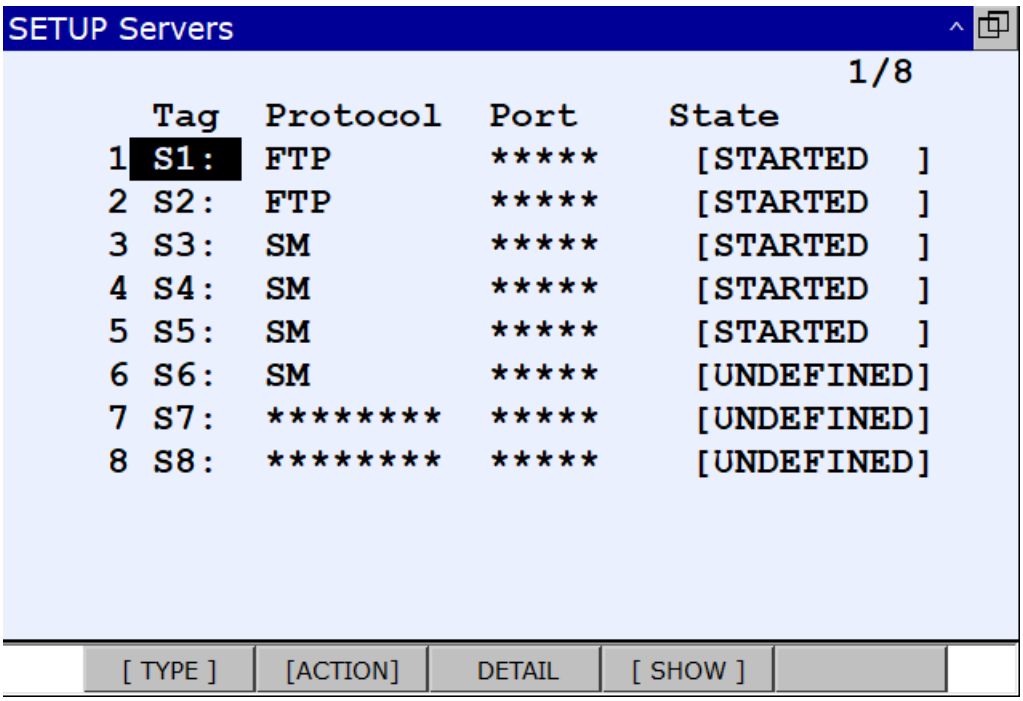
První z příkazů zavolá jednotlivé skripty nutné k připojení ke kontroléru. Pro uživatele je však důležitější druhý příkaz, který volá skripty nutné pro hru. Okno,

ve kterém bude tento příkaz použit, následně slouží jako konzolové rozhraní pro komunikaci s lidským protivníkem.

9.4 Instalace balíčku robocheckers na kontrolér

V této kapitole je stručně popsáno, jakým způsobem probíhá instalace ovladače RoboIO na kontrolér manipulátoru. Jedná se o ovladače, které slouží k ovládání vstupů/výstupů kontroléru. Nejprve je nutné z přílohy na flash zkopírovat disk soubory *roboio.pc* a *robolib.pc*. Ten se následně připojí ke kontroléru a dříve zmiňované soubory se zde přesunou pomocí teach pendantu.

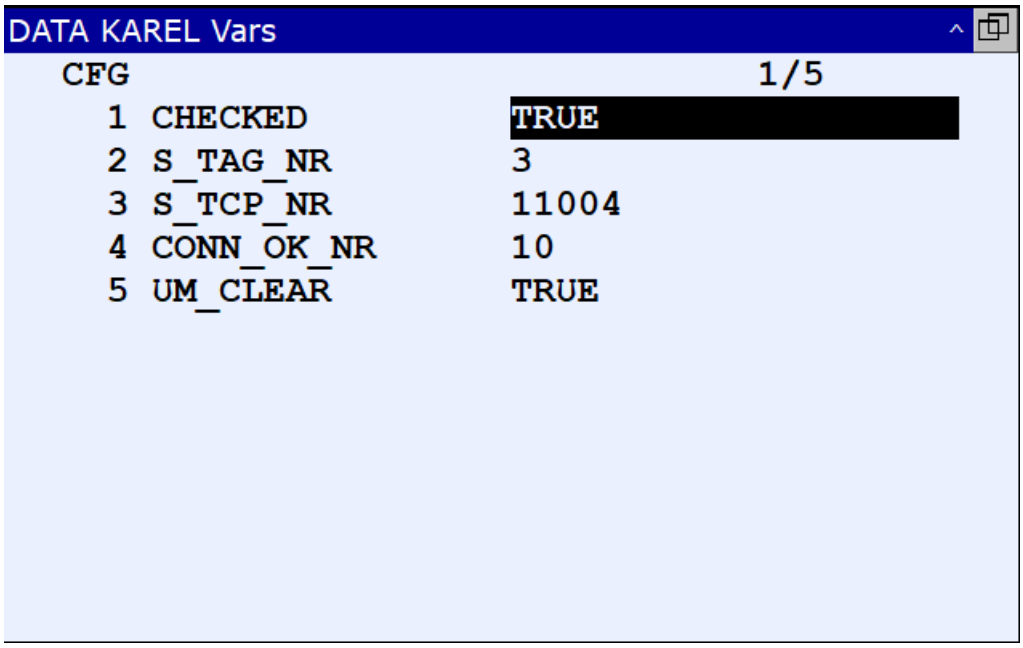
Obdobně jako programy *ros_state* a *ros_traj* je potřeba pro program *roboio.pc* vyhradit jeden server tag. Server tagy se nastavují v podmenu „Host Comm“, do kterého se přistupuje možnostmi „Menu -> SETUP -> Host Comm“. Dále je nutné stisknout „SHOW -> Servers“, čímž se zobrazí nastavené TCP/IP socket servery stejně jako na obrázku 9.4.



Obr. 9.4: Nastavení socket serveru.

Zde je zapotřebí vybrat libovolný nepoužitý server tag a nastavit jeho protokol na *SM (Socket messaging)*. Dále je nutno tento server definovat a spustit, čehož je docíleno za pomoci označení příslušného server tagu a následného zvolení možností „ACTION -> DEFINE“ a „ACTION -> START“. Zvolený server tag by nyní měl mít popisek *RUNNING*.

Nyní je nutné upravit vnitřní proměnné programu, aby věděl jaký server tag, nebo port má použít. Nejdříve je potřeba program zvolit. Toho je dosaženo stiskem klávesy „Select“ na teach pendantu, následným výběrem programu *ROBOIO* (tady pozor, je důležité vybrat program *roboio*, ne *robolib*) a stiskem klávesy „Enter“. Následně je zapotřebí stisknou na teach pendantu klávesu „Data“ a pokračovat možnostmi „Type -> KAREL Vars“. Otevře se seznam všech proměnných zvoleného programu. Dále volíme „CFG_T“ a potvrzujeme klávesou „Enter“.

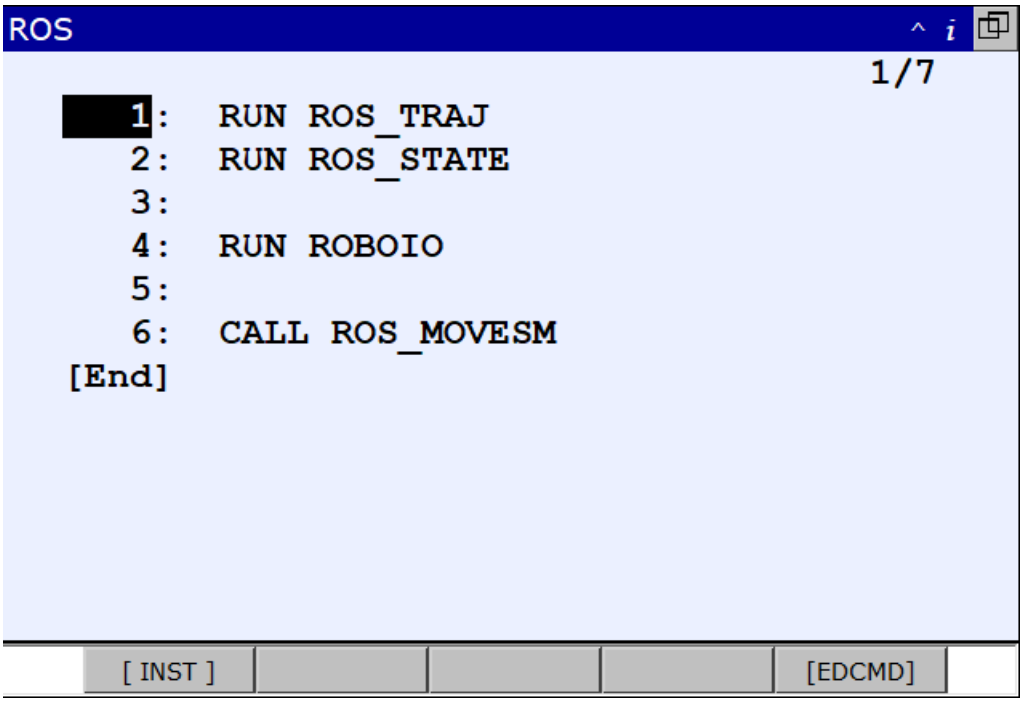


Obr. 9.5: Nastavení vnitřních proměnných programu *roboio*.

V tomto momentě je nutné nastavit proměnné „CHECKED“ a „UM_CLEAR“ na hodnoty „TRUE“ a „S_TCP_NR“ na hodnotu 11004. Vnitřní proměnná programu „S_TCP_NR“ představuje port, na kterém bude probíhat TCP/IP komunikace s platformou ROS, její hodnota je pevně daná. Proměnnou „S_TAG_NR“ je potřeba nastavit dle zvoleného server tagu, tedy pokud byl zvolen server tag „S3“, jeho hodnota bude 3. Poslední proměnnou je „CONN_OK_NR“, která nese informaci o flagu, který požívá k signalizaci, že přišla nová data. Doporučuje se vybrat adresa takového flagu, která ještě není obsazena. Všechny tyto vnitřní proměnné jsou k vidění na obrázku 9.5.

Posledním krokem je nutno přidat funkci volání programu *roboio* z programu „ROS“. Nejprve je tedy zapotřebí stisknout klávesu „Select“, zvolit program „ROS“ a potvrdit klávesou „Enter“. Dále je potřeba přidat prázdné řádky mezi instrukce „RUN ROS_STATE“ a „CALL ROS_MOVESM“ a na nich vytvořit další instrukci volání „RUN ROBOIO“. Je nutné použít instrukci „RUN“, instrukce „CALL“ nesmí

být použita z důvodu, že by se na ní zacyklil a nepokračoval by dále. Instrukce „RUN“ spouští programy na pozadí. Výsledek této úpravy lze vidět na obrázku 9.6.



Obr. 9.6: Nastavení vnitřních proměnných programu *roboio*.

Nyní při spuštění programu „ROS“ dojde i ke spuštění ovladače „RoboIO“ sloužícího k obsluze vstupů a výstupů kontroléru manipulátoru ze strany platformy ROS.

9.5 Možná rozšíření

Přestože je systém velmi obsáhlý, stále je mnoho možných rozšíření, o která by tento systém bylo možné vylepšit. Některá taková vylepšení již byla diskutována v rámci kapitoly 7. Tato podkapitola je však věnována stručnějšímu výčtu dalších uvažovaných vylepšení.

Vylepšená herní logika

Prvním velkým možným vylepšením je tvorba nové herní logiky. Aktuální uzel *game_logic* je sice dostačující, ale zato daleko od ideálu. Obtížnost je nižší a ve své podstatě téměř neměnná. Některé tahy při zvolené hloubce minimax algoritmu na hodnotu sedm trvaly až tři a půl minuty. To vypovídá o další nevýhodě zvoleného kódu, kterou je nízká rychlost.

Systém byl od počátku koncipován maximálně modulárně, aby k takovým změnám mohlo dojít jednoduše za předpokladu zachování dosavadní vnitřní komunikace.

Tvorba standalone systému

Dalším velkým uvažovaným vylepšením již od počátku diplomové práce je nahrání celé platformy ROS na externí hardware v podobě například Raspberry Pi. Toto Raspberry by následně bylo za pomoci rozhraní Ethernet skrze router propojeno s kontrolérem robotického manipulátoru. Zmiňovaný router by sloužil jako přístupový bod (drátový i bezdrátový).

Na zmiňovaném externím hardwaru by bylo zapotřebí do systému implementovat další uzel, který by spouštěl lokální webový server, který by sloužil jako uživatelské rozhraní. Uživatel by zde měl možnost spouštět a ukončovat jednotlivé hry, vybírat si obtížnost, barvu svých figurek, nebo i zobrazit aktuální situaci na herní ploše. K dispozici by byly i možnosti jako spouštění jednotlivých kalibračních sekvencí nebo editace konfiguračních souborů.

Uživatel by měl být schopen se jednoduše k tomuto webovému serveru připojit skrze dříve zmiňovaný router například za pomoci mobilního telefonu a Wi-Fi, nebo počítačem skrze kabelové připojení přes rozhraní Ethernet. Takto vytvořený systém by po uživateli nevyžadoval nic jiného než libovolné elektronické zařízení s nainstalovaným internetovým prohlížečem.

Společný topic pro zasílání chybových hlášek

S rozšiřujícím se systémem je vhodné přidat i společný topic pro sdílení chybových hlášek. Každý jednotlivý uzel by posílal své chybové zprávy se speciálním označením (například identifikační číslo uzlu, timestamp a kód chyby) na společný ROS topic. Z něj by následné mohl číst další uzel, který by zprávy logoval, nebo vypisoval uživateli. Pokud by byl celý systém předělán na standalone verzi popisovanou v předchozí podkapitole 9.5, mohlo by součástí webové stránky lokálního webového serveru být i konzole pro výpis takto přijímaných chybových zpráv.

Automatická kalibrace páru robot–šachovnice

Pokud dojde k odmontování držáčků šachovnice, musí proběhnout ruční kalibrační sekvence, výpočet a následné zapsání hodnot do souboru *dimension_data.yml* v adresáři *config*. Proces je poměrně zdlouhavý a nepřesný, manipulátorem v manuálním režimu *T1* jsou rozestavěny tři figurky na šachovnici a je zjištěna jejich přesná poloha. Následně je zjištěna jejich poloha v obraze kamery. Z těchto hodnot je vypočten offset pro obě osy, který je používán pro přepočet pozic všech figurek.

Návrhem pro automatickou kalibrační sekvenci systému je umístit figurky do zásobníku na dámy, kde mají všechny sloty pevně stanovenou pozici v souřadném systému manipulátoru (je definována v souboru *static_positions.yml*). Postupně tyto figurky umísťovat manipulátorem do jednotlivých rohů a středu šachovnice (pět figurek celkem). Jakmile jsou figurky rozmístěny, následuje nasnímání situace na herní ploše. Proběhne stejný výpočet jako u manuální sekvence páru robot–šachovnice a výsledné hodnoty jsou uloženy do souboru.

Výroba a napojení přechodové skříně pro IO kontroléru

Velmi vhodným rozšířením, které by mohl použít jakýkoliv budoucí projekt, je napojení všech signálů konektorů *CRMA58* a *CRMA59* do přechodové skříně. Podle zdroje [9] se jedná o pouhých 55 signálů (20 digitálních vstupů, 20 digitálních výstupů a 15 systémových signálů), 2 vstupy externího napájení a 2 potenciály (0 V a jištěných 24 V). Přechodovou skříň by bylo možné vyrobit z 19 trojpatrových svorek pro signály (7, 7, a 5), dvou svorek pro vstupy externího napájení a dvou potenciálových svorkovnic pro rozvod potenciálů 0 V a 24 V. Silně doporučuji pérové provedení svorek, protože není nutností lankové vodiče zakončovat dutinkou, ani pocínovat.

Toto rozšíření by bylo za účelem rapidního urychlení a zjednodušení budoucího vylepšování nejen této práce, ale i jakéhokoliv budoucího projektu spojeného s tímto konkrétním manipulátorem.

Update celé platformy ROS a vytvořených skriptů

Posledním uvažovaným rozšířením je update celé platformy ROS na nejnovější verzi a přechod na Pythonu verze 3. V tomto případě by bylo zapotřebí přepsat veškeré zdrojové kódy vytvořené v průběhu vývoje této práce tak, aby odpovídaly standardům, právě třetí verze Pythonu. Jedná se o vylepšení, které taktéž umožní jednoduší implementaci budoucích rozšíření (například webový server, nebo vylepšená herní logika).

Závěr

V první kapitole jsou čtenáři předvedeny, popsány a zhodnoceny celkem tři již existující systémy, které jsou schopny hrát dámu či jinou podobnou deskovou hru. Zpočátku je terčem popisu diplomová práce, která proběhla na stejném ústavu v roce 2012, akorát za použití jiného manipulátoru, kontroléru i systému. Následně je zmíněna práce skupinky lidí na projektu robotické ruské dámy. Zde bylo zdůrazněno, že systém používá pokročilejší herní AI AlphaZero a byl zmíněn inovativní systém pro indikaci „emocí“ robotického systému. V poslední řadě je předveden systém Square Off Swap. Ten měl zastoupit sériově vyráběné produkty v kategorii robotických systému pro hru deskových her.

Téma druhé kapitoly je zaměřeno na způsob ovládání manipulátoru. Ten je popsán z pohledu dění na kontroléru manipulátoru. Čtenáře seznamuji s principy vnější komunikace s platformou ROS na externím zařízení nejen za použití existujícího balíčku *fanuc_experimental*. Dopodrobna i rozebírám postup návrhu a tvorby mnou vytvořeného balíčku na ovládání vstupů/výstupů manipulátoru, jeho napojení na platformu ROS a tvorbu komunikačního protokolu. Následně popisuji myšlenkové pochody stojící za rozhodnutími týkajících se ovládání informačního LED panelu, který jsem mimo jiné musel i vlastnoručně zapojit. Kapitola nemá dostatek zdrojů, protože materiály o systémech Fanuc je těžko dohledatelný. Drtivá většina informací v této kapitole pochází hlavně z nabytých zkušeností v průběhu různých prací na manipulátorech Fanuc a také částečně z čtení cizích kódů (například Fanuc ROS ovladače).

Třetí kapitola pojednává o koncepci celého systému. Popisuji zde vnitřní strukturu systému ROS a jsou zde také rozebrány dopodrobna jednotlivé uzly tohoto systému. Nejdůležitější jsou však jednotlivé komunikační protokoly, které jsem pro komunikaci mezi jednotlivými uzly navrhl a realizoval. Proto jsou rozebrány velmi dopodrobna včetně formátů jejich zpráv. Ke konci popisuji, jakým způsobem je vhodné celý systém na straně PC spouštět a jak jsem jednotlivé spustitelné souboru navrhl, aby byly schopny předat programům v jazyce Python informace skrze parametrický server.

Kapitola čtvrtá je věnována fyzickým modelům hry. Probírám zde rozložení různých součástí pro hru v manipulačním prostoru robotického ramene, které jsem navrhl tak, aby využívaly prostor a kinematické vlastnosti manipulátoru co nejefektivněji. Mezi nimi jsou hlavně šachovnice, figurky, zásobník na dámy a koš na vyřazené herní kameny. Ke konci kapitoly dále představuji nový informační panel pro lepší komunikaci s lidským hráčem a vysvětluji jeho jednotlivé stavy. Od minulé práce došlo především ke změně barev figurek a kromě šachovnice jsou všechny ostatní zde popsané součásti zcela nové.

Pátou kapitolou je řešena problematika ohledně detekce herní plochy v reálném čase. Na začátek ukazuji pokrok, kterým uzel herní detekce prošel od výsledků minulé práce. Ten je znatelně rychlejší, přesnější a hlavně spolehlivější. Dále dopodrobna rozebírám kalibrační algoritmy samotné kamery a následně i páru kamera–šachovnice, který jsem vytvořil za účelem rychlejšího průběhu hlavní smyčky detekce. Pokračuji podrobným popisem celého detekčního algoritmu, kde nejprve dochází k aplikaci kalibračních dat a barevné filtraci na každý snímek pořízený v reálném čase. V profiltrovaných obrazech dochází k vyhledávání pixelových shluků a k určení hráčských příslušností a hodností. Nakonec je rozebírám, jakým způsobem jsem navrhl korekci dle perspektivy snímání kamery a přepočet na hodnoty reálného kartézského souřadnicového systému.

Šestá kapitola stručně probere několik vhodných algoritmů (u kterých byl nalezen alespoň jeden open source program) a následně vybere již existující program na principu jednoho z nich. Dále v ní popisuji implementaci zvoleného kódu, kde jsem musel kromě oprav chyb vytvořit knihovnu a navrhnout komunikační protokol, abych jej mohl napojit na zbytek systému uvnitř platformy ROS.

Kapitola sedmá pojednává o aktuálním stavu bezpečnosti manipulátoru pro běžného uživatele. Ta po dohodě s vedoucím nebyla doposud zajištěna z důvodu potřeby nejprve lépe danou problematiku rozebrat a až následně vybrat konkrétní vhodné řešení. Dále popisuji různé návrhy na její zavedení, například instalací světelné závory. V druhé části je diskutuji možné metody zpracování lidského zásahu. V některých variantách však bude nutné implementovat oboustrannou komunikaci mezi manipulátorem a platformou ROS.

V osmé kapitole je řešena převážně kontrola dodržování pravidel ze strany lidského hráče. Ty jsou kontrolovány dvoufázově a ve dvou různých uzlech. První fáze probíhá v uzlu *game_interface* a jedná se o kontrolu platnosti vzhledem k pravidlům, kterou jsem vytvořil za účelem přesné klasifikace tahu. Druhá fáze kontroly je provedena dvakrát, tedy v uzlech herního rozhraní a *game_logic*. Ta zjišťuje, jestli je provedený tah platný vzhledem k návaznosti na předchozí situaci.

V rámci poslední kapitoly jsem vytvořil několik jednoduchých návodů, které například popisují práci s konfiguračními soubory, spouštění kalibračních sekvencí a samotného systému, nebo postup instalace balíčku Robocheckers na kontrolér manipulátoru. V druhé části kapitoly diskutuji o možných budoucích rozšířeních systému. Jedná se o nápady, které bych chtěl sám implementovat, ale bohužel z časového hlediska to není možné. Například přetvoření systému na standalone zařízení, ke kterému by uživatel potřeboval pro ovládání pouze libovolné elektronické zařízení s internetovým prohlížečem.

Výsledkem této práce je systém schopen plnohodnotné hry dámy proti lidskému protivníkovi. Pokud však nedojde k upravení bezpečnostních podmínek, hry by se

neměl účastnit obyčejný laik. Demonstrační video zachycující funkčnost systému za těchto podmínek je součástí přílohy. Práce by nadále měla směřovat k vytvoření standalone systému, na který se bude možné připojit pomocí jakéhokoliv zařízení bez jakýchkoliv speciálních specifikací.

V průběhu realizace této práce mi značně pomohly zkušenosti, které jsem nabyl v rámci letní brigády roku 2021. Nejen, že jsem dostal omezený přístup k několika manuálům Fanuc, ale naučil jsem se i základy jazyků TP a KAREL. Ty jsem poté značně využil při psaní knihoven pro ovládání vstupů a výstupů kontroléru z platformy ROS. Celá práce je od úplného počátku koncipována tak, aby na ni bylo kdykoliv jednoduché navázat, což stále považuji za nejsilnější stránku celého systému.

Literatura

[1] LICHOSYT, David. *Programování výukového robotického manipulátoru*. Brno, 2020. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/126934. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Tomáš Lázna.

[2] LICHOSYT, David. *Robotická hra dáma*. Brno, 2021. MPC-RBT - Semestrální projekt. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Tomáš Lázna.

[3] FIREŠ, M. *Demonstrační úloha pro robotický manipulátor EPSON*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2012. 73 s. Vedoucí diplomové práce doc. Ing. Luděk Žalud, Ph.D..

[4] Ekaterina E. Kopets, Artur I. Karimov, Georgii Y. Kolev, Lorenzo Scalera, and Denis N. Butusov. *Interactive robot for playing russian checkers*. Robotics, 9(4), 2020. Dostupné také z: https://www.mdpi.com/2218-6581/9/4/107, doi:10.3390/ robotics9040107. 28

[5] NAIR, Surag. *A Simple Alpha(Go) Zero Tutorial* [online]. Prosinec 2017 [cit. 2021-12-21]. Dostupné z: https://web.stanford.edu/~surag/posts/alphazero.html

[6] SQUARE OFF, Inc. SWAP. *SquareOffNow.com* [online]. © 2015-2021 [cit. 2021-12-26]. Dostupné z: https://squareoffnow.com/product/swap

[7] CES 2019: Square Off Smart Chessboard. In: *Youtube* [online]. 14. 2. 2019 [cit. 2021-12-26]. Dostupné z: https://www.youtube.com/watch?v=6SON5sEAbX0&t=12s. Kanál uživatele nosillacast.

[8] GvdHoorn. *Installation of the ROS-Industrial driver on Fanuc controllers*. ROS Wiki [online]. Last modified on 9.10. 2019 [cit.2021-22-12]. Dostupné z: https://wiki.ros.org/fanuc/Tutorials/hydro/Installation#Prerequisites

[9] FANUC America Corporation. *R-30iB Mate/and R-30iB Mate Plus Controller Maintenance Manual*. Manuál údržby B-83525EN/09. © FANUC CORPORATION, 2012.

[10] Kaustubh Sadekar, Satya Mallick. *Camera Calibration using OpenCV*. LearnOpenCV [online]. Last modified on 25.02. 2020 [cit.2022-05-08]. Dostupné z: https://learnopencv.com/camera-calibration-using-opencv/

[11] Satya Mallick. *Geometry of Image Formation*. LearnOpenCV [online]. Last modified on 20.02. 2020 [cit.2022-05-08]. Dostupné z: https://learnopencv.com/geometry-of-image-formation/

[12] LearnOpenCV. Blob Detection Using OpenCV. *learnopencv.com* [online]. 2007-2022 [cit.2022-05-09]. Dostupné z: https://learnopencv.com/blob-detection-using-opencv-python-c/

[13] OpenCV. cv::SimpleBlobDetector Class Reference. *docs.opencv.org* [online]. Last modified on 23.12. 2016 [cit.2022-05-09]. Dostupné z: https://docs.opencv.org/3.2.0/d0/d7a/classcv_1_1SimpleBlobDetector.html

[14] *ES.268 The Mathematics in Toys and Games*, Spring 2011. [online]. (Massachusetts Institute of Technology: MIT OpenCourseWare), http://ocw.mit.edu [cit.2022-05-09]. Dostupné z: https://ocw.mit.edu/courses/es-268-the-mathematics-in-toys-and-games-spring-2010/resources/mites_268s10_ses4_ai/

[15] Richard S. Sutton, Andrew G. Barto. *Reinforcement Learning: An Introduction*. [online]. Cambridge, Massachusetts, London, England, 2014, 2015 [cit.2022-05-10]. Dostupné z: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

[16] Leslie P. Kaelbling, Michael L. Littman, Andrew W. Moore. *Reinforcement Learning: A Survey*, 1996 [online]. [cit.2022-05-11]. Dostupné z: https://www.jair.org/index.php/jair/article/view/10166

[17] Christopher J.C.H. Watkins, Peter Dayan. *Q-learning*. Kluwer Academic Publishers, Boston, 1992 [online]. [cit.2022-05-11]. Dostupné z: https://link.springer.com/article/10.1007/BF00992698

[18] Checkers [online]. Almostimplemented, 29.05.2017 [cit. 2022-05-11] Dostupné z: https://github.com/almostimplemented/checkers

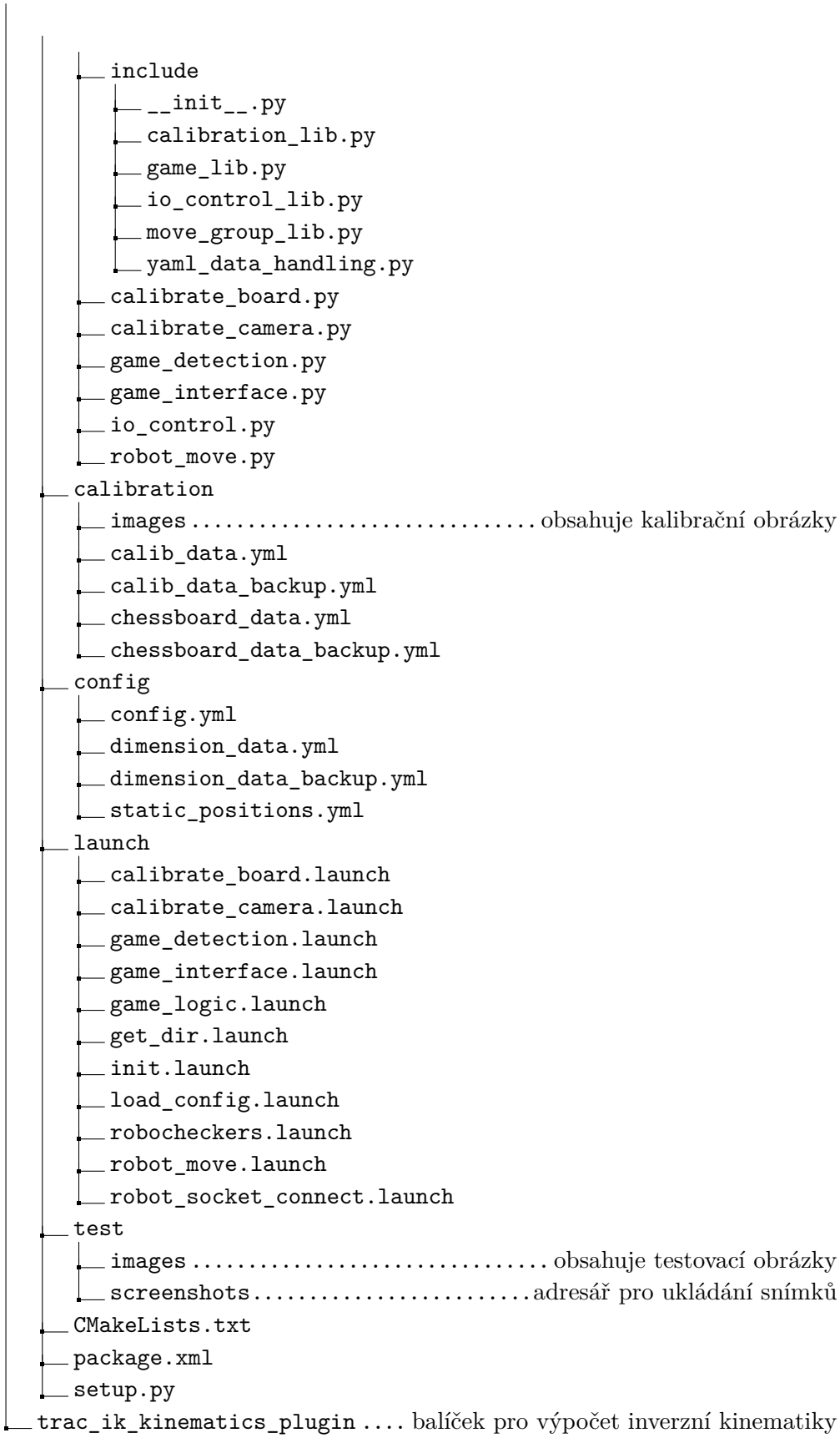
[19] History Computer Staff. Checkers, Arthur Samuel – Biography, History and Inventions. *history-computer.com* [online]. 19.10.2021 [cit. 2022-05-11] Dostupné z: https://history-computer.com/arthur-samuel-biography-history-and-inventions/

[20] Rachůnek Filip. Americká dáma. Brainking.com [online]. ©2002-2022 [cit. 2022-05-12]. Dostupné z: https://brainking.com/cz/GameRules?tp=7

A Seznam souborů

V příloze jsou celkem tři základní adresáře. První z nich jsou modely užité pro hru a systém. Další jsou programy vytvořené pro kontrolér manipulátoru, které slouží pro komunikaci s platformou ROS. Posledním adresářem je balíček platformy *ROS* verze Melodic Morenia obsahující všechny programy nutné pro spuštění hry. Samotné programy byly vytvořeny v jazyce Python verze 2.7.17.

```
/ ..... kořenový adresář přiloženého archivu
├── robocheckers.mp4 ..... demonstrační video
├── 3d_modely ..... fyzické modely
│   ├── king.STL
│   ├── king_recognition_part.STL
│   ├── kings_holder_base.STL
│   ├── kings_holder_left_leg.STL
│   ├── kings_holder_right_leg.STL
│   ├── pawn.STL
│   ├── status_panel_base.STL
│   ├── status_panel_front.STL
│   ├── status_panel_rear.STL
│   └── status_panel_top.STL
├── KAREL ..... programy vytvořené pro kontrolér manipulátoru
│   ├── include
│   │   ├── robolib_h.kl
│   │   └── robolib_t.kl
│   ├── roboio.kl
│   ├── roboio.pc
│   ├── robolib.kl
│   └── robolib.pc
└── Robocheckers ..... balíček platformy ROS
    └── src
        ├── almostimplemented ..... balíček herní logiky
        │   ├── bin
        │   │   ├── agent.py
        │   │   ├── arthur.py
        │   │   ├── game .py
        │   │   ├── checkers.py
        │   │   ├── logfile
        │   │   └── robocheckers_link.py
        │   ├── CMakeLists.txt
        │   ├── package.xml
        │   └── setup.py
        ├── fanuc ..... základní balíček Fanuc
        ├── fanuc_experimental ..... rozšíření Fanuc balíčku o LR Mate 200iD/4S
        └── robocheckers ..... balíček RoboCheckers řídící hru
            └── bin
```


include
__init__.py
calibration_lib.py
game_lib.py
io_control_lib.py
move_group_lib.py
yaml_data_handling.py
calibrate_board.py
calibrate_camera.py
game_detection.py
game_interface.py
io_control.py
robot_move.py
calibration
images ..... obsahuje kalibrační obrázky
calib_data.yml
calib_data_backup.yml
chessboard_data.yml
chessboard_data_backup.yml
config
config.yml
dimension_data.yml
dimension_data_backup.yml
static_positions.yml
launch
calibrate_board.launch
calibrate_camera.launch
game_detection.launch
game_interface.launch
game_logic.launch
get_dir.launch
init.launch
load_config.launch
robocheckers.launch
robot_move.launch
robot_socket_connect.launch
test
images ..... obsahuje testovací obrázky
screenshots ..... adresář pro ukládání snímků
CMakeLists.txt
package.xml
setup.py
trac_ik_kinematics_plugin .... balíček pro výpočet inverzní kinematiky