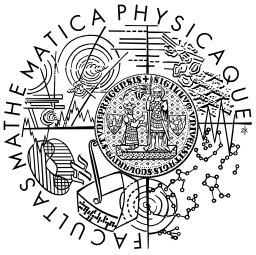


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Roman Staněk

Sledování průběhu deskových her pomocí kamery a hloubkového senzoru

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Martin Kruliš, Ph.D.

Studijní program: Informatika

Studijní obor: Softwarové a datové inženýrství

Praha 2019

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Děkuji svému vedoucímu práce RNDr. Martinu Krulišovi, Ph.D. za cenné rady, připomínky, nápady a čas, které mi během psaní bakalářské práce věnoval.

Poděkování patří i mé rodině a přátelům za neocenitelnou podporu při studiu a pomoc s testováním a opravou stylistických chyb.

Název práce: Sledování průběhu deskových her pomocí kamery a hloubkového senzoru

Autor: Roman Staněk

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Martin Kruliš, Ph.D., Katedra softwarového inženýrství

Abstrakt: Deskové hry byly oblíbenou kratochvílí lidstva již od nepaměti. Během posledních dekád se objevilo mnoho pokusů jak pomocí RGB kamery rozpoznat aktuální stav hry, nebo dokonce sledovat průběh celých zápasů. V této práci k danému účelu používáme senzor Kinect verze 2 od společnosti Microsoft. Jedná se o levný senzor obsahující jak RGB kameru, tak snímáč hloubky obrazu. Cílem práce je analyzovat vlastnosti snímače hloubky při pozorování malých scén a navrhnout způsoby, jakými mohou prostorová data vylepšit výsledky dosavadních postupů používaných k určení pozice herního plánu a figurek. Pro demonstraci jsme vyvinuli program schopný sledovat průběh šachové partie. Dokáže v obrazu určit pozici šachovnice, rozpoznat přítomnost černých a bílých figurek na jednotlivých polích, validovat tahy hráčů v průběhu hry a pořizovat záznam partie, který může být uložen na disk a znovu nahrán do programu.

Klíčová slova: deskové hry, šachy, počítačové vidění, Kinect

Title: Board Games Tracking Using Camera and Depth Sensor

Author: Roman Staněk

Department: Department of Software Engineering

Supervisor: RNDr. Martin Kruliš, Ph.D., Department of Software Engineering

Abstract: Board games have been the everlasting amusement of mankind. During the last few decades, there were many attempts to recognize the state of the games or even track the whole matches using an RGB camera. In this thesis, we use low-cost sensor containing both RGB camera and depth sensor, namely Kinect v2 from Microsoft Corporation, to do such a task. The goal is to analyze properties of depth sensor on small scenes and propose ways in which can depth data improve results of existing solutions used to locate game board and chess pieces. To demonstrate those techniques, we developed a program able to track the chess match. It can locate the chessboard, detect the presence of black and white chess pieces on individual squares, validate movements, and record game so that it can be saved and loaded later.

Keywords: board games, chess, computer vision, Kinect

Obsah

Úvod

Obor informatiky zvaný počítačové vidění, který se zabývá analýzou a zpracováním dat přijímaných skrze optické senzory, neustále nachází nová uplatnění v praxi. Ať se jedná o populární témata, jako jsou například samořiditelná auta, sledování osob, zpracování satelitních snímků, nebo konvenční aplikace z oblasti logistiky, řízení dopravy a automatizace výrobních linek.

Mezi méně známé aplikace počítačového vidění lze zařadit deskové hry. Ve spojení s počítači jsou známy především díky umělé inteligenci schopné porazit člověka. Jedná se o opravdovou výzvu, neboť i hra s malým herním plánem a jednoduchými pravidly může nabývat astronomického množství stavů, se kterými si algoritmy řešící tuto problematiku musí poradit. Mediálně nejznámější událostí je poražení mistra světa šachu Garri Kasparova roku 1997 počítačem Deep Blue. Velkých úspěchů také nedávno dosáhla umělá inteligence AlphaGo [16] hrající hru Go, nebo DeepStack [12] hrající poker.

Na tyto úspěchy navazuje úsilí vytvořit programy schopné rozpoznat, v jakém stavu se hra právě nachází. Umožňuje to hráčům trénovat a studovat hru umělé inteligence nejen na obrazovce počítače, ale i u hrací desky stejně tak, jako by hráli proti lidskému soupeři. Při vysoké spolehlivosti rozpoznání stavu hry by se daly využít k validaci tahů a pořizování záznamu hry, což je dodnes často lidskou záležitostí. Uplatnit lze tento postup jak na šachových turnajích, tak při přímých přenosech z nich i při hrách s časovým limitem, kdy hráči nemají kapacitu hrát a zároveň provádět záznam.

Tomuto tématu se věnovalo již mnoho prací. Jejich společným elementem bývá snímání prostoru pomocí barevné kamery s vysokým rozlišením a rozdělení úlohy na dvě části. První se věnuje nalezení herního plánu a druhá lokalizací kamenů na očekávaných místech a jejich rozpoznání. Některé práce se snaží úlohu řešit v její plné šíři, jiné si ji uměle zjednodušují, například připevněním speciálních značek na herní plán, netradičním obarvením herního plánu a jednotlivých kamenů, či pozorováním hry pouze ze specifických úhlů.

V této práci se zabýváme výhodami a nevýhodami použití senzoru Kinect 2 od společnosti Microsoft na sledování deskových her. Senzor kromě barevné kamery obsahuje také hloubkový senzor, který vysílá do prostoru infračervené záření a na základě rychlosti jeho návratu je schopen spočítat bitmapu vzdáleností jednotlivých obrazových bodů od senzoru. Tyto dodatečné informace rozšiřují možnosti při pozorování reálného světa. Primárně byl tento senzor vyvinut jako ovládací prvek k herní konzoli Xbox, aby uživatel mohl interagovat s konzolí pohybem svého těla. Díky nízké ceně a velkému rozšíření si však našel cestu k mnoha jiným aplikacím, od mapování interiéru budov, přes interaktivní umělecké instalace až po diagnostiku v medicíně.

Pro demonstraci metod počítačového vidění využívajících jak barevnou kameru, tak prostorová data, vyvíjíme program schopný sledovat průběh šachové partie. Šachy jsou ze široké škály deskových her zvoleny díky dobře rozlišitelným figurkám a přehlednému hernímu plánu. To při vyhodnocování snižuje počet faktorů, které by se musely brát v úvahu u složitějších her. Od programu se očekává použitelnost v praktických situacích, například

na šachových turnajích. Proto mezi požadované schopnosti patří nejen určení pozice herního plánu a rozpoznání stavu hry, ale i schopnost stav hry udržet, reagovat na jeho změny a vyrovnat se s rušivými elementy.

Cíle práce

Práce si klade za cíl ověřit, zda použití senzoru Kinect na sledování deskových her může přinést významné výhody oproti použití samotné barevné kamery. Zaprvé je potřeba analyzovat, jestli má senzor dobré vlastnosti při pozorování malých scén, kterými deskové hry bezpochyby jsou. Také je potřeba určit, zda prostorová data přináší užitečné informace navíc, a při kterých úkonech analýzy scény je vhodné je použít.

Jako demonstrace použitelnosti senzoru bude vyvinut program schopný sledovat šachový zápas. Protože se očekává použitelnost programu v praxi, například na šachových turnajích, je vyžadováno, aby program sám monitoroval průběh celého zápasu a uměl se vyrovnat s rušivými elementy, které se ve scéně mohou objevit. V neposlední řadě bude program schopný validovat jednotlivé tahy hráčů a pořizovat záznam celého zápasu.

Přehled kapitol

První část se věnuje krátkému seznámení se šachy a senzorem Kinect. U senzoru jsou podrobně rozebrány klady a zápory technologie použité na snímání hloubky, neboť přímo ovlivňují výběr dalších technik zpracování obrazu.

Následuje analýza problému sledování partie z pohledu počítačového vidění. Ta je provedena diskuzí použitelnosti algoritmů na jednotlivé úkony, od nalezení desky se šachovnicí, přes určení pozic šachovnice a figurek, až po zajištění stability sledování v průběhu času.

Implementační část popisuje návrh programu jako celku. Věnuje se vzájemné interakci mezi moduly, které zpracovávají obraz, udržují stav hry, analyzují ji a komunikují informace uživateli.

Výsledný program je na závěr podroben měření při různých úhlech pozorování. Naměřená data jsou poté interpretována a naznačují směr, kterým by bylo možno práci dále rozvíjet.

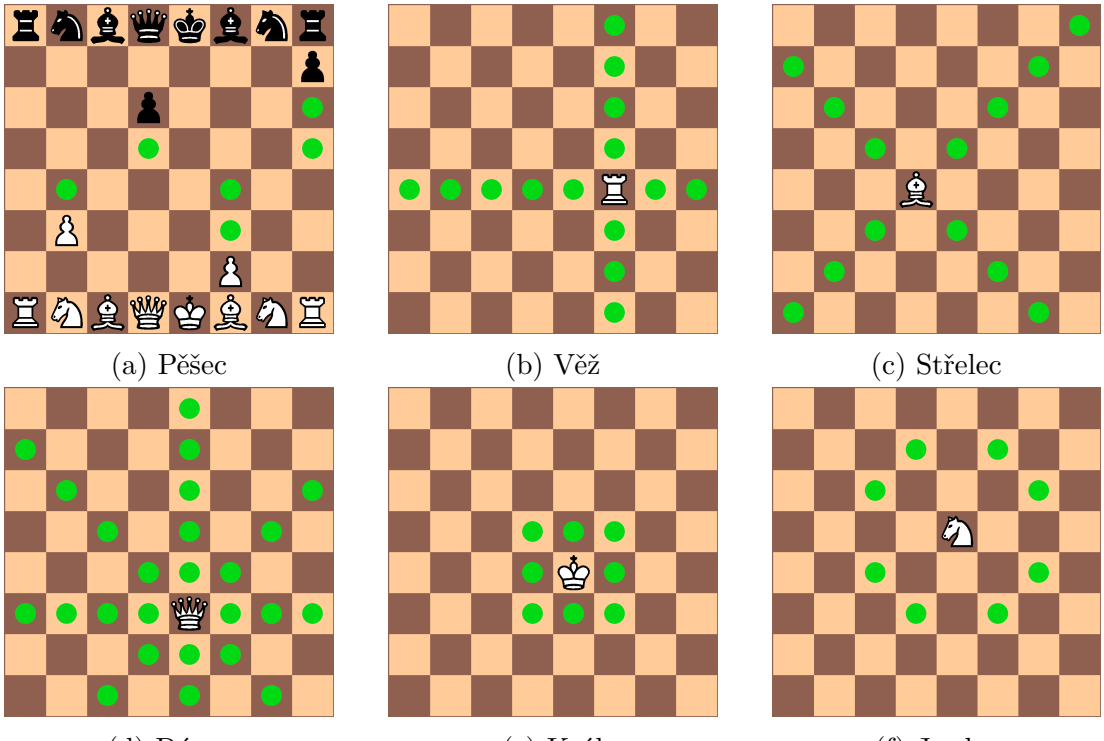
1. Úvod do problematiky

1.1 Šachy

Šachy jsou jedním z klasických a dobře známých reprezentantů deskových her. Pravidelně se v nich pořádají turnaje na mnoha dovednostních úrovních, od oblastních soutěží až po světový žebříček. Následující odstavce obsahují stručný úvod do pravidel hry.

Hrací plán se nazývá šachovnice, což je čtvercová mřížka o 64 (8x8) polích, která jsou střídavě tmavá a světlá. Oba hráči mají identickou sadu figurek lišící se pouze barvou.

Hru začíná hráč se světlými figurkami, dále se v tazích hráči střídají. Hráč může během svého tahu buď pohnout libovolnou figurkou na volné pole, nebo svou figurkou sebrat soupeřovu. Jednotlivé typy figurek se mohou pohybovat pouze podle předem daných pravidel, viz obrázek 1.1.

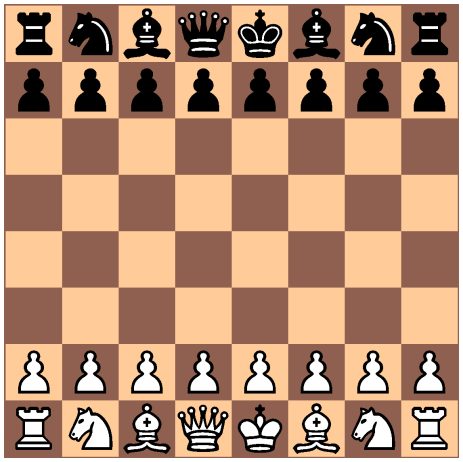


(a) Pěšec (b) Věž (c) Střelec

(d) Dáma (e) Král (f) Jezdec

Obrázek 1.1: Povolené pohyby figurek po šachovnici

Pěšec se může pohybovat pouze o jedno pole (kromě prvního tahu kdy může jít o dvě pole) dopředu, tj. směrem k soupeřově straně, a brát figurky o jedno pole dopředu po diagonále. Věže se pohybují buď po řadách, nebo po sloupcích. Střelci se pohybují po diagonálách. Dáma se buď může pohybovat po řadách a sloupcích jako věž, nebo po diagonálách jako střelec. Věže, střelci ani dámy však nemohou při svém pohybu přeskakovat ostatní figurky. Král se může pohybovat na všechna sousední pole a to včetně diagonálních. Jezdec se vždy musí pohnout o dvě pole v řadě a jedno pole ve sloupci, nebo naopak, přičemž jako jediná figurka může přeskakovat ostatní.



Obrázek 1.2: Základní rozestavení šachovnice

Hra začíná v základním postavení, viz obrázek 1.2, to znamená, že ve druhé řadě od hráče jsou umístěni všichni jeho pěšci a v první řadě všechny ostatní figurky. Ty jsou rozmístěny tak, že věže jsou na krajích, vedle nich se nacházejí jezdci, dále střelci a uprostřed se nachází dáma a král. Dáma zaujímá pole odpovídající její barvě.

Cílem hry je dát soupeři mat, což je situace kdy hráč ohrozí soupeřova krále tak, že by ho mohl v příštím kole sebrat, ale soupeř s králem nemůže uniknout z ohrožení pohybem, nemůže sebrat figurku, která ho ohrožuje, ani nemůže do cesty postavit jinou ze svých figurek. Hra ovšem může skončit i jiným způsobem, například patem, což je šachová obdoba remízy, kdy hráč nemůže soupeřova krále přímo sebrat, soupeř ale nemůže v dalším kole udělat žádný povolený tah. Na profesionální úrovni pak hra může skončit časovým limitem, odehráním určitého počtu tahů bez braní, uznáním prohry, apod.

Hra obsahuje i několik speciálních tahů. Mezi ně patří rošáda, kdy zatím netažený král postoupí o dvě pole do strany a věž, ke které postoupil, se posune na místo, které král přeskočil. Pokud pěšec dojde až na konec hracího pole, nastane tzv. výměna, kdy se pěšec může změnit na jakoukoli jinou figuru podle hráčovy vůle kromě krále a pěšce.

1.2 Kinect

První verze senzoru Kinect spatřila světlo světa v listopadu 2010. Již několik let předtím se konkurenční společnosti v oblasti herních konzolí vyvinuly ovladače schopné zaznamenávat pozici některých částí těla. Příkladem může být Wii Remote od společnosti Nintendo, nebo Playstation Move od Sony.

Firmě Microsoft se podařilo vyvinout pohybový senzor, jenž nevyžaduje držení ovladače, ale je sám schopen sledovat celé tělo v prostoru před kamerou. Skládá se z barevné kamery a hloubkového senzoru.

Esenciální součástí senzoru je předtrénovaný klasifikátor lidských postav [15]. Ten je schopen z hloubkové mapy určit, které obrazové body patří jednotlivým postavám před senzorem. Postavy jsou dále segmentovány na jednotlivé části těla. Z těchto informací se nakonec odvodí předpokládané pozice významných kloubů.

Listopadu 2013, tj. tři roky od vydání první verze, byl vydán jeho nástupce

Obrázek 1.3: Senzor Kinect verze 2 [2]

Kinect 2, viz obrázek 1.3. Přinesl s sebou rozšířené funkce, například sledování většího počtu osob najednou, zvýšenou přesnost díky většímu rozlišení, ale také změnu technologie snímání hloubkové mapy. Parametry jsou uvedeny v tabulce 1.1.

Na podzim roku 2017 Microsoft oficiálně oznámil konec produkce druhé verze Kinectu [17]. Za třetí verzi projektu Kinect se označuje použití a další vylepšení technologie v brýlích pro rozšířenou a virtuální realitu HoloLens.

V květnu 2018 byla na konferenci Microsoft Build představena budoucnost celého projektu jménem Kinect for Azure[1], který má přinést další vylepšení pozorovacích schopností. Technologie má být integrována do další generace HoloLens, a zároveň bude k prodeji jako samostatný senzor stejně tak, jako první a druhá verze. Nově by se však již nemělo jednat o doplněk k herní konzoli, ale o senzor určený k širokému spektru použití [9].

Parametry	Hodnoty
počet pozorovatelných osob	6
počet rozlišitelných kloubů	26
barevná kamera:	
rozlišení (pixel)	1920 x 1080
zorné pole (stupně)	84.1 x 53.8
frekvence (Hz)	30
hloubková kamera:	
rozlišení (pixel)	512 x 424
zorné pole (stupně)	70.6 x 60
frekvence (Hz)	30
minimální měřitelná hloubka (m)	0.5
maximální měřitelná hloubka (m)	4.5

Tabulka 1.1: Hlavní charakteristiky senzoru Kinect verze 2

[1] `https://azure.microsoft.com/en-us/campaigns/kinect/`

1.2.1 Hloubkový senzor

Hloubkový senzor u první verze fungoval na principu strukturovaného světla. Prostor před kamerou je osvícen infračervenými paprsky ze senzoru, které na objektech vytváří malé tečky. Paprsky jsou do prostoru vyzařovány podle předem daného vzoru. Jelikož objekty jsou různě daleko od senzoru a mají různé tvary, tento vzor se zdeformuje. Snímač zaznamená zdeformovaný vzor a ze znalosti původního je schopen dopočítat, jak jsou jednotlivé tečky vzdáleny od senzoru. Výsledkem je hloubková mapa, kde hodnoty jednotlivých obrazových bodů představují jejich vzdálenost od senzoru.

Druhá verze senzoru využívá na rozdíl od té první technologii Time-of-Flight. Jak název napovídá, důležitým faktorem měření vzdálenosti je čas. Vysílač krátce ozáří scénu infračerveným světlem. Každý pixel snímače obsahuje několik jednoduchých senzorů, které se kaskádovitě sepnou, a každý z nich zaznamenává dopadající světlo. Díky znalosti rychlosti světla, času kdy jsme světlo vyslali, času kdy se senzory spouští a množství světla zaznamenaného senzory jsme schopni dopočítat vzdálenost prostoru, na který je pixel zaměřen [11]. Vzdálenost pro spolehlivé měření doporučená od výrobce je 0,5-4,5 metru, viz tabulka 1.1, méně přesná data lze získat až na vzdálenost 8 metrů.

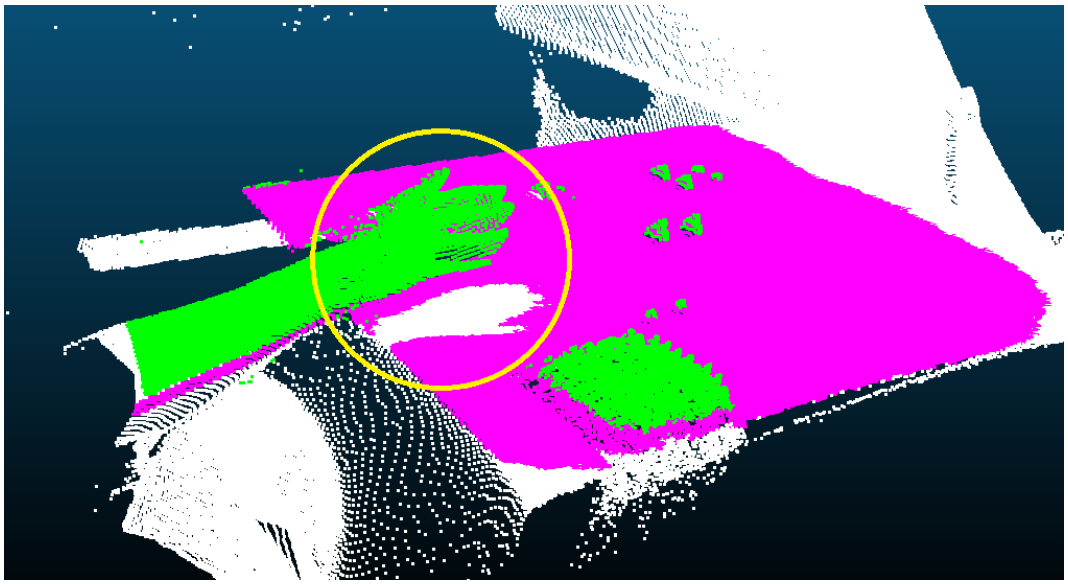
Mezi hlavní výhody použité technologie patří možnost použití více senzorů na jednu scénu, neboť je u nich menší pravděpodobnost, že se budou navzájem rušit. Díky způsobu, jakým probíhá snímání a výpočet vzdálenosti je také senzor stabilnější při pozorování scén s dalšími zdroji záření, například se zářením od slunce.

Bohužel lze pozorovat i několik záporů. První významný problém je známý jako vícecestné rušení [5]. Výpočet vzdálenosti předpokládá, že scénu iluminujeme a odražené světlo dopadne přímo na senzor. To ovšem v reálném světě nemusí platit a může docházet k odrazům, například v rozích, nebo k lomu světla, například v průhledných materiálech, a tyto odražené paprsky dopadnou na senzor také. To naruší správnost výpočtu. Na obrázku 1.4 lze před rukou pozorovat bílé pixely nacházející se pod rovinou stolu, to je způsobeno právě vícecestným rušením.

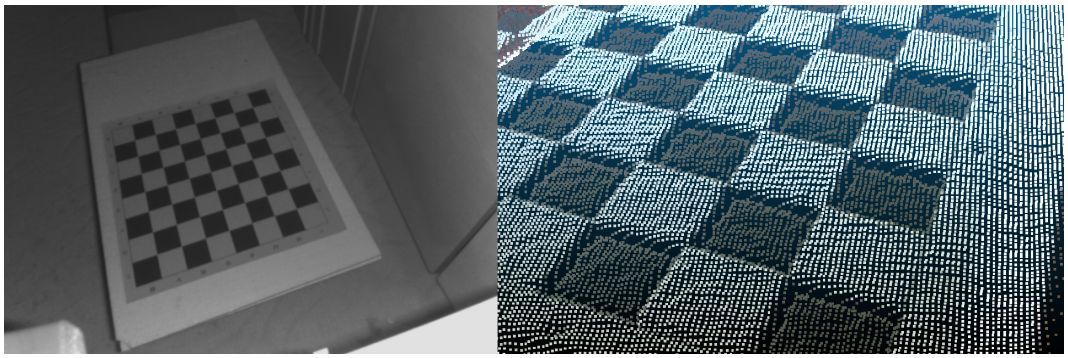
Dalším problémem je pozorování materiálů se špatnými vlastnostmi vzhledem k odrazu infračerveného záření. Například vysoká absorpce záření materiálem způsobí, že se do senzoru vrátí jen velmi málo informace, z níž vzdálenost nelze určit. Tento efekt je dobře viditelný na obrázku 1.5, kde se černá pole nacházejí o několik milimetrů níže oproti zbytku šachovnice.

Posledním zmíněným problémem jsou odlétávající pixely na okrajích objektů. Například pokud budeme pozorovat objekt vzdálený jeden metr a za ním bude pozadí vzdálené dva metry, pak pixely na okrajích objektu nebudou schopny přijímat správně informaci pouze z jednoho objektu, ale budou vracet smíšenou hodnotu z obou, takže vypočítaná vzdálenost okrajových pixelů bude ležet v intervalu mezi jedním a dvěma metry. To může vytvářet netriviální deformace objektů s malým povrchem, jak tomu je u figurek na obrázku 1.6.

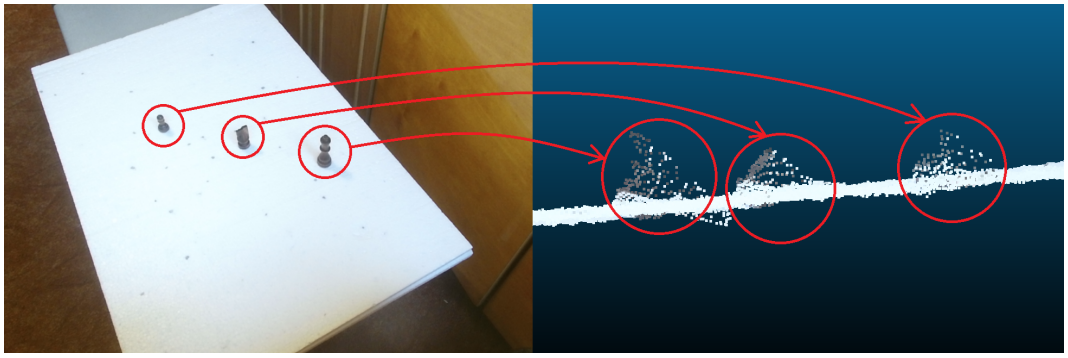
[2]Všechny vizualizace mračen bodů byly vytvořeny v programu CloudCompare - `https://www.danielgm.net/cc/`



Obrázek 1.4: Vizualizace problému vícecestného rušení - Fialové body jsou předpokládaná deska stolu. Zelené body jsou body umístěné nad deskou. Ve žlutém kruhu je ruka těsně nad stolem.[2]



Obrázek 1.5: Vizualizace problému nevhodných materiálů. Vlevo je snímek množství odraženého infračerveného světla ze scény (světlejší = více světla). Vpravo detail šachovnice jako mračna bodů.



Obrázek 1.6: Vizualizace problému odlétávajících pixelů. Vlevo snímek scény z barevné kamery. Vpravo boční pohled na mračno bodů představující figurky.

1.2.2 Aplikační rozhraní

Pro druhou verzi Kinectu existují alespoň dvě udržovaná aplikační rozhraní (dále jen API). První z nich je Kinect SDK 2.0[3]. Jedná se o oficiální ovladače a knihovní podporu přímo od Microsoftu. Druhou alternativou je open source implementace ovladačů OpenKinect[4] a na ni navázané open source knihovny.

V této práci jsme se rozhodli použít oficiální Kinect SDK, u kterého jsme očekávali méně problémů s instalací a kompatibilitou. To se vyplnilo pouze částečně, neboť v jedné z počátečních fází vývoje přestaly po aktualizaci Windows ovladače rozpoznávat zařízení, což vyústilo v několikatýdenní zpoždění vývoje. Stinnou stránkou oficiálních ovladačů je jejich omezení pouze na operační systém Windows, zatímco OpenKinect podporuje i Linux a MacOS. Nevýhodou OpenKinect je, že ačkoli do samotných ovladačů přispívá mnoho vývojářů a lze je nalézt na jednom místě, knihovny umožňující jejich provolání z různých jazyků jsou většinou osobní projekty několika jednotlivců na různých repozitářích na internetu.

Samotné API je vystaveno v jazycích C++ a C# a je poměrně jednoduché. Pro přístup k senzoru slouží třída KinectSensor, která obsahuje zapínání a vypínání senzoru, rovněž se uživatel může zaregistrovat k příjmu událostí, jako je například zaznamenání RGB obrazu, hloubkové mapy či pozice jednotlivých osob. Také lze používat vstup z vestavěného mikrofonu. Důležitou součástí jsou statické mapovací funkce, které jsou schopny namapovat pixely z barevného obrazu na hloubkové souřadnice a naopak. Toto mapování je potřeba, neboť barevná a hloubková kamera jsou umístěny na jiném místě senzoru a proto snímají obraz z jiného úhlu. Obrazové body z hloubkové mapy lze také namapovat na body ve 3D. Jejich pozice se počítá ze vzdálenosti od senzoru a znalosti optických vlastností snímající čočky, jako je například zorné pole, či ohnisková vzdálenost.

1.2.3 Podobné senzory

Alternativami k senzoru Kinect 2 jsou Kinect 1 a v budoucnu Kinect for Azure. Existují také konkurenční zařízení dosahující podobné kvality a použitelnosti. Jedním z nich je Orbbec Astra od firmy Orbbec[5], který poskytuje barevnou i hloubkovou kameru, SDK schopné tyto informace zpracovávat a stejně jako Kinect sledovat postavy. Pro snímání hloubky obrazu je použito principu strukturovaného světla, stejně jako u Kinect 1. Dalšími konkurenty, ale už s omezenější použitelností jsou Leap od společnosti Leap Motion[6] určený primárně pro přesné snímání pohybu rukou, nebo senzor od VicoVR[7], který je určený ke sledování lidských postav a interakci s mobilními zařízeními.

Mnoho hloubkových senzorů také využívá zatím nezmíněných technologií. Jednou z nich je pozorování prostoru senzorem se dvěma, nebo více, barevnými kamerami. Mezi nimi se dá určit podobnost obrazu, který zachycují a pomocí

[3] `https://docs.microsoft.com/en-us/previous-versions/windows/kinect`
[4] `https://openkinect.org/wiki/Main_Page`
[5] `https://orbbec3d.com/`
[6] `https://www.leapmotion.com/`
[7] `https://vicovr.com/`

jednoduché geometrie dopočítat, jak daleko jsou jednotlivé pixely. Výpočet je ovšem netriviálně náročný na výkon. Za zmínku rovněž stojí tzv. LIDAR. Jedná se o způsob snímání prostoru pomocí zaměřeného laserového paprsku, který se velmi rychle pohybuje z místa na místo. Dosahuje vysoké přesnosti, ale nevýhodou je vyšší cena. Využití má hlavně v mapování terénu a jako významný senzor například v samořiditelných autech.

Kinect verze 2 byl z výše zmíněných alternativ vybrán hlavně díky celosvětovému rozšíření a nízké ceně. Ve své cenové hladině vychází vzhledem k rozlišení snímačů a knihovním funkcím na rozpoznávání jako jedna z nejlepších možností.

2. Analýza problému

Při výběru algoritmů počítačového vidění bylo kromě standardních kritérií, jako je složitost implementace, rychlost a chybovost, důležité i to, zda jsou schopny využít prostorová data, neboli hlavní přednost senzoru Kinect. Podobné práce [10, 6, 1] rozdělují úlohu na část hledání šachovnice a následně detekci figurek, která je díky znalosti pozice šachovnice o něco jednodušší. V této práci využíváme stejného rozdělení, ale jako předstupeň hledání šachovnice provádíme hledání herní roviny, na které leží šachovnice, neboť prostorová data ze senzoru jsou k této úloze vhodná a ušetří práci v navazujících částech.

Jak bylo zmíněno, předstupněm hledání šachovnice je hledání desky, na které se hra odehrává. Může se jednat o stůl, šachový stolek, či jinou rovnou desku. Vymezení dalšího hledání pouze na nalezenou desku sníží počet obrazových bodů, kterými bychom se jinak v průběhu každého dalšího zpracování museli zabývat, a zároveň sníží množství okolního šumu pro algoritmy hledající specifické rysy v datech. Dále umožní přesnější nalezení pracovního prostoru, definujeme tak hrací prostor.

Při hledání šachovnice extrahujeme z barevného obrázku významné rysy, jako jsou hrany a rohy šachovnice. Využíváme k tomu osvědčených postupů z podobných prací [10, 6, 1], které hojně využívají práci s barvami, hranové detektory a jiné filtry. Získané informace pak použijeme na určení pozice šachovnice v prostoru, které provádíme pomocí vlastního algoritmu inspirovaného pravděpodobnostními metodami používanými na segmentaci prostorových dat [7].

Určení pozic figurek na šachovnici provádíme ověřením, zda se nad jednotlivými políčky šachovnice vyskytuje dostatečné množství bodů indikujících přítomnost figurky. Vlastnosti technologie použité v senzoru na snímání hloubky však způsobují netriviální deformace a zašumění prostorových dat. Proto je použito několik různých způsobů filtrování, abychom dosáhli čistoty dat potřebné pro snadnou detekci přítomnosti figurky.

V neposlední řadě se věnujeme faktu, že sledování figurek není jednorázová záležitost, ale jedná se o kontinuální proces, který musí pracovat v reálném čase. Potřebujeme proto způsob, kterým budeme udržovat aktuální stav šachovnice, a který bude schopen reagovat na jednotlivé tahy hráčů. Kromě tahů hráčů se mohou objevovat i nežádoucí jevy, příkladem může být pohyb se senzorem, zaclonění scény, pohybující se předměty ve scéně a podobně. Ty je potřeba brát v úvahu a přizpůsobit aplikaci tak, aby nenarušovaly průběh sledování, a aplikace tak byla použitelná v praxi.

Mezi menší úlohy, které je potřeba rozmyslet, patří způsob určení barvy figurky, nebo způsoby, jakými by mohla znalost stavu šachové partie pomoci jednotlivým částem sledování.

2.1 Hledání herní roviny

Najít v mračnu bodů herní rovinu v praxi znamená najít rovinu, která má v malé vzdálenosti od sebe mnoho bodů a ty jsou podél roviny rozumně rozdistribuovány. Problém, kdy se v množině bodů snažíme najít geometrická

primitiva, nebo matematické modely, je důkladně studován. V matematice se tyto metody využívají například k vytváření modelů na základě předchozích dat a předpovídání budoucích událostí. V geometrii je lze využít na takzvanou segmentaci prostředí, kdy jsme z předem nezpracovaného mračna bodů schopni odvodit, zda a kde se v něm nachází geometrická primitiva.

Na řešení bylo vyvinuto mnoho různých přístupů [13]. Příkladem mohou být metody regresní, nebo pravděpodobnostní využívající malého vzoru dat k odhadu modelu, nebo metody snažící se o prohledání všech možných řešení v rozumném čase. Tato podkapitola je věnována diskuzi čtyř v praxi často používaných řešení, jejich použitelnosti, požadavcích na data, rychlosti, správnosti a výstupu, který jsou schopni poskytnout.

Protože hlavním zaměřením této práce není segmentace prostoru, rozhodli jsme se úlohu zjednodušit tak, šachovnice musí ležet na největší rovině v záběru a tato rovina musí být z pohledu senzoru po dobu hledání šachovnice spojitá.

Dalším zjednodušením práce bude předpoklad, že podstava senzoru směřuje směrem k zemi, tj. není vertikálně otočený oproti jeho předpokládanému použití. Tato vlastnost nám pomůže v praktických případech, kdy například sledujeme stůl se šachovnicí a v záběru máme také stěnu nacházející se za stolem. Tato stěna by mohla být detekována jako největší rovina v záběru, ale protože má vůči senzoru nevhodný sklon (její horní část v záběru je blíže, než ta spodní), nemohla by na ní bez přichycení stát šachovnice. Proto budeme takto nevhodně natočené roviny penalizovat, například poměrovým snížením jejich úspěšnosti v algoritmech, čímž dosáhneme příjemnější použitelnosti, neboť program bude méně často zachycovat nevhodné roviny.

2.1.1 RANSAC

Název algoritmu je zkratka z anglického „Random sample consensus“ [7]. Jedná se o pravděpodobnostní algoritmus. Jeho podstatou je, že ze všech bodů vybere minimální množství bodů, ze kterých lze utvořit požadovaný model. Například v případě hledání roviny ve 3D prostoru vybere ze vstupního mračna náhodně 3 různé body, neboť to je minimální počet na jednoznačné určení roviny. Z těchto bodů si vytvoří model, v našem případě reprezentaci roviny. Poté spočítá vzdálenost vypočítaného modelu od každého bodu v mračnu. Pokud je vzdálenost menší než předem definovaná mezní hodnota, můžeme o bodu říci, že splňuje konkrétní model. Čím více má daný model bodů, kterými je splňován, tím lépe odpovídá vstupním datům.

Tento postup lze libovolně iterovat, neboť body na začátku vždy vybíráme náhodně. Spolehlivost algoritmu spočívá právě na počtu iterací, neboť s každou další iterací roste pravděpodobnost, že jsme aspoň v jedné vybrali 3 body z hledané roviny a tím vytvořili dobrý model. Tento vztah lze vyjádřit matematicky [18]. Označme si n počet iterací algoritmu, p požadovanou pravděpodobnost s jakou chceme najít vhodný model, w odhadovaný poměr bodů modelu(zde roviny) oproti všem bodům a k počet bodů které potřebujeme k vytvoření modelu, pak platí následující vztah:

$$n = \frac{\log(1-p)}{\log(1-w^k)}$$

Výhodou tohoto přístupu je možnost měnit poměr správnost/rychlost pomocí nastaveného počtu iterací. Rovněž se jedná o metodu vycházející ze vstupních dat, takže zbytečně nezkouší možnosti, které by jinak ani nepřipadaly v úvahu.

Nevýhodou je potřeba nastavení mezní hodnoty, což je často specifické pro konkrétní problém. Algoritmus není konečný, takže není zaručeno, že vyzkoušel všechny rozumné varianty. Také model vytvořený přímo z dat nemusí být ideální, takže může být vhodné využít metody, které jsou schopny výsledky ještě vylepšit.

Algoritmus RANSAC:

Vstup: množina bodů B, počet iterací n, *Výstup:* výsledný model M

- Inicializuj M jako prázdný model a $m = 0$ jako jeho ohodnocení
- Pro n iterací:
  - Z B náhodně vyber minimální množství bodů potřebné k vytvoření modelu a vytvoř z nich model M'
  - Vypočti u jako počet bodů z B, které odpovídají modelu M'
  - Pokud $u > m$, neboli novější model je lepší než poslední zaznamenaný, pak $M := M'$ a $m := u$

2.1.2 Houghova transformace

Tento algoritmus [8] se oproti předchozím nesnaží hlavně o rychlost, ale o projití značné části prostoru, v níž by se měl nacházet hledaný model. Běžně se využívá při hledání geometrických primitiv, jako jsou přímky, kružnice, apod. v 2D grafice. Princip algoritmu se dobře demonstruje na hledání přímky procházející body v rovině.

Vychází z principu, že každou přímku můžeme vyjádřit dvěma parametry. Například z klasické rovnice $y = ax + b$ lze přímka jednoznačně vyjádřit pomocí dvojice koeficientů (a,b). Problémem u této reprezentace je, při určitých natočeních přímky, potenciálně neomezená hodnota koeficientu a. Proto se často využívá reprezentace $r = x\cos\theta + y\sin\theta$, kde r je vzdálenost od počátku k nejbližšímu bodu na přímce a θ úhel mezi osou x a úsečkou spojující počátek a nejbližší bod přímky. Výsledná dvojice je v tomto případě (r,θ).

Prostor parametrů (r,θ) lze rozdělit na přihrádky. Provedeme to tak, že si pro jednotlivé parametry nastavíme granularitu, například pro r granularitu 1 a pro θ 10°. Vznikne nám například přihrádka, do které spadají všechny přímky s parametry $0 < \mathrm{r} < 1$ a $0° < \theta < 10°$. Každá přihrádka bude obsahovat počítadlo počtu bodů, které do ní patří.

Na počátku algoritmu si definujeme akumulátor, to je struktura obsahující všechny přihrádky. Teoreticky by akumulátor mohl být nekonečný, neboť parametr r může nabývat hodnot od nuly do nekonečna. V praxi však je akumulátor konečný, neboť i obrázek je konečný a všechny přímky v něm lze popsat konečným množstvím přihrádek.

V průběhu algoritmu projdeme každý obrazový bod a provedeme v něm hlasování, tj. proceduru, při které se pro daný bod vypočte, do kterých

přihrádek patří přímky, na kterých může ležet, a zvětší jejich počítadla o jedna. Určení přihrádek pro konkrétní bod vypadá následovně: Vygenerujeme všechny násobky granularity θ ležící v intervalu 0° až 359°, tzn. 0°, 10°, 20°,.. , 350°. Pro každou vygenerovanou θ vypočítáme r tak, aby přímka (r,θ) procházela právě zpracovávaným bodem. Parametry takto vygenerovaných přímek nám přímo určují požadované přihrádky.

Na konci algoritmu najdeme přihrádku s maximální hodnotou. V ní bude ležet rozmezí parametrů, které se v obrazu nacházelo nejčastěji, tzn. nejčastější typ přímky ve smyslu hodnoty parametrů.

Z průchodu algoritmu je vidět, že se snaží projít celý prostor, takže zaručuje, že pokud existuje nejlepší model, pak ho najde.

Nevýhodou jsou vlastnosti akumulátoru. Počet jeho dimenzí je roven počtu parametrů, které potřebujeme na definici modelu. Také není na první pohled, jak nastavit granularitu jednotlivých parametrů, aby transformace vrátila očekávané výsledky. Další překážkou je počet modelů, které musíme v každém bodě vygenerovat, neboť exponenciálně roste s počtem dimenzí modelu.

Algoritmus Houghova transformace:

Vstup: množina bodů B, *Výstup:* výsledný model M

- Inicializuj prázdný akumulátor A
- Pro každý bod b z B proveď:
  - Vypočti přihrádky, do kterých b může patřit a zvětši jejich počítadla v akumulátoru A o jedna
- Vytvoř model M z parametrů, které patří k nejčetnější přihrádce v akumulátoru A

Úlohu lze přirozeně rozšířit na hledání roviny v trojrozměrném prostoru. Pro definici roviny jsou potřeba tři parametry, takže i akumulátor bude mít 3 dimenze. V průběhu algoritmu pak pro každý bod potřeba vygenerovat všechny relevantní přihrádky, kterých je mnohem více, než je tomu u přímky.

2.1.3 Region growing

Předchozí algoritmy hledaly model globálně přes všechna data ze vstupu. Přístup pomocí rostoucích regionů [3] se snaží problém řešit lokálně. Vybírá ze vstupu náhodně, nebo pomocí heuristiky, počáteční body. Z počátečního bodu se postupně šíří vlna, která do aktuálního regionu přidává nové body, ale pouze tehdy, pokud jejich hodnota je v mezní hodnotě od modelu celého regionu.

Algoritmus se chová lokálně, tzn. zatímco RANSAC či Houghova transformace mohly spojit dva nesouvisející regiony splňující podobný model, tento algoritmus funguje hlavně pro spojité regiony. Spojování oddělených částí lze řešit později.

Nevýhodou algoritmu je potenciálně velká výpočetní náročnost, citlivost na šum v datech a potřeba nadefinovat mezní hodnotu a funkci, která rychle odhaduje průměrnou hodnotu regionu. Významný vliv na výsledek může mít také výběr počátečních bodů. Každý nevhodně vybraný bod ovlivní výstup

algoritmu více, než například u RANSAC algoritmu.

Algoritmus Region growing:

Vstup: množina bodů B, *Výstup:* výsledný model M

- Vytvoř prázdný seznam modelů M
- Vyber náhodně množinu počátečních bodů X z B
- Pro každý bod x z X:
  - Z x a jeho okolí vytvoř model N
  - V cyklu:
    - Najdi bod k, který je nejblíže x a zároveň není zahrnut v modelu N
    - Pokud k splňuje model N tak ho přidej do N a příslušně uprav N
    - Jinak přeruš cyklus
  - Přidej N do M

2.1.4 Lineární regrese

Lineární regrese je statistická metoda, která je pro množinu dat schopna najít lineární funkci vstupních proměnných vzhledem k výstupní proměnné takovou, že nejvíce vyhovuje zadaným datům. Z pohledu geometrie hledáme v prostoru nadrovinu, která nejvíce vyhovuje naměřeným bodům.

Kritérií, podle kterých hodnotíme, jak moc je daná rovina vzhledem vhodná vzhledem k datům, se v praxi používá mnoho. Nejčastější je tzv. metoda nejmenších čtverců, která hledá nadrovinu, jejíž suma čtverců vzdáleností ke všem bodům je minimální. Můžeme však použít i součet euklidovských, nebo maximových vzdáleností, a podobně.

Metodu je vhodné použít, pokud očekáváme, že většina dat bude mít od vytvořeného modelu malou odchylku. I menší množství velmi vzdálených bodů může významně znehodnotit vypočtený model. V případě data budou generována ze 2 rovnoběžných nadrovin, pak model pravděpodobně nebude vhodný ani pro jednu z nich.

Časová složitost algoritmu, řešícího lineární regresi soustavou lineárních rovnic, je $O(nc^2)$, kde n je počet bodů a c je počet proměnných definujících nadrovinu.

2.1.5 Vybraný algoritmus

Houghova transformace i růst regionů jsou velmi úspěšné algoritmy. Jejich použití ovšem vyžaduje netriviální množství kódu a vhodných datových struktur. Také potřebují dokončit svůj běh, aby mohly poskytnout výsledek. Výpočet transformace v případě roviny na velkém množství bodů trvá dlouho a také může nastat potíž s nastavením akumulátoru a funkce, která odhaduje lokální výskyt modelu. U rostoucích regionů jsou problémem hlavně nekvalitní

data, která by se musela v reálném čase netriviálně upravovat. Také výběr počátečních bodů není v tomto případě přímočarý.

Lineární regrese je pro hledání stolu nevhodná, kvůli citlivosti na šum v datech.

Pro implementaci byl vybrán algoritmus RANSAC, neboť má jednoduchou implementaci a lze v něm ovlivňovat mnoho parametrů, například jak dlouho poběží, nebo jak přesně bude vzorkovat úspěšnost modelu. Také lze v tomto případě přidat zajímavé heuristiky, například pro způsob, jakým jsou vybírány body, ze kterých se odvozuje model. Z dat bude tedy vypočten model herní roviny. Všechny body do určité vzdálenosti od roviny budou označeny jako body herní roviny. Odhad modelu bude nakonec ještě zpřesněn pomocí lineární regrese, neboť nyní lze předpokládat, že nehledáme v obecných datech, ale už máme body rozumně daleko od modelové roviny.

Úlohu si ještě zjednodušíme tak, že šachovnice musí ležet pouze na jedné rovině, tzn nemůže například ležet každým rohem na jiném stolu. Proto bude z bodů stolu vybrána největší spojitá oblast, vzhledem k jejich pozicím na bitmapě snímané senzorem.

2.2 Hledání šachovnice

Určením herní roviny jsme vymezili prostor, na kterém se může nacházet šachovnice, což nám pomohlo snížit jak vliv okolního rušení, tak objem dat, kterým se dále budeme zabývat. Nyní určíme přesně pozici a otočení samotné šachovnice. Podíváme se na to, jak je tento problém řešen u úloh, které se jím zabývají v praxi, a jak k němu přistupují ostatní práce zabývající se určením stavu šachové hry z barevného obrazu. Jeden z přístupů vybereme, abychom určili významné rysy šachovnice a jejich umístění v obrazu, a poté použijeme vlastní algoritmus, který je využije na určení pozice šachovnice v prostorových datech.

2.2.1 Rozbor přístupů

Problém hledání šachovnice má více využití. Prázdná šachovnice o různém počtu polí se často používá pro kalibraci kamer. Pomocí tohoto postupu se dají odstranit některé neduhy nastavení kamery, jako je například radiální rozostření. V knihovnách věnujících se počítačovému vidění lze dokonce najít funkce věnované přímo tomuto problému. Nelze je však použít na hledání obsazené šachovnice, neboť k tomuto účelu nejsou určeny. Dalo by se pro fungování programu omezit, že by se lokalizovala prázdná šachovnice pomocí těchto vestavěných funkcí a až poté by se na ni postavily figurky. To je ovšem v reálné situaci velmi nepraktické, a pokud by došlo k posunutí šachovnice nebo kamery, musel by se celý postup odstranění a znovupostavení figurek opakovat. Druhý potenciální přístup je snímat šachovnici z ptačí perspektivy a pokusit se v rámci předzpracování obrázku co nejvíce omezit vliv figurek na výsledek. Jedná se však o velmi specifické nastavení snímající kamery, které nevede k flexibilnímu použití.

Využití detekce rohů

Hledání se proto často realizuje pomocí jiných metod počítačového vidění. Například se může jednat o detekci hran nebo rohů. Při hledání pomocí detekce rohů se barevný obraz transformuje do stupňů šedi a aplikuje se na něj algoritmus rohové detekce. Roh je v tomto kontextu definován jako rychlá změna kontrastu ve více směrech. Myšlenkou je najít rohy všech polí, neboli místa kde se stýkají 2 bílá a 2 černá pole. Metoda je ovšem velmi náchylná na nastavení parametrů, neboť v praxi může jako rohy zachycovat i některé přechody figurek a pozadí, či umělecky dekorované okraje šachovnice, popřípadě objekty mimo šachovnici. Pokud se podaří najít dostatečný počet rohů, lze aplikovat navazující metodu, která odhadne, kde se šachovnice nachází.

Příkladem použití tohoto způsobu je práce Koray a Sümer [10]. Ta využívá výše zmíněného principu napozicování barevné kamery přímo na šachovnici tak, aby byla viditelná z tzv. ptačího pohledu. Poté dojde k předzpracování obrázku a detekci rohů pomocí knihovní funkce jazyka Matlab.

Využití detekce hran

Detekce hran má velmi podobný průběh jako u rohů. Obrázek se převede na stupně šedi, nebo pomocí prahování (binarizace) na pouze bílou a černou barvu. Algoritmus hledání hran v tomto případě rovněž hledá rychlou změnu kontrastu, ale pouze v jednom směru. Ta se na šachovnici vyskytuje na rozhranních mezi černými a bílými políčky. Tato metoda je benevolentnější, co se nastavení parametrů týče. Zatímco jedna figurka může při rohové detekci vytvořit nežádoucí roh, v hranové detekci nevytvoří sama celou hranu. Při hranové detekci může také dojít k nalezení nežádoucích hran. Ty se ovšem dají jednoduše různě filtrovat, třeba na základě délky či sklonu.

Tento způsob je dobře popsán v práci Danner a Kafafy [6], kde se snaží o rozpoznání stavu hry pomocí barevné kamery. Protože je zaměřena hlavně rozpoznání typu figurek a ty se kvůli stejné barvě rozpoznávají od šachovnice těžce, rozhodli se autoři celý problém zjednodušit tak, že zatímco figurky mají černou a bílou barvu, šachovnice je zeleno-červená. Jejich řešení spočívá v potlačení jiných barev, než je červená a zelená, následné binarizaci obrázku, detekci hran a nalezení jednotlivých hran šachovnice pomocí Houghovy trasformace uvedené v kapitole 2.1.2. Z hran nakonec vyberou 2 největší skupiny s podobným sklonem a označí je za horizontální a vertikální hrany v šachovnici. Nalezenou šachovnici pak převedou na čtverec pomocí perspektivní transformace, aby s ním mohli dále pracovat.

Bezpochyby praktičtějším a úspěšnějším následníkem je článek od A Czyzewski a kol. [1]. Autoři v něm používají stejný postup jako v předchozím případě. Každou část však podrobněji rozpracovávají, používají strojové učení a vyhodnocování pozice šachovnice nedělají jednou, ale vícekrát a postupně zlepšují svůj odhad. S tímto přístupem dosahují i v obecném případě velmi dobrých výsledků.

2.2.2 Detekce hran

Pro nalezení významných rysů šachovnice jsme si z výše zmíněných metod vybrali hranovou detekci za použití Houghovy transformace. Její hlavní výhodou by měla být odolnost vůči figurkám, které obraz šachovnice narušují, a zároveň její snadná implementace, neboť knihovny na zpracování obrazu většinou podporují všechny potřebné algoritmy. Abychom mohli transformaci použít, je potřeba obrázek předzpracovat.

Rozhranní polí na šachovnici jsou tvořena na místech s velmi vysokým kontrastem (standardní šachovnice obsahují velmi tmavá či velmi světlá pole), proto převedeme barevný obrázek na stupně šedi a následně provedeme prahování (binarizaci), viz obrázek 2.1c. Tato metoda převádí tmavé pixely na černou a světlé pixely na bílou barvu. Výstupem je tedy dvoubarevný obrázek, který obsahuje všechny důležité informace o pozicích polí a figurek, ale nevyskytuje se v něm zbytečný šum různých barev a intenzit světla. Přístupů k binarizaci existuje několik, například lze nastavit mezní hodnotu pro rozdělení barev, nebo tuto hodnotu nechat vygenerovat adaptivně, pomocí Otsuovy metody [14]. Ta pracuje na principu minimalizace rozptylu původních barev bílých a černých pixelů. V této úloze bude výstup jednotlivých přístupů velmi podobný, proto se jimi nebudeme hlouběji zabývat.

Dvoubarevný obrázek použijeme jako vstup do hranového detektoru. V tomto případě není hranou myšlena hrana šachovnice, nýbrž změna náhlá změna kontrastu v obrázku. Ty lze pozorovat například při rychlé změně barvy na povrchu objektu, nebo na rozhranní objektu a jeho pozadí. Hranových detektorů existuje mnoho, v této práci jsme se rozhodli pro, v praxi velmi používaný, Cannyho detektor [4], neboť hledá změny kontrastu ve všech směrech. Jeho aplikace je vidět na Obrázku 2.1d

Takto připravený obrázek již lze vložit do Houghovy transformace, která v něm nalezne všechny dlouhé úsečky představující hrany. Nemusí se však jednat o všechny hrany šachovnice, ani o hrany šachovnice obecně. Mohou se objevit například úsečky představující hrany stolu a podobně.

Kvůli tomu se získané hrany nakonec vhodně vyfiltrují. První možností je vybírat podle maximální a minimální délky. Další, stěžejní metodou je najít dvě největší skupiny hran tak, že v rámci jedné skupiny hran jsou všechny skoro rovnoběžné. Všechny ostatní hrany už nebereme v úvahu. Výsledkem je tedy skupina, která představuje hrany mezi sloupci šachovnice, a druhá skupina představující hrany mezi řádky šachovnice.

V této chvíli máme potřebné informace o šachovnici z barevného obrázku. Výsledek je vidět na Obrázku 2.1f. Nyní můžeme přistoupit k určení pozice šachovnice v prostoru pomocí dat z hloubkového senzoru.

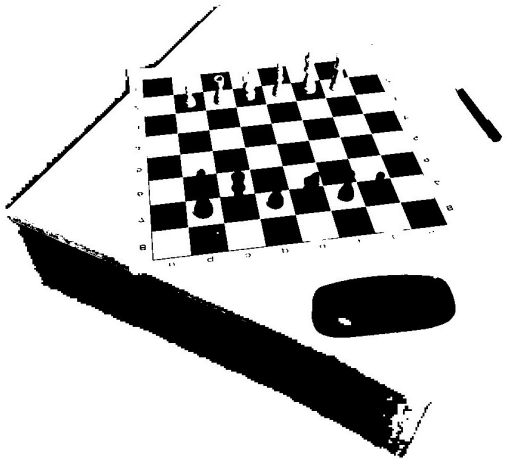
2.2.3 Lokalizace v prostoru

Kdybychom z barevného obrázku měli určeny všechny hrany šachovnice, mohli bychom použít přímočaré postupy namapování těchto čar na 3D body. Situaci ovšem komplikuje několik faktů.

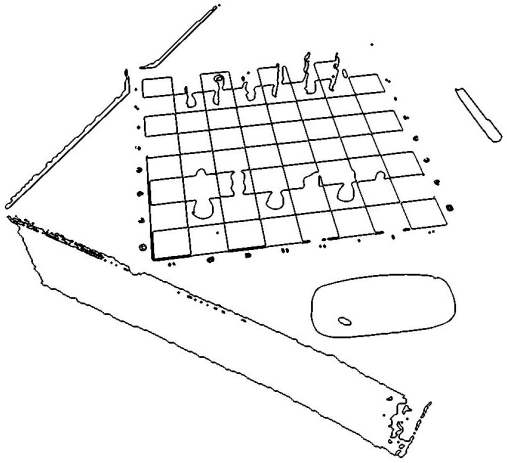
Detekce hran nemusí zachytit všechny hrany šachovnice. Děje se tak například v případě, kdy je hrana zakryta velkým množstvím figurek a Houghova transformace ji není schopna rozpoznat. Následné filtrování snažící

(a) Barevný snímek šachovnice

(b) Stupně šedi



(c) Prahování

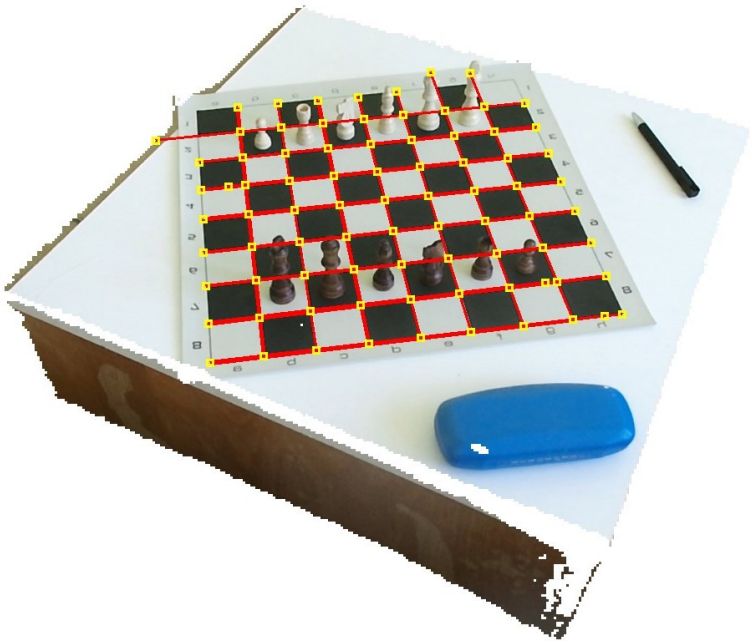


(d) Hranový detektor

(e) Houghova transformace

(f) Filtrování hran

Obrázek 2.1: Jednotlivé kroky detekce hran



Obrázek 2.2: Zobrazení průsečíků hran - červené úsečky představují hrany, žluté čtverce průsečíky a konce hran

se odstranit nežádoucí hrany pomocí rozdělení hran na dvě skupiny nemusí odstranit všechny nedostatky. Pokud v záběru existuje hrana rovnoběžná s jednou skupinou, například zdobený kraj šachovnice, nebo hrana stolu, pak bude i ona zahrnuta do skupiny jako potenciální hrana šachovnice. V neposlední řadě je potřeba myslet na fakt, že pokud bychom z barevného obrázku vzali bod, který nyní považujeme za hranu šachovnice a namapovali ho na prostorový bod, může se stát, že se bude jednat o bod figurky zakrývající zamýšlenou hranu. Získáme tak nesprávnou informaci o pozici v prostoru.

Navrhované řešení

Z výše zmíněných důvodů proto navrhujeme algoritmus, který se snaží vypořádat se všemi nepřesnostmi a určit nejpravděpodobnější pozici šachovnice v prostoru.

Jeho myšlenkou vybrat vhodné body, které nesou důležitou informaci, namapovat je do prostoru a proložit jimi mřížku šachovnice tak, aby jim byla co nejblíže, ale zároveň nepodléhala bodům, které nebyly namapovány správně.

Vhodnými body, na které zaměříme naši pozornost, jsou průsečíky vertikálních a horizontálních hran šachovnice, které máme k dispozici. Ty by měly představovat velmi přesně určené rohy políček šachovnice. Nesou tedy významnou informaci o pozici i tvaru šachovnice, jak je vidět na obrázku 2.2. Také můžeme předpokládat, že hrany nepatřící do šachovnice nebudou tyto body vytvářet, nebo jich vytvoří pouze málo oproti hranám šachovnice. Dalšími body, které je vhodné zahrnout, jsou konce hran šachovnice. Ty nesou informaci o ohraničení šachovnice, a jsou významné v případě, kdy nedetekujeme okrajovou hranu šachovnice, a proto nedojde k vytvoření průsečíků na tomto okraji, jak je vidět u horní, levé a pravé hrany šachovnice na obrázku 2.2.

Pomocí mapovací funkce senzoru mezi barevným obrazem a prostorovými daty získáme souřadnice těchto bodů ve 3D. Na ně aplikujeme algoritmus myšlenkově podobný RANSAC algoritmu zmíněném v předchozí kapitole. Jeho průběh je ilustrován na menším problému, proložení šachovnice 2x2 devíti body, na obrázku 2.3.

Pro každý získaný bod, nazývejme ho B, vybereme několik nejbližších bodů z okolí. V ideálním případě by měly stačit 4, ale kvůli případným nepřesnostem je vhodné zvolit vyšší číslo, například 6. Myšlenkou tohoto postupu je, že pro bod B, jakožto předpokládaného rohu políčka šachovnice, vybíráme další body, se kterými dané políčko sdílí. Viz obrázek 2.3a.

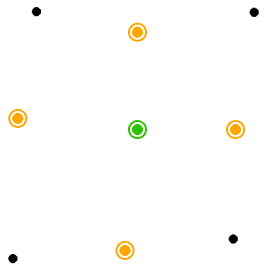
Nyní z okolních bodů vygenerujeme všechny možné dvojice, které splňují podmínku, že jsou přibližně stejně daleko od B, a přímky vedoucí od nich k B svírají přibližně pravý úhel. Všechny takové dvojice jsou pomocí úseček stejné barvy znázorněny na obrázku 2.3b. Myšlenka výběru těchto polí předpokládá, že jsme právě k bodu B vybrali další dva body, které spolu tvoří 3 rohy jednoho políčka šachovnice.

Pokud je tento předpoklad správný, máme nyní určenu pozici a velikost jednoho pole šachovnice. Nevíme ale, které z 64 možných polí to je. Vygenerujeme tedy všech 64 možných modelů šachovnic, ve kterých by naše pole mohlo ležet. Pro každou šachovnici vypočítáme pozice jednotlivých rohů polí. Viz obrázek 2.3d.

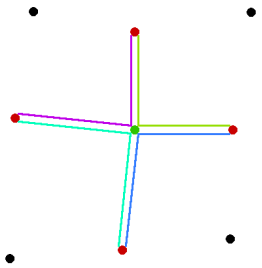
Algoritmus proložení šachovnice body:

Vstup: množina bodů B(průsečíky), parametry e_1 (odchylka délky vektorů definujících šachovnici, očekávaně 5-10 mm), e_2 (odchylka úhlu svíraného vektory definujícími šachovnici od pravého úhlu, očekávaně 1-2°), p (počet uvažovaných sousedů každého bodu, očekávaně 4-8) *Výstup:* nejlepší model šachovnice M

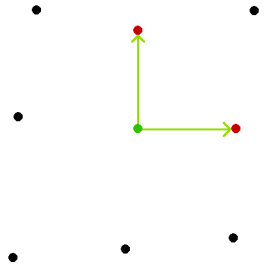
- Inicializuj M jako prázdný model šachovnice
- Pro každý bod b z B:
  - Vybereme p nejbližších sousedů b z B, označíme je S
  - Z S vybereme všechny dvojice ($d1$,$d2$) navzájem různých bodů, označíme je D
  - Pro každou dvojici ($d1$,$d2$) z D ověříme, zda úsečky b–$d1$ a b–$d2$ mají podobnou délku s maximální odchylkou e_1 a svírají pravý úhel s maximální odchylkou e_2, jinak dvojici odstraníme z D.
  - Pro každou dvojici d z D
    - Vygenerujeme všechny navzájem různé šachovnice takové, že úsečky b–$d1$, b–$d2$ spolu tvoří hrany právě jednoho políčka šachovnice. Označíme je N.
    - Pro každou n z N:
      - Spočteme míru úspěšnosti n vůči B, pokud je n lepší než M, pak $M := n$



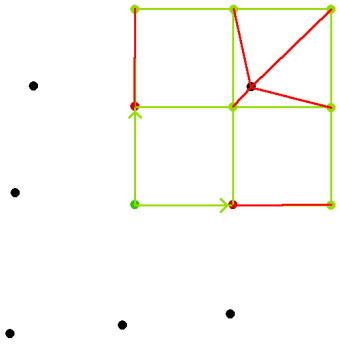
(a) Zpracovávaný bod (zeleně), a jeho sousedé (oranžově)



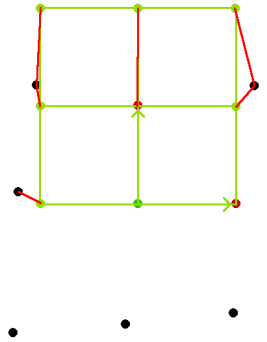
(b) Dvojice přímek



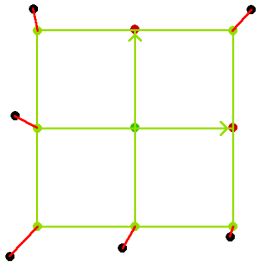
(c) Zpracovávaná dvojice



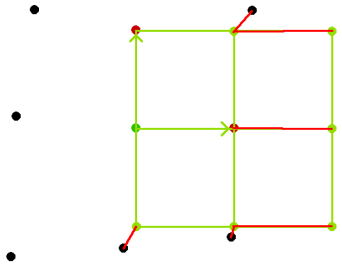
(d) První model šachovnice



(e) Druhý model šachovnice



(f) Třetí model šachovnice



(g) Čtvrtý model šachovnice

Obrázek 2.3: Algoritmus proložení šachovnice body - zelené mřížky jsou model šachovnice, červené úsečky jsou odchylky od reálných dat

Míru úspěšnosti dané šachovnice spočítáme následovně. Pro každý bod modelové šachovnice najdeme nejbližší bod z naměřených dat a určíme jejich vzájemnou euklidovskou vzdálenost. Neděláme perfektní párování, tzn. bod z naměřených dat může být pro výpočet úspěšnosti modelu použit vícekrát. Součet vzdáleností pro všechny body modelu šachovnice je její míra úspěšnosti. Pokud bude hodnota malá, znamená to, že její body odpovídají naměřeným datům a je tedy vhodným modelem.

2.3 Lokalizace figurek

Všechny dále uvedené přístupy předpokládají, že lokalizace figurek navazuje na hledání šachovnice, díky níž máme vymezena políčka šachovnice, tedy místa, kde se figurky mohou nacházet.

Nejjednodušším přístupem [10] jak rozpoznat, zda se na políčku nachází figurka, je definovat si region uprostřed políčka, které nás bude zajímat. Pokud v něm pozorujeme dostatečné odlišnosti od jiných polí, můžeme usoudit, že se na něm nachází figurka. Tento postup je vhodný hlavně při pohledu na šachovnici shora, při pohledu ze strany není obecně jasné, kde přesně by měl ležet region, který nás bude zajímat. Dalším omezením může být i vysoká podobnost figurek a polí, což může zapříčinit malou odlišitelnost figurek od šachovnice.

Komplexnější metody [6] se snaží pomocí aplikace konvolučních filtrů (například detekce hran) vydobýt z obrazu siluety jednotlivých figurek. Se znalostí siluety figurky, nebo její části, lze provádět detailnější analýzu přítomnosti figurek. Zjistíme tak například, přes která pole se nám promítá figurka, nebo je možné lépe odhadnout, zda pozorujeme siluetu jedné figurky, nebo se nám díky zakrytí sloučily dvě. Pokud se vyskytne dostatečná indikace, že se na poli nachází figurka, je možné ještě aplikovat rozpoznávací algoritmus, který se pokusí z obrysu figurky určit, jakého je druhu.

Nejlepších výsledků pak dosahují algoritmy využívající konvoluční neurální sítě [1]. Ty by měly být při dostatečném rozlišení kamery schopny z obrázku figurky a jejího okolí určit jak přítomnost figurky, popřípadě její obalující obdelník, tak typ a barvu.

Zajímavou heuristikou, kterou lze použít jak na detekci přítomnosti, tak na určení druhu figurky, je pomoci si statistikami šachových enginů o pravděpodobnosti výskytu a druhu figurky na jednotlivých polích.

2.3.1 Zvolené řešení

Předchozím zpracováním jsme získali pozici šachovnice, to znamená i pozice jednotlivých polí. Díky faktu, že pracujeme s 3D daty můžeme použít poměrně přímočarý postup na určení přítomnosti a barvy figurky, podobný nejjednoduššímu z výše zmíněných přístupů. Na rozdíl od něj však bude díky prostorovým datům fungovat i při pohledu ze strany a ne jen shora.

Zaměřme se na kvádr, jehož podstavu tvoří políčko šachovnice a má výšku nejvyšší z pozorovaných figurek. Pokud obsahuje figurku, budou se v něm nalézat jak body figurky, tak případně body samotného políčka šachovnice. Potenciálních bodů políčka šachovnice se lze zbavit tak, že do určité výšky

budeme vše ignorovat. Tím nám zůstanou pouze případné body figurky, které nám určují, zda se zde figurka nalézá a jakou má barvu. U nich můžeme nastavit mezní počet, který znamená výskyt figurky, popřípadě dělat sofistikovanější úsudky, například zvětšením důležitosti bodů vyskytujících se blíže středu políčka, neboť očekáváme, že hlavně tam se bude nacházet figurka a spíše u okrajů se bude vyskytovat šum od ostatních figurek.

Zmíněný postup lze použít, nejdříve je však potřeba vyčistit 3D data šachovnice, neboť technologie použitá v senzoru je nesnímá ideálně a obsahují mnoho šumu, jak je vidět na Obrázku 2.4b. Prvním problémem jsou černá pole, která se místy jeví netriviálně propadlá pod rovinu šachovnice, při jiných úhlech pozorování mohou naopak vystupovat nad. Jak již bylo zmíněno v kapitole o senzoru, příčinou je malá odrazivost černých polí vzhledem k infračervenému záření, což má za následek odchylky v pozorování. Neboť senzor poskytuje informace o množství světla, které se do daného obrazového bodu vrátilo, lze tyto body s nízkou spolehlivostí zcela odfiltrovat. Data po odstranění těchto bodů jsou na Obrázku 2.4c.

Druhým problémem je deformace figurek způsobená odlétáváním okrajových bodů figurek směrem od senzoru. Ta způsobuje, že se body figurky na jednom poli objeví i na poli za ním. Velké figurky typu dáma a král mohou kvůli odlétávání na polích za sebou vygenerovat takové množství chybných bodů, které by normálně generoval pěšec, nebo věž.

V této práci jsme se zaměřili na odstranění těchto nežádoucích bodů, aby zůstaly jen správné informace, podle kterých se pak budeme rozhodovat. Alternativními přístupy by mohlo být použití informací z barevného obrazu, jako u výše zmíněných prací, nebo vyvinutí způsobu, který se s odlétávajícími body dokáže vyrovnat, nebo je využít, protože i když je jejich pozice daleko od té skutečné, stále nesou informaci o obrysu figurky.

Odstranění odlétávajících bodů provedeme následujícím způsobem. Hloubková data převedeme na bitmapu se stupni šedi. Na bitmapu spustíme již zmíněný Cannyho detektor, viz obrázek 2.5. Ten v tomto případě nebude hledat hrany na základě změny barev, ale na základě změny šedé barvy, která v tomto případě představuje hloubku jednotlivých bodů. Pokud tedy budeme mít objekt a za ním vzdálené pozadí, jsme schopni určit obrys figurky v prostorových datech, který následně odstraníme, viz obrázek 2.4d. Zbavíme se tak problému odlétávajících bodů, na druhou stranu z figurky nemusí zbýt mnoho pozorovaných bodů.

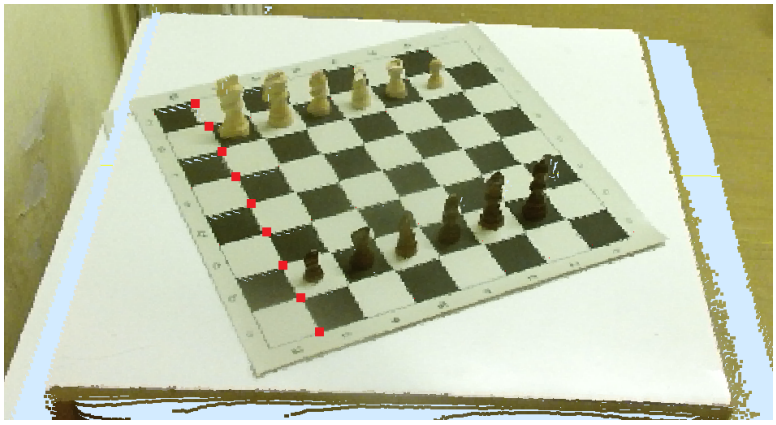
Výsledek, ze kterého určujeme pozici a barvu figurky je na obrázku 2.4e.

2.4 Stabilita sledování

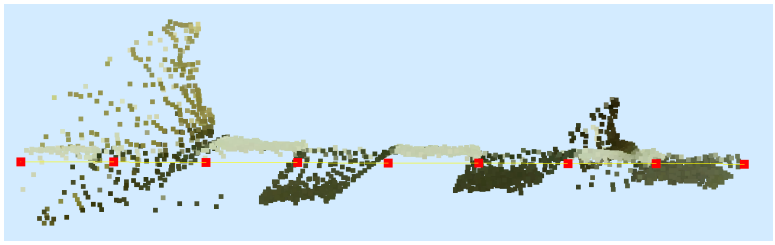
Sledování je kontinuální proces odehrávající se v reálném čase, a proto je potřeba se kromě jednorázového rozpoznání vypořádat i s jevy, které s sebou přináší scéna a její změny. Jejich popis a nástin řešení je obsažen v následujících sekcích.

Detekce barvy figurky

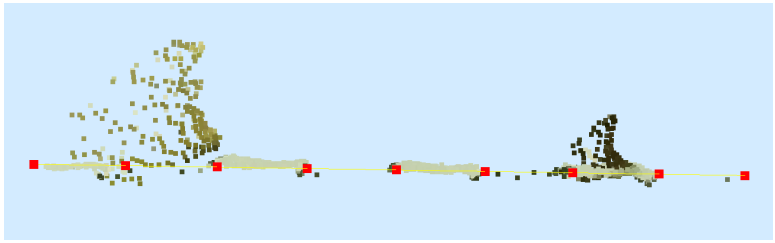
Proces, kterým jsme se ještě nezabývali, ale je nutný pro praktické fungování sledování, je rozpoznání barvy figurek. V obecném případě se jedná o těžký



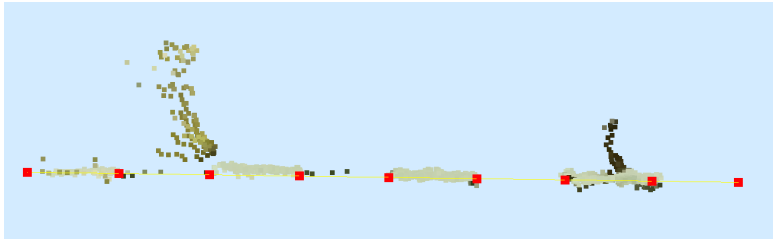
(a) Mračno bodů obsahující šachovnici



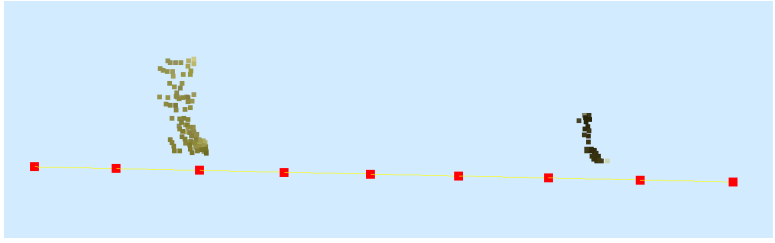
(b) Pohled zleva na první sloupec zleva obsahující figurky



(c) Data po odstranění černých polí



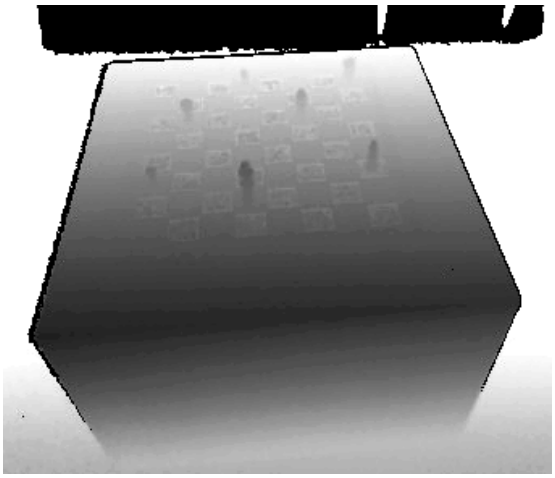
(d) Data po odstranění odlétávajících bodů



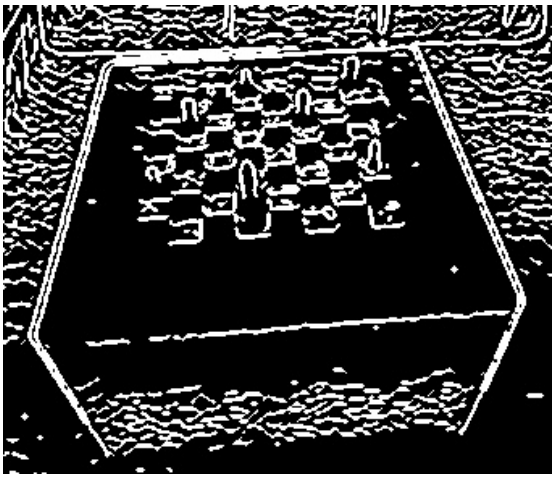
(e) Data po odstranění bodů z políček šachovnice

Obrázek 2.4: Jednotlivé kroky lokalizace figurek

(a) Barevný snímek scény



(b) Hloubková data jako stupně šedi



(c) Cannyho hranový detektor na hloubkových datech. Bílé pixely nebudou využity k detekci figurek

Obrázek 2.5: Aplikace Cannyho hranového detektoru na hloubková data

problém, neboť šachové figurky se vyrábí v různých odstínech a materiálech, světelné zdroje ve scéně se mohou lišit intenzitou, nebo barvou světla, a na šachovnici se mohou objevovat stíny vržené figurkami, nebo objekty v okolí šachovnice.

V této práci se tímto problémem hlouběji nezabýváme a poskytujeme jednoduché, ale ne vždy spolehlivé řešení. To je založeno na myšlence, že černé figurky by měly být tvořeny převážně tmavými pixely, bílé světlými. Pro každý bod figurky tedy spočteme jasovou složku, a tuto hodnotu zprůměrujeme přes všechny body figurky. Pokud je tato hodnota větší, než empiricky nastavený práh, je figurka označena jako bílá, pokud je menší, je označena jako černá. Zároveň uživateli poskytujeme možnost tímto prahem manipulovat, neboť empiricky stanovená hodnota není vhodná pro všechny intenzity osvětlení. Hlavním problémem tohoto přístupu je, že pokud je na část šachovnice vržen stín, pak mohou být některé černé figurky mimo stín identifikovány jako světlejší, než zastíněné bílé figurky, a ve výsledku nebude existovat práh, který by figurky správně odlišil.

S tímto problémem se lze částečně vyrovnat, pokud budeme znát stav hry. Pokud na poli má stát bílá figurka, můžeme práh vychýlit tak, aby snáze registroval bílou figurku, a naopak. Při tahu můžeme předpokládat, že pokud na jednom poli zmizela bílá figurka, a na jiném poli se figurka objevila, bude pravděpodobně bílá. To ovšem funguje pouze pro přesun. Při braní figurek, kdy je možnost sebrat více nepřátelských figurek, to nemusí být směrodatné.

Udržování stavu hry

Jelikož se ve scéně mohou pohybovat předměty, vrhat stíny a měnit barvy, a sám senzor má mez spolehlivosti, dostáváme v každém okamžiku trochu jiná data (milimetrové odchylky u hloubkových dat), i pokud se na šachovnici reálně nic nezměnilo. Pokud bychom počítali pozice figurek pouze z aktuálního snímku, může se snadno stát, že na po sobě jdoucích snímcích dojde k nereálným změnám pozic, které nebudou v rámci šachové hry dávat smysl a sledování skončí v nesmyslném stavu.

Ve snaze zamezit podobné situaci nepočítáme pozice figurek pouze z aktuálního snímku, ale z několika snímků do minulosti. Prvním místem vhodným k použití tohoto přístupu je samotná bitmapa, která přijde ze vstupu. Příchozí bitmapu nepokračuje do výpočtu v podobě, ve které přišla, ale hodnotu obrazového bodu spočítáme například jako průměr, medián, nebo klouzavý průměr přes několik bitmap do minulosti.

Tento postup byl v rámci vývoje otestován, ale není obsažen ve finální implementaci. Z vizuálního hlediska se v obrazu vyskytovalo mnohem méně šumu. Negativním následkem však byly prodlevy mezi změnami ve scéně a jejich promítnutí do výpočtu. Navíc program po implementaci průměrování nejevil známky zlepšení kvality sledování.

Místem, kde se výhody průměrování posledních stavů projevily naplno, je výpočet pozic figurek. Pokud vypočteme chybný stav rozmístění figurek a pošleme ho dále, kde už může ovlivňovat stav hry, můžeme si přivodit potíže s výše zmíněným nekonzistentním stavem hry. Proto v tomto místě provádíme průměrování přes určitý časový interval do minulosti, konkrétně dvě vteřiny.

To znamená, že drobné poruchy sledování by musely mít nejméně poloviční převahu nad správnými výpočty, aby negativně ovlivnily stav hry. Naopak pokud dojde k výrazné změně, například přesunu figurky hráčem, stačí časový úsek o něco větší než vteřina, aby došlo ke korektní změně stavu, což je z pohledu uživatele skoro okamžitě.

Stran samotného průměrování si při sledování můžeme pomoci předchozími pozicemi figurek. Jak již bylo zmíněno u barevnosti figurek, pokud na místě má být bílá figurka, a víme, že tam stojí figurka, je rozumné, abychom uvažovali, že daná figurka bude pravděpodobně bílá. Stejně tak můžeme rozmyslet, že pokud na poli má stát figurka, můžeme být v rámci detekce figurky na tomto poli tolerantnější.

Stinnou stránkou ovšem je, že větší tlak na stabilitu barev a pozic figurek způsobuje, že případná změna stavu musí být výraznější, aby překročila práh rozpoznání. Proto jsou tyto metody v implementaci v základu vypnuty a uživatel si může nastavit, jak moc velký vliv má předchozí pozice/barva mít.

Strategií, která by mohla pomoci s odhadem nové pozice, popřípadě barvy figurky, je predikce za pomoci znalostí pravidel šachové hry. Pokud předpokládáme, že uživatel se snaží hrát hlavně validní tahy, můžeme vygenerovat všechny možné, popřípadě nejpravděpodobnější tahy, což nám může pomoci lépe predikovat místo, kde hledat přesunutou figurku, popřípadě jakou by měla mít barvu.

Očekávané zásahy do scény

V průběhu šachové partie lze od hráčů očekávat, že budou provádět tahy figurkami, pokládat sebrané figurky blízko šachovnice, nebo hýbat objekty na stole. S objekty mimo šachovnici by postupy navržené v této práci neměly mít problém, pokud nezačnou zastiňovat šachovnici. Samotné tahy už z principu šachovnici i figurky zastiňují, a přítomnost ruky může znehodnocovat prostorová data ve svém okolí. Proto je vhodné ošetřit tah jako speciální situaci zásahu do scény.

Zvažovanou možností bylo použít detekci pohybu, kterou nabízí některé knihovny počítačového vidění. Ta je schopna ze sekvence snímků určit, ve kterých místech obrazu probíhá pohyb. V případě, že by pohybující se části začaly zastiňovat šachovnici, určování pozice by se pozastavilo, a po jejich zmizení by se znovu spustilo. Problémem pro tento přístup je ošetření situace, kdy se uživatel dotkne figurky a chvíli váhá, kam potáhne, a nehýbe při tom rukou.

Implementovaným přístupem je detekce libovolných objektů ve vzdušném prostoru nad šachovnicí. Aby uživatel mohl provést tah, musí se jeho ruka dostat nad šachovnici. V tom okamžiku výpočet v prostorových datech nalezne významné množství bodů v prostoru nad šachovnicí a pozastaví výpočet pozic do chvíle, kdy ruka z prostoru zmizí. Nevýhodou použití tohoto postupu jsou situace, kdy do prostoru nad šachovnicí zasahují objekty, které nelze odstranit. Potenciálním řešením může být tyto objekty detekovat při rozpoznání šachovnice, a po zbytek sledování je ignorovat a všímat si pouze objektů, které se objeví nově.

Nežádoucí zásahy do scény

Kromě očekávaných zásahů do scény se mohou objevit i nežádoucí události, se kterými je nutno počítat, aby implementace byla použitelná v praxi. Příkladem může být pohyb kamerou, nebo šachovnicí v rámci sledování. Protože druhá verze senzoru Kinect (na rozdíl od první) neobsahuje gyroskop ani akcelerometr, detekce pohybu kamery lze odvodit pouze z obrazu.

V této práci se pohyby kamerou ani šachovnicí neošetřujeme. Uživateli je poskytnuta možnost resetovat sledování, pokud se do takové situace dostane.

Možným řešením pro narušení pohybem kamerou by mohlo být využití detekce pohybu z obrazu, jako bylo zmíněno u detekce ruky, nebo pro scénu počítat vhodné charakteristiky, které by při větší změně resetovaly hledání, popřípadě by program mohl být schopen určit, o jakou změnu se jednalo a dopočítat pozici šachovnice bez resetu.

Dalším zásahem týkajícím se senzoru může být zastínění šachovnice, nebo celého snímače senzoru. Za této situace může program pouze udržovat stav hry a čekat, než se scéna vrátí do původního stavu, popřípadě po určitém časovém intervalu může upozornit uživatele, nebo zkusit resetovat hledání. Aktuální implementace neobsahuje žádný interval a pouze čeká na obnovení scény.

Posledními uvažovanými jsou světelné změny ve scéně. Můžeme uvažovat buď postupnou změnu osvětlení způsobenou například změnou pozice slunce, nebo okamžitou změnu osvětlení, například prasknutí žárovky v místnosti, či vrhání stínu objektem. Pro postupné změny bychom do programu mohli zavést proměnné, které by akumulovaly stav barev ve scéně přes delší časové období, a ovlivňovaly by výpočet barvy figurek. Okamžité změny by mělo být možné ošetřit za pomoci stavu z několika posledních vteřin, stejně se však může jednat o netriviální problém, například pokud se rychle změní světelné podmínky během tahu hráče.

Vliv pozice senzoru na sledování

Uvážení pozice senzoru vůči šachovnici by také mohlo zlepšit kvalitu sledování. Z určení herní roviny víme, jaký sklon má šachovnice vůči senzoru. Stojí za zvážení, zda by pro jiné úhly nemělo být použité jiné nastavení parametrů algoritmů, popřípadě jiné algoritmy.

Díky znalosti hloubkových dat lze jednoduše určit, které figurky jsou blíže senzoru, a které jsou od něj dále. Pro bližší figurky bychom mohli aplikovat jiné prahy na detekci přítomnosti a barvy, než pro ty vzdálenější.

2.5 Technická analýza

Knihovna pro komunikaci se senzorem Kinect SDK podporuje jazyky C# a C++. Pro implementaci byl použit jazyk C# z důvodu osobních preferencí a lepší podpory uživatelských rozhraní.

Nad .NET Frameworkem[1] jsou udržovány minimálně 2 standardní knihovny pro tvorbu uživatelského rozhranní, Windows Presentation Foundation[2] (WPF) a Windows Forms[3]. V této práci je použita knihovna starší knihovna Windows Forms, hlavně kvůli větším osobním zkušenostem. Knihovna je také podporována v prostředí Mono[4], což by v budoucnu mohlo vést k podpoře programu na více platformách, kdyby se eliminovala závislost knihovny pro komunikaci se senzorem na Windows. Alternativa v podobě WPF by byla také vhodná, neboť se jedná o moderní framework, který podporuje širokou škálu přizpůsobení vzhledu, rychlejší kopírování bitmap, atd.

V případě potřeby zobrazení mračen prostorových bodů, nebo pro augmentaci scény vzhledem ke stavu šachové hry, lze uvažovat knihovny podporující vykreslování ve 3D. Příkladem vhodné knihovny je OpenGL[5], nebo její nástupce Vulkan[6].

Co se týče dalších užitečných algoritmů, jsou potřeba matematické knihovny obsahující algoritmy pro lineární regresi, a také práci nad jednoduchými geometrickými útvary, například reprezentaci úsečky a algoritmus výpočtu průsečíků úseček. Ve zpracování obrazu je potřeba mnoho konvolučních filtrů, práce s barvou obrázku a Houghova transformace. Tyto algoritmy poskytují knihovny určené ke zpracování obrazu, například část knihovny Accord.NET[7], nebo EmguCV[8], která poskytuje C# rozhraní pro volání funkcí knihovny OpenCV[9] napsané v C++.

Pro algoritmus RANSAC lze nalézt již hotové implementace, například v C++ knihovně Point Cloud Library[10]. Od algoritmu ovšem očekáváme možnost většího ovlivňování jeho parametrů, než poskytuje knihovna, proto bude algoritmus implementován vlastním kódem.

[1] `https://dotnet.microsoft.com/`
[2] `https://docs.microsoft.com/en-us/dotnet/framework/wpf/`
[3] `https://docs.microsoft.com/cs-cz/dotnet/framework/winforms/`
[4] `https://www.mono-project.com/`
[5] `https://www.opengl.org/`
[6] `https://www.khronos.org/vulkan/`
[7] `http://accord-framework.net/`
[8] `http://www.emgu.com`
[9] `https://opencv.org/`
[10] `http://pointclouds.org/`

3. Implementace

3.1 Použité technologie

Program je implementovaný v jazyce C# na platformně .NET. Vývoj probíhal ve Visual Studiu 2017. Při vývoji byl použit systém na správu verzí Git. Zdrojový kód je volně dostupný na veřejném repozitáři služby Github[1].

Použité knihovny:

- *Kinect SDK 2.0* - oficiální knihovna pro práci se senzorem Kinect v2. Poskytuje API pro jazyky C# a C++.
- *EmguCV* - poskytuje API v jazyce C# ke knihovně OpenCV vyvíjené v C++, která je dnes de facto standardem pro vývoj aplikací spojených s počítačovým viděním.
- *Windows Forms* - starší knihovna pro tvorbu uživatelského rozhraní běžící na .NET Framework. I přes to, že je jednodušší než její nástupce Windows Presentation Foundation, tak poskytuje všechny potřebné funkce. Navíc existuje její podpora v prostředí Mono, což umožňuje běh na více platformách.
- *MathNet*[2] - opensource matematická knihovna. Zde je využita díky široké škále regresních metod používaných ke zpřesnění výsledků a geometrickým objektům.
- *Accord.NET* - knihovna pro strojové učení a zpracování obrázků. Zde je využita pro práci s geometrickými objekty.

3.2 Struktura projektu

Zdrojové kódy jsou rozděleny do složek podle jmenných prostorů, ve kterých se nacházejí. Následující odrážky obsahují popis jednotlivých složek a významné třídy, které jsou v nich umístěny.

- `UserInterface` - obsahuje jak formuláře, které se zobrazují uživateli a umožňují reagovat na jednotlivé události vyvolané jeho akcemi, tak vstupní a výstupní fasády, které tvoří styčný bod pro interakci s programem.
  - Potomci třídy `Form` - jsou jednotlivé formuláře. Obsahují obsluhu událostí uživatelského rozhraní a funkce, které ho mění.
  - `UserInterfaceInputFacade` - vstupní fasáda, přes kterou procházejí všechna volání z uživatelského rozhraní dále do programu.

[1] `https://github.com/standa42/ChessTracking`
[2] `https://www.mathdotnet.com/`

    - `UserInterfaceOutputFacade` - výstupní fasáda, skrze kterou může program měnit uživatelské rozhraní, popřípadě se dotazovat na jeho stav.

- `Game` - poskytuje model šachové hry a zajišťuje načítání/ukládání her z disku.

    - `GameController` - je styčným bodem logiky šachové hry se zbytkem programu. Drží většinu objektů týkajících se šachů a navenek vystavuje funkce, které může volat zbytek programu.
    - `GameData` - obsahuje všechny informace o stavu konkrétní hry.
    - `GameFactory` - obsahuje statické metody schopné vytvořit novou hru, nebo načíst uloženou hru ze souboru.
    - `GameRenderer` - je schopen vykreslit vizualizaci šachové hry do bitmapy.
    - `GameValidator` - slouží k rozpoznání provedeného tahu a jeho validaci.

- `ControllingElements` - obsahuje objekty, které spravují stav programu, komunikují se zpracováním obrazu a zajišťují správný překlad mezi reprezentací výsledků sledování a reprezentací stavu hry v šachovém modulu.

    - `TrackingManager` - slouží pro komunikaci s logikou provádějící zpracování obrazu. Drží vstupní konec fronty, jejíž výstup dostává logika zpracování obrazu a stejně tak v něm končí fronta, skrze kterou zpracování obrazu zasílá data o výsledcích, nebo popis chyb. Také při příkazu ke sledování vytváří objekt `Kinect`, který získává data ze senzoru a zasílá je do logiky zpracování obrazu. Výsledky přicházející ze sledování jsou přeposílány objektu `TrackingResultProcessing`.
    - `TrackingResultProcessing` - do tohoto objektu přichází data ze sledování, na kterých se provádí průměrování a vhodná rotace, aby splňovala požadavky pro porovnání s reprezentací šachové hry.
    - `ProgramState` - je implementací stavového automatu. Jeho rozhraní tvoří důležité události života programu. Při zavolání události na `ProgramState` může dojít k přechodu do jiného stavu či postranním efektům, jako například ukončení příjmu dat ze senzoru, nebo zamknutí některých ovládacích prvků v uživatelském rozhraní.

- `MultithreadingMessages` - zprávy zprostředkovávající výměnu informací mezi jednotlivými vlákny.

- `ImageProcessing` - obsahuje logiku zajišťující zpracování obrazu a potřebné algoritmy.

    - `ProcessingController` - má podobnou úlohu jako `TrackingManager`. Stará se informace, které mu přichází od zbytku programu a ovlivňuje jimi `Pipeline`.

- `Pipeline` - obsahuje logiku průběhu zpracování obrazu, to jest, kdy probíhá kalibrace a kdy už sledování, stejně tak jako že nejdříve probíhá detekce stolu, pak šachovnice a nakonec figurek. Pokud výpočty doběhnout úspěšně, odesílá výsledek sledování objektu `TrackingManager`.
- `Kinect` - obsahuje všechen kód obsluhující senzor. Přijímá snímaná data a naplňuje jimi objekt `KinectData`, který posílá do zpracování obrazu.

- `Utils` - obsahuje obecné algoritmy a objekty, které nemají přímou vazbu k této práci, jedná se například o rotace dvoudimenzionálních polí, nebo hluboké kopírování objektů.

3.3 Návrh programu

Samotný program je z důvodu responzivity uživatelského rozhraní a oddělení kódu rozdělen na dvě vlákna: hlavní vlákno a vlákno věnující se zpracování obrazu. Zatímco vlákno zpracování obrazu pracuje s daty ze senzoru a provádí na nich časově náročné výpočty, hlavní vlákno může reagovat na akce uživatele, nebo pracovat s reprezentací šachové hry. Obě vlákna také pracují na jiných třídách objektů a vzájemně si vyměňují informace pouze pomocí objektů vícevláknové komunikace, kterými se budeme zabývat později. Na obrázku 3.1 je znázorněn diagram toku důležitých informací v programu.

Hlavní vlákno pracuje s uživatelským rozhraním, šachovou hrou, objekty z `ControllingElements` a objektem `Kinect`. Pokud chce uživatel pracovat se šachovou hrou, například vytvořit novou hru, načíst ji ze souboru, nebo ji uložit, komunikuje skrze objekty uživatelského rozhraní přímo s objekty ze složky `Game`.

Při požadavku na začátek sledování šachovnice dojde k odeslání informace o zahájení sledování skrze `TrackingManager` do `ProcessingController`. Zároveň `TrackingManager` vytvoří objekt `Kinect`, který začne přijímat data ze senzoru a ukládat je do `KinectBuffer`. Vlákno zpracování obrazu pasivně čeká v objektu `ProcessingController` na příkazy týkající se sledování. Po přijmutí zprávy o zahájení sledování začne odebírat data z `KinectBuffer` a provede nejprve kalibraci, kdy dojde k nalezení šachovnice, a poté se až do příkazu k ukončení věnuje sledování figurek.

Výsledky sledování jsou odesílány hlavnímu vláknu, které na nich provede potřebná průměrování, překlopení a otočení tak, aby mohly být porovnány se stavem šachové hry. Pokud stav sledování neodpovídá stavu hry, pokusí se šachový validátor zjistit, zda došlo k platnému tahu, nebo se jedná nevalidní stav, a informuje o tom uživatele.

Při požadavku na ukončení sledování dojde k destrukci objektu `Kinect` a informování výpočtu zpracování obrazu.

Komunikace mezi vlákny

Pro vícevláknovou komunikaci je v programu využito tří přístupů. Prvním jsou vícevláknové fronty, skrze které si `TrackingManager` a `ProcessingController` vyměňují zprávy o zahájení/ukončení sledování a jeho výsledcích. Jedná se instance objektů `BlockingCollection`, což je thread-safe fronta ve standardní knihovně přímo uzpůsobená pro návrhový vzor producent-konzument. Každý objekt, který se posílá přes fronty musí být potomkem třídy `Message` definované v programu.

Druhým způsobem vícevláknové komunikace je objekt `KinectBuffer`. Do něj z objektu `Kinect` přichází data ze senzoru a na začátku zpracování výpočtu zpracování obrazu se data odebírají. Lze skrze něj kontrolovat, kolik instancí dat ze senzoru právě proudí programem, což je důležité hlavně pro paměťové požadavky programu.

Posledním způsobem komunikace je `UserDefinedParametersPrototype-Factory`. Aby při změně parametrů výpočtu na formuláři nemuselo docházet k propagování změny přes celý program, objekty uživatelského rozhraní i zpracování obrazu drží instanci této factory. Ta obsahuje prototyp parametrů, které může měnit uživatel. Pokud chce uživatel parametry ovlivnit, změní skrze uživatelské rozhraní tento prototyp. Při spuštění každého výpočtu zpracování obrazu se potřebné parametry naklonují z prototypu a jsou tedy pro výpočet vždy aktuální.

Výpočetní vlákno

Jak již bylo zmíněno, při požadavku na zahájení sledování hry se informuje `ProcessingController`, a objekt `Kinect` začne plnit `KinectBuffer` aktuálními daty ze senzoru. Na objektu `ProcessingController` pasivně čeká vlákno zpracování obrazu, které po obdržení příkazu k zahájení sledování vybere z `KinectBuffer` data, z `UserDefinedParametersPrototypeFactory` parametry výpočtu, a spustí kalibraci.

Kalibrace má 3 fáze: hledání herní roviny, hledání šachovnice a určení figurek, viz obrázek 3.2. Mezi jednotlivými fázemi se posílají datové objekty, jejich data jsou strukturována podle toho, v jaké fázi výpočtu se získala. Každá fáze je uvnitř rozdělena na další podčásti, například při hledání roviny se jedná o RANSAC, lineární regresi, hledání spojitých oblastí, a podobně. Paralelní výpočty v kalibraci probíhají pouze na úrovni jednotlivých algoritmů, za pomoci `Parallel.For`, neboť většina částí zpracování přímo navazuje na výsledky předchozích výpočtů.

Po úspěšném dokončení kalibrace je informováno hlavní vlákno, a vlákno zpracování obrazu zahájí sledování. Při sledování, stejně jako u kalibrace, dojde k získání dat z `Kinect` a parametrů výpočtu. Výpočet sledování vlákno neprovádí samo, ale vytvoří `Task`, neboli úlohu určenou ke zpracování v thread poolu, který provede zbytek jednoho výpočtu sledování. Rozdělení výpočtu sledování je stejné jako u kalibrace. Paralelizace sledování tedy probíhá jak na úrovni jednotlivých výpočtů sledování, neboť jsou samostatnými instancemi `Task`, tak na úrovni jednotlivých algoritmů, kde se, stejně jako v kalibraci, využívá `Parallel.For`.

Uživatelské rozhraní

Uživatelské rozhraní se skládá ze čtyř formulářů: `MainGameForm`, `VizualizationForm`, `CalibrationSnapshotForm`, `AdvancedSettingsForm`. První dva jsou viditelné po celou dobu běhu programu, zbylé dva může uživatel volitelně otevírat a zavírat. `MainGameForm`, viditelný na obrázku 3.3, je hlavním formulářem celého programu. Jsou na něm umístěna tlačítka ovládající šachovou hru a sledování, také se na něm zobrazují výsledky sledování, průběh šachové hry, záznam šachové hry a log.

`VizualizationForm` obsahuje upravený snímek scény, kterou senzor právě sleduje. Ze snímku můžeme například vyčíst, kde byl rozpoznán stůl, kde leží šachovnice, nebo na jakých místech výpočet indikuje figurky a jakou mají barvu, viz obrázek 3.4.

Na formuláři `CalibrationSnapshotForm` je možné šipkami přepínat bitmapy, které znázorňují jednotlivé fáze zpracování snímku během kalibrace. Tato vizualizace může sloužit jako nápověda uživateli v případě, že se během kalibrace objevují chyby. Příkladem je snímek po provedení prahování, viz obrázek 3.5.

Posledním formulářem je `AdvancedSettingsForm`, který obsahuje většinu nastavení programu. Jedná se převážně o nastavení konstant, nebo přepínače, které ovlivňují průběh výpočtu sledování. Jeho jednotlivé prvky můžeme vidět na obrázku 3.6.

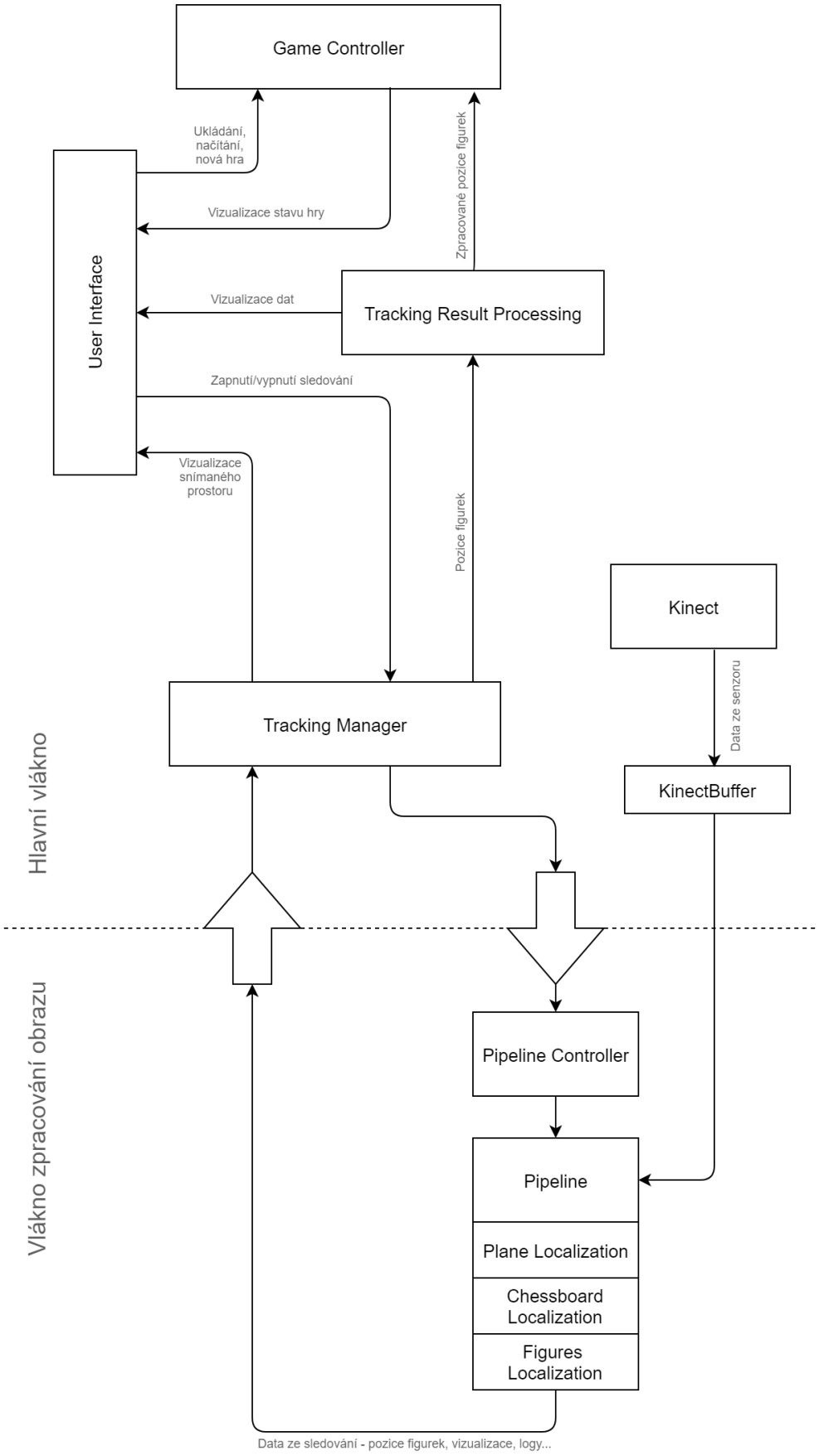
Parametry výpočtu ovlivnitelné uživatelem

Za účelem příjemnějšího ladění a testování programu, ale i pro jeho praktické použití při různých podmínkách, byly do programu přidány ovládací prvky, skrze které může uživatel ovlivňovat vybrané prvky výpočtu sledování šachovnice. Na obrázcích 3.3 a 3.6 je tučnými červenými číslicemi vyznačena jejich poloha. Číslice se vztahují k jejich popisu v následujícím seznamu:

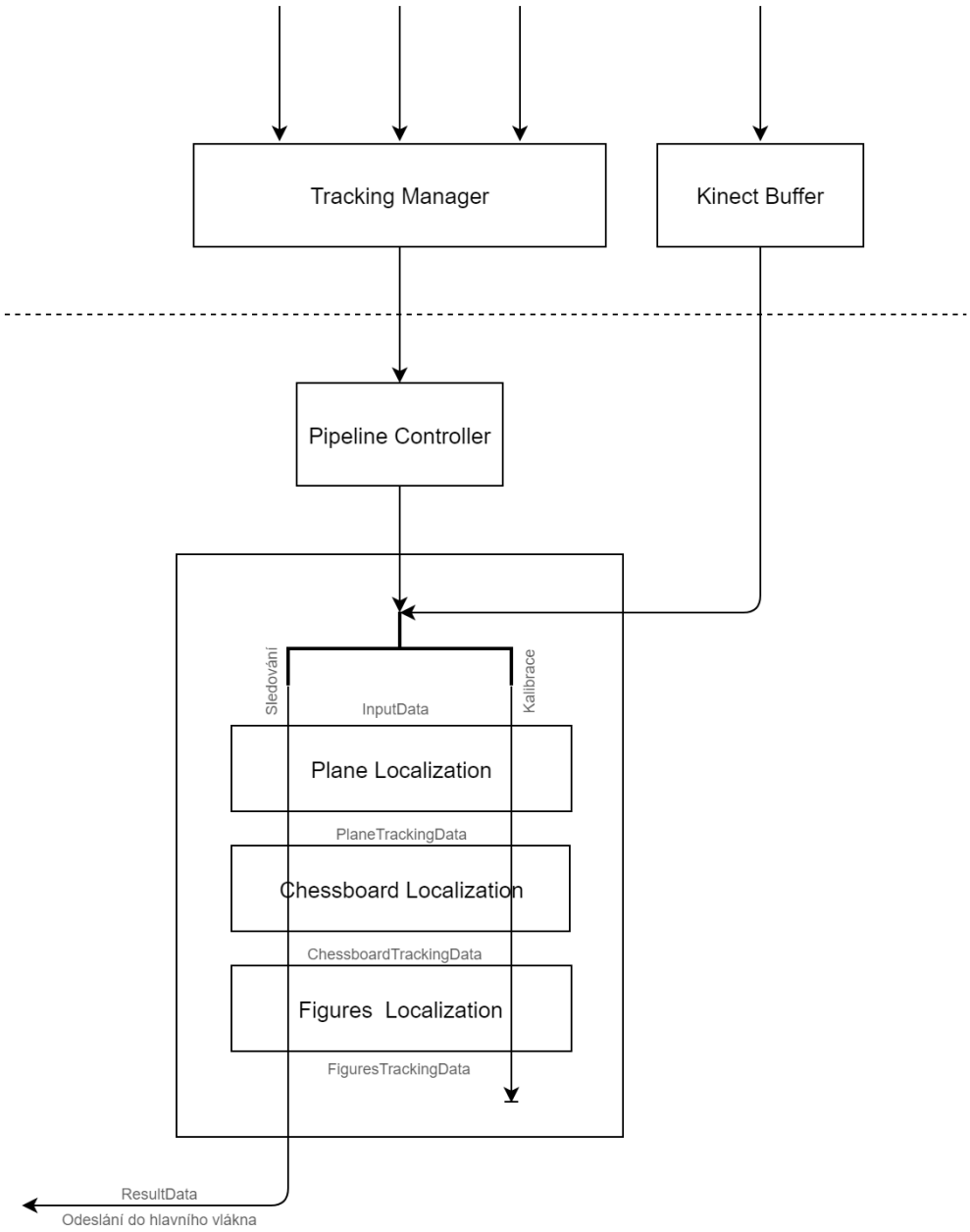
1. možnost pohnout již nalezenou šachovnicí do stran o velikost jednoho pole.
2. možnost měnit způsob výpočtu barvy figurek a konstantu, která rozděluje černé a bílé figurky.
3. změnu způsobu, jakým se provádí prahování obrazu při hledání šachovnice.
4. nastavení prodlevy mezi jednotlivými výpočty sledování pro snížení zátěže programu na počítač.
5. změnu vzdálenosti od šachovnice, do které dochází k ořezu bodů figurky a šachového pole.
6. nastavení počtu bodů nad polem šachovnice, který indikuje přítomnost figurky.
7. nastavení míry vlivu stavu hry na výpočet přítomnosti a barvy figurek.
8. možnost měnit způsob výpočtu úspěšnosti modelu při prokládání šachovnice reálnými daty.

Tyto parametry mají při spuštění nastaveny hodnoty, které se během vývoje empiricky ukázaly jako spolehlivé. Příkladem může být nastavení výšky ořezu figurky. Pro něj byla vybrána hodnota 10 mm, při které jsou ořezány body pole a zároveň je odstraněno nejméně bodů figurky.

Kompletní popis prvků, a jejich umístění, i parametrů, které jsou jimi ovlivňovány, lze nalézt v uživatelské dokumentaci nacházející se v elektronické příloze A.1.

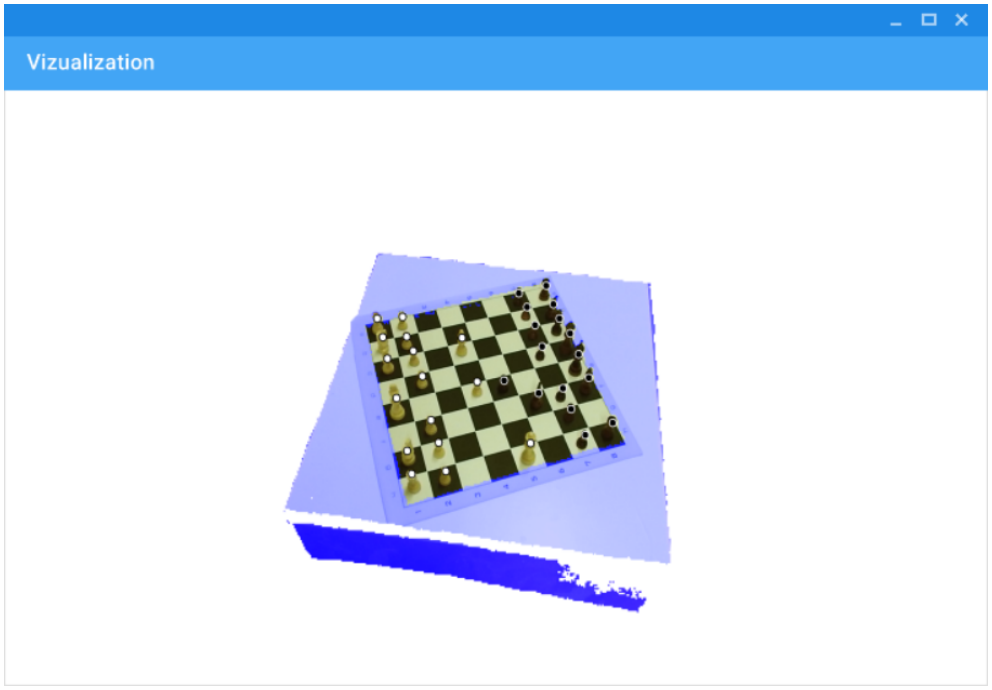


Obrázek 3.1: Diagram toku informací v programu

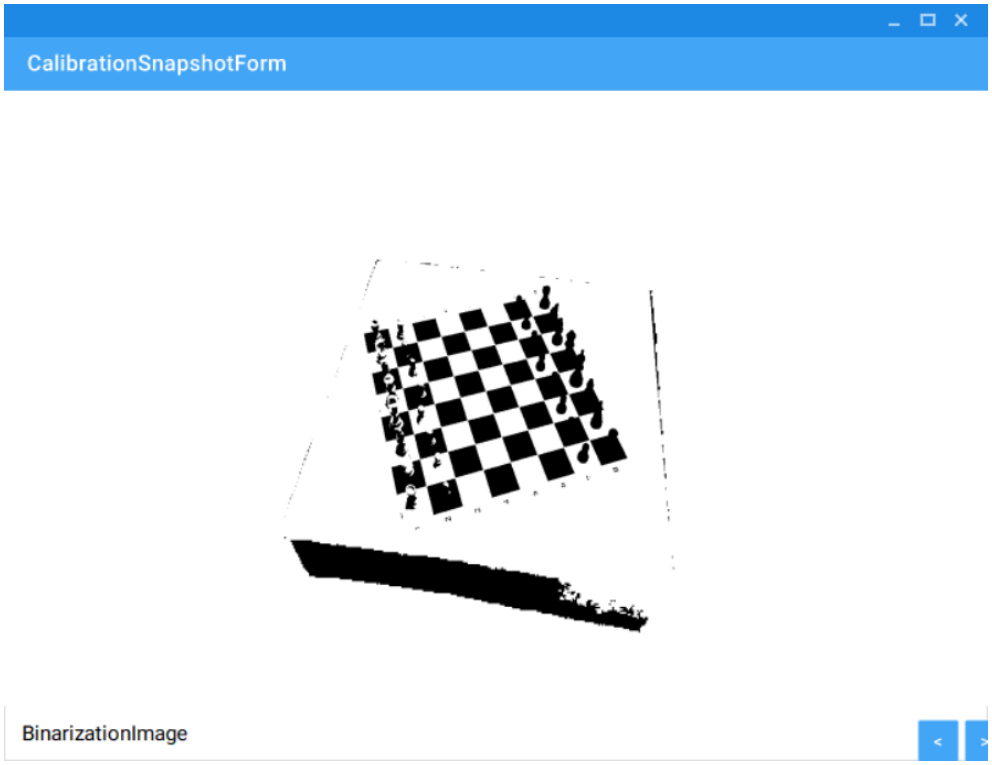


Obrázek 3.2: Diagram průběhu zpracování dat ve výpočetním vlákně

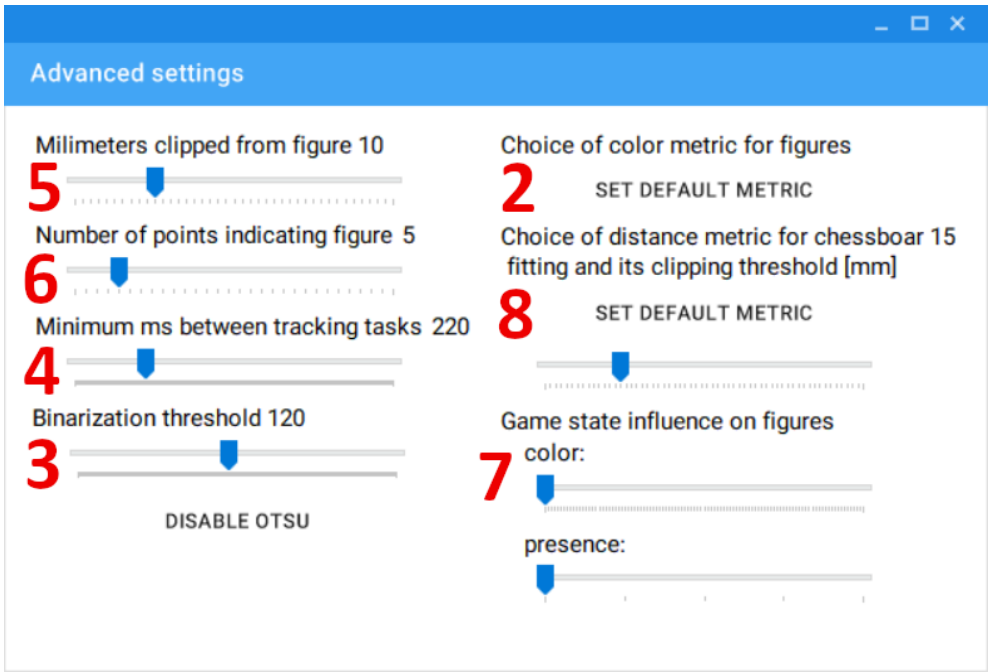
Obrázek 3.3: Hlavní formulář programu (`MainGameForm`)



Obrázek 3.4: Vizualizační formulář (`VizualizationForm`)



Obrázek 3.5: Formulář se snímky z kalibrace (`CalibrationSnapshotForm`)



Obrázek 3.6: Formulář pokročilého nastavení (`AdvancedSettingsForm`)

4. Měření

V minulých kapitolách byl vytvořen program sledující šachovou hru. Pro demonstraci funkčnosti byly navrženy dva experimenty. První experiment se věnuje spolehlivosti hledání šachovnice. Senzor je umisťován do různých úhlů vzhledem k šachovnici a je testováno, kolikrát dokáže správně určit pozici šachovnice a figurek. Druhý má za cíl otestovat, zda program dokáže bez problémů fungovat v praktické situaci, tj. senzor bude umístěn nad šachovnici do pozice, ze které by neměl mít problém se sledováním, a bude monitorovat šachovou partii od začátku do konce.

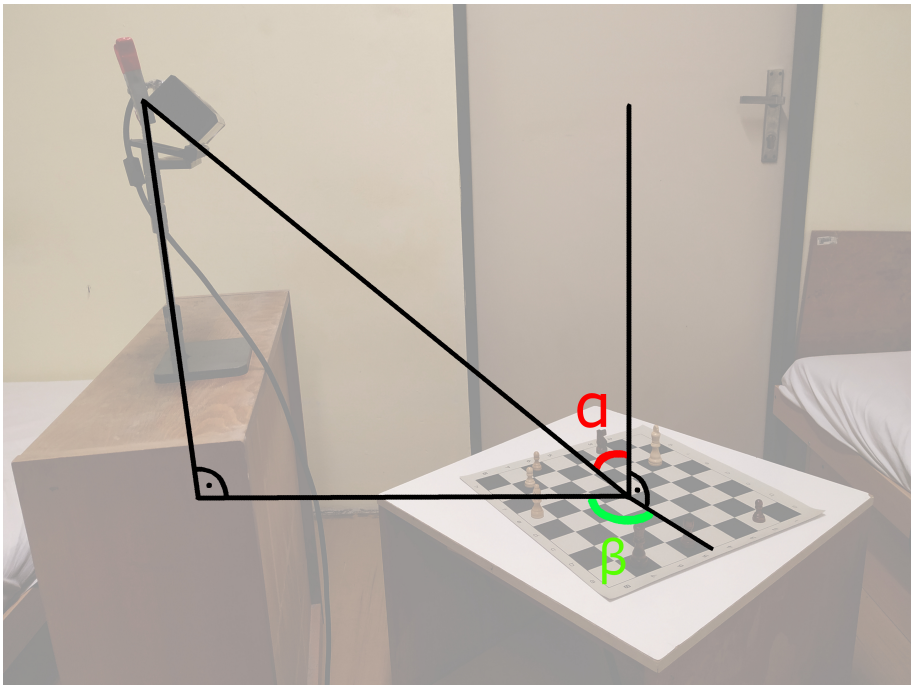
Podmínky testování

Testovacím prostředím byl interiér, který je vidět na obrázku 4.1. Scéna je osvětlena pouze zářivkami ve stropě a skládá se ze šachovnice s velikosti mřížky 38x38 cm a figurek velikosti 3,5-7,5 cm. Zpracování probíhalo na notebooku střední třídy s procesorem Intel i5-3210M.

Obrázek 4.1: Snímek testovacího prostředí

4.1 Hledání šachovnice

Cílem testování bylo zjistit, jakou má program úspěšnost při lokalizaci šachovnice. Senzor byl od středu šachovnice vzdálen jeden metr a byl umisťován do různých pozic okolo šachovnice. Tyto pozice je možné unikátně identifikovat dvěma úhly, viz obrázek 4.2. První je úhel mezi normálou šachovnice a úsečkou senzor-střed šachovnice, nazveme ho α. Druhý je otočení šachovnice na stole



Obrázek 4.2: Vizualizace úhlů při hledání šachovnice. α - úhel pozorování vzhledem k normále, β - úhel otočení šachovnice

vzhledem k senzoru, nazveme ho β. Tento úhel je nulový, pokud se senzor dívá na šachovnici z pohledu hráče s bílými figurkami, a roste při otočení šachovnice ve směru hodinových ručiček okolo svého středu.

Úhel α byl testován v rozpětí 10-90° a to vždy s odstupem 10°. Úhel β byl testován pod úhly 0°, 30°, 45°, 60° a 90°. Všechny ostatní úhly do 360°, dělitelné 30, nebo 45, jsou postavením figurek symetrické k nějakému z testovaných úhlů. Šachovnice na sobě měla rozestaveny figurky v základním postavení.

V každém nastavení bylo provedeno deset pokusů o lokalizaci šachovnice. Pro ujištění, že lokalizace proběhla úspěšně a monitorování funguje, bylo navíc při každém nalezení šachovnice taženo krajními pěšci jak bílé, tak černé barvy a následně oběma jezdci obou barev na jedinou volnou pozici. Toto opatření by mělo zamezit případům, při kterých se šachovnice na začátku jeví v pořádku, ale při hře dochází k poruchám pozic figurek, které činí sledování prakticky nemožným.

Všechny zaznamenávané informace o lokalizaci jsou detailně popsány v legendě přílohy A.2, která obsahuje také naměřené výsledky. Záznam obsahuje, zda:

- lokalizace šachovnice i sledování figurek proběhlo bez chyby
- objevil se problém s posunem šachovnice o řádky, či sloupce, ale bylo možné ho napravit nastavením parametrů v uživatelském rozhraní.
- objevil se problém s ořezem figurek, ale bylo možné ho napravit nastavením parametrů v uživatelském rozhraní.
- objevila se chyba při tažení figurkami, která znemožňovala sledování

- objevily se artefakty (figurky náhodně mizí/objevují se/mění barvu), které naznačují, že ve sledování se objevují chyby, které však díky průměrování nenarušují průběh hry.

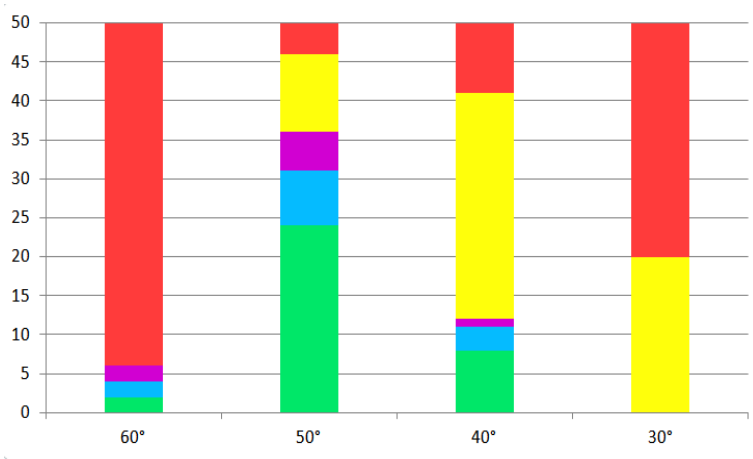
Výsledky

Prvním poznatkem z měření je, že úhel β měl na měření vliv pouze při α rovno 60°, ve kterém docházelo k detekcím šachovnice pouze pod úhlem β rovným 30°, nebo 45°. Pod těmito úhly pravděpodobně figurky zastiňují šachovnici tak, že je možné ji částečně rozpoznat.

Dále z měření jasně plyne, že sledování je vhodné pod úhlem α okolo 50–40°. Při větším úhlu α, kdy se na šachovnici díváme více z boku, 60° a více, nelze správně určit šachovnici. Při menších úhlech, kdy se na šachovnici díváme shora, 30° a méně, dochází ke špatné lokalizaci figurek, popřípadě rozpoznání jejich barev, což lze odůvodnit zmenšováním velikosti plochy, kterou senzor vidí z figurky.

Z provedených 50 měření na jednu hodnotu úhlu α byl počet bezchybných sledování poměrně malý, jak je vidět na obrázku 4.3. Pokud však budeme uvažovat počet chyb, které jsou opravitelné, neboli chyby, které může uživatel opravit v uživatelském rozhraní a přivést tak program do stavu stabilního sledování, úspěšnost se razantně zlepší. Jedná se o chyby způsobené špatným určením šachovnice o 1-2 řádky/sloupce, které lze opravit manuálním posunem šachovnice, a chyby způsobené špatnou detekcí figurek kvůli nevhodnému ořezu, které lze opravit zvětšením ořezu figurky a snížením počtu bodů požadovaných na její detekci, nebo kombinaci obou chyb.

Důležitým poznatkem získaným během měření je fakt, že hodnoty nastavení ořezu figurky a nastavení počtu bodů na detekci figurky, které byly empiricky vybrány během vývoje programu, nemusí být vhodné za všech situací. Dobře fungují hlavně u α rovno 50°, pokud α začneme snižovat na 40° a 30°, je lepší ořez zvýšit o několik milimetrů a toleranci na počet bodů snížit. Proto lze ve



Obrázek 4.3: Graf výsledků hledání šachovnice v závislosti na úhlu α. Zelená - sledování bez chyb, Modrá - chyby s určením šachovnice a ořezem figurek, Fialová - chyby s určením šachovnice, Žlutá - chyby s ořezem figurek, Červená - neopravitelné chyby

výsledcích pozorovat velké množství chyb opravitelných změnou těchto proměnných. To může být námětem pro zlepšení programu, tyto proměnné by se například mohly adaptivně měnit podle úhlu α.

Závěrem lze odvodit dvě závislosti, které jsou již naznačeny ve druhém odstavci, ale lze je zobecnit: Čím větší je úhel α, tím více dochází ke špatnému určení šachovnice. Naopak čím menší je α, tím menší plocha je vidět z jednotlivých figurek, a proto dochází k chybám detekce figurek. Oba trendy jsou dobře viditelné na grafu 4.3.

Pro ověření myšlenky, že u malého úhlu α není z figurky vidět mnoho bodů, podle kterých by šla dobře určit, jsme provedli doplňkové měření pod úhlem α rovno 30°. Tentokrát jsme však vzdálenost senzoru od středu šachovnice snížili na 75 cm, aby senzor viděl větší množství bodů ze šachovnice a figurek. Ukázalo se, že došlo k výraznému zlepšení sledování, ovšem jen pro úhel α rovno 30°, menší úhly, 20° a méně, stále nebyly schopné rozpoznávat figurky.

4.2 Praktické použití

Cílem bylo prozkoumat, zda je program schopný sledovat delší šachové partie. Celkem bylo odehráno pět her o délkách 25, 37, 35, 33 a 64 tahů. Jejich podmínky, záznam a popis chyb sledování v průběhu lze nalézt v příloze A.3. Hry nedávají strategicky žádný smysl, jejich hlavním cílem bylo otestovat, zda správně funguje přesun a braní figurek, popřípadě jak se chová sledování na různých konfiguracích šachovnice.

Výsledky

Program zvládal vyhodnocovat hry rychlostí 4 až 6 snímků za vteřinu, to je zcela postačující pro plynulé sledování. Sledování zvládá vyhodnotit stav hry, pokud je mezi jednotlivými tahy prostor nad šachovnicí prázdný minimálně po dobu 1,5-2 vteřin.

Ve dvou hrách z pěti bylo potřeba na začátku hledat šachovnici vícekrát kvůli horšímu určení pěšců. Kromě artefaktů na obrazovce okamžitého stavu hry, které nemají vliv na kvalitu sledování, se objevil jen jeden větší problém. Tím bylo zaclonění pole a později pěšce dámou. Přítomnost dámy způsobovala větší poruchy detekce barvy a přítomnosti pěšáka. Dáma sama o sobě vytvářela dostatek chybných bodů, aby i na poli za ní byla detekována figurka, která tam ve skutečnosti nebyla.

Vhledem k provedenému měření, lze pro praktické použití říci, že senzor by se měl umisťovat 70-100 cm od šachovnice, pod úhlem 30-50° od normály. Pokud se na začátku sledování vyskytují chyby, lze buď sledování resetovat, nebo přes uživatelské rozhraní změnit některé parametry výpočtu. V případě, že je sledování na začátku stabilní, nebo stabilizováno uživatelem, neměly by se v průběhu sledování objevovat výrazné potíže s funkčností.

Závěr

Podařilo se nám vytvořit plně funkční program schopný v reálném čase sledovat průběh šachové partie senzorem Kinect verze 2. Při spuštění lze načíst hru ze souboru, nebo spustit novou, a zahájit sledování. Program v obrazu nalezne šachovnici, ověří, zda jsou figurky správně rozmístěny, a začne monitorovat zápas. Během něj umí validovat tahy a pořizovat záznam hry, který může být později uložen na disk. Největší předností pro praktické použití je schopnost odolávat rušivým elementům ve scéně, jako jsou například objekty pohybující se v okolí šachovnice, nebo krátkodobé zaclonění senzoru.

Měření provedená na výsledné implementaci ukázala, že nejvhodnější úhel, pod kterým by senzor měl sledovat hru, se pohybuje kolem 40 a 50 stupňů od normály šachovnice. Obecně lze pozorovat, že při zvýšení úhlu dochází ke zhoršení detekce šachovnice, a naopak při zmenšení úhlu dochází k horšímu určování pozic figurek. Během měření zkoumajícího praktickou použitelnost se často objevovaly krátkodobé problémy s určením pozice, nebo barvy figurky. Díky průměrování v čase však neovlivňovaly stav hry. Při měření pod různými úhly trpělo velkou chybovostí správné určení figurek, což bylo ovšem často zaviněno nevhodnou konstantou pro ořez figurek. Změna této konstanty, nebo její adaptivní přizpůsobení by mohla přinést významné zlepšení výsledků programu při zahájení sledování.

Použití Kinectu namísto RGB kamery přineslo výhody jak při hledání herního plánu, tak při určování pozice figurek. Najít, díky prostorovým datům, jako první rovinu se šachovnicí silně omezuje prostor, který musí navazující algoritmy prohledávat, což snižuje jak chybovost, tak čas zpracování. Detekce figurek by v ideálním případě byla jen přímočarý průzkum prostoru nad jednotlivými políčky. Zde se však ukázalo, že technologie použitá na snímání hloubky má zápory, které výrazně zhoršují kvalitu dat. To pro použitelnost Kinectu na sledování deskových her znamená, že hloubkový senzor je dobře použitelný pouze u her, jejichž figurky/kameny jsou z matriálu dobře odrážejícího infračervené záření a mají dostatečnou velikost, aby byly rozpoznatelné od šumu.

Možnosti dalšího rozšíření

Práci lze ze současného stavu rozšiřovat několika směry. Po stránce spolehlivosti a rychlosti programu by bylo vhodné zaměřit se na algoritmus hledání šachovnice v prostorových datech. Při použití vhodných heuristik, například podle shody barevnosti reálných dat a predikované šachovnice, by mělo být možné dosáhnout zvýšení spolehlivosti pozice výsledné šachovnice. Zrychlení by šlo dosáhnout lepší paralelizací jednotlivých algoritmů, či agresivnějším zmenšováním dat, na který algoritmy pracují. Pokud by došlo k dostatečnému zrychlení lokalizace šachovnice, mohl by ji program provádět automaticky bez pokynu uživatele.

Vylepšeny mohou být také algoritmy zabývající se detekcí figurek, obzvláště co se týče detekce barvy a ořezu figurek.

Vhodným rozšířením programu může být podpora pro načítání více formátů

šachových notací, popřípadě interaktivní procházení záznamu hry a možnost vrátit se do předchozích tahů. Funkčnost by šlo také rozšířit o pozorování dalších her odehrávajících se na pravidelné mřížce.

Literatura

[1] A Czyzewski, M., Laskowski, A. a Wasik, S. (2018). Chessboard and chess piece recognition with the support of neural networks.

[2] Amos, E. Xbox one's kinect. [online]. [cit. 08.07.2019]. Dostupné z: `https://en.wikipedia.org/wiki/File:Xbox-One-Kinect.jpg`.

[3] C Gonzalez, R. a E Woods, R. (2002). *Digital Image Processing (2nd Edition)*. ISBN 0201180758.

[4] Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PAMI-8**(6), 679–698. ISSN 0162-8828. doi: 10.1109/TPAMI.1986.4767851.

[5] Chen, C., Lu, J. a Ma, K. (2017). *Computer Vision – ACCV 2016 Workshops: ACCV 2016 International Workshops, Taipei, Taiwan, November 20-24, 2016, Revised Selected Papers*. Number díl 2 in Lecture Notes in Computer Science. Springer International Publishing. ISBN 9783319544274.

[6] Danner, C. a Kafafy, M. (2015). Visual chess recognition ee 368 , spring 2015.

[7] Fischler, M. A. a Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, **24**(6), 381–395. ISSN 0001-0782. doi: 10.1145/358669.358692.

[8] Hassanein, A. S., Mohammad, S., Sameer, M. a Ragab, M. E. (2015). A survey on hough transform, theory, techniques and applications. *CoRR*.

[9] Kipman, A. (2018). Introducing project kinect for azure. [online]. [cit. 08.07.2019]. Dostupné z: `https://www.linkedin.com/pulse/introducing-project-kinect-azure-alex-kipman`.

[10] Koray, C. a Sümer, E. (2016). A computer vision system for chess game tracking.

[11] Li, L. (2018). Time-of-flight camera – an introduction.

[12] Moravčík, M., Schmid, M., Burch, N., Lisý, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M. a Bowling, M. (2017). Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, **356**(6337), 508–513. ISSN 1095-9203. doi: 10.1126/science.aam6960.

[13] Nguyen, A. a Le, B. (2013). 3d point cloud segmentation: A survey. pages 225–230. ISBN 978-1-4799-1201-8. doi: 10.1109/RAM.2013.6758588.

[14] Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, **9**(1), 62–66. ISSN 0018-9472. doi: 10.1109/TSMC.1979.4310076.

[15] Shotton, J., Fitzgibbon, A., Blake, A., Kipman, A., Finocchio, M., Moore, B. a Sharp, T. (2011). Real-time human pose recognition in parts from a single depth image. IEEE.

[16] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. a Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, **529**, 484–503.

[17] Wilson, M. (2017). Exclusive: Microsoft has stopped manufacturing the kinect. [online]. [cit. 08.07.2019]. Dostupné z: `https://www.fastcompany.com/90147868/exclusive-microsoft-has-stopped-manufacturing-the-kinect`.

[18] Yaniv, Z. (2010). Random sample consensus (ransac) algorithm, a generic implementation.

Seznam obrázků

Seznam tabulek

A. Přílohy

A.1 Obsah přiloženého CD

- `/UzivatelskaDokumentace/UzivatelskaDokumentace.pdf` - uživatelská dokumentace
- `/GenerovanaDokumentace/index.html` - generovaná dokumentace k programu
- `/Kod` - zdrojové kódy
- `/Text/text.pdf` - kopie této práce

A.2 Data z měření hledání šachovnice

Legenda:

- Úhel α - velikost úhlu od normály šachovnice.
- Úhel β - otočení šachovnice na stole vzhledem k senzoru. Úhel 0° je z pohledu hráče s bílými figurkami. Úhel roste při otáčení šachovnice okolo jejího středu ve směru hodinových ručiček.
- Pokus - číslo pokusu s konkrétním nastavením.
- Prázdný řádek - nebyla nalezena šachovnice
- Ok - měření proběhlo zcela bez problémů
- Posun - program nalezl šachovnici o 1-2 sloupce/řádky vedle. Oprava šla provést pomocí tlačítek posunu šachovnice.
- Výška - figurky nebyly na začátku ideálně lokalizovány. Oprava šla provést jednoduše pomocí táhla ořezu figurek a nastavení počtu bodů na indikaci figurky.
- Chyba pohybu - figurky již byly v základním postavení správně nalezené, ale při pohybech se ukázalo, že dochází k chybám (mizejícím/přebývajícím figurkám, změnám jejich barev), které znemožňují plynulé sledování
- Artefakty - v průběhu sledování se na vizualizaci aktuálního stavu sledování objevovaly artefakty (mizející/přebývající figurky, změny jejich barev, ...), které nenarušily plynulost hry, ale mohou indikovat potíže narušující sledování při delší hře.

* Vzdáleností senzoru od šachovnice je myšlena vzdálenost přední strany senzoru od středu sledované šachovnice

Úhel α	Úhel β	Pokus	Ok	Posun	Výška	Chyba pohybu	Artefakty
60°	0°	1					
60°	0°	2					
60°	0°	3					
60°	0°	4					
60°	0°	5					
60°	0°	6					
60°	0°	7					
60°	0°	8					
60°	0°	9					
60°	0°	10					
60°	30°	1					
60°	30°	2		x			
60°	30°	3		x	x	x	
60°	30°	4					
60°	30°	5					
60°	30°	6		x	x		
60°	30°	7		x		x	
60°	30°	8					
60°	30°	9		x			
60°	30°	10		x	x		
60°	45°	1					
60°	45°	2	x				
60°	45°	3					
60°	45°	4					
60°	45°	5					
60°	45°	6	x				
60°	45°	7					
60°	45°	8		x		x	
60°	45°	9					
60°	45°	10					
60°	60°	1					
60°	60°	2					
60°	60°	3					
60°	60°	4					
60°	60°	5					
60°	60°	6					
60°	60°	7					
60°	60°	8					
60°	60°	9					
60°	60°	10					

Vzdálenost senzoru od šachovnice: 1 m.

Úhel α	Úhel β	Pokus	Ok	Posun	Výška	Chyba pohybu	Artefakty
60°	90°	1					
60°	90°	2					
60°	90°	3					
60°	90°	4					
60°	90°	5					
60°	90°	6					
60°	90°	7					
60°	90°	8					
60°	90°	9					
60°	90°	10					
50°	0°	1			x		
50°	0°	2	x				
50°	0°	3	x				x
50°	0°	4	x				x
50°	0°	5	x				
50°	0°	6	x				
50°	0°	7			x		
50°	0°	8	x				
50°	0°	9				x	
50°	0°	10	x				
50°	30°	1		x	x		
50°	30°	2	x				
50°	30°	3		x			
50°	30°	4		x	x		x
50°	30°	5		x			
50°	30°	6		x			
50°	30°	7	x				
50°	30°	8		x		x	
50°	30°	9			x		
50°	30°	10	x				
50°	45°	1			x		x
50°	45°	2		x	x		
50°	45°	3	x				
50°	45°	4					
50°	45°	5	x				
50°	45°	6	x				
50°	45°	7		x	x		x
50°	45°	8		x	x		x
50°	45°	9		x			
50°	45°	10			x		x

Vzdálenost senzoru od šachovnice: 1 m.

Úhel α	Úhel β	Pokus	Ok	Posun	Výška	Chyba pohybu	Artefakty
50°	60°	1			x		
50°	60°	2		x	x		x
50°	60°	3		x			
50°	60°	4	x				
50°	60°	5			x		
50°	60°	6	x				
50°	60°	7	x				
50°	60°	8			x		
50°	60°	9	x				
50°	60°	10	x				
50°	90°	1	x				
50°	90°	2	x				
50°	90°	3	x				
50°	90°	4					
50°	90°	5	x				
50°	90°	6			x		x
50°	90°	7	x				
50°	90°	8			x		
50°	90°	9		x	x		
50°	90°	10	x				
40°	0°	1			x	x	
40°	0°	2			x	x	
40°	0°	3	x				
40°	0°	4					
40°	0°	5	x				
40°	0°	6			x		
40°	0°	7			x		
40°	0°	8			x		
40°	0°	9			x		
40°	0°	10		x	x	x	
40°	30°	1			x		
40°	30°	2			x		
40°	30°	3		x	x		
40°	30°	4	x				x
40°	30°	5		x	x		x
40°	30°	6			x		
40°	30°	7			x		
40°	30°	8			x		x
40°	30°	9			x		
40°	30°	10	x				

Vzdálenost senzoru od šachovnice: 1 m.

Úhel α	Úhel β	Pokus	Ok	Posun	Výška	Chyba pohybu	Artefakty
40°	45°	1		x			x
40°	45°	2	x				
40°	45°	3			x	x	
40°	45°	4	x				
40°	45°	5			x		
40°	45°	6			x		
40°	45°	7			x		
40°	45°	8	x				x
40°	45°	9			x	x	
40°	45°	10			x		x
40°	60°	1			x		x
40°	60°	2			x		
40°	60°	3			x		
40°	60°	4			x		
40°	60°	5			x	x	
40°	60°	6		x	x		
40°	60°	7			x		x
40°	60°	8			x		
40°	60°	9			x		x
40°	60°	10			x	x	
40°	90°	1	x				x
40°	90°	2			x		
40°	90°	3			x		
40°	90°	4			x		
40°	90°	5			x	x	
40°	90°	6			x		
40°	90°	7			x		
40°	90°	8			x		
40°	90°	9			x		
40°	90°	10			x		
30°	0°	1			x		x
30°	0°	2			x		x
30°	0°	3			x	x	
30°	0°	4			x		x
30°	0°	5			x	x	
30°	0°	6			x	x	
30°	0°	7			x	x	
30°	0°	8			x	x	
30°	0°	9			x		x
30°	0°	10			x	x	

Vzdálenost senzoru od šachovnice: 1 m.

Úhel α	Úhel β	Pokus	Ok	Posun	Výška	Chyba pohybu	Artefakty
30°	30°	1			x		x
30°	30°	2			x	x	
30°	30°	3			x	x	
30°	30°	4			x		x
30°	30°	5			x	x	
30°	30°	6			x		x
30°	30°	7			x	x	
30°	30°	8			x	x	
30°	30°	9			x	x	
30°	30°	10			x		x
30°	45°	1			x		x
30°	45°	2			x		x
30°	45°	3			x		x
30°	45°	4			x	x	
30°	45°	5			x	x	
30°	45°	6			x		x
30°	45°	7			x	x	
30°	45°	8			x		x
30°	45°	9			x	x	
30°	45°	10			x		x
30°	60°	1			x	x	
30°	60°	2			x		x
30°	60°	3			x	x	
30°	60°	4			x	x	
30°	60°	5			x		x
30°	60°	6			x	x	
30°	60°	7			x	x	
30°	60°	8			x		x
30°	60°	9			x	x	
30°	60°	10			x	x	
30°	90°	1			x	x	
30°	90°	2			x	x	
30°	90°	3			x		x
30°	90°	4			x		x
30°	90°	5			x	x	
30°	90°	6			x	x	
30°	90°	7			x	x	
30°	90°	8			x		x
30°	90°	9			x	x	
30°	90°	10			x	x	

Vzdálenost senzoru od šachovnice: 1 m.

Úhel α	Úhel β	Pokus	Ok	Posun	Výška	Chyba pohybu	Artefakty
30°	0°	1			x		
30°	0°	2			x		
30°	0°	3			x		
30°	0°	4			x		
30°	0°	5			x		x
30°	30°	1			x		x
30°	30°	2			x		
30°	30°	3			x		
30°	30°	4			x		x
30°	30°	5			x		
30°	45°	1			x		
30°	45°	2			x		
30°	45°	3			x		
30°	45°	4			x		x
30°	45°	5			x		
30°	60°	1			x		x
30°	60°	2			x		
30°	60°	3			x		
30°	60°	4			x		
30°	60°	5			x		
30°	90°	1			x		
30°	90°	2			x		
30°	90°	3			x		
30°	90°	4			x		
30°	90°	5			x		

Vzdálenost senzoru od šachovnice: 0,75 m.

A.3 Data z měření praktického použití

Všechny hry byly odehrány pod úhlem β rovným 90°, ze vzdálenosti 90 cm.

Hra číslo 1.

Podmínky: úhel α rovný 40°. Upravený ořez figurek 13 mm, tolerance 3 body.

Záznam: Pe2-e4; Pd7-d5; Pb2-b4; Bc8-g4; Qd1xg4; Nb8-c6; Pe4xd5; Nc6xb4; Qg4-g5; Ph7-h5; Qg5xe7; Ke8xe7; Bf1-b5; Pa7-a5; Ph2-h4; Ra8-a6; Bb5xa6; Nb4xc2; Ke1-d1; Pb7xa6; Kd1xc2; Qd8xd5; Ng1-f3; Qd5xa2; Ra1xa2

Popis průběhu: na polích b7, a6 a a5 docházelo k problikávání pěšců černé barvy. Problikáváním je myšlen chvilkový výpadek detekce figurky. Dáma na poli g5 zastínila pole h5, a později černého pěšce na h5. V prvním případě pole občas vypadalo, že na něm stojí figurka měnící se barvy, v případě že tam stála figurka, se taktéž stávalo, že se měnila její barva. Černý jezdec na poli c4 občas zbělal, ale neovlivnilo to hru.

Hra číslo 2.

Podmínky: úhel α rovný 40°. Upravený ořez figurek 13 mm, tolerance 3 body.

Záznam: Ph2-h4; Pg7-g5; Pf2-f4; Pe7-e5; Pd2-d4; Pc7-c5; Pb2-b4; Pa7-a5; Pa2-a4; Pb7-b5; Pc2-c4; Pd7-d5; Pe2-e4; Pf7-f5; Pg2-g4; Ph7-h5; Pa4xb5; Ph5xg4; Pb4xc5; Pg5xf4; Pc4xd5; Pf5xe4; Pd4xe5; Qd8xd5; Bc1xf4; Bf8xc5; Qd1-d4; Qd5xe5; Qd4xe4; Bc5-a7; Bf4xe5; Pa5-a4; Qe4-b4; Ng8-h6; Be5-f6; Ba7xg1; Qb4-e7; 1-0

Popis průběhu: kvůli problikávání některých pěšců bylo potřeba dvakrát resetovat hledání šachovnice, teprve poté byl stav uspokojivý. V průběhu hry pěšci na středu šachovnice občas problikávali, nemělo to však vliv na hru.

Hra číslo 3.

Podmínky: úhel α rovný 40°. Upravený ořez figurek 13 mm, tolerance 3 body.

Záznam: Pf2-f4; Pc7-c5; Pg2-g4; Pb7-b5; Pg4-g5; Pb5-b4; Bf1-h3; Pd7-d6; Pe2-e3; Bc8-d7; Pf4-f5; Pg7-g6; Pa2-a3; Pf7-f6; Pg5xf6; Pg6xf5; Nb1-c3; Nb8-c6; Pa3xb4; Pc5xb4; Ra1xa7; Ra8xa7; Ke1-f1; Nc6-d4; Nc3-a2; Ra7xa2; Pe3xd4; Ng8-h6; Qd1-h5; Nh6-f7; Qh5-d1; Nf7-e5; Pd4xe5; Pd6xe5; Qd1-h5; 1-0

Popis průběhu: bez jakýchkoli problémů.

Hra číslo 4.

Podmínky: úhel α rovný 45°. Upravený ořez figurek 13 mm, tolerance 3 body.

Záznam: Pd2-d4; Pd7-d5; Bc1-g5; Ph7-h6; Bg5xe7; Qd8xe7; Pe2-e4; Pd5xe4; Qd1-e2; Qe7-e5; Qe2xe4; Bc8-f5; Bf1-c4; Bf8-c5; Ng1-f3; Ng8-f6; Nf3-g5; Nf6-d5; Nb1-c3; Nb8-c6; Ke1-d2; Ke8-d7; Kd2-d3; Kd7-d6; Pf2-f4; Pf7-f6; Rh1-e1; Ra8-e8; Re1-e3; Re8-e6; Pg2-g4; Pb7-b5; Pb2-b4

Popis průběhu: bylo navíc provedeno jedno hledání šachovnice kvůli špatně rozpoznatelným pěšcům. Pěšci v průběhu hry problikávali, nemělo to však vliv na hru.

Hra číslo 5.

Podmínky: úhel α rovný 30°. Upravený ořez figurek 13 mm, tolerance 3 body.

Záznam: Ph2-h4; Pg7-g5; Pf2-f4; Pe7-e5; Pd2-d4; Pc7-c5; Pb2-b4; Pa7-a5; Ph4xg5; Pe5xf4; Pd4xc5; Pa5xb4; Pg5-g6; Pf4-f3; Pc5-c6; Pb4-b3; Pg2xf3; Ph7xg6; Pa2xb3; Pb7xc6; Rh1xh8; Ra8xa1; Qd1xd7; Ke8xd7; Nb1-c3; Ng8-f6; Ng1-h3; Ra1xc1; Ke1-f2; Nf6-e4; Pf3xe4; Rc1xf1; Kf2xf1; Pg6-g5; Nh3xg5; Bf8-b4; Ng5xf7; Bb4xc3; Nf7-d6; Kd7xd6; Rh8-e8; Qd8xe8; Pe4-e5; Qe8xe5; Pb3-b4; Nb8-a6; Pb4-b5; Bc8-b7; Pb5xa6; Bb7xa6; Kf1-g1; Ba6xe2; Kg1-f2; Qe5-e4; Kf2-g1; Bc3-d4; Kg1-h2; Qe4xc2; Kh2-h1; Qc2-g6; Kh1-h2; Be2-f1; Kh2-h1; Qg6-g2; 0-1

Popis průběhu: pěšec na e2 mizel tak, že to na 3 vteřiny přerušilo hru. Černá věž na a1 občas měnila barvu na bílou, neovlivnilo to hru. Dále docházelo k občasnému problikávání pěšců a věží, neovlivnilo to však hru. Černá dáma na konci hry na c2 vrhala stín na pole za sebou, které pak začínalo indikovat přítomnost figurky. Opět to ale nemělo vliv na hru.