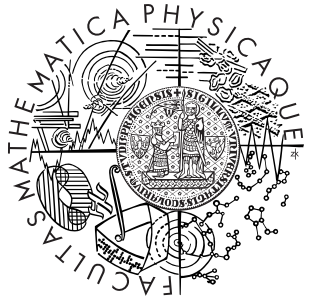
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

# BAKALÁŘSKÁ PRÁCE




Andrej Mikulík

## Navigace robota na šachovnici

Katedra softwarového inženýrství

Vedoucí bakalářské práce: Mgr. Viliam Holub
Studijní program: Informatika, obecní informatika

2007

Rád by som poďakoval vedúcemu mojej bakalárskej práce Mgr. Viliamovi Holubovi za rady pri návrhu algoritmov a konštruktívne pripomienky pri písaní práce. Ďalej by som chcel poďakovať Ing. Anne Mikulíkovej a Márii Šoltésovej za trpezlivosť a psychickú odolnosť pri jazykovej korektúre a študentom Pavlovi Juskovi, Martinovi Kupcovi a Tomášovi Petrůškovi za výdatnú pomoc pri stavbe robota, na ktorom som výsledky tejto práce testoval.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 28.5.2007 Andrej Mikulík

# Obsah

Název práce: Navigace robota na šachovnici
Autor: Andrej Mikulík
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí bakalářské práce: Mgr. Viliam Holub
e-mail vedoucího: holub@dsrg.mff.cuni.cz

Abstrakt: Predložená práca sa zaoberá lokalizáciou mobilného robota. Lokalizácia je postavená na počítačovom videní. Zo snímaného obrazu navrhnuté algoritmy rozpoznávajú šachovnicovú podložku, po ktorej sa robot pohybuje. Pomocou nej je analyzovaný uhol natočenia a pozícia robota.

Navrhnutý algoritmus bol otestovaný a zhodnotený na reálnom robotovi.

Klíčová slova: navigace, robotika, lokalizace, počítačové vidění

Title: Robot Navigation on chessboard
Author: Andrej Mikulík
Department: Department of Software Engineering
Supervisor: Mgr. Viliam Holub
Supervisor‘ s e-mail address: holub@dsrg.mff.cuni.cz

Abstract: Offered thesis investigates localization of mobile robot. Localization is based on computer vision. Projected algorithms recognize square-patterned pad on which robot moves from recorded image. Inclination angle and position of the robot is then analyzed from image.

Proposed algorithm was discussed according to tests on real robot.

Keywords: robot, navigation, chessboard, computer vision

# Kapitola 1

# Úvod

Témou tejto bakalárskej práce je lokalizácia mobilného robota založená na analýze šachovnicovej podlahy snímanej kamerou umiestnenou na robotovi.

## 1.1 Motivácia

Lokalizácia je jeden zo základných problémov mobilných robotov. Informácia o aktuálnej pozícii robota je kľúčovým poznatkom a základným predpokladom nutným pre riešenie ďalších užitočných úloh autonómnych robotov.

Existuje viacero typov lokalizácie. Opierajú sa o rôzne typy senzorov, sú vhodné do rôznych typov prostredí. Každá z nich je vhodná na iný typ robota a úlohu, pre ktorú je postavený. Rád by som vytvoril algoritmus, ktorý by bol nezávislý na type robota a ktorý by bol natoľko spoľahlivý, aby ho bolo možné použiť ako základ autonómnych robotov plniacich najrôznorodejšie úlohy. Šachovnicová podlaha, ktorá je predpokladom pre navrhnutý typ lokalizácie, síce obmedzuje pôsobnosť robota do špeciálne upravených interiérov, ale mala by na druhú stranu zabezpečiť dobre rozpoznateľný pohyb a tým spoľahlivosť.

Aplikovaný by mohol byť napríklad ako lokalizácia pre autonómneho robota skladníka, kde namaľovať na podlahu skladu šachovnicu, alebo použiť vhodné štvorcové dlaždice predstavuje oproti inej skladovej technike minimálnu investíciu.

## 1.2 Cieľ

Cieľom bakalárskej práce bude navrhnúť a otestovať algoritmus, ktorý dokáže udržiavať známu pozíciu robota pomocou analýzy obrazu. Algoritmus bude v obraze rozpoznávať šachovnicu, po ktorej sa bude robot pohybovať a mal by dokázať v každom okamžiku správne určiť polohu a natočenie robota. Počiatočná poloha bude algoritmu poskytnutá zvonku. Obraz by mal robot získavať pomocou kamery, ktorú si ponesie sám.

V druhom kroku je cieľom navrhnutý algoritmus implementovať do skutočného mobilného robota a otestovať jeho funkčnosť v praxi. Algoritmus by

mal byť čo možno najrobustnejší, odolný voči zlej kvalite obrazu spôsobenej rôznym šumom a zlým osvetlením scény.

## 1.3 Prehľad jednotlivých kapitol

V druhej kapitole definujeme základné pojmy a problémy lokalizácie. Načrtneme alternatívne druhy lokalizácie mobilného robota, popíšeme ich vlastnosti a zhrnieme niekoľko základných kritérií, podľa ktorých môžeme jednotlivé riešenia porovnávať.

V rámci tretej kapitoly podrobne popíšeme návrh algoritmu a funkčnosť jednotlivých častí. Zaoberať sa budeme okrem iného hranovou detekciou, analýzou čiar, určovaním uhla natočenia a detekciou pohybu mobilného robota.

Štvrtú kapitolu venujeme implementácií algoritmu lokalizácie a jeho napojenie na riadiaci program reálneho robota. Súčasťou tejto kapitoly bude aj programátorská dokumentácia.

Zhodnotenie, dosiahnuté výsledky, presnosť a vlastnosti algoritmu budú popísané v poslednej piatej kapitole. Budú tu taktiež spomenuté nedostatky a navrhnuté ďalšie kroky vývoja algoritmu.

# Kapitola 2

# Rôzne typy riešení a spôsoby ich porovnávania

## 2.1 Základné pojmy súvisiace s problémom lokalizácie

Za pozíciu robota budeme považovať usporiadanú trojicu $p = (x, y, \alpha)$, kde $x$, $y$ budú súradnice v nejakej karteziánskej súradnicovej sústave a $\alpha$ je uhol, ktorým je robot natočený. Lokalizovať robota potom znamená určiť pozíciu $p$ z priestoru všetkých možných pozícií. Výška, v ktorej sa robot nachádza ma v tomto prípade nezaujíma. Lokalizácia bude určená pre robota pohybujúceho sa po zemi a teda výška môže byť definovaná napríklad mapou (ak je mapa známa).

Niektoré algoritmy tzv. pravdepodobnostné namiesto trojice $(x, y, \alpha)$ určia pravdepodobnostnú funkciu, ktorá udáva, s akou pravdepodobnosťou sa robot vyskytuje na nejakom mieste. Ďalej netreba zabudnúť rozlišovať pozíciu robota, ktorú určí lokalizácia, a pozíciu, na ktorej sa robot v skutočnosti nachádza. Ak lokalizácia určí pozíciu robota zle a to až za hranicou prípustnej chyby daného algoritmu, budeme pozíciu robota považovať za stratenú.

## 2.2 Dve úlohy lokalizácie

Úlohy kladené na lokalizáciu môžeme rozdeliť do dvoch typov. Prvý z nich je problém lokalizácie robota na známej mape, v ktorej sa robot nachádza a to bez predošlej informácie o polohe. S ním spojená je tzv. relokalizácia alebo znovunájdenie robota po predošlej strate správnej pozície. Druhý problém lokalizácie je sledovanie pohybu (tracking) a schopnosť udržať si správnu pozíciu pri ľubovoľnom možnom pohybe robota.

V prípade, že je lokalizácia natoľko robustná, aby sa robot počas pohybu nestrácal, je možné za predpokladu, že je počiatočná pozícia známa, neriešiť prvý typ problému, globálnu lokalizáciu, a spoľahnúť sa iba na sledovanie zmeny pozície. Na takýto typ riešenia sa zameriame aj my. Rovnako je možné vynechať zisťovanie globálnej pozície v prípade, ak má lokalizácia za úlohu

navigovať robota iba na konkrétom mieste pre riešenie nejakej lokálnej úlohy. V takom prípade sa predpokladá, že o globálnu pozíciu sa stará iný systém, resp. ak nie je potrebná, nemusí sa riešiť vôbec.

Naopak existujú iné typy lokalizácii (napr. GPS), ktoré už zo svojej podstaty riešia lokalizáciu výhradne určovaním globálnej pozície. V takom prípade je schopnosť správneho sledovania pohybu zbytočná.

Veľa robustných lokalizácií ale rieši obe úlohy. Prvú preto, že je to nutné pri prípadnej strate robota. Druhú napríklad preto, že býva z pravidla oveľa jednoduchšia na výpočtový výkon a je teda možné zisťovať pozíciu rádovo rýchlejšie. Iný dôvod pre sledovanie pohybu je ten, keď globálna lokalizácia funguje iba za podpory tej lokálnej. V prípade, že robot nedokáže detekovať stratu, je možné globálnu lokalizáciu vykonávať v pravidelných intervaloch.

## 2.3 Kritéria hodnotenia rôznych typov lokalizácií

Úloha lokalizácie nie je žiadna novinka. Ako základný problém ju vývojári mobilných robotov riešia od samých začiatkov mobilnej robotiky. Exituje preto celá rada metód a riešení lokalizácie.

Na tomto mieste vymenujeme hlavné kritéria, podľa ktorých je možné jednotlivé riešenia porovnávať alebo kategorizovať. Ich dôležitosť potom úzko súvisí s úlohou, pre ktorú je robot s danou lokalizáciou určený.

- **Presnosť lokalizácie.** Rôzna aplikácia vyžaduje rôznu presnosť. Pre roboty určené do vonkajšieho prostredia (outdoor) bývajú zaujímavé presnosti v rádovo desiatkach centimetrov až jednotkách metrov. Samozrejme sa to líši podľa veľkosti a typu úlohy, ktorú ma robot vykonávať. Oproti tomu menšie mobilné roboty určené pre vykonávanie drobnejších úloh potrebujú poznať svoju polohu s presnosťou na niekoľko centimetrov.

- **Schopnosť udržať si správnu pozíciu.** Niektoré metódy sú nezávislé na vzdialenosti, ktorú robot prekoná, iné využívajú na obnovenie svojej polohy informáciu získanú v predošlom kroku. Hlavne metódy druhého typu sú často náchylné na akumulovanie chyby a postupné strácanie robota.

- **Schopnosť riešiť znovunájdenie sa („kidnapped robot“ problém).** Ide o problém relokalizácie pozície robota po predošlej strate, či už zapríčinenej nedokonalosťou lokalizačného algoritmu, zmenením vonkajších podmienok, alebo napríklad pôsobením inej vonkajšej sily, ktorú robot nedokáže rozpoznať. Môže ísť o náraz alebo prenesenie robota na iné miesto. Základný predpoklad pre riešenie tohto problému je samotná schopnosť detekcie straty pozície.

- **Závislosť na kvalite vonkajších podmienok, podmienky kladené na konštrukciu robota.** Rozhodujúce môžu byť povrch po ktorom sa

robot pohybuje, kvalita osvetlenia, poveternostné podmienky alebo nutnosť vybaviť prostredie prvkami, s ktorými lokalizácia spolupracuje alebo ich využíva.

Lokalizácia môže tiež od robota vyžadovať nejaké konštrukčné predpoklady. Ak využíva metóda kompas, ten je nutné umiestniť čo najďalej od akéhokoľvek zdroja elektromagnetického šumu, menovite čo možno najďalej od motorov. Čo sa týka prostredia, pre správne fungovanie kompasu by malo byť v okolí čo najmenej kovov, ktoré porušia homogenitu magnetického pola. Typ lokalizácie využívajúci kompas, bude teda pravdepodobne vyžadovať vonkajšie prostredie. Použitie kamery bude zase často predpokladať jej vhodné umiestnenie, správnu výšku a sklon.

- **Náročnosť na kvalitu a výpočtový výkon hardvéru potrebného pre lokalizáciu.** Na tomto poli bývajú medzi algoritmami často priepastné rozdiely. Kým na implementáciu jedného postupu bohato postačuje 8-bitový mikrokontrolér s taktovacou frekvenciou v jednotkách MHz alebo niekoľko jednoúčelových lacných integrovaných obvodov, iné už zo svojej podstaty vyžadujú hardvér podstatne dokonalejší a výpočtový výkon rádovo väčší, rátaný v stovkách MHz alebo dokonca v jednotkách GHz.

  Príkladom nenáročných systémov sú lokalizácie založené na odometrii, sonaroch, svetlocitlivých diódach a podobne. Naopak veľmi náročné na výpočtový výkon bývajú lokalizácie, ktoré využívajú spracovanie obrazu.

- **Zložitosť implementácie.** Práve toto kritérium býva neprávom podhodnotené. Je to ale často hlavný dôvod nasadenia alebo nenasadenia konkrétneho riešenia v praxi. Darmo je nejaký algoritmus o kúsok lepší ako iný, ak nasadenie toho druhého vyžaduje len zlomok úsilia.

  Zložitosť implementácie by sme mohli rozdeliť na zložitosť vlastného algoritmu a na zložitosť aplikovania lokalizácie na konkrétneho robota.

## 2.4 Prehľad niektorých metód lokalizácie

**Odometria.** Jeden z najzákladnejších a najjednoduchších prístupov je odometria. Jedná sa o metódu, ktorá je závislá na meraní otočenia kolies. Na tento účel sa využívajú enkodéry, ktoré poskytujú informáciu či a prípadne ktorým smerom sa pootočilo koleso. Pri pohybe sa potom táto informácia nahromadí a následne podľa typu robota, predovšetkým jeho podvozku, je možné namodelovať posun robota oproti minulej pozícii.

Výhody tejto metódy sú jednoduchá implementácia a nenáročnosť na výpočtový výkon. Základným nedostatkom tejto metódy je postupná akumulácia chyby, ktorou je zaťažený každý výpočet posunutia. Veľkosť chyby je veľmi závislá na kvalite a rozlíšení použitých enkodérov ale aj na hladkosti povrchu a priľnavosti kolies.

Ešte väčší problém ako akumulácia chyby kvôli nepresnosti hardvéru nastane v prípade podkĺznutia kolesa s enkodérom. Robot príme otočenie kolesa a premietne ho do zmeny svojej polohy napriek tomu, že sa koleso vôbec nemuselo pohnúť z miesta. Bez ďalších mechanizmov to nie je možné detekovať a informácia o pozícii sa stane v tú chvíľu nepoužiteľnou.

**Orientovanie sa v známej mape pomocou detekcie stien.** Pomocou sonarov, infračervených alebo iných diaľkomerov môže robot sledovať vzdialenosti stien v okolí robota. Po zmapovaní dostatočne veľkého okolia je za predpokladu jeho jedinečnosti možné priradiť zmapovanú časť k úseku na mape.

Presnosť lokalizácie je závislá na presnosti a rozlíšení diaľkomerov. Lokalizácia toho typu neakumuluje chybu, môže byť odolná voči strácaniu a dokáže riešiť znovunájdenie robota. Mapa musí byť vhodného typu. Tento spôsob lokalizácie je typický pre orientáciu v bludisku, v chodbách budovy alebo v miestnostiach.

**Lokalizácia pomocou GPS.** Je vhodná najmä pre robotov určených do vonkajšieho prostredia. Jej presnosť sa pohybuje v jednotkách metrov a nehrozí pri nej ani akumulovanie chyby ani strata pozície (snáď okrem prípadu, keď GPS príjímač stratí signál). Rovnako, nie je problém znovunájdenie uneseného robota. Pre implementáciu je potrebný špeciálny hardvér tzv. GPS prijímač, ktorý na základe informácii vysielaných z GPS družíc spočíta svoju polohu.

Ďalšou nespornou výhodou pre použitie GPS v mobilnom robote je vďaka špeciálnemu hardvéru aj veľmi jednoduchá implementácia. Všetky potrebné výpočty za nás vykoná GPS prijímač, ktorý nám spravidla v pravidelných intervaloch sprostredkováva výslednú pozíciu.

**Monte Carlo lokalizácia (MCL).** Je založená na kombinácií viacerých metód lokalizácie a modelovaní ich nepresností. Konkrétne pre pohyb po šachovnici sa dá s úspechom použiť informácia enkodérov, ktorú ďalej spresňujú snímače v podvozku. Tie musia byť schopné rozlíšiť, na ktorom z dvoch typoch políčok sa robot nachádza. Na čierno-bielej šachovnici to typicky bude svetlocitlivý senzor schopný rozlíšiť svetlé biele políčko čierneho.

Kombinácia metód lokalizácie pomocou odometrie a detekcie stien s využitím MCL ponúka robustné riešenie pre lokalizáciu aj znovunájdenie robota [1]. Táto metóda na rozdiel od ostatných spomenutých je ako jediná pravdepodobnostná, čo znamená, že skutočná poloha robota je daná pravdepodobnostnou funkciou, v ideálnom prípade Gaussovou. Podobnou metódou je Markovova lokalizácia [2], ktorá sa od MCL líši diskrétnym rozdelením mapy na menšie úseky, medzi ktorými sa algoritmus snaží nájsť ten, v ktorom sa robot nachádza s najväčšou pravdepodobnosťou.

**Lokalizácia pomocou kamery.** Je to typ lokalizácie, ktorá je spracovaná v tejto práci. Snímaný obraz poskytuje oproti iným metódam veľmi komplexnú informáciu o okolí robota, čo ju robí najuniverzálnejšou.

Presnosť, s akou tento typ lokalizácie pracuje, je úmerná rozlíšeniu spracovávaného obrazu a môže sa pohybovať už od niekoľkých milimetrov. Spomedzi spomínaných metód je táto najnáročnejšia z pohľadu výpočtového výkonu. Práve tento fakt častokrát znemožní jej nasadenie v malých mobilných robotoch.

Čo sa týka rozlíšenia, v dnešnej dobe nebýva limitujúcim prvkom kamera, ale práve výpočtový výkon palubného počítaču robota, ktorý musí obraz dostatočne rýchlo spracovávať a vyhodnotiť pozíciu alebo jej zmenu. Výpočtový výkon počítačov sa neustále zvyšuje, rozmery zmenšujú a ich cena klesá. Vďaka tomu sa čím ďalej tým častejšie stretávame s mobilnými robotmi vybavenými kamerou, ktoré získavajú informáciu o polohe práve z analýzy obrazu.

# Kapitola 3

# Návrh algoritmu

Algoritmus je navrhnutý s dôrazom na nasledujúce požiadavky:

- **Presnosť lokalizácie.** Navrhnutý algoritmus lokalizácie by mal byť schopný určovať svoju pozíciu s presnosťou rádovo v jednotkách centimetrov. Výslednú presnosť očakávame v rozsahu do 5 cm vzdialenosti od skutočnej pozície robota.

- **Schopnosť riešiť znovunájdenie sa.** Algoritmus nerieši ani znovunájdenie robota ani detekciu straty.

- **Schopnosť udržať si správnu pozíciu.** Kritérium, na ktoré budeme klásť jednoznačne najväčší dôraz. Algoritmus lokalizácie staviame bez schopnosti znovunájdenia a detekcie straty robota, z čoho okamžite vyplýva silná závislosť úspešnosti na schopnosti nestratiť sa.

  Na rozdiel od iných metód lokalizácie má detekcia šachovnice dobré predpoklady pre schopnosť nestrácať sa. Vyplýva to z diskrétneho charakteru samotnej šachovnice. Presnejšie sa tomu budem venovať neskôr v oddiele 3.4 Detekcia posunu.

- **Závislosť na kvalite vonkajších podmienok.** Táto práca si kladie za cieľ vytvoriť algoritmus, ktorý sa bude v najväčšej možnej miere adaptovať vonkajším podmienkam ako zmeny viditeľnosti, osvetlenia, nevýrazná šachovnica, prekrytie časti výhľadu inými predmetmi.

- **Zložitosť implementácie.** Pri návrhu algoritmu bol v neposlednom rade kladený dôraz na to, aby nasadenie na skutočného robota vyžadovalo čo možno najmenej úsilia. Takmer všetky parametre takmer všetkých integrovaných algoritmov sa prostrediu prispôsobujú automaticky. Nevyhnutné je nastaviť algoritmu naozaj iba parametre, ktoré sám už z princípu zistiť nemôže. Potrebné parametre bližšie rozoberieme v oddiele 3.1.

Podmienky a predpoklady pre navrhnutý typ lokalizácie:

- **Kamera.** V ideálnom prípade digitálna, v opačnom prípade, je nutný ďalší hardvér pre zdigitalizovanie analógového obrazu. Kamera musí poskytovať dostatočné rozlíšenie pre požadovanú presnosť lokalizácie a dostatočne vysokú frekvenciu snímkou.

  Veľmi nepriaznivý jav pre lokalizáciu pomocou kamery je rozmazaný obraz. Ten môže vznikať pri rýchlom pohybe a zvlášť rýchly pohyb v kombinácií so zlým osvetlením môžu napríklad bežným digitálnym webkamerám spôsobiť nemalé problémy. Veľmi vhodná by teda bola kamera s uzávierkou, ktorá vznik rozmazaného obrazu minimalizuje.

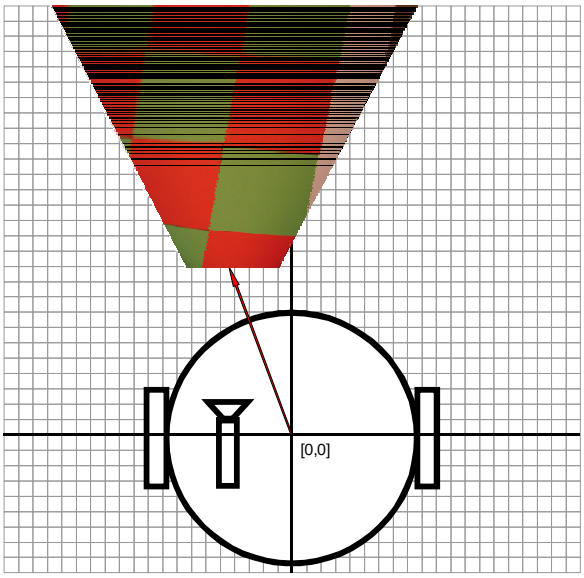
  Kamera musí byť umiestnená na robotovi, ktorý lokalizáciu využíva a musí byť nasmerovaná dopredu čo sa týka horizontálneho smeru. Vo vertikálnom smere je uhol kamery ľubovoľný (algoritmus je navrhnutý tak, aby sa tomuto uhlu prispôsobil). Konkrétna pozícia na robotovi je rovnako ľubovoľná. Slovo „ľubovoľný“ je potreba brať s rezervou. Aby mohla kamera vôbec niečo detekovať musí mať aspoň trochu vhodný výhľad na šachovnicu, čo ale na druhú stranu nie je príliš obmedzujúca podmienka.

- **Výpočtový výkon.** Nielen kamera je v tomto type lokalizácie kľúčová. Robot musí disponovať dostatočne veľkým výpočtovým výkonom pre spracovanie daného počtu snímkov obrazu v danom rozlíšení. Veľkosť výkonu samozrejme úzko súvisí s kvalitnou implementáciou algoritmov a hlavne s použitím vhodných a rýchlych metód. Treba teda pamätať na to, že robot musí spracovať obraz v reálnom čase.

- **Šachovnica.** Počiatočná myšlienka lokalizácie sa opierala o šachovnicu namaľovanú na podložke resp. podlahe, po ktorej sa bude robot pohybovať. Šachovnica preto, lebo sú na nej dobre viditeľné rozhrania dvoch farieb a pravidelnosť vzoru.

  Počas návrhov a vývoja algoritmov však dospejeme k riešeniu, ktoré dvojfarebnosť šachovnice nevyužíva. Na správnu lokalizáciu postačuje podlaha, na ktorej je výrazný štvorcový vzor. To samozrejme pri zachovaní ostatných požiadaviek na presnosť a nestrácanie sa, čím sa stal algoritmus univerzálnejší a podstatne ľahšie implementovateľný. Vhodná je napríklad už aj podlaha zo štvorcových dlaždíc, medzi ktorými je viditeľná špára. V ďalšom texte budeme hovoriť o šachovnicovej podložke, ale s platnosťou predošlého tvrdenia.

V predošlom odstavci sme často používali slová ako „dostatočne“ alebo „vhodné“. Chceli sme totiž poukázať hlavne na obecné podmienky pre navrhnutý lokalizačný algoritmus. Dostatočnosť a vhodnosť parametrov je silne závislá na výsledkoch, ktoré od lokalizácie očakávame. O konkrétnych hodnotách, s ktorými algoritmus otestujeme, budeme pojednávať v kapitole 5.




Obr. 3.1: Zobrazuje definíciu pozície snímaného obrazu vzhľadom na pozíciu robota. Pohľad zhora.

## 3.1 Vstupy a výstupy algoritmu

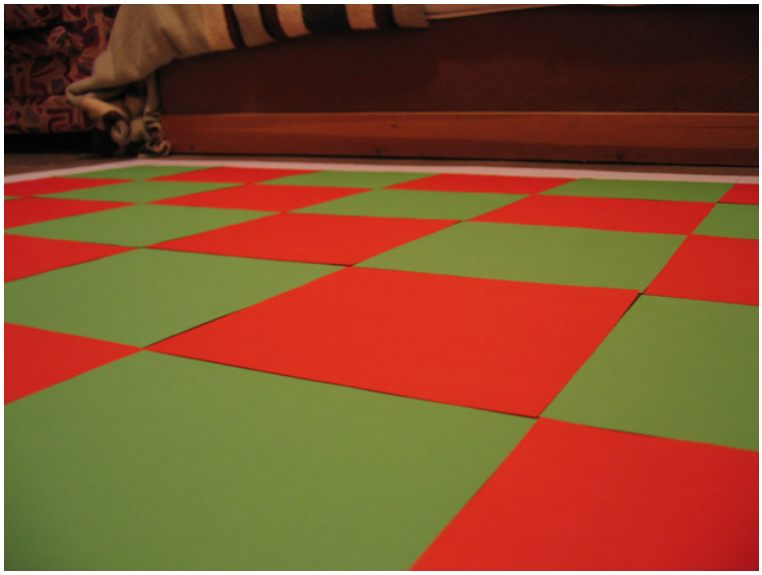
Algoritmus pre svoje fungovanie potrebuje poznať pozíciu kamery, resp. pozíciu snímaného obrazu vzhľadom na pozíciu robota. Tá sa udáva v centimetroch v súradnicovej sústave so stredom v aktuálnej pozícii robota a natočeného v smere osi $y$. Presnejšie zobrazuje situáciu obrázok 3.1. (Samozrejme je možné zadať skutočnú vzdialenosť nielen v centimetroch, ale v ľubovoľnej dĺžkovej jednotke. Tú potom treba dodržiavať na všetkých miestach. My budeme v tejto práci používať centimetre.)

Ďalej je potrebné udať skutočnú dĺžku strany štvorca na šachovnici v centimetroch. (Robot musí byť schopný rozhodnúť, o koľko centimetrov sa posunul, ak prešiel jeden štvorec.)

Ďalším vstupom algoritmu je posledná známa pozícia robota (v $i$-tom kroku trojica $(x_{i-1}, y_{i-1}, \alpha_{i-1})$. Na začiatku to bude pravdepodobne pozícia zvonku definovaná, a v nasledujúcich krokoch to môže byť pozícia zistená týmto algoritmom vždy z predchádzajúceho kroku. Nič ale nebráni tomu, aby bol algoritmus použitý ako súčasť iného lokalizačného systému, ktorý môže napríklad rôzne kombinovať výsledky získané z viacerých zdrojov. Môže ísť dokonca aj o pravdepodobnostné lokalizácie, ktoré by tento algoritmus využili ako jeden zdroj spresnenia. V takom prípade nadradený lokalizačný algoritmus poskytne „svoju“ pozíciu robota, o ktorej si myslí, že je správna, ako vstup do tohto algoritmu.

Konečne posledným vstupom je digitálny obraz zosnímaný kamerou, ktorý poskytuje dobrý výhľad na šachovnicovú podložku (obrázok 3.2).

Jediným hlavným výstupom z algoritmu je nová poloha (v $i$-tom kroku trojica $(x_i, y_i, \alpha_i)$). Ďalšími informatívnymi výstupmi môžu byť priemerný po-



Obr. 3.2: Typický pohľad z kamery robota

čet spracovaných snímkov za sekundu alebo doby behu jednotlivých súčastí lokalizačného algoritmu.

## 3.2 Mapa algoritmu

Algoritmus je rozdelený na dve hlavné fázy. Fáza jedna, spočíva v detekcii uhla natočenia robota. Jej súčasťou je algoritmus na detekciu hrán navrhnutý Cannym [3]. Hrany detekované v obraze ďalej využívame ako vstup pre Houghovu transformáciu [4], úlohou ktorej je detekovať čiary v obraze. Tie potom pomocou perspektívnej transformácie zmeníme tak, aby zodpovedali čiaram, ktoré by sa našli, ak by kamera snímala daný výhľad priamo zhora. Získané stransformované čiary už nesú časť informácie o uhle natočenia $\alpha_i$. Uhol zo získaných čiar analyzuje súčasť algoritmu, ktorá je nazvaná „angle detector“. Na jeho analýzu potrebuje poznať ešte natočenie robota v predošlom kroku $\alpha_{i-1}$.

Okrem zistenia nového uhla natočenia je úlohou angle detectoru kalibrovať perspektívnu transformáciu. V priebehu fázy jedna sa ďalej kalibruje Houghov algoritmus tak, aby detekoval vždy vhodný počet najjasnejšie viditeľných čiar.

Tejto časti sa budeme podrobne venovať oddiele 3.3.

Vstupom do fázy dva je nové natočenie robota detekované v prvej fáze a posledná známa pozícia z predošlého kroku – teda trojica $(x_{i-1}, y_{i-1}, \alpha_i)$.

Tejto časti algoritmu sa venuje analyzátor, ktorý je nazvaný „chessboard finder“. Ten si v prvom kroku vytvorí vlastnú štvorcovú sieť podľa vstupnej informácie (stará poloha a nový uhol) a vytvorí zoznam mrežových bodov $A$, ktoré by sa mohli vyskytovať na snímanom obraze. Ďalej analyzuje snímku z kamery. Presnejšie analyzuje hrany detekované Cannyho algoritmom [3]. Po-

mocou algoritmu pre hľadanie kontúr vyhľadá všetky kontúry v obraze a vyberie z nich tie, ktoré majú vlastnosti podobné štvorcom šachovnice. Z nich získa vrcholy štvorcov šachovnice a vytvorí zoznam mrežových bodov $B$.

Ďalej chessboard finder porovná získané zoznamy $A$ a $B$. Vyhľadá v nich také dvojice bodov, pre ktoré existuje predpoklad, že ide o jeden mrežový bod. Z týchto dvojíc je možné určiť vektory posunutia, z ktorých vhodnou kombináciou získame výsledné posunutie robota. Ten stačí pričítať k polohe z minulého kroku a vytvoriť novú pozíciu $(x_i, y_i, \alpha_i)$.

Okrem tejto úlohy má chessboard finder na starosti kalibráciu pomeru počet pixlov na centimeter dĺžky.

Fáze dva sa venujeme podrobne v oddiele 3.4

Implementácia algoritmu v nezanedbateľnej miere využíva prostriedky knižnice „Open source Computer Vision Library“ (OpenCV), ktorú zastrešuje Intel Corporation [5].

Nakoniec ešte spomenieme, že z dôvodu, že ide o algoritmus, ktorý musí bežať v reálnom čase, sme boli nútený niektoré algoritmy upraviť alebo niektoré nápady úplne vypustiť, podľa toho nakoľko dobrý mali pomyselný pomer užitočnosti ku dobe výpočtu. Upravený bol napríklad Cannyho algoritmus a úplne sme vypustili pôvodný nápad dvoch súčastí „color detector“ a „surround cutter“, ktorým sa budeme podrobnejšie venovať v oddiele 3.5.

## 3.3 Detekcia uhla natočenia

Detekcia uhla natočenia sa skladá z troch častí. Prvá z nich sa využíva v priebehu celého algoritmu (nie len detekcia uhla natočenia). Ide o algoritmus pre detekciu hrán. Spomedzi viacerých kandidátov sme vybrali pre svoje vynikajúce vlastnosti Cannyho detektor.

Po detekovaní hrán nasleduje hľadanie čiar v obraze. O to sa postará známy algoritmus Houghová transformácia.

No a nakoniec hľadané čiary analyzuje angle detector, ktorý už zistí konkrétny uhol natočenia robota. Okrem toho má za úlohu podľa detekovaných čiar kalibrovať výšku horizontu pre správnu perspektívnu transformáciu.

Jednotlivými časťami sa budeme podrobne zaoberať v nasledujúcich oddieloch.

### 3.3.1 Detekcia hrán

Detekcia hrán sa stala jednou zo základných pomôcok vo veľkom počte algoritmov v počítačovom videní. Jej pomocou môžeme výrazne znížiť množstvo dát a ponechať iba naozaj dôležitú informáciu o štruktúre obrazu, jeho hrán. Ostatné dáta, ktoré nenesú informáciu o tvare alebo obrysoch útvarov v obraze, sú v tomto prípade pre nás nezaujímavé a budú odfiltrované.

Pri hľadaní hrán sa vychádza z 2 možných stratégií, 2 „definícií“ hrán. Prvá, metóda, ktorá využíva gradient, definuje hranu v obraze na mieste, ktoré ma

oproti svojmu okoliu maximálnu alebo minimálnu intenzitu. Ak si predstavíme intenzitu v jednorozmernom obraze ako funkciu polohy, potom môžeme hľadať nulové body v jej prvej derivácií. V dvojrozmernom prípade potom môžme vytvoriť gradient z parciálnych derivácií v smere osí $x$ a $y$.

V počítačovej grafike sa kvôli potrebnej rýchlosti ako aproximácia používajú tzv. konvolučné masky, ktoré sú konštante veľké a z pravidla rádovo menšie ako analyzovaný obraz. Tie sa potom prikladajú na každý bod obrazu a určuje sa z nich veľkosť gradientu. Jeden predstaviteľ takýchto detektorov je Sobelov detektor hrán [6]. Ten využíva pre výpočet gradientov $G_x$, $G_y$ dve matice.

$$\text{matica pre gradient v smere osi } x \quad \begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix}$$

$$\text{matica pre gradient v smere osi } y \quad \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{pmatrix}$$

Veľkosť je potom určená ako $G = \sqrt{G_x^2 + G_y^2}$, čo sa často kvôli rýchlosti výpočtu aproximuje ako $G = |G_x| + |G_y|$. Smer gradientu a teda smer hrany $\Theta$ spočítame ako

$$\Theta = \begin{cases} \arctan \frac{G_y}{G_x} & G_x \neq 0 \\ 0 & G_y = 0 \wedge G_x = 0 \\ \frac{\pi}{2} & \text{inak} \end{cases} \tag{3.1}$$

V tomto prípade smer ešte nevyužijeme, ten sa nám ale zíde ďalej pri Cannyho hranovom detektore.

Existuje potom niekoľko ďalších variant, ktoré sa líšia iba použitými maticami. Za všetky napríklad Prewittov gradientový detektor hrán [7] s maticami:

$$\text{pre gradient v smere osi } x \quad \begin{pmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{pmatrix}$$

$$\text{pre gradient v smere osi } y \quad \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{pmatrix}$$

ktorý produkuje veľmi podobný výsledok ako Sobelov detektor hrán. Porovnanie je možné vidieť na obrázku 3.3.

Druhá Laplacian metóda pri detekcii hrán spočíva v hľadaní nulových bodov v druhých deriváciách. Rovnako ako v predošlom prípade pre potreby počítačovej grafiky aproximujeme konvolučnou maskou. Laplace používa jednu 5x5 masku (3.2) pre druhú deriváciu v oboch smeroch.

Obr. 3.3: Porovnanie hranových detektorov s rôznymi maskami. Zľava: originálny snímok, sobelov detektor hrán, prewittov detektor hrán [8].

$$\begin{pmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & 24 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{pmatrix} \tag{3.2}$$

V tomto algoritme je použitá Cannyho detekcia [3], pre niektoré jej veľmi dobré vlastnosti:

- **Nízka chybovosť.** Algoritmu sa veľmi dobre darí detekovať hrany tam, kde naozaj sú, nedetekovať, kde nie sú.
- **Hrany sú veľmi dobre lokalizované.** Vzdialenosť medzi detekovanou hranou a skutočnou hranou v obraze je minimálna.
- **Jedna odozva pre jednu hranu.** V tomto prípade ide pre nás o najdôležitejšiu vlastnosť. Je zabezpečená potlačením nemaximálnych bodov pri skutočnej hrane.
- **Odolnosť voči šumu.** Tú zabezpečuje preprocesing obrazu rozmazaním Gaussovým filtrom. My však túto vlastnosť nevyužijeme a filter na obraz nespustíme. Výsledok bez filtru je porovnateľne dobrý a zároveň nezanedbateľne rýchlejší (čo je pri algoritme, ktorý musí bežať v reálnom čase, zvlášť vhodné).

Algoritmus sa delí na niekoľko krokov.

V prvom kroku sa aplikuje Gaussov filter. Ten je možné diskrétne aproximovať a spočítať ho pomocou masky (3.3). Podľa Greena [9] je vhodná matica $5 \times 5$ s parametrom $\sigma = 1.4$. Ako sme spomenuli vyššie, tento krok sme v implementácií vynechali.

$$\frac{1}{115}\begin{pmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 15 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{pmatrix} \tag{3.3}$$

Ako nasledujúci krok je použitý Sobelov detektor hrán, presne tak, ako je opísaný vyššie. Tentokrát je už ale využitý aj smer gradientu $\Theta$, ktorý udáva smer hrany v intervale $(0, \pi)$. Aby bol algoritmus dostatočne rýchly aproximujeme aj tento smer a zaradíme ho do jednej zo skupín podľa (3.4).

$$\begin{array}{lll} 0^\circ & \text{pre} & 0^\circ \leq \Theta < 22.5^\circ \vee 157.5^\circ \leq \Theta < 180^\circ \\ 45^\circ & \text{pre} & 22.5^\circ \leq \Theta < 67.5^\circ \\ 90^\circ & \text{pre} & 67.5^\circ \leq \Theta < 112.5^\circ \\ 135^\circ & \text{pre} & 112.5^\circ \leq \Theta < 157.5^\circ \end{array} \tag{3.4}$$

Vo chvíli, keď je rozhodnuté, ktorým smerom vedie hrana, je možné túto hranu sledovať a všetky okolité body (veľkosti gradientu $G$), ktoré nie sú lokálnymi maximami, potlačiť (nastaviť ich veľkosť na 0). To je presne tá chvíľa, keď sa z jednej hrany stáva jedna ostrá linka.

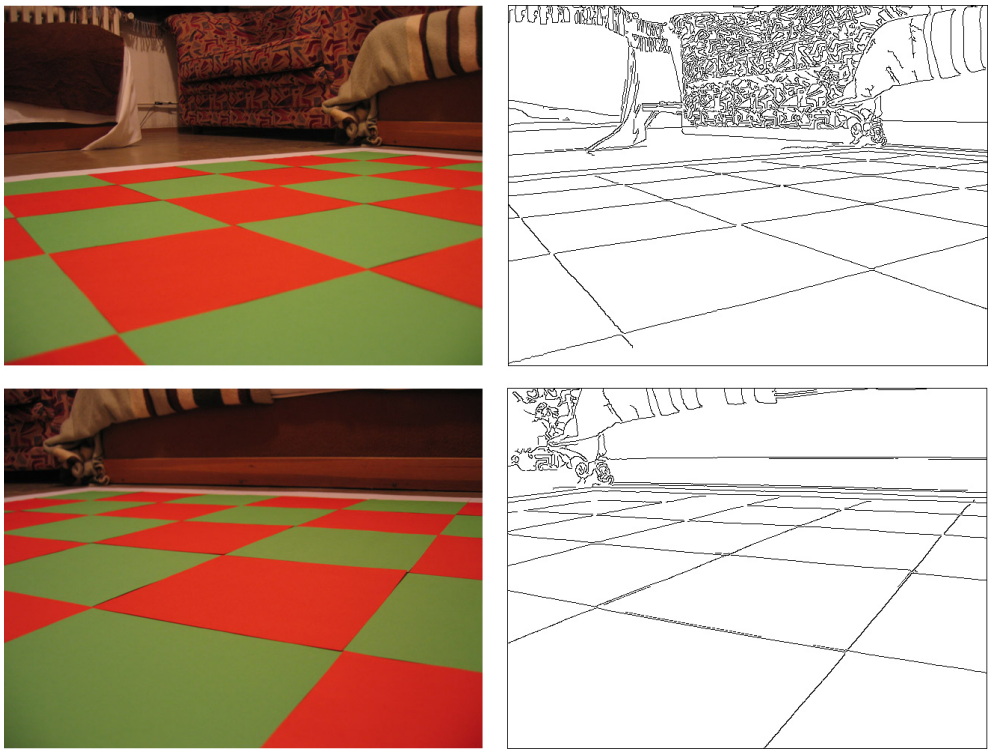
Nakoniec robí Cannyho algoritmus poslednú úpravu nájdených hrán. Pre rozhodovanie, či je ten ktorý pixel hranou, potrebuje dve prahové hranice $T_1$ a $T_2$, pre ktoré $T_1 \geq T_2$. Najskôr sa použije $T_1$ a za hrany sa označia všetky body, pre ktoré $G \geq T_1$. Ak by sme v tejto chvíli prestali (čo spravíme ak $T_1 = T_2$), tak získame hrany, ktoré ale môžu byť značne prerušované. Teraz však prichádza na rad druhá, menšia hranica $T_2$. Tá označí za hrany všetky body, pre ktoré platí $G \geq T_2$ a existuje bod v 8-okolí, ktorý je hranou.

Asymptotická zložitosť spomínaných hranových detektorov je $\mathcal{O}(nm)$, kde $m \times n$ sú rozmery obrázku.

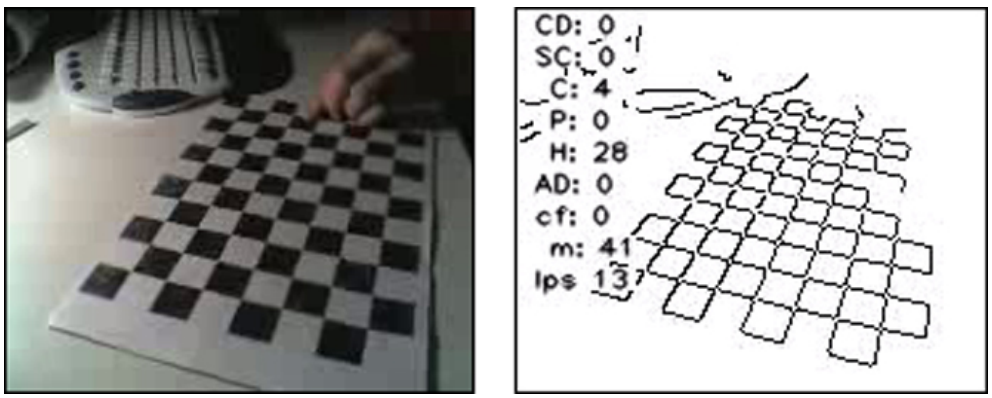
Získané výsledky po Cannyho detekcií hrán už môžeme predviesť na vlastných snímkach z kamery: obrázky 3.4 a 3.5, a video [i] na priloženom CD. Na videu si je možné všimnúť, že Cannyho algoritmus funguje bez problémov aj pri pomerne slabom osvetlení. To, že je osvetlenie slabé, sa samozrejme prejavuje. Konkrétne pri prudšom pohybe má použitá web-kamera problém zaostriť a môžu sa strácať okraje šachovnice. Tomu je jednoduché zamedziť správnym posunutím zvolených prahových hodnôt. Nie je to ale potrebné. Sú to síce jediné dva parametre, ktoré sa v lokalizácií nekalibrujú a sú nastavené napevno, ale Cannyho algoritmus dáva dostatočne dobré výsledky pri ľubovoľnej situácii. Okrem toho sa výsledku Cannyho algoritmu prispôsobujú nasledujúce algoritmy, ktoré sú kalibrované automaticky.

### 3.3.2 Detekcia čiar

Pre detekciu hrán použijeme Houghovu transformáciu vo svojej klasickej verzii podľa [4]. Tá ako svoj vstup používa obraz vo formáte bitovej mapy. Kvôli efektivite je tu veľmi vhodné použiť obraz detekovaných hrán napríklad práve Cannyho detektorom.



Obr. 3.4: Detekcia hrán algoritmom Canny.




Obr. 3.5: Algoritmus Canny. Snímka z videa [i].

Houghov algoritmus pre detekciu čiar reprezentuje priamku dvoma parametrami $(r, \theta)$, kde $r$ je vzdialenosť priamky od bodu $(0, 0)$ a $\theta$ je, uhol ktorý zviera normála priamky s osou $x$. Oproti klasickej reprezentácií $y = ax + b$ táto nemá problém s vertikálnymi alebo skoro vertikálnymi čiarami. Úloha Houghovho algoritmu je teda nájsť v dvojrozmernom tzv. Houghovom priestore parametrov $(r, \theta)$ tie body, ktoré reprezentujú čiaru, ktorá prechádza čo najväčším počtom bodov s hodnotou 1 v bitovej mape.

Prvý krok algoritmu je diskretizovanie Houghovho priestoru, pretože nekonečne veľký sa pochopiteľné preskúmať nedá. Vytvoríme teda pre neho akumulátor, čo bude dvojrozmerné pole, kde jeden rozmer je pre parameter $r$ a druhý pre parameter $\theta$. Veľkosť týchto rozmerov je daná zvoleným rozlíšením.

V ďalšom kroku prechádza obraz bod po bode a pre každý bod, ktorý je nastavený na 1 pričíta do akumulátora +1 pre každú čiaru, ktorá ním prechádza.

V poslednom kroku je teda potrebné preskúmať akumulátor a vybrať z neho maximá, ktoré budú predstavovať čiary, ktoré prechádzali najväčším počtom bodov. To je možné s ohľadom na rýchlosť najjednoduchšie spraviť zvoleným prahom akumulovanej hodnoty. Bod akumulátora $(r, \theta)$, ktorý sa svojou hodnotou dostal nad danú hranicu, je prehlásený za detekovanú čiaru. Nakoniec ešte detekované čiary zotriedime podľa ich intenzity (hodnoty akumulátoru) a niekoľko prvých, podľa požiadavky vrátime ako výstup algoritmu.

Vhodný počet čiar je empiricky stanovený na sedem. Je to dostatočný počet na to, aby niesli silnú informáciu o uhle natočenia robota s dostatočne malou chybou, a pritom sa neobjavovali detekované hrany mimo šachovnice. Hranica pre prijímanie čiar z akumulátoru sa dynamicky upravuje podľa predchádzajúceho počtu nájdených hrán. Tým zamedzíme tomu, aby boli pri prechádzaní akumulátora zakaždým vyhľadaných tisíce čiar, z ktorých potom až zdĺhavým triedením dostaneme tie najviditeľnejšie.

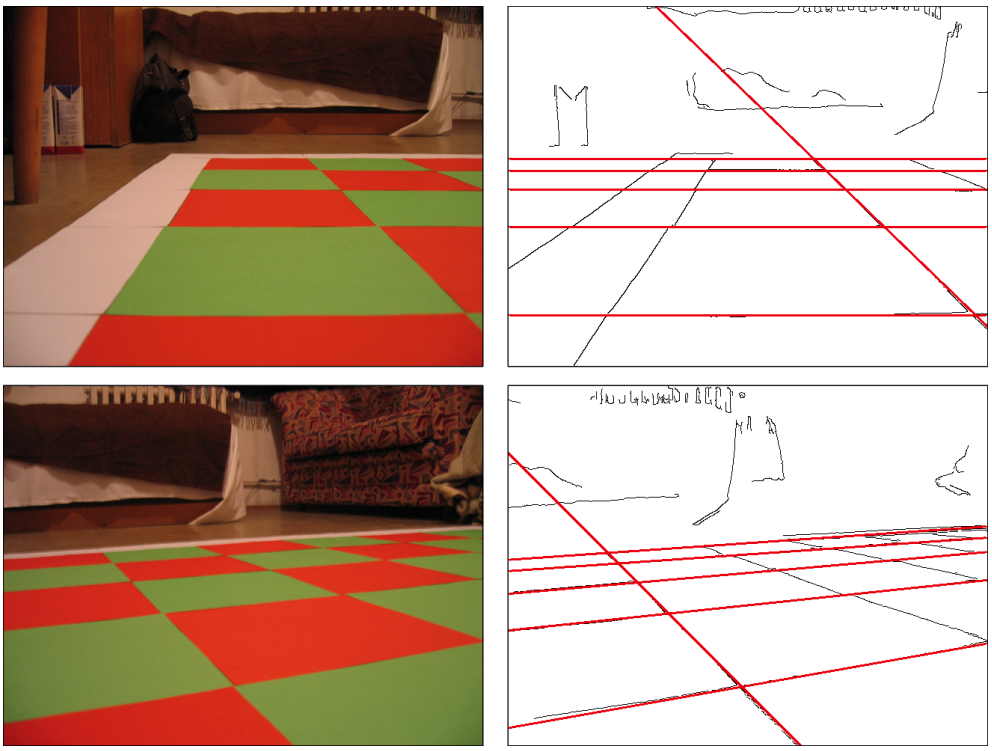
```
double adaptacia = 0.02;  //relatívna úprava hranice
int lineNumber = 7; //správny počet nájdených čiar

//adaptácia houghovho algoritmu na viditeľnosť
if (lineStorage->cols > lineNumber)
    threshold = int(threshold * (1 + adaptacia));
else if (lineStorage->cols < lineNumber)
    threshold = max(int(threshold * (1 - adaptacia)),1);
```

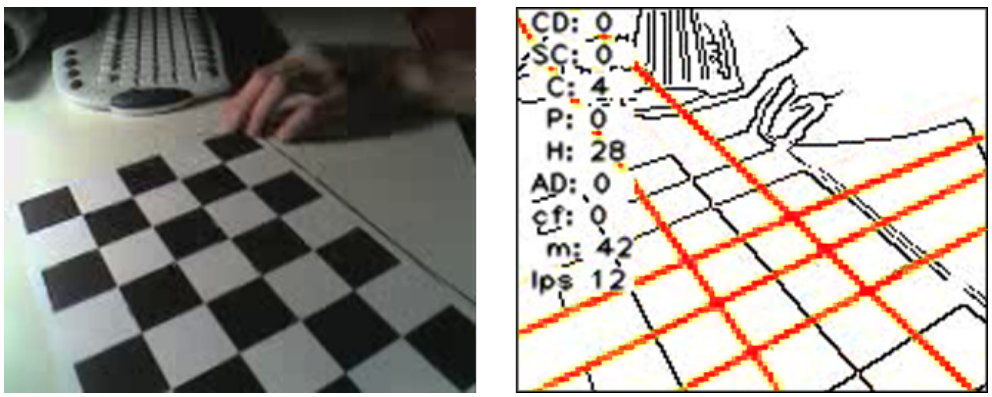
Výsledky detekcie čiar zobrazujú obrázky 3.6 a 3.7 a video [ii].

### 3.3.3 Analýza získaných čiar

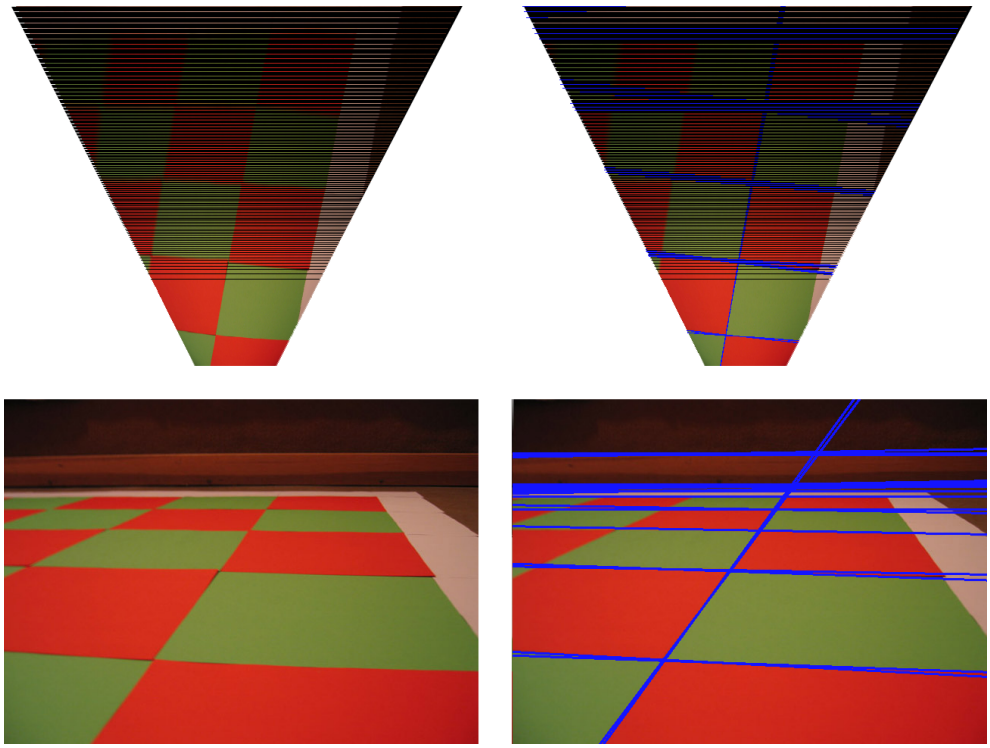
Ako sme už spomenuli v mape algoritmu (oddiel 3.2), po detekcií hrán a detekcií čiar v obraze nasleduje analýza uhla natočenia robota. O túto analýzu sa v tejto implementácií stará angle detector. Ešte predtým, ako bude možné z detekovaných čiar určiť uhol natočenia, je potrebné perspektívny obraz z kamery transformovať na obraz, ktorý by sme získali, ak by bola kamera natočená kolmo dolu. V skutočnosti nie je nutné transformovať celý obraz. Angle



Obr. 3.6: Detekcia čiar Houghovou transformáciou.




Obr. 3.7: Houghova transformácia. Snímka z videa [ii].



Obr. 3.8: Použitie perspektívy na obraz zosnímaný kamerou.

detector potrebuje vedieť iba to, ako budú vyzerať detekované čiary po transformácií. Implementovaná je však aj transformácia celého obrazu. Užitočné je to ako pre vlastnú kontrolu pri programovaní, tak aj ako názorná ukážka pri prezentácií algoritmu, obrázok 3.8.

Angle detector ako svoj vstup potrebuje poznať zoznam detekovaných čiar, uhol natočenia $\alpha_{i-1}$ z minulej pozície a horizont snímaného obrazu (priamka, v ktorej sa stretávajú rovnobežky z vodorovnej podlahy, šachovnice). Práve angle detector je navrhnutý tak, aby vedel horizont kalibrovať, takže na začiatku stačí nejaký implicitne zvolený. Horizont je reprezentovaný ako reálne číslo relatívne k výške obrazu. Ak teda je napríklad horizont rovný 0.5, znamená to, že sa nachádza v obraze presne v strede, čo nastane vtedy, ak je kamera vo vertikálnom smere natočená rovnobežne z podložkou. To ale nie je úplne ideálna poloha kamery, pretože užitočná informácia sa v takom prípade nachádza iba v spodnej časti obrazu. A čím bližšie horizontu sa nachádzame, tým horšie rozlíšenie šachovnice máme. Ak bude horizont rovný 1, znamená to, že sa nachádza na hornom okraji snímaného obrazu.

Za uhol natočenia budeme považovať uhol, pod ktorým vidíme jedny z rovnobežiek šachovnice po perspektívnej transformácií. Nový uhol teda zistíme ako priemer uhlov detekovaných čiar s tým, že čiary, ktoré budú od pôvodného uhla natočené viac ako o 45° budeme považovať za kolmé k smeru natočenia (druhý typ rovnobežiek zo šachovnice) a predtým, ako ich zahrnieme do priemeru, od nich odčítame 90°. Pre prípad, že by sa objavila medzi detekovanými čiarami nejaká úplné zlá, napríklad taká, ktorá bola výrazná v obraze

mimo šachovnice, jednu čiaru s najväčšou odchýlkou od pôvodného smeru do priemeru nezahrnieme.

Perspektívu transformáciu pre nejaký bod počíta nasledujúca metóda. Tá dostane pôvodný bod v súradnicovej sústave so stredom v snímanom obraze dole v strede.

```
CvPoint2D64f tPerspective::perspectiveTransform(CvPoint2D64f point) {
    //z = 0 na horizonte, z~= 1 v~spodnom riadku, zaporne nad horizontom
    double z~= (horizont * imgHeight - point.y) / (horizont * imgHeight);
    double posunY = imgHeight * (1 - z) / z;
    double posunX = cx * point.x / z;

    CvPoint2D64f transformovanyBod;
    const double c = 10000000; //konštanta, posúva body do nekonečna
    transformovanyBod.x = (point.x + posunX * c) * scale / c;
    transformovanyBod.y = (point.y + posunY * c) * scale / c;
    return transformovanyBod;
}
```

Nájdem odchýlky jednotlivých čiar od predchádzajúceho uhla natočenia robota:

```
vector<double> diffs;
for (unsigned int i = 0; i < angles.size(); i++) {
    double diff = angles[i] - (-oldPosition.angle);
    //ak je rozdiel noveho od stareho vacsi ako pi/4,
    //povazujem najdenu ciaru za kolmu k~smeru robota
    while (diff >  CV_PI/4) diff -= CV_PI/2;
    while (diff < -CV_PI/4) diff += CV_PI/2;
    diffs.push_back(diff);
}
```

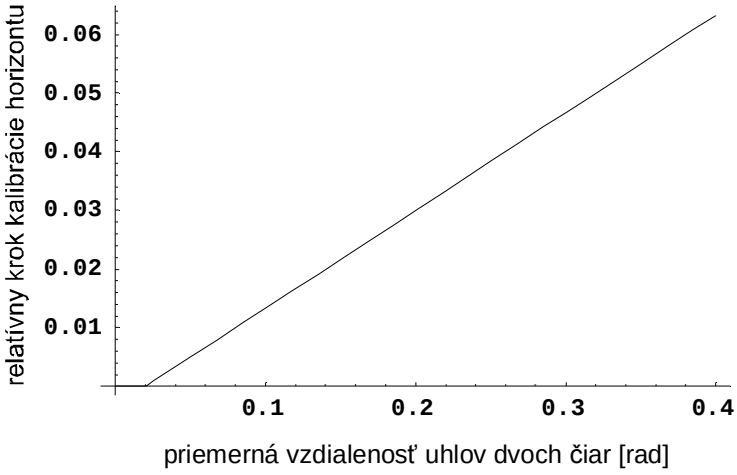
A nakoniec zo zoznamu odchýlok, po zahodení najhoršej odchýlky, spočítame priemer. Výnimočne v prípade, že sa v obraze našla len jedna čiara, tak ju použijeme a nezahodíme žiadnu.

```
sort(diffs.begin(), diffs.end(), &tAngleDetector::absComp);
unsigned int p = diffs.size() > 1 ? diffs.size() - 1 : diffs.size();
double avg = 0;
for (unsigned int i = 0; i < p; i++) avg += diffs[i] / p;
```

Výslednú odchýlku potom už len pripočítame k smeru natočenia robota. Tu treba dať pozor na opačný charakter otáčania robota oproti šachovnici. Ak sa totiž šachovnica otáča na obraze v kladnom smere, znamená to že sa robot otáča v smere zápornom.

```
newPosition.angle += (-avg);
```

Počas perspektívnej transformácie nájdených čiar vykonávame ešte kalibráciu horizontu nasledujúcim spôsobom. Ako referenčnú hodnotu horizontu použijeme poslednú nájdenú. Nasledovne všetky detekované čiary transformujem 3x. Raz pre hodnotu horizontu z predchádzajúceho kroku, druhýkrát




Obr. 3.9: Graf zobrazujúci zvolenú závislosť pre kalibráciu perspektívnej transformácie.

posunieme horizont v obraze smerom dole o zvolený krok a tretíkrát smerom hore o rovnaký krok.

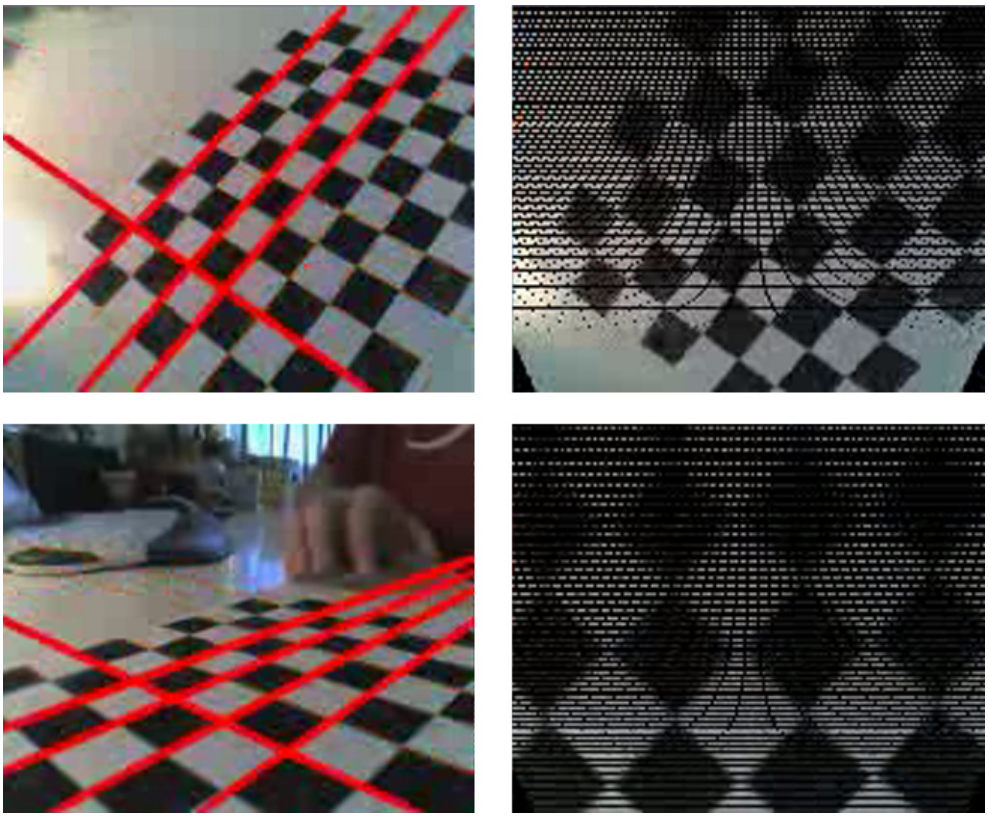
V ideálnom prípade očakávame ako výsledok dokonale rovnobežné čiary (tie kolmé najprv otočíme). To, nakoľko rovnobežné čiary sme dostali, spočítame priemernou vzdialenosťou uhla medzi každými dvoma čiarami. To vykonáme pre každé nastavenie horizontu a transformáciu, ktorá poskytne čiary čo najrovnobežnejšie (budú mať najmenšiu priemernú vzdialenosť svojich uhlov) prehlásime za tú správnu, použijeme ju a v nasledujúcom kroku nastavíme ako referenčnú výšku horizontu tú k nej prislúchajúcu.

Výsledok takto navrhnutej kalibrácie je silne závislý na zvolenom kroku. My ho spravíme úmerne veľký voči kĺzavému priemeru posledných niekoľko priemerných vzdialeností uhlov dvoch čiar. Zvolenú závislosť zobrazuje graf na obrázku 3.9. Takto som dosiahneme rôznu rýchlosť kalibrácie horizontu. Pri veľkej výchylke sa horizont bude posúvať rýchlejšie, ale vo chvíli, keď bude nastavený skoro dobre, bude sa dokalibrovávať už len o malé kúsky.

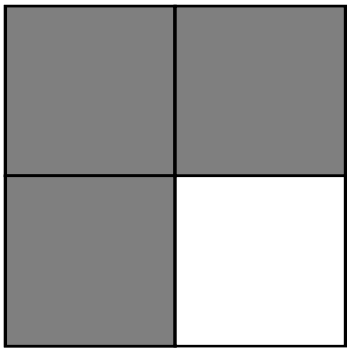
Na reálnom robotovi sa očakáva, že kamera bude pripevnená pevne a horizont sa nebude meniť. Nastaviť ho ručne však môže byť neľahké a to hlavne z príliš subjektívneho určovania rovnobežnosti. Nejaké dokonalejšie ručné nastavovanie by na druhú stranu bolo zbytočne komplikované a pri malej zmene natočenia kamery by ho bolo nutné vykonať opätovne. Túto automatickú kalibráciu sme spravili za cieľom výrazného zrýchlenia a zjednodušenia implementácie algoritmu na robota.

Detekciu uhla natočenia je možné vidieť na videu [iii]. Ktoré bolo natočené ešte bez perspektívnej transformácie. Výsledok transformácie celého obrazu zobrazujú obrázky 3.8 a 3.10. Úspešnosť kalibrácie horizontu je možné vidieť na videu [v].

Treba ešte dodať, že transformovanie celého obrazu slúži iba ako kontrola správnej funkcie a preto je spravené jednoducho (akýsi draft). Transformované boli postupne všetky body z pôvodného obrazu a tie, ktoré sa po transformácií



Obr. 3.10: Perspektívna transformácia. Snímky z videa [v]



Obr. 3.11: Maska $2 \times 2$ používaná na dilatáciu bitmapy po Cannyho hranovej detekcii.

premietli do nového obrazu boli po zaokrúhlení priamo vykreslené. Tam kde sa „netrafil“ po transformácií žiaden bod, ostal nevyplnený čierny podklad.

## 3.4 Detekcia posunutia

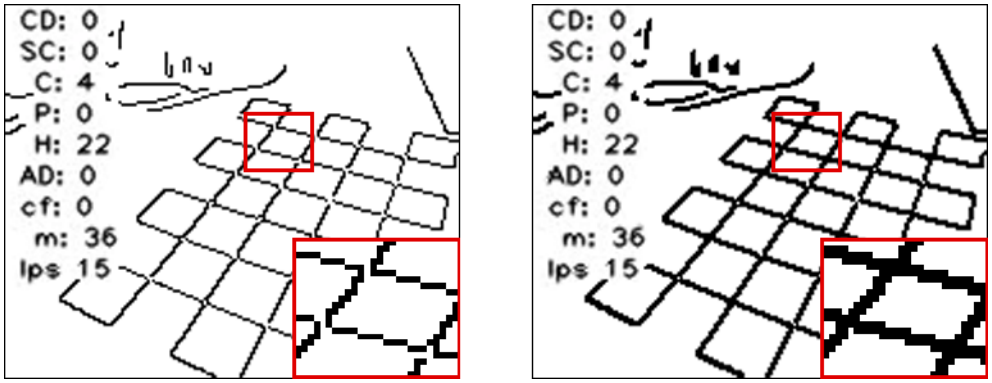
V tejto chvíli už máme za sebou hranovú detekciu a analýzu uhla. Môžeme sa teda pustiť do detekcie posunutia. Vstup pre túto časť algoritmu bude nový uhol $\alpha_i$, obraz po Cannyho detekcií hrán (bitová mapa) a pôvodná poloha robota $(x_{i-1}, y_{i-1})$.

Kľúčovým v tejto fáze algoritmu bude správne určiť mrežové body šachovnice. K tomu využijeme algoritmus implementovaný v knižnici OpenCv [5] pre hľadanie kontúr [10]. Ak totiž objavíme kontúry štvorcov na šachovnici a použijeme ich vrcholy, dostaneme práve hľadané mrežové body (video [iv]).

Hľadanie kontúr v bitovej mape spočíva v jej prehľadávaní. Vo chvíli, keď narazíme na nastavený bit, môžme začať obchádzať kontúru. Po zistení jej obvodu pokračujeme ďalej ako na začiatku, ale s tým, že si pamätáme objavenú kontúru, aby sme ju nemuseli obchádzať viackrát. Ako výstup dostaneme zoznam kontúr.

Tesne predtým je však potrebné spraviť ešte jednu úpravu. Napriek výborným vlastnostiam Cannyho detektora hrán sa môže výnimočne stať, že sú niektoré hrany prerušené. Tomu zabránim tzv. dilatáciou. Zvolili sme masku $M$ (obrázok 3.11), ktorou prejdeme celým obrazom. Konkrétny pixel $(x, y)$ v bitovej mape potom po priložení masky bude mat hodnotu $\max(x + \bar{x}, y + \bar{y})$, kde $(\bar{x}\bar{y}) \in M$. Výsledok je vidieť na obrázku 3.12.

Teraz už môžeme spustiť hľadanie kontúr. Tie sú ale v takejto „surovej“ podobe ešte nepoužiteľné, pretože kontúra štvorca na šachovnici bude s pravidla tvorená desiatkami až stovkami bodov. Preto nasleduje zjednodušenie nájdenej krivky. Pre tento účel som použil Douglasov-Peuckerov algoritmus [11]. Ten dramaticky zníži počet vrcholov potrebných na opísanie kontúry pri




Obr. 3.12: Použitie dilatačnej masky na obraz po Cannyho hranovej detekcii. Vľavo originál, v pravo aplikovaná dilatačná maska z obrázku 3.11.

zachovaní dobrej presnosti. Ako parameter potrebuje číslo, udávajúce počet bodov, o ktoré sa môže aproximovaná kontúra odchýliť od pôvodnej.

Práve tento parameter nie je jednoduché správne nájsť, aby sa nájdená kontúra aproximovala štvoruholníkom. Spravíme to tak, že budeme postupne maximálnu povolenú odchýlku aproximovanej kontúry zväčšovať, až do chvíle než dostaneme štvoruholník alebo odchýlka bude už dostatočne veľká, aby sa kontúra mala šancu aproximovať štvoruholníkom. Ak sa ani pri maximálnej povolenej odchýlke tak nestane, je to pravdepodobne preto, že nájdená krivka nebola štvorcom zo šachovnice. Ak sa podarí krivku aproximovať štvoruholníkom, je potrebné spraviť ďalšie kontroly. Nie každý nájdený štvoruholník je totiž detekovaný štvorec šachovnice. Kontrolujeme konvexnosť štvoruholníka, jeho veľkosť, ktorá musí presiahnuť danú minimálnu úroveň, aby sa odfiltrovali maličké kontúry, ktoré vznikli pravdepodobne šumom v obraze a nakoniec dĺžky jednotlivých strán, ktoré sa pri štvoruholníku nemôžu výrazne líšiť.

Krivky, ktoré prešli všetkými testami už považujeme za nájdené štvorce šachovnice a ich vrcholy si uložíme do zoznamu.

```
//Postupne vyhľadáme všetky kontúry v˜obraze.
while (contour = cvFindNextContour(image))
{
    if (area(contour) >= min_size )
    {
        for (int approxLvl = minLvl; approxLvl <= maxLvl; approxLvl++)
        {
            contour = cvApproxPoly(contour, approxLvl);
            if (contour->total == 4) break; //teraz máme štvoruholník
        }

        //Zaujímajú nás konvexné štvoruholníky, podobné štvorcu.
        if (contour->total == 4
            && cvCheckContourConvexity(contour)
            && likeSquare(contour))
        {
            //získali sme mrežové body
            for (int i = 0; i < 4; i++ )
```

```
                newCrossPoints.push_back(contour[i]);
            }
        }
    }
```

V ideálnom prípade budeme mať v zozname vrcholov štvorcov každý mrežový bod šachovnice na snímke 4-krát, pretože je vrcholom štyroch nájdených štvorcov. Tieto vrcholy treba zlúčiť do jedného mrežového bodu. Vrcholy, ktoré sú jedným mrežovým bodom, majú k sebe spravidla naozaj blízko. Vyplýva to z vlastnosti Cannyho hranovej detekcie a jeho potlačenia nie maximálnej hrany, takže jednu hranu detekuje iba raz. Na výsledok Cannyho detekcie sme však spustil ešte dilatáciu na odstránenie prípadných dier. Maximálnu vzdialenosť pre spájané vrcholy teda nastavíme na 5.

Teraz, keď máme zoznam mrežových bodov, existujú dve možnosti prístupu. Prvá z nich je, že ich môžeme porovnať s mrežovými bodmi rovnako získanými z predošlej snímky. V prvom rade treba nejako zistiť, ktoré mrežové body z dvoch po sebe idúcich snímok k sebe korešpondujú (je to ten istý mrežový bod, len posunutý). Posunutý by potom mal byť presne o toľko, o koľko sa posunul robot, len v opačnom smere (ak sa šachovnica posunie smerom „dozadu“ (na snímke smerom dole), robot sa musel posunúť dopredu). Keď máme mrežové body spárované, môžeme odčítať pozíciu starších bodov v páre od tých novších bodov a získať tak vektor posunutia robota.

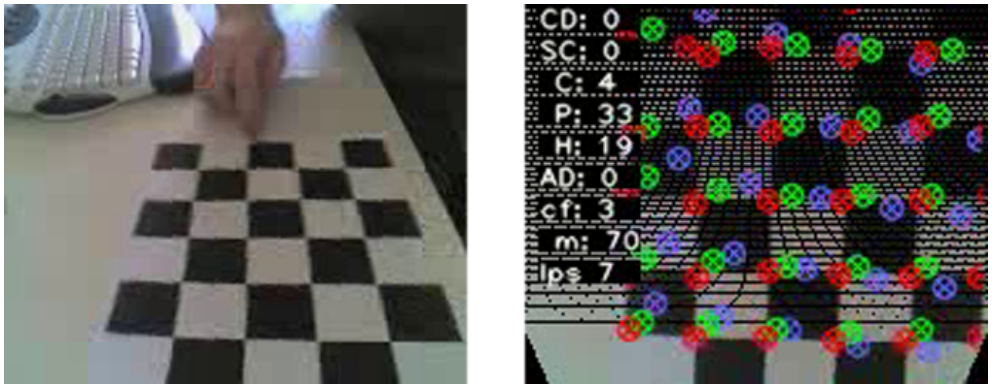
S tým je spojených niekoľko problémov. Po prvé, nedá sa očakávať že sa spárujú všetky body. To, koľko bodov a ktoré body sa v snímke nájdu sa môže veľmi meniť. Zvlášť pri zlých viditeľnostných podmienkach dávajú bežné kamery obraz silno zašumený a detekcia konkrétneho mrežového bodu sa niekedy podarí a niekedy nie. To však nie je neprekonateľný problém. Kladie to akurát požiadavky na vlastnosti šachovnice a umiestnenie kamery, a to také, že by mala kamera vidieť vždy dostatok mrežových bodov aby mala šancu spárovať vždy aspoň nejaký minimálny počet. O tom ako by mala vhodná šachovnica vyzerať, diskutujeme v oddiele 5.3.

Párovanie potom prebieha hľadaním najbližšieho bodu, pričom je určená maximálna vzdialenosť, pri ktorej sa ešte body považujú za pár. Tá je empiricky nájdená a vychádza z veľkosti štvorca.

```
//sú to 3/5 dĺžky strany štvorca v~bodoch
const double maxShiftDistance = 3.0/5 * SQUARE_WIDTH * scale;
```

Takéto zistenie translačného posunu robota však môže fungovať iba v prípade, že sa robot medzi dvoma po sebe idúcimi snímkami neotáčal. Pretože v takom prípade sa vzdialenejšie body posunú od tých minulých viac ako body bližšie osi otáčania. To ale nie je možné vylúčiť. Pretože ale v tejto chvíli už poznáme uhol o ktorý sa robot pootočil (spočítal nám ho angle detector), môžeme najprv všetky mrežové body zo starej snímky pootočiť o uhol $\alpha_i - \alpha_{i-1}$ okolo osi otáčania robota v polohe $(x_{i-1}, y_{i-1})$.

Takto otočené body už budú na všetkých miestach posunuté rovnako. Výsledný vektor posunutia spočítame ako ich vážený priemer. Vážený preto, aby



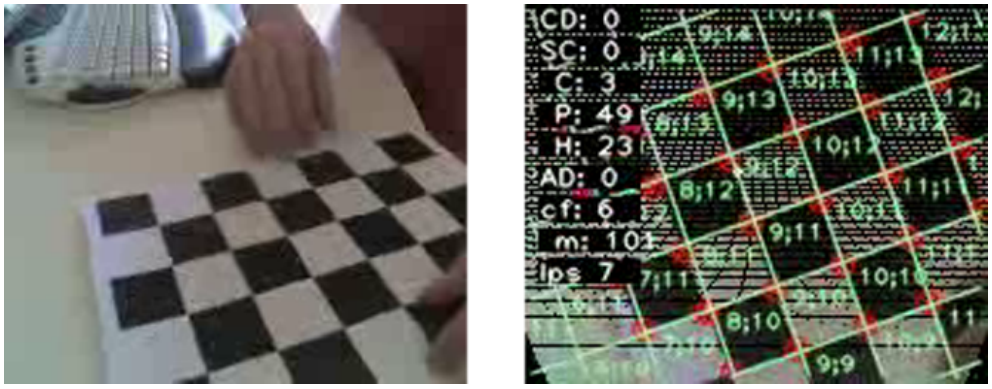

Obr. 3.13: Zobrazuje posun mrežových bodov pri otáčaní robota okolo vlastnej osi. Červené mrežové body sú detekované na súčasnej snímke, modrou farbou mrežové body z minulej snímky a zelenou farbou mrežové body z minulej snímky pootočené o uhol otáčania. Šachovnicou pohybujeme smerom doľava. Snímka z videa [vi].

som mohol prisúdiť väčšiu váhu bodom ktoré sú bližšie pri robotovi, pretože ich kamera vidí s lepším rozlíšením, ako tie ďalej.

Tento postup je vidieť na obrázku 3.13 a videu [vi]. Ako sme ale spomenuli vyššie, existujú dve možnosti ako určovať translačný posun z mrežových bodov. A táto prvá je tá zlá, ako bolo overené v prax. Jej veľký nedostatok je rovnaký, ako trápi veľa iných lokalizácií – akumulovanie chyby. Na videu [vi], kde robot stál na mieste a otáčané bolo šachovnicu v podstate na mieste, sa robot podľa lokalizácie posunul o takmer 8 cm od pôvodnej pozície z počiatočnej polohy $(0, 0)$ do polohy $(2.1, 7.3)$ pričom video je dlhé 9 sekúnd a obsahuje 136 snímok (súradnice sú v centimetroch). Červené body na videu sú detekované mrežové body na súčasnej snímke. Modrou farbou sú nakreslené mrežové body z minulej snímky a zelenou farbou mrežové body z minulej snímky pootočené o uhol otáčania.

Druhý prístup je skoro v celom postupe totožný s tým prvým. Líši sa ale v jednej podstatnej veci. V tom, že nebudeme porovnávať mrežové body šachovnice z novej snímky s tými zo starej, ale s mrežovými bodmi „internej“ šachovnice. Tú si môže robot spočítať zo svojej pozície $(x_{i-1}, y_{i-1}, \alpha_i)$ za predpokladu, že vie, kde na šachovnici je pozícia $(0, 0)$, a ktorým smerom je definované natočenie šachovnice.

Spočítame teda, ktoré mrežové body by sme mali vidieť na obraze a s nimi budeme zachádzať presne tak ako sme to popísali v predošlom postupe. Budeme ich pokladať za mrežové body z prvej snímky. Týmto spôsobom sme akékoľvek akumulovanie chyby znemožnili. Výsledok vidieť na obrázku 3.14 a videu [vii].




Obr. 3.14: Sledovanie šachovnice. Pohľad s perspektívnou transformáciou. Snímka z videa [vii].

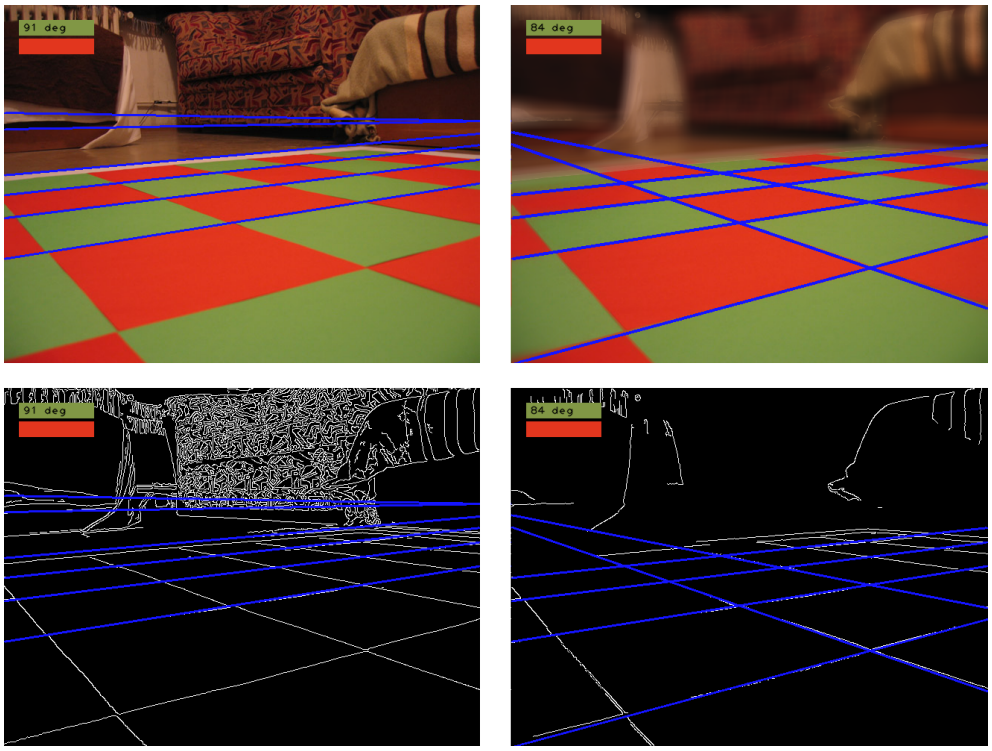
## 3.5 Vypustené súčasti

Ako sme už spomenuli v oddiele Mapa algoritmu 3.2, z dôvodu malej efektívnosti sme boli nútený vypustiť dve plánované súčasti „color detector“ a „surround cutter“. Tie mali bežať na začiatku analýzy ako predspracovanie obrazu pred detekciou hrán.

Cieľom týchto dvoch algoritmov bolo odstránenie nežiadúcich hrán v obraze, ktoré po hranovej detekcii spracovávajú ostatné algoritmy. Ich účinnosť ale nebola dostatočne veľká na to, aby vyvážila časovú stratu spôsobenú ich výpočtom. Preto sme sa nakoniec rozhodli nepoužiť ich.

Color detector mal za úlohu detekovať dve najčastejšie vyskytujúce sa farby v obraze. Za predpokladu, že aspoň na začiatku behu algoritmu bude mať robot dostatočne dobrý výhľad na šachovnicu bolo možné očakávať, že sa nájdu obe farby šachovnice.

V nasledovnom, kroku túto informáciu prevzal surround cutter a rozmazal oblasti v obraze, ktoré neboli podobné detekovaným farbám. Silu rozmazania sme skúšali nastaviť úmerne vzdialenosti farieb alebo diskrétne. V každom prípade bolo rozmazávanie príliš pomalé. Jednoduché odstránenie bodov s nevhodnou farbou fungovalo síce rýchlo, ale skôr kontraproduktívne. Po odstránených bodoch ostávali niekedy dokonca jasnejšie hrany ako tie na šachovnici.




Obr. 3.15: Porovnanie hranovej detekcie a detekcie hrán bez a s použitím algoritmu surround cutter. Dve najčastejšie vyskytujúce sa farby (dva obdĺžniky v ľavom hornom rohu) detekoval color detector.

# Kapitola 4

# Implementácia algoritmu do reálneho robota

## 4.1 Testovacie prostredie

Pre testovacie účely bol použitý mobilný robot s trojkolesovým podvozkom, kde dve kolesá sú nezávisle poháňané pomocou dvoch motorov. Motory sú ovládané riadiacimi čipmi ATmega8, ktoré s palubným počítačom komunikujú prostredníctvom sériovej linky. Palubný počítač pozostáva zo základnej dosky formátu ITX a procesoru VIA Epia s taktovacou frekvenciou 1.5 GHz. Takáto konštrukcia umožňuje priamy pohyb dopredu, dozadu, aj pohyb s ľubovoľne prudkým otáčaním poprípade otáčanie na mieste okolo vlastnej osi. Kameru (Logitech Quickcam Fusion) má robot pripevnenú vo výške 33 cm.

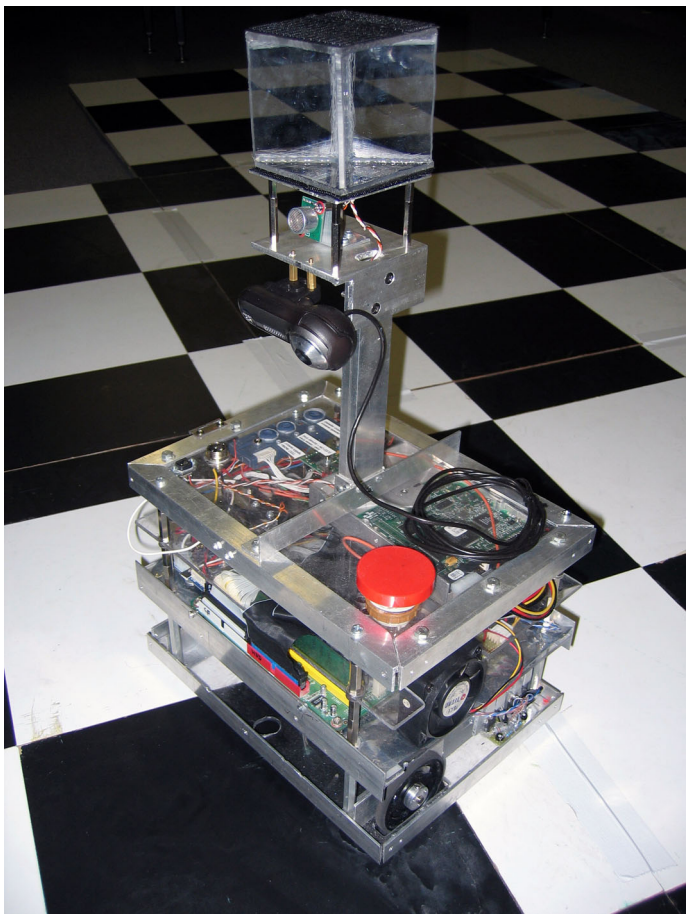
Zo softvérovej stránky je robot vybavený operačným systémom Gentoo Linux. Riadiaci program vrátane ovládačov pre hardware je napísaný v C++. Pre pomocné funkcie na spracovanie obrazu je nainštalovaná knižnica OpenCV [5].

Robota spolu s riadiacim programom som vytvoril v tíme s ďalšími tromi študentmi na súťaži Eurobot (nie je súčasťou tejto bakalárskej práce). Pre lokalizáciu na šachovnici som naprogramoval modul grafického analyzátora, ktorý popíšeme v nasledujúcej kapitole.

## 4.2 Programátorská dokumentácia

Implementáciu lokalizácie pomocou analýzy obrazu rozdelíme na dve súčasti – moduly. Prvý modul slúži pre spracovanie obrazu všeobecne. Pozostáva z dvoch tried. Tou prvou je `CamGrabber`. Stará o získavanie obrázkov z kamery a prostredníctvom metódy `getImage(IplImage **image)` poskytne poslednú získanú snímku.

V prípade, že analýza obrazu bude trvať dlhšie ako získavanie snímku z kamery začne `CamGrabber` snímky zahadzovať (buffer má dĺžku 1, príchod novej snímky z kamery tú starú prepíše). Zahodenie neaktuálnej snímky je nutné pre to, aby grafický analyzátor dostal vždy najčerstvejšiu snímku a aby mohla analýza bežať v reálnom čase.



Obr. 4.1: Robot použitý na testovanie.

Občasná strata snímky nemusí byť problémom. Dôležité však je, aby samotná analýza obrazu dokázala spracovať dostatočný počet snímok za sekundu. Minimálny počet je pritom závislý od rýchlosti, ktorou sa robot pohybuje (konkrétne hodnoty spomenieme v kapitole 5).

Aby mohlo získavanie obrazu z kamery (pomalá hardvérová záležitosť) a jeho spracovanie bežať paralelne, je implementované v osobitných vláknach. Triedy, ktoré majú hlavný cyklus pre vlákno musia byť (podľa návrhu riadiaceho programu robota) potomkami triedy `RThread`.

Druhou triedou je abstraktný `GraphicsAnalyzer`. Ten obsahuje hlavný cyklus pre svoje vlákno, v ktorom získava vstupný obraz, spúšťa analýzu a ak je treba, tak nahráva video. Obraz je možné získavať z troch rôznych vstupov: kamera, videozáznam alebo statická fotka. Typ vstupu je daný premennou `inputDevice`, ktorá sa nastaví parametrom v konštruktore. Ak to typ vstupu vyžaduje, je v konštruktore naplnená aj premenná `inputFile`, ktorá udáva cestu k vstupnému súboru.

```
void RGraphicsAnalyzer::mainLoop() {
    if (setSourceImage()) {
        getMe()->analyze();
        if (showControl) rControlPanel->showControl();
        videoRecorder();
    } else die(); //nedostávam vstupný obraz, vyvolám ukončenie threadu
}
```

V prípade, že je ako vstup vybraná kamera, `GraphicAnalyzer` bude získavať aktuálny obraz pomocou objektu triedy `CamGrabber`. Od abstraktnej triedy musia dediť všetky analyzátory obrazu, ktorými robot disponuje. V prípade tejto bakalárskej práce je to len jeden. Analyzátor rozpoznávajúci šachovnicu a pohyb robota `GraphicAnalyzerChessboard`. Pri prípadnom rozširovaní (oddiel 5.4) je tak `GraphicAnalyzer` pripravený na ďalšie analyzátory.

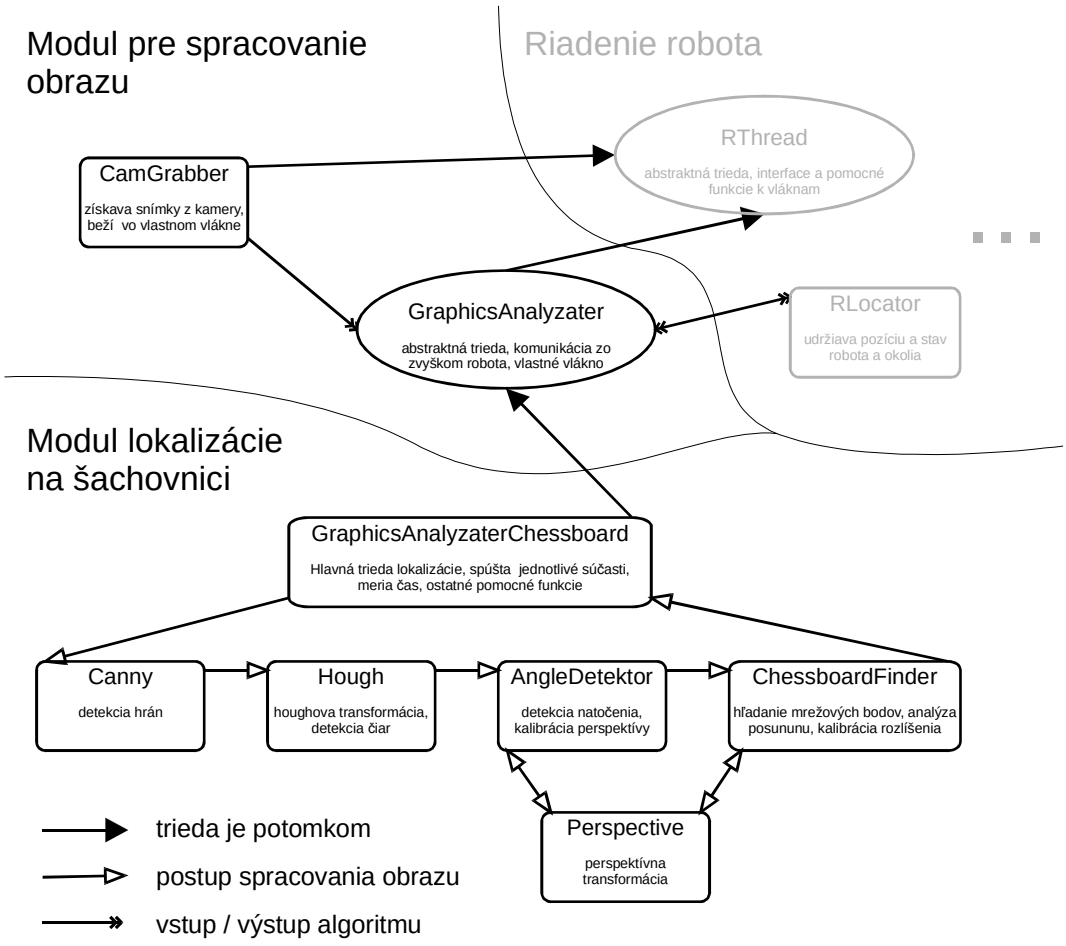
Druhou súčasťou implementácie je samotný modul pre analýzu šachovnice a pohybu robota. Pozostáva z triedy `GraphicAnalyzerChessboard`, ktorá dedí od triedy `GraphicAnalyzer` a z tried `Canny`, `Hough`, `Perspective`, `AngleDetector` a `ChessboardFinder`, ktoré implementujú jednotlivé časti algoritmu tak ako sme ich popísali v kapitole 3 návrh algoritmu. Všetky tieto triedy sú singletonmi podľa Mayersovho vzoru. Do tried sme jednotlivé algoritmy rozdelili iba kvôli prehľadnosti.

Náčrt hlavnej metódy analyzátoru šachovnice:

```
void RGraphicsAnalyzerChessboard::analyze(){

    //preberiem pozíciu z~lokátoru
    position = rLocator->getMyPosition();

    canny->make(inputImage);
    hough->make(inputImage);
```




Obr. 4.2: Implementácia grafického analyzátora a jej napojenie na riadiaci program robota.

```
    position = angleDetector->make(inputImage, position);
    position = chessboardFinder->make(inputImage, position);

    //kresliace metódy určené pre snímky a natočenie videa
    angleDetector->drawAnalyzedLines(inputImage);
    chessboardFinder->drawCrossPoints(inputImage);
    chessboardFinder->drawChessboard(inputImage);
    perspective->make(inputImage);

    //odovzdám pozíciu lokátoru
    rLocator->setMyPosition(position);
}
```

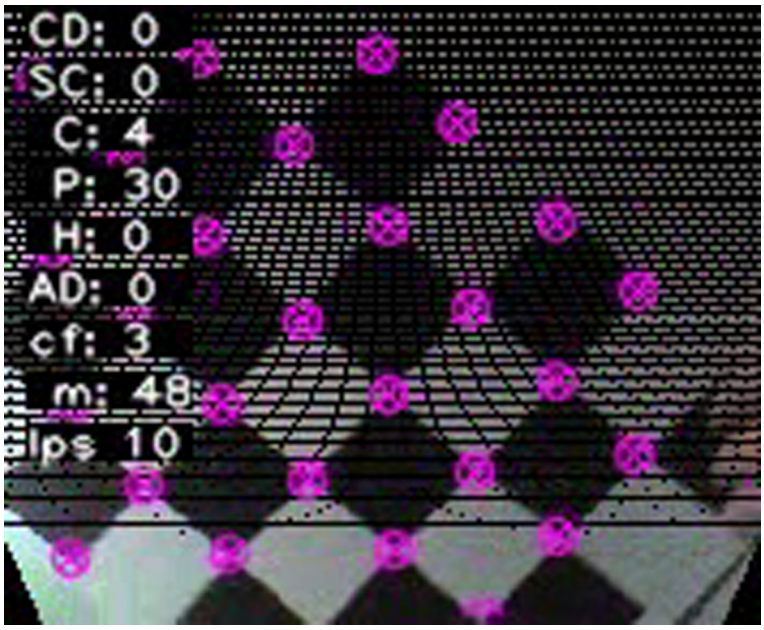
Zaujímavé pasáže z jednotlivých pomocných tried na analýzu obrazu sme už opísali v návrhu algoritmu, kapitola 3. Diagram zobrazujúci priebeh spracovania a dedičnosť tried je nakreslený na obrázku 4.2.

Nakoniec ešte ozrejmíme čísla, ktoré sa vyskytujú na videách a snímkach z nich. Ide o odstopované dĺžky spracovania jednotlivých algoritmov. V hlavnom cykle analyzátora jednotlivé pomocné triedy voláme v skutočnosti takto:

```
stopwatch[HOUGH].start();
hough->make(img_bw);
stopwatch[HOUGH].stop();
```

`stopwatch[HOUGH]` je objekt triedy `Stopwatch` na meranie času jednotlivých algoritmov. Výsledky je možné vidieť na všetkých videách (viz. obrázok 4.3), kde jednotlivé skratky sú popísané nasledujúcej tabuľke:

| | | | |
|---|---|---|---|
| CD | ColorDetector | AD | AngleDetector |
| SC | SurroundCutter | cf | ChessboardFinder |
| C | Canny | m | main loop (celá analýza) |
| P | Perspective | lps | loops per second (počet hlavných |
| H | Hough | | cyklov za sekundu) |




Obr. 4.3: Snímka z videa [iv]. Na ľavej strane obrázku sú zobrazené doby trvania jednotlivých algoritmov.

# Kapitola 5

# Zhodnotenie algoritmu

Testovanie navrhnutého algoritmu bolo zamerané na body podľa požiadaviek, kladených na lokalizáciu pri návrhu v kapitole 3. Meraná bola presnosť určenej polohy na rôznych šachovniciach, odskúšané boli schopnosti lokalizácie pri zhoršenej viditeľnosti, bola zistená odolnosť voči strácaniu a pritom boli nájdené najvhodnejšie parametre povrchu, po ktorom sa robot môže pohybovať.

Pri všetkých testoch bol robot ovládaný pomocou ovládača.

## 5.1 Schopnosť udržať si správnu pozíciu

Strata správnej pozície môže pri tomto spôsobe lokalizovania nastať z dvoch rôznych príčin.

Prvou je „strata uhla“. Ako sme opísali v oddiele 3.3.3, uhol natočenia získavame z detekovaných čiar. Tie na štvorcovej sieti sú dvoch typov. Jedny sú rovnobežné s uhlom natočenia (pre jednoduchosť môžme predpokladať, že počiatočný uhol natočenia bol v smere niektorých rovnobežiek), a tie druhé sú na tento smer kolmé. Tie, ktoré sú kolmé určíme podľa toho, že je ich uhol vzhľadom na smer natočenia väčší ako 45°.

Z toho okamžite vyplýva, že uhol natočenia bude určený zle, ak sa robot stihne otočiť medzi dvoma snímkami o viac ako 45°.

Druhou možnosťou straty pozície je „strata štvorca“. Mrežové body z poslednej známej pozície sa s každou novou snímkou párujú s detekovanými bodmi v obraze. Párovanie prebieha podľa vzájomnej vzdialeností mrežových bodov. Podobne ako pri strate uhla, môže nastať strata štvorca v prípade, ak sa robot medzi dvoma snímkami posunie o dostatočne veľký úsek, aby sa mrežové body popárovali zle.

Veľkosť úseku závisí od smeru natočenia. V najhoršom prípade je to polovica dĺžky štvorca, ak sa robot pohybuje zarovno štvorcovej siete. Ak by sme využili farbu políčka šachovnice, bolo by možné túto minimálnu vzdialenosť nutnú pre stratenie štvorca zväčšiť. Prišli by sme tak ale o možnosť navigácie na jednofarebnej štvorcovej sieti. Ideálne by teda bolo implementovať to ako voliteľnú funkciu, ako je navrhnuté v oddiele 5.4.

## 5.2 Presnosť lokalizácie

Meranie presnosti lokalizácie je možné robiť dvomi metódami. Prvou z nich je meranie polohy referenčného bodu robota od počiatočnej pozície a porovnanie tejto hodnoty z pozíciou, ktorú vyhodnotil algoritmus. Tento spôsob je problematický na väčšie vzdialenosti a úplne nepoužiteľný pri pohybe robota.

Druhou metódou je odčítanie vzdialenosti mrežových bodov šachovnice nakreslenej algoritmom od ich skutočnej polohy. Týmto spôsobom je možné z natočeného videa relatívne presne určiť chybu, s akou lokalizácia pracovala a tentokrát už aj v pohybe.

Táto metóda má však tiež svoje nevýhody. Nahrávanie videa spotrebovávalo nemalý výpočtový výkon robota a počet spracovaných snímok za sekundu tak klesol z počtu 15 (maximum, ktoré sprostredkovala kamera) na polovicu. To sa pravdepodobne nezanedbateľnou mierou pripísalo na dosiahnuté výsledky. Napriek tomu, na správnom type šachovnice boli výsledky stále výborné. Akou veľkou mierou prispelo k nepresnosti zníženie počtu spracovaných snímok za sekundu však môžeme iba odhadovať. Istý si môžeme byť ale tým, že vo chvíli, keď sa robot nepohybuje, nie je presnosť algoritmu nahrávaním videa ovplyvnená.

Namerané hodnoty uvediem pre každý typ šachovnice v nasledujúcom oddiele.

Presnosť lokalizácie je tiež veľmi závislá na rozlíšení snímaného obrazu. Čím väčšie je použité rozlíšenie, tým presnejšie lokalizované môžu byť mrežové body, a teda presnejšie určená pozícia robota. Veľkosť rozlíšenia je ale vždy obmedzená kamerou a výpočtovou silou palubného počítača.

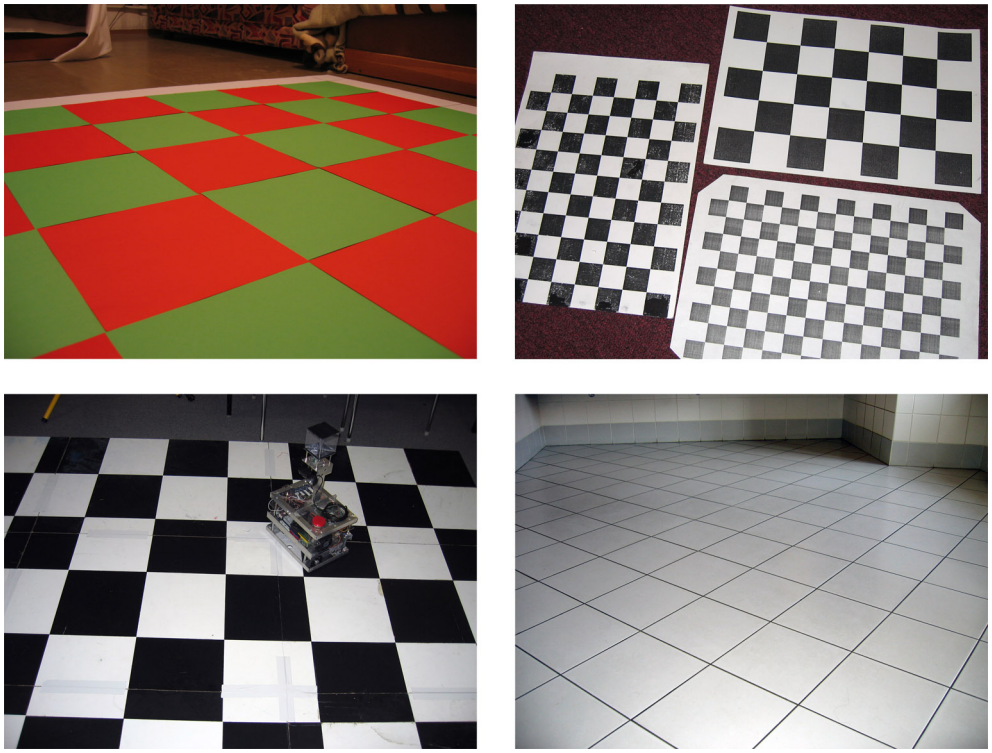
Ja som pri testoch som používal rozlíšenie obrazu $176 \times 144$, ktoré poskytovala kamera. Bolo dostatočne veľké na to, aby lokalizácia dosahovala dobré výsledky a zároveň dostatočne malé na to, aby palubný počítač robota, na ktorom bol algoritmus testovaný, stihol analyzovať 15 snímkov za sekundu.

## 5.3 Vhodné parametre šachovnice

Počas testovania bolo vyskúšaných čo možno najviac typov povrchov, aby bolo možné určiť ideálne parametre pre vytvorenú lokalizáciu.

Jedným z testovaných povrchov bola červeno-zelená šachovnica so štvorcami s hranou dĺžky 21 cm vyrobená z farebných papierov. Matný papier a výrazné farby veľmi dobre poslúžili pri detekcií hrán. Takáto konštrukcia bohužiaľ nebola príliš vhodná pre prechádzky robota a tak bola využitá hlavne pre analýzu statických obrázkov.

Pre ďalšie testovanie bol použitý šachovnicový motív rôznych veľkostí vytlačený na kancelársky papier. Pri takejto podložke nebol vďaka presnej a výraznej tlače nikdy problém z detekciou hrán, čiar alebo uhla. Veľkým nedostatkom sú však príliš malé štvorce, ktoré znemožňovali správnu lokalizáciu pri prudšom pohybe. Lokalizácia vtedy často strácala štvorce. Oproti tomu vyvolať stratu uhla bolo skoro nemožné. Pri frekvencií 15 snímok za sekundu to znamenalo



Obr. 5.1: Testované povrchy.

otočiť šachovnicou o viac ako 45° za 66 ms. To sa samozrejme stihnúť dá, ale robot sám sa tak rýchlo otáčať nedokáže.

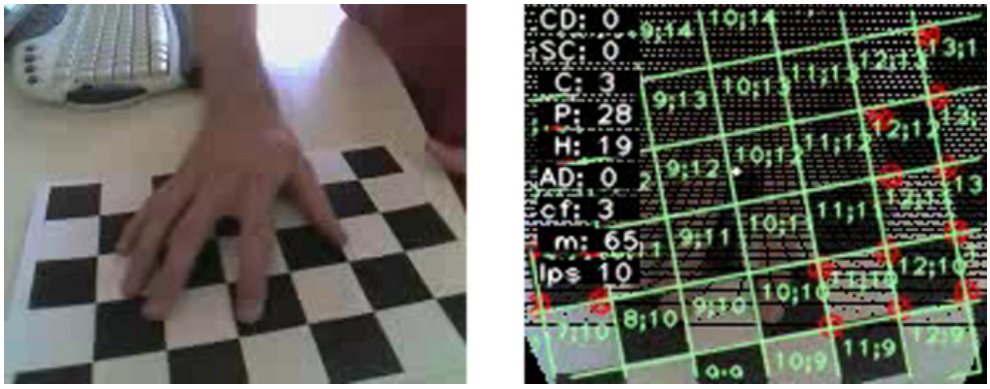
Pri pomalom pohybe poskytovala šachovnica s malými štvorcami výbornú presnosť lokalizácie polohy, ktorá sa pohybovala do 1 cm od skutočnej pozície. Algoritmus mal možnosť nájsť veľký počet štvorcov a tým získať dostatočný počet referenčných bodov pre presné určenie polohy.

S týmito šachovnicovými podložkami bola tiež odskúšaná odolnosť lokalizácie na rôzne svetelné podmienky. Vďaka výrazným a ostrým prechodom nebolo slabé osvetlenie problémom. Ak však bolo svetlo už príliš slabé, začal byť obraz veľmi zašumený a rozmazaný a lokalizácia sa stala nepoužiteľnou. V takomto prípade by pravdepodobne pomohla špeciálna kamera alebo vlastné svetlá namontované na robotovi.

Ako vidieť na videu [vii], vďaka malým štvorcom bola lokalizácia úspešná aj po prekrytí veľkej časti obrazu (obrázok 5.3).

Obecne platí, čím väčšie štvorce šachovnica má, tým rýchlejší pohyb robota je lokalizácia schopná ustrážiť bez straty štvorca. Podmienkou však je aby kamera mala vždy dostatočne dobrý výhľad. Ten je použiteľný približne od dvoch detekovaných štvorcov na snímku. Čím viac štvorcov sa podarí algoritmu detekovať, tým presnejší výsledok je možné dosiahnuť.

Ako najlepší povrch, ktorý bola možnosť otestovať, sa ukázala veľká čiernobiela drevená šachovnica. Jej štvorce mali veľkosť 30 cm. Nastavenie kamery




Obr. 5.2: Detekovanie papierovej šachovnice po zakrytí významnej časti obrazu. Snímka z videa [vii]

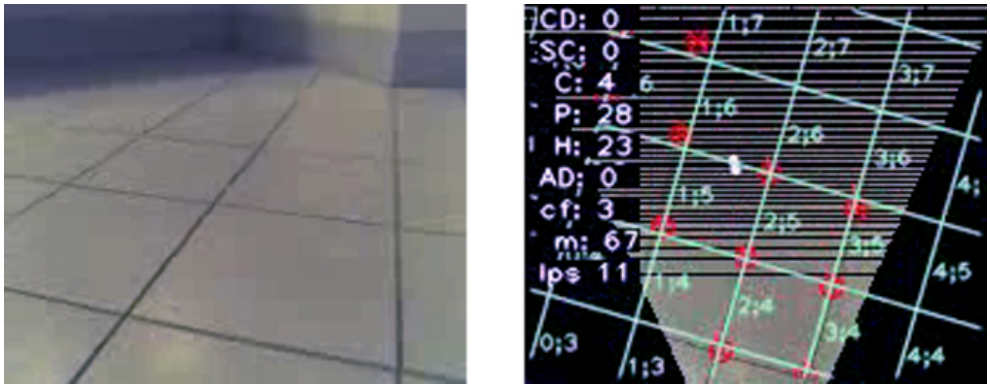
umožnilo detekovať v priemere 4 štvorce na snímku. To už postačovalo na presnosť lokalizácie do 3 cm od skutočnej polohy. Pri maximálnej rýchlosti robota, ktorá je približne 120 cm za sekundu, sa lokalizácia vplyvom rozmazaného obrazu zhoršila približne na úroveň 8 cm od skutočnej pozície. Vďaka dostatočnej veľkosti štvorcov šachovnice to ani pri ľubovoľne prudkých pohyboch, ktoré dokázal robot vykonať, neznamenalo stratu štvorca. Video [viii] zobrazuje jednu z prechádzok po tejto šachovnici.

Nakoniec boli ešte vyskúšané dve rôzne podlahy tvorené jednofarebnými dlaždicami.

V prvom prípade išlo o svetlú lesklú dlažbu s bielou špárou. Dĺžka strany dlaždice bola 30 cm. Osvetlenie, ktoré sa na dlažbe odrážalo spolu s nevýraznosťou farbou špáry beh lokalizácie úplne znemožnili. Podarilo sa síce po dlhšej dobe automaticky vykalibrovať perspektívnu transformáciu aj veľkosť štvorca, ale pri pohybe sa robot okamžite stratil (video [ix]).

V druhom prípade to bola biela lesklá dlažba s čiernou špárou. Veľkosť strany jednej dlaždice bol rovnako 30 cm. Na tomto povrchu fungovala lokalizácia bez problémov do 5 cm od skutočnej pozície (obrázok 5.3 a video [x]). Oproti čiernobielej šachovnici bola ale detekcia mrežových bodov o niečo zložitejšia a pri prudkých pohyboch hrozila strata štvorca. Veľký podiel na tom má lesklosť dlaždíc. Ak by boli dlaždice matné, pravdepodobne by nevznikalo pri detekovaní hrán toľko šumu a hľadanie mrežových bodov by bolo úspešnejšie. To už ale bohužiaľ nebola možnosť overiť v praxi.

Odchýlka určenia uhla natočenia sa vo všetkých prípadoch pohybovala do 2°. Tejto presnosti sa dosiahlo dokonca aj na dlažbovej podlahe s bielou špárou, kde lokalizácia polohy nefungovala.




Obr. 5.3: Lokalizacia na bielej lesklej dlažbe s čiernou špárou. Snímka z videa [x]

## 5.4 Ďalší vývoj

Čo sa týka algoritmickej časti odporúčam pre ďalší vývoj nasledujúce kroky:

- Namiesto klasickej Houghovej transformácie implementovať niektorú z vylepšených verzií, ktoré môžu byť výrazne rýchlejšie. Ponúka sa napríklad pravdepodobnostná Houghova transformácia [12] alebo jej rozšírenie progresívna pravdepodobnostná Houghova transformácia podľa Matasa a kol. [13].
- Navrhnúť a implementovať kalibráciu Cannyho algoritmu. Ako sme uviedli v oddiele 3.3.1, nie je kalibrácia bezpodmienečne nutná, ale mohla by čiastočne zrýchliť beh Houghovu transformácie.
- Navrhnúť a zapracovať kombináciu s iným typom lokalizácie. Algoritmus by sa stal naozaj robustným, ak by dokázal využívať pre spresnenie napríklad odometriu. Tá môže byť na malých úsekoch dostatočne presná a mohla by tak úplne zamedziť strate štvorca alebo uhla.
- Detekcia obsadeného políčka šachovnice. Robot by sa nezanedbateľne priblížil prípadnému nasadeniu v priemysle napríklad ako manipulátor paliet v skladoch.

Ďalej je možné vylepšiť lokalizáciu rozšírením implementácie:

- Umožniť nastaviť ľubovoľné natočenie kamery nielen v smere vertikálnom, ale aj v horizontálnom. Tiež by bolo možné dorobiť perspektívnu transformáciu, ktorá by umožnila použiť kameru namontovanú šikmo, t.j. horizont by už nebola priamka na snímke rovnobežná s osou $x$.
- Zrýchliť a spresniť perspektívnu transformáciu pre celý obraz. Algoritmus ju síce nevyužije, ale ako názorná ukážka funkčnosti je veľmi užitočná.

- Možnosť využiť iného ako štvorcového vzoru na podlahe. Napríklad kachličky obdĺžnikového tvaru. Ďalej možnosť parametrizovať algoritmus pre pohyb na šachovnici alebo štvorcovej sieti. Na šachovnici by potom mohol naviac využiť dvojfarebnosť políčok.

- Umožniť pridanie ďalšej alebo niekoľko ďalších kamier na robota pre zaistenie lepšej spoľahlivosti. V prípade, že sa robot dostane do rohu miestnosti tak, že kamera stratí výhľad na podlahu, mohla by v lokalizácii poslúžiť druhá kamera umiestnená na opačnej strane robota.

## 5.5 Záver

Vzhľadom na dosiahnuté výsledky počas testovania algoritmu na rôznych povrchoch môžem považovať ciele, ktoré si táto bakalárska práca stanovila, za dosiahnuté a to dokonca nad pôvodné očakávania. Dosiahnutá presnosť, ktorá sa na vhodnom povrchu (čierno-biela šachovnica s hranou štvorcov veľkosti 30 cm) pohybovala do 3 cm spolu výbornou odolnosťou voči strácaniu, je známkou spoľahlivosti tejto lokalizácie. Skutočnosť, že algoritmu pre správne lokalizovanie postačuje aj štorcová sieť vytvorená napríklad jednofarebnou dlažbou ho zase umožňuje použiť na v mnohých miestach bez nutnosti úpravy podlahy.

Presnosť lokalizácie je zároveň veľmi dobre škálovatelná. Jednoduchým zvýšením rozlíšenia snímaného obrazu sa spresní lokalizácia mrežových bodov, čo automaticky vedie k presnejšiemu určovaniu polohy. Ďalšie spresnenie by mohlo byť dosiahnuté pridaním ďalších kamier, čo by vyžadovalo iba malé zmeny v implementácii. Rozlíšenie snímaného obrazu a počet kamier je však silno limitované výpočtovou kapacitou palubného počítača robota.

Automatickou kalibráciou sa úspešne dosiahlo zníženie počtu parametrov vyžadujúcich konfiguráciu na minimum.

# Odkazy na CD

[i] Video ilustrujúce funkciu Cannyho algoritmu.
`CD:\video\detekciaHranCanny.avi`

[ii] Hľadanie čiar pomocou Houghovu transformácie.
`CD:\video\detekciaCiarHough.avi`

[iii] Detekcia uhla angle detectorom, bez perspektívnej transformácie.
`CD:\video\detekciaUhla.avi`

[iv] Hľadanie mrežových bodov.
`CD:\video\hladanieMrezovychBodov.avi`

[v] Kalibrácia perspektívy.
`CD:\video\kalibraciaPerspektivy.avi`

[vi] Rotovanie mrežových bodov z predošlej snímky.
`CD:\video\rotovanieStarychMrezovychBodov.avi`

[vii] Sledovanie papierovej šachovnice s prekrytím.
`CD:\video\sledovaniePapierovejSachovnice.avi`

[viii] Prechádzka po šachovnici.
`CD:\video\prechadzkaPoSachovnici.avi`

[ix] Testovanie kachličiek s nevýraznou špárou.
`CD:\video\kachlickySBielouSparou.avi`

[x] Testovanie kachličiek s výraznou špárou.
`CD:\video\kachlickySCiernouSparou.avi`

# Literatúra


[1] Dellaert F., Fox D., Burgard W., Thrun S.: *Monte Carlo Localization for Mobile Robots*, IEEE International Conference on Robotics and Automation (ICRA99), May 1999.

[2] Fox D., Burgard W., Thrun S.: *Markov Localization for Mobile Robots in Dynamic Environments*, Journal of Artificial Intelligence Research, Volume 11, pp. 391-427. Nov 1999.

[3] Canny J.: *A Computational Approach to Edge Detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 8, Number 6, Nov 1986.

[4] Hough P. V. C.: *Machine Analysis of Bubble Chamber Pictures*, International Conference on High Energy Accelerators and Instrumentation, CERN, 1959.

[5] Open source Computer Vision Library (OpenCV), `http://opencvlibrary.sourceforge.net/`

[6] Sobel, I., Feldman G.: *A 3x3 Isotropic Gradient Operator for Image Processing*, prednášané na Stanford Artificial Project v roku 1968, nepublikované ale často citované, orig. Duda R., Hart P., Wiley J. and Sons: *Pattern Classification and Scene Analysis*, pp. 271-272, 1973

[7] Prewitt J. M. S.: *Picture Processing and Psychophysics*, A. Rosenfeld and B. S. Lipkin, eds. pp. 75-149, New York: Academic Press, 1970.

[8] Masayuki Katahira, `http://www.mis.med.akita-u.ac.jp/~kata/image/sobelprew.html`

[9] Bill Green:, *Canny Edge Detection Tutorial*, 2002 `http://www.pages.drexel.edu/~weg22/can_tut.html`

[10] Suzuki S., Abe K.: *Topological structural analysis of digital binary images by border following*, CVGIP, Volume 30, n.1, pp. 32-46, 1985

[11] Douglas D., Peucker T.: *Algorithms for the reduction of the number of points required to represent a digitized line or its caricature*, The Canadian Cartographer 10(2), pp. 112-122, 1973

[12] Kiryati N., Eldar Y., Bruckstein A. M.: *A probabilistic Hough Transform*, Pattern Recognition, Volume 24, pp. 303–316, 1991.

[13] Matas J., Galambos C., Kittler J.: *Progressive probabilistic Hough Transform*, Proc. British Machine Vision Conf. BMVC98, Sep 1998.