Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

Bakalářská práce

Odhad polohy pohybujícího se předmětu z obrazových dat

PLZEŇ, 2013 Lukáš Kiráľ

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: **2012/2013**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Lukáš KIRÁĽ**
Osobní číslo:	**A10B0456P**
Studijní program:	**B3918 Aplikované vědy a informatika**
Studijní obor:	**Kybernetika a řídicí technika**
Název tématu:	**Odhad polohy pohybujícího se objektu z obrazových dat**
Zadávající katedra:	**Katedra kybernetiky**

Zásady pro vypracování:

1. Vyberte vhodnou metodu detekce objektu v obraze.
2. Seznamte se s metodami odhadu polohy pohybujícího se objektu.
3. Navrhněte algoritmus odhadu polohy pohybujícího se objektu na základě obrazových dat.

Rozsah grafických prací:	**dle potřeby**
Rozsah pracovní zprávy:	**30-40 stránek A4**
Forma zpracování bakalářské práce:	**tištěná**
Seznam odborné literatury:	

Dodá vedoucí bakalářské práce

Vedoucí bakalářské práce:	**Ing. Miroslav Flídr, Ph.D.** Katedra kybernetiky
Datum zadání bakalářské práce:	**1. listopadu 2012**
Termín odevzdání bakalářské práce:	**17. května 2013**

Doc. Ing. František Vávra, CSc.
děkan

L.S.

Prof. Ing. Josef Psutka, CSc.
vedoucí katedry

V Plzni dne 1. listopadu 2012

P R O H L Á Š E N Í

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne 13. 5. 2013

……

vlastnoruční podpis

Poděkování

Tímto bych chtěl poděkovat vedoucímu mé bakalářské práce Ing. Miroslavu Flídrovi Ph.D. za velkou ochotu a trpělivost, poskytnutí informací i cenných rad.

Anotace

Tato práce se zabývá sledováním pohybujícího se objektu na základě obrazových dat. V první části je uveden přehled možných metod detekce objektu v obraze. Dále jsou využity dvě kamery pro triangulaci a získání polohy objektu v prostoru. Součástí práce je také filtrování naměřených dat pomocí Kalmanova filtru. Nakonec jsou zhodnoceny výsledky navrhovaného postupu.

Klíčová slova: počítačové vidění, detekce pohybu, triangulace, Kalmanův filtr

Abstract

This work deals with estimation of moving object position based on visual data. In first part, there is an overview of the possible methods of object detection in the image. Then two cameras are used to triangulate and obtain the position of an object in space. The work also includes filtering the measured data using a Kalman filter. Finally, the evaluation of the results of the proposed procedure is given.

Keywords: computer vision, motion detection, triangulation, Kalman filter

Obsah

Kapitola 1

Úvod

Hlavním cílem této bakalářské práce je vytvořit systém pro lokalizaci robotů v místnosti. V našem případě se bude jednat o robota s automatickým řídícím systémem. Tento systém bude mít k dispozici informaci o přibližné poloze robota z jeho senzorů. Úkolem této práce je přidat objektivní informaci vnějšího pozorovatele. Výstupem tak budou souřadnice sledovaného robota v rovině a také jeho rychlost pohybu.

Asi nejjednodušším řešením by bylo využití GPS. To nám může poskytnout informaci o poloze s přesností až 1 cm. Problém je ale s příjmem GPS signálu v budovách. Proto tento přístup nebude vhodný pro využití v místnosti.

Jako další možnost se nabízí použití kamerového systému. V tomto přístupu je mnoho variant, a to jak v oblasti výběru hardwaru, tak v oblasti softwaru. Cílem této práce je právě využití informaci z kamerového systému. Stěžejní částí tak bude otestovat různé přístupy získávání informaci z obrazových dat.

Měření polohy robota musí splňovat určité požadavky. Nejdůležitější je přesnost měření. Pro navigaci v místnosti bychom potřebovali informaci o poloze s přesností na centimetry. Dalším požadavkem je rychlost běhu programu. Pro sledování pohybujícího se objektu v reálném čase budeme potřebovat zpracovat dostatečné množství snímků za sekundu. V případě pomalého běhu programu bychom měli zastaralou informaci o poloze robota. Dalším hlediskem při návrhu systému je ekonomická náročnost. To je třeba zohlednit hlavně při výběru hardwaru. Je samozřejmě možné použít velké množství kamer a různých senzorů, ale takové řešení by bylo příliš drahé.

V první části práce se budeme zabývat výběrem hardwaru a softwaru. V další kapitole budeme testovat různé metody na detekci pohybujících se objektů v obraze. Ve čtvrté kapitole provedeme kalibraci systému a triangulací získáme prostorové souřadnice sledovaného robota. V páté kapitole se podíváme na filtraci naměřených údajů.

Kapitola 2

Výběr hardwaru a softwaru

Ke sledování polohy pohybujícího se robota lze využít mnoho různých přístupů. V této části práce se budeme zabývat výběrem vhodného vybavení. To lze rozdělit na hardware (kamery a senzory) a software (knihovny a další programové prostředky).

2.1 Hardware

K určení souřadnic určitého objektu v prostoru je zapotřebí informace z více zdrojů. Jednou možností je využití více kamer rozmístěných v místnosti.

Nejprve ale otestujeme možnosti spojení kamery a senzoru měřícího vzdálenost od objektu. Tyto kamery, které kromě obrazu poskytují i jeho hloubku se označují jako RGB-D kamery. Do této kategorie patří také Kinect od Microsoftu. Ten obsahuje normální RGB kameru s rozlišením 640x480 bodů při frekvenci 30 snímků za sekundu, a také disponuje infračerveným hloubkovým senzorem. Je vhodný zejména pro použití v místnosti, protože intenzivní sluneční svit oslepuje jeho infračervený senzor. Přesnost měření vzdálenosti je velmi dobrá. Chyba při měření hloubky do 4 metrů je menší než 1 cm[1].

Hlavní výhodou Kinectu je tedy schopnost získat relativně přesné údaje o poloze určitého objektu v prostoru pouze z jednoho zařízení. Má ale také určitá omezení. RGB kamera disponuje relativně nízkým rozlišením, které nemusí při sledování objektů daleko od Kinectu stačit. Také hloubkový senzor má omezený dosah. Navíc cena při srovnání s obyčejnou webkamerou je výrazně vyšší. Jako menší mínus můžeme také zmínit potřebu instalace několika ovladačů a podpůrných knihoven.

Pro pokrytí celé místnosti budeme potřebovat více těchto zařízení. V tom případě bude ekonomicky výhodnější použití několika kamer, které jsou podstatně levnější. Konkrétně budeme mít k dispozici 3 webkamery Logitech c525. Ty mohou pracovat až v rozlišení 720p. Hlavní výhodou oproti Kinectu jsou tedy širší možnosti a větší potenciál do budoucna. Tento přístup bude ale zároveň vývojově náročnější, protože souřadnice se budou muset získávat minimálně ze dvou kamer paralelně a následně ještě triangulovat. Rozmístěním kamer v místnosti a triangulací se budeme zabývat v jedné z následujících kapitol.

2.2 Software

Ačkoliv můžeme celou aplikaci začít psát od nuly, existuje celá řada již napsaných knihoven, které nám usnadní práci. Zaměříme se především na otevřená řešení. Jednou z možností je použití nástroje Processing[2], který je založený na jazyce Java. Processing obsahuje řadu metod pro manipulaci s obrazem. Ke stažení je také velké množství dodatečných knihoven z různých oblastí, včetně počítačového vidění. Například pro použití Kinectu je k dispozici knihovna simple-openni[3]. Dále existují knihovny na rozpoznávání obličeje nebo detekci blobů a spousta dalších.

Processing plně dostačuje pro úlohy základního zpracování obrazu. Pokud bychom ale chtěli využít složitější algoritmy, bude lepší zaměřit se na jinou knihovnu. Asi nejrozšířenější knihovnou v oblasti počítačového vidění je knihovna OpenCV[4]. OpenCV je nativně psána v jazyce c++, ale obsahuje také rozhraní pro jazyky Python a Java. K dispozici je jak verze pro Windows, tak také pro Linux, Mac OS, iOS a Android. Knihovna obsahuje řadu modulů zaměřených na různé oblasti počítačového vidění. Následující moduly budou zajímavé pro naše účely:

- core - The Core Functionality

  základní funkce pro práci s obrazem

- imgproc - Image Processing

  metody pro zpracování obrazu, obsahuje také implementované detektory (Canny, Harris)

- calib3d - Camera Calibration and 3D Reconstruction

  metody pro kalibraci kamer a také triangulaci

- features2d - 2D Features Framework

  obsahuje implementaci dalších detektorů (FAST, MSER) a deskriptorů

- objdetect - Object Detection

  nástroje pro detekci objektů s využitím natrénovaného klasifikátoru

- gpu - GPU-accelerated Computer Vision

- ocl - OpenCL-accelerated Computer Vision

  tyto moduly umožňují akceleraci výpočtů využitím grafické karty. Modul gpu je zaměřen pouze na grafické karty podporující CUDA, zatímco ocl je možné použít také s ostatními grafickými kartami podporujícími OpenCL.

Pro OpenCV je k dispozici také řada přídavných balíčků. Např. Intel © Threading Building Blocks (TBB) pro využití více-jádrových procesorů, nebo OpenNI Framework pro práci s Kinectem. Knihovna OpenCV nám tedy poskytne vhodný základ pro vytvoření požadovaného systému pro sledování polohy pohybujícího se robota v místnosti.

Kapitola 3

Detekce objektu v obraze

Před vlastním sledováním polohy robota je potřeba jej v obraze z kamer nalézt. Existuje velké množství metod, které umožňují detekovat určité zajímavé oblasti v obraze. Každá má své výhody i nevýhody a hodí se pro jiné účely. Proto nejprve provedeme rozbor různých přístupů k tomuto problému.

3.1 Metoda s trénováním klasifikátoru

Pokud chceme nalézt pouze jeden konkrétní objekt, tak se jako jeden z nejlepších způsobů může jevit natrénování klasifikátoru na detekci daného objektu. K tomuto účelu lze využít knihovnu OpenCV, která obsahuje řadu užitečných metod pro práci s obrazem, a kterou budeme používat prakticky v celé této práci.

Budeme potřebovat velké množství obrázků sledovaného předmětu. To lze získat natočením asi minutového videa našeho objektu. Při frekvenci 25 snímků za sekundu získáme 1500 obrázků. Ty by měly zachytit pozorovaný předmět ze všech možných úhlů. Dále bude třeba také velký počet snímků prostředí bez sledovaného objektu. Pro získání pozitivních i negativních vzorků z videa lze využít nástroj Positive Builder[5]. Ten umožňuje označení objektu v obraze. Označit hledaný předmět ručně je samozřejmě potřeba ve všech pozitivních snímcích. Po vytvoření trénovací množiny už jen použijeme knihovnu OpenCV pro natrénování klasifikátoru. K tomu slouží metoda haartraining. Trénování klasifikátoru může trvat i několik dní, v závislosti na počtu vzorků. Samozřejmě s počtem použitých vzorků roste také přesnost výsledného klasifikátoru a zároveň i čas potřebný pro trénink. Po natrénování je ještě potřeba převést výsledná data do xml souboru, který poslouží jako vstupní parametr při volání klasifikátoru. Pro tento účel lze použít metodu convertcascade.

Výsledky tohoto přístupu jsou rozporuplné. Přesnost vzorového klasifikátoru na rozpoznávání obličeje byla uspokojivá. Avšak přesnost mého klasifikátoru nebyla dostatečná pro reálné využití. Navíc má tento přístup řadu nevýhod. Jako největší problém bych zmínil vysokou hardwarovou náročnost. Na průměrném PC je program schopen vyhodnotit asi 5 snímků za sekundu, což není pro sledování objektů v reálném čase dostatečná rychlost. Pro využití této metody by byla zapotřebí výkonná

grafická karta. Dalším omezením je schopnost detekovat pouze jeden konkrétní objekt. Pokud by se náš předmět změnil, bude třeba provést znovu náročný trénink. Mezi výhody je možné zahrnout fakt, že při správném fungování získáme přímo hledaný objekt a ne pouze určité oblasti v obraze.

3.2 Metoda detekce významných bodů

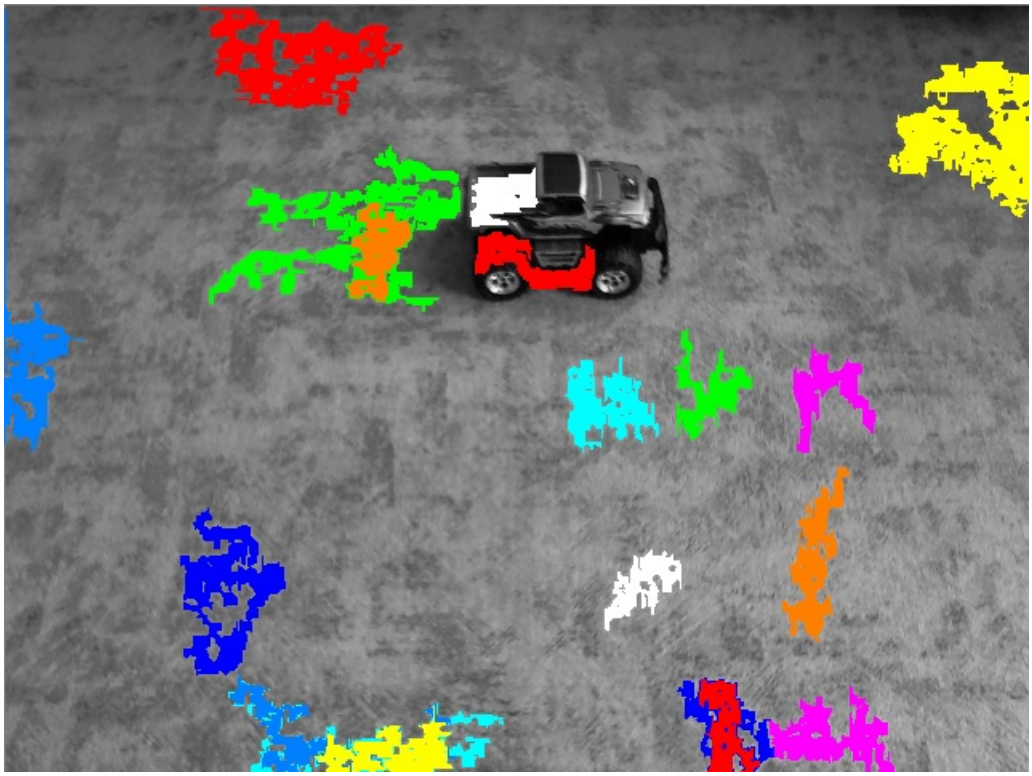
Detekce významných bodů slouží k získání určitých informací z obrazu. Neexistuje univerzální definice toho, co je významný bod. Obecně se dá tvrdit, že významný bod je určitá oblast v obraze, která je něčím zajímavá. Na detekci těchto oblastí bylo vyvinuto velké množství různých algoritmů, označovaných jako detektory. Ty se dají rozdělit do několika skupin. V zásadě to jsou Blob detektory (detektory oblastí), Edge detektory (detektory hran) a Corner detektory (detektory rohů), přičemž některé algoritmy mohou patřit do více skupin. Z každé skupiny vybereme jednoho zástupce a na jednoduché scéně otestujeme vhodnost daného přístupu pro naši úlohu. Testovací scéna (obrázek 3.1) obsahuje pouze sledovaný objekt (auto na dálkové ovládání) umístěný na koberci. Všechny následující algoritmy jsou implementovány v knihovně OpenCV a my pouze nastavíme vhodné parametry pro danou scénu.

Obrázek 3.1: Testovací scéna

Blob detektory

Tato kategorie detekuje celé oblasti v obraze, na základě podobnosti pixelů. Jako zástupce této kategorie otestujeme algoritmus MSER (Maximally stable extremal regions)[6]. Tento algoritmus je založený na prahování. Postupně se prochází sekvence snímků při použití různých hodnot prahu. Detekovány jsou takové oblasti, které se pro různé hodnoty prahu příliš nemění.

Výsledky algoritmu MSER pro naši scénu zobrazuje obrázek 3.2. Je zřejmé, že použití blob detektoru nebude pro detekci robota příliš výhodné. Jedním problémem jsou detekované oblasti okolo robota, což je dáno členitostí koberce. To ukazuje, že pro složitější scény by v obraze bylo detekováno více objektů. Druhým problémem je fakt, že samotný robot je detekován poze částečně a navíc jako více oblastí. Ani pro různé nastavení parametrů se nepodařilo detekovat robota jako jeden objekt. To je dáno složitostí sledovaného objektu. Navíc algoritmus potřebuje přibližně 200 ms na vyhodnocení obrázku, takže není vhodný pro běh v reálném čase.



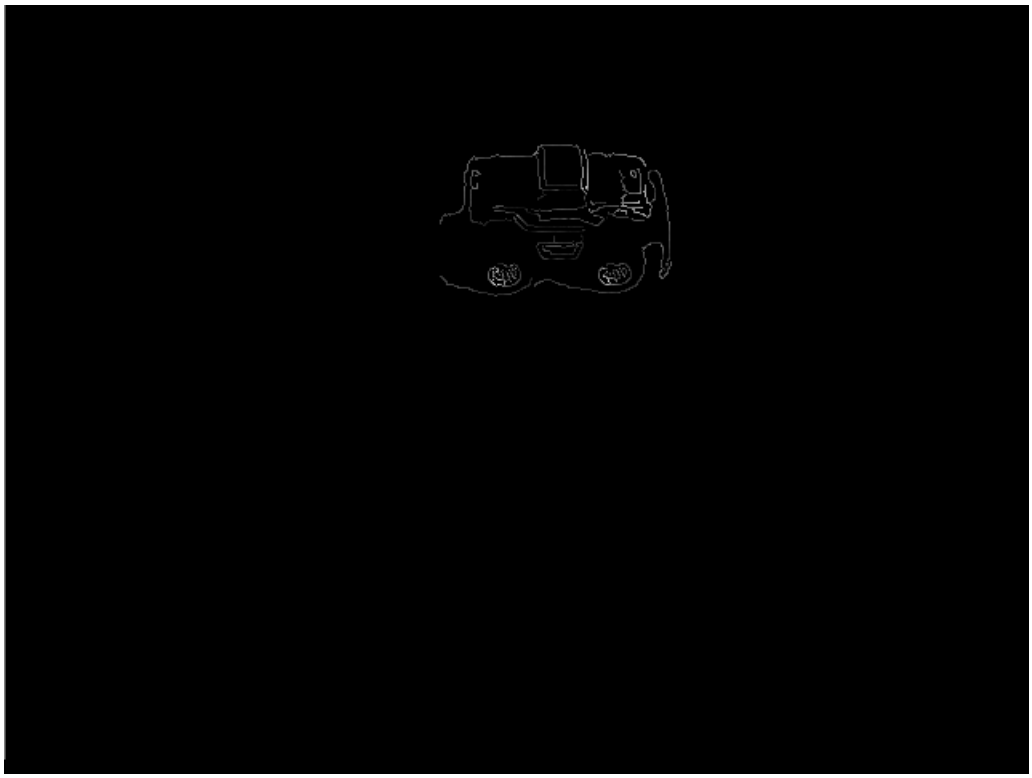
Obrázek 3.2: Výsledek aplikace algoritmu MSER

Detektory hran

Další kategorií detektorů jsou hranové detektory. Ty mají za úkol nalézt v obraze body nespojitosti. Tyto body tvoří křivky označované jako hrany. Zde budeme testovat algoritmus Canny[7]. Jedná se o vícefázový detektor vyvinutý v roce 1986 Johnem Cannym. V první fázi se redukuje šum za pomoci Gaussovského filtru. Poté se nalezne gradient intenzity - výsledný úhel je zaokrouhlen na jeden ze čtyř možných úhlů ve stupních (0, 45, 90, 135). Následně se potlačí nemaximální hodnoty, čímž se odstraní pixely, které nejsou součástí hrany. Nakonec se provádí hystereze, kde se používají dva prahy (horní a dolní):

- pokud je gradient pixelu větší než horní práh, je pixel považován za hranu
- pokud je gradient pixelu menší než dolní práh, není pixel považován za hranu
- pokud je gradient mezi horním a dolním prahem, je to hrana, pouze je-li hodnota gradientu sousedního pixelu vetší než horní práh

Výsledky pro testovací scénu jsou na obrázku 3.3. Obrys sledovaného objektu zde byl detekován velmi dobře. Při vhodně nastavených parametrech je to také jediný detekovaný objekt. Tento přístup by byl pro naše účely vhodnější.



Obrázek 3.3: Výsledek aplikace algoritmu Canny

Detektory rohů

Poslední kategorií jsou detektory rohů. Roh může být definován jako místo, kde se střetávají dvě hrany. Většina rohových detektorů ale nedetekuje pouze rohy, ale obecně "zajímavé"body. Tímto bodem budeme rozumět místo, které lze dobře identifikovat, např. lokální minimum nebo maximum. Zde vyzkoušíme dva algoritmy. Prvním bude SURF (Speeded Up Robust Features)[8] a pak také FAST (Features from Accelerated Segment Test)[9][10].

SURF je založený na výpočtu determinantu Hessovy matice. Může pracovat velmi rychle v případě, že se jedná o integrální obrazy. Integrální obraz je taková reprezentace obrazu, kde každý bod představuje součet hodnot předchozích pixelů vlevo a nahoře. FAST zkoumá okolí každého bodu. Pokud je dostatečné množství okolních pixelů podstatně světlejších nebo tmavších, potom je zkoumaný pod označen jako "významný".

Výsledky jsou na obrázcích3.4 a 3.5. Oba algoritmy detekují praktický stejné body. Všechny tyto body odpovídají sledovanému objektu. Pro jednoduchou scénu, bylo by možné využít také tyto detektory. V případě algoritmu SURF bychom ale museli všechny snímky nejprve převést na integrální obrazy, protože při aplikaci na běžná data není rychlost zpracovávání dostatečná pro běh v reálném čase.

Obrázek 3.4: Výsledek aplikace algoritmu SURF

Obrázek 3.5: Výsledek aplikace algoritmu FAST

3.3 Metoda detekce pohybu

Pro sledování pohybujícího se objektu se jako nejlepší možnost přímo nabízí detekovat pohyb v obrazové sekvenci. Hlavní výhodou tohoto přístupu je nezávislost na složitosti pozorované scény a také složitosti sledovaného objektu. Pro tyto účely lze v zásadě použít dva různé přístupy. Jednou možností je přímá detekce rozdílů ve dvou následujících snímcích a druhou je modelování pozadí.

Rozdíl snímků

Nejjednodušší způsob detekce pohybu v obrazové sekvenci je porovnání dvou sousedních snímků. Případné rozdíly mezi nimi nám pravděpodobně indikují pohyb. Samozřejmě se může jednat také o změnu osvětlení nebo automatickou korekci kamery, ale tyto faktory se objevují spíše sporadicky a nemají dlouhodobější charakter. Proto budeme předpokládat, že rozdíly ve snímcích značí pohybující se objekt.

Implementace této metody je velice jednoduchá. Stačí oba snímky přebarvit do stupňů šedé barvy a odečíst. Potom jen určíme práh a pixely s menší hodnotou obarvíme černě a pixely s větší hodnotou obarvíme bíle. Ještě je vhodné použít erozní filtr na odstranění šumu. Výsledek je vidět na obrázku 3.6.

Obrázek 3.6: Metoda rozdílu v sousedních snímcích

Výsledky tohoto přístupu jsou celkem uspokojivé. Sledovaný objekt není detekován celý, ale pro určení jeho polohy to není stěžejní. Při dostatečné frekvenci snímků za sekundu je i relativně rychlý pohyb detekován dostatečně přesně. Je samozřejmě patrné mírné rozmazání ve směru pohybu, ale při pomalejším pohybu se tento problém neprojeví. Také rychlost zpracování obrazu je velmi vysoká. Algoritmus potřebuje pro vyhodnocení pohybu pouze něco málo přes 1 ms, takže budeme omezeni pouze snímkovací frekvencí kamery.

Problém této metody nastane v okamžiku, kdy se sledovaný robot přestane pohybovat. V tom případě jej tato metoda detekovat nebude. To lze vyřešit tím, že pokud nedetekujeme žádný objekt, budeme za aktuální polohu hledaného robota považovat poslední známé souřadnice.

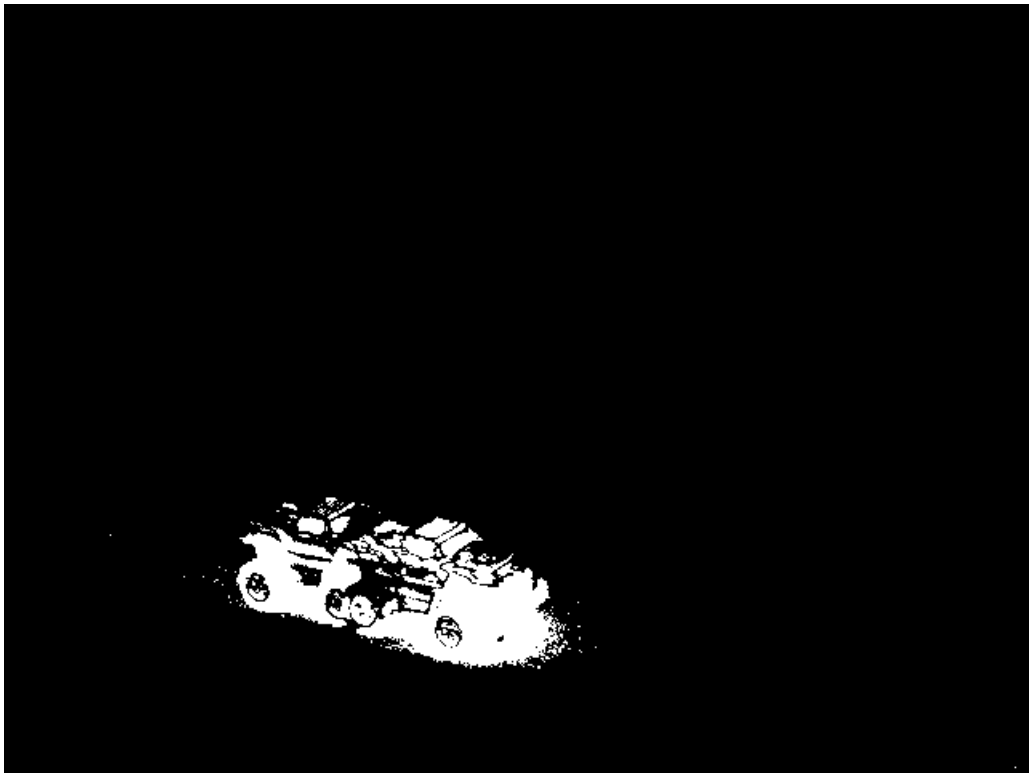
Modelování pozadí

Princip modelování pozadí spočívá ve vytvoření referenčního obrazu scény. Ten by měl obsahovat pouze pozadí scény, tedy ty části snímku, které se v čase nemění. Potom bychom na základě rozdílů aktuálního a referenčního snímku mohli lokalizovat případné pohybující se objekty. Pro modelování pozadí bylo vypracováno velké množství metod. Některé jsou jednoduché, zatímco jiné jsou poněkud složitější. K

otestování různých metod nám výborně poslouží knihovna BGSLibrary [11], která má implementováno více než 20 různých algoritmů na modelování pozadí. Pro testování použijeme jednoduchou scénu, kde jediný pohybující se objekt bude sledovaný robot.

Pro nejjednodušší metody, jako je například "vážená střední hodnota"nebo "dočasná střední hodnota"je výsledek prakticky totožný s metodou rozdílu snímků. Proto nemá smysl se těmito metodami hlouběji zabývat.

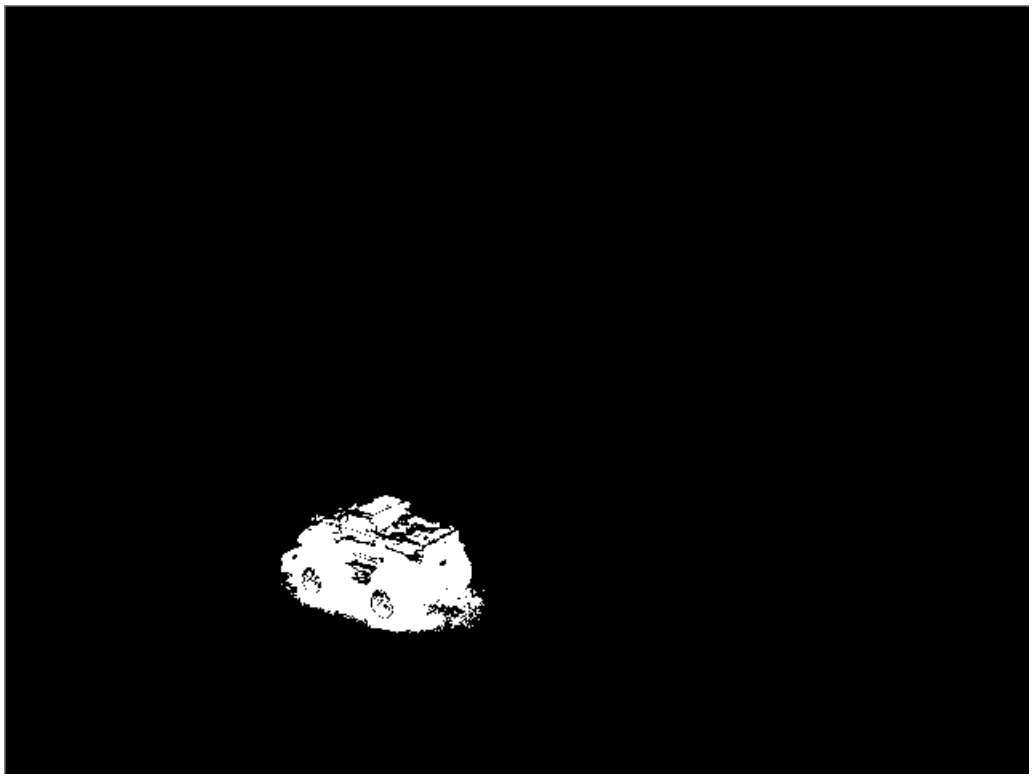
Potom existují mírně složitější metody, např. "adaptivní učení pozadí". Tyto metody modelují pozadí jednoduchým způsobem. Nejprve se jako pozadí označí první snímek. Následně se pozadí postupně "posouvá"směrem k aktuálnímu snímku. Výsledek tohoto přístupu zobrazuje obrázek 3.7. Tato metoda se hodí zejména pro sledování rychlých objektů, které příliš často neopakují svoji polohu. Při pomalejším pohybu je zřejmé, že objekt za sebou zanechává stopu. To je dáno pomalou aktualizací pozadí. Pokud bychom rychlost posouvání pozadí k aktuálnímu snímku zvětšovali, postupně bychom dostali jako referenční obraz poslední snímek - přešli bychom do modelu rozdílu sousedních snímků. Výhodou tohoto algoritmu je schopnost pozorovat sledovaný objekt ještě určitou dobu po zastavení pohybu. Na druhu stranu, když se objekt znovu začne pohybovat, je také chvíli pozorován na místě, kde předtím stál.



Obrázek 3.7: Metoda adaptivního pozadí

Posledním přístupem, který zmíníme je metoda modelování pozadí pomocí směsi gausiánů. Zde existuje mnoho různých algoritmů a modifikací. Princip je založen na tom, že hodnota intenzity každého pixelu je reprezentována jako směs několika Gaussových rozdělení. To je výhodné pokud se v obrazové sekvenci vyskytuje opakující se pohyb.

Výsledek je vidět na obrázku 3.8. Sledovaný objekt je detekován téměř celý. Není rozmazaný a nezobrazuje se ani stopa. Stejné výsledky jsou dosaženy nezávisle na rychlosti pohybu. Robot je také detekován ještě několik sekund po zastavení. Rychlost zpracování je sice nižší než u jednodušších metod, ale pořád je dostatečná i pro běh v reálném čase. Pokud se robot zastaví na delší dobu, zahrne se do pozadí. Pokud se v takovém případě začne znovu pohybovat, bude jej systém nějakou dobu detekovat jak v místě, kde stál, tak také v místě, kde se skutečně pohybuje.



Obrázek 3.8: Metoda směsi gausiánů

3.4 Shrnutí

Otestovali jsme řadu přístupů k detekci pohybujících se objektů v obraze. Některé jsou pro naši úlohu vhodné a jiné se hodí spíše pro jiné účely. Jako nejlepší se jeví využití pohybu v obraze. Hlavní výhodou oproti detekci významných bodů je nezá-

vislost na složitosti pozorované scény. Také rychlost zpracování je více než dostatečná pro běh v reálném čase. Pokud bude sledovaný robot jediný objekt, který se v obraze pohybuje, potom jsme schopni jednoznačně určit jeho polohu ve snímku. V případě více detekovaných objektů bude ještě nutné rozhodnout o tom, zda je mezi nimi hledaný robot.

Pro vyhodnocení polohy robota nepotřebujeme jeho celý obraz. Postačující je obrys, proto bude stačit jednoduchá metoda rozdílu snímků. Pokud se robot zastaví, není příliš velký problém to, že jej systém nebude detekovat. Můžeme předpokládat, že se pořád nachází na stejných souřadnicích. Velkou výhodou je také rychlost zpracování, protože bude potřeba šetřit výkon na další potřebné výpočty (triangulace, filtrace...).

Kapitola 4

Kalibrace systému a triangulace

Jestliže se nám podaří v obraze lokalizovat hledaného robota, dalším krokem bude výpočet jeho polohy v prostoru. K tomu bude zapotřebí informace minimálně ze dvou nezávislých zdrojů, protože kamery nám poskytují pouze 2D obraz. V této práci se omezíme pouze na dvě kamery. Ty bude třeba umístit tak, aby pokrývali co největší oblast a aby se jejich zorná pole co nejvíce překrývala. Nejjednodušší řešení je umístit kamery vedle sebe. Tím získáme stereovidění a můžeme určit hloubku. Nejprve je ale třeba systém kamer nakalibrovat.

4.1 Stereo kalibrace

Pro popsání vztahu, kterým se transformují souřadnice reálného světa do obrazu z kamer využijeme rovnici (4.1).

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A[R|t]M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{23} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.1}$$

kde:

- u, v jsou souřadnice bodu v obraze v pixelech
- A je matice vnitřních parametrů
- $[R|t]$ je matice vnějších parametrů
- f_x, f_y jsou ohniskové vzdálenosti
- c_x, c_y jsou souřadnice hlavního bodu v pixelech, obvykle je uprostřed obrazu
- R je rotační složka matice
- t je translační složka matice

Matice A je matice parametrů kamery, je nezávislá na pozorované scéně a proto zůstává konstantní, dokud nedojde ke změně ohniskové vzdálenosti (v případě zoomu). Matice $[R|t]$ slouží k transformaci reálných souřadnic do koordinačního systému vzhledem ke kameře.

K získání těchto matic nám opět velmi dobře poslouží knihovna OpenCV, konkrétně použijeme metody calibrateCamera. Ta na základě několika snímků známého objektu provede odhad potřebných parametrů. Pro kalibraci se využívá tzv. kalibrační vzor. V našem případě to bude šachovnice velikosti papíru A4.

Kamery umístíme jen pár centimetrů od sebe, aby se jejich obrazy co nejvíce překrývali. Pro zjištění vzájemné polohy kamer slouží metoda stereoCalibrate. Ta dokáže také odhadnout matice vnějších a vnitřních parametrů pro obě kamery. Vzhledem k velkému počtu odhadovaných parametrů se ale z důvodu větší přesnosti doporučuje provést nejprve kalibraci pro každou kameru zvlášť. Potom je třeba získat obrazy kalibrační šachovnice z obou kamer. Pro správnou kalibraci je potřeba 10-20 párů snímků. Důležitá je také různorodost snímků. Šachovnici je třeba umístit do různých částí obrazu a také ji různě natočit. Musí ale být vždy vidět celá v obou snímcích.

Na závěr kalibračního procesu je ještě potřeba zavolat funkci stereoRectify. Ta na základě parametrů poskytnutých metodou stereoCalibrate vytvoří projekční matice pro obě kamery, které budou zapotřebí pro následnou triangulaci. V případě horizontálního sterea mají tvar (4.2),

$$P1 = \begin{bmatrix} f & 0 & cx_1 & 0 \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} P2 = \begin{bmatrix} f & 0 & cx_2 & T_x * f \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.2}$$

kde T_x je horizontální posun mezi kamerami.

Po nakalibrování kamerového systému již nesmíme s žádnou kamerou pohnout. Pokud se změní poloha některé z nich, přestanou zjištěné projekční matice platit a bude třeba provést celý kalibrační proces znovu.

4.2 Triangulace

Úloha triangulace spočívá ve zjištění prostorových souřadnic určitého objektu. Máme-li projekční souřadnice objektu v obrázku z kamery, můžeme tvrdit, že pozorovaný objekt leží na dané epipolární přímce. V případě dvou kamer máme dvě přímky. Náš objekt musí ležet na každé z nich. Z toho vyplývá, že naší úlohou bude nalézt jejich průsečík v prostoru. To se může jevit jako jednoduchá úloha, ale vzhledem k různým poruchám a šumu se tyto dvě přímky pravděpodobně nikdy neprotnou. Problémem triangulace tedy bude nalézt takové řešení, které bude minimalizovat chybu.

Ačkoliv knihovna OpenCV obsahuje také metodu pro triangulaci, její přesnost není příliš dobrá. Proto se zaměříme na jinou metodu, konkrétně na iterativní lineární triangulaci. Ta je popsána v [12]. Nejprve vyjdeme z lineární triangulace. Začneme rovnicí (4.3),

$$w(u, v, 1)^T = Px \tag{4.3}$$

kde w je neznámý faktor měřítka, $(u, v, 1)^T$ jsou homogenní souřadnice bodu v obraze, P je projekční matice a x jsou souřadnice v reálném prostoru. Nyní pokud označíme $i-$tý řádek matice P jako P_i, získáme rovnice (4.4).

$$wu = P_1^T x, \qquad wv = P_2^T x, \qquad w = P_3^T x \tag{4.4}$$

Nyní použijeme k eliminaci neznámé w třetí rovnici a získáme (4.5).

$$uP_3^T x = P_1^T x, \qquad vP_3^T x = P_2^T x \tag{4.5}$$

Ze dvou kamer získáme čtyři lineární rovnice, které můžeme psát ve tvaru $Ax = 0$. Tyto rovnice definují x, ale pro zašuměná data nemají přesné řešení. Proto hledáme nejlepší možné řešení, např. ve smyslu metody nejmenších čtverců. Za předpokladu, že hledaný bod neleží v nekonečnu, můžeme zavést $x = (x, y, z, 1)^T$. Tím dostaneme soustavu čtyř rovnic pro tři neznámé. Poté pomocí metody nejmenších čtverců můžeme najít nejlepší řešení.

Lineární triangulace samostatně není příliš přesná. Chybu můžeme snížit zavedením iterativního přístupu. Myšlenkou iterativní lineární triangulace je změnit váhy jednotlivých rovnic tak, aby rovnice korespondovaly s chybou měření souřadnic v obraze.

Uvažujme první z rovnic (4.5). Obecně pro získané x tato rovnice nebude splněna a bude obsahovat chybu (4.6).

$$\epsilon = uP_3^T x - P_1^T x \tag{4.6}$$

My ale chceme minimalizovat odchylku změřené souřadnice u a projekce x, která má tvar (4.7).

$$\epsilon' = u - P_1^T x / P_3^T x \tag{4.7}$$

Vydělením pravé strany rovnice (4.6) výrazem $P_3^T x$ lze přejít k rovnici (4.7). Protože $P_3^T x = w$, budeme všechny rovnice váži výrazem $1/w$. Samozřejmě nemůžeme hned takto vážit rovnice, protože neznáme x, dokud nevyřešíme soustavu rovnic. Proto budeme postupovat iterativně. Nejprve zvolíme všechny váhy $w_i = 0$. Tím získáme první řešení, které odpovídá jednoduché lineární triangulaci. Následně můžeme přepočítat váhy na základě získaného řešení a tento proces dále opakujeme. Iterační cyklus se zastaví, pokud se váhy již výrazně nemění. Po několika iteracích by mělo řešení konvergovat.

Hlavní výhodou tohoto algoritmu je jeho jednoduchost. S tím je spojená jednoduchá implementace a také vysoká rychlost výpočtu. I v případě iterativní metody je zpracování dostatečně rychlé pro použití v aplikaci, která musí běžet v reálném čase.

Kapitola 5

Úloha filtrace

Mohlo by se zdát, že pokud jsme schopni najít sledovaného robota v obraze a triangulací získat jeho prostorové souřadnice, problém je vyřešen. Pokud by vše fungovalo naprosto přesně, potom by to byla pravda. Získané souřadnice ale nemusí zcela odpovídat skutečnosti. V průběhu celého procesu se do měření zanáší chyby a výsledek tak odpovídá pouze přibližně. Díky filtraci je možné přesnost výsledků výrazně zvýšit.

Pro objasnění filtrace vyjdeme z [13].Úloha filtrace bývá také nazývána úlohou odhadu (estimace) systému, protože se snažíme odhadnout, jak se systém bude chovat. Základem pro následnou estimaci je model systému. Například v našem případě není možné, aby se sledovaný robot přemístil okamžitě z jednoho rohu do druhého. Proto jeho následující polohu budeme očekávat v blízkosti poslední známé polohy. Konkrétní údaje nám poskytne právě model systému.

5.1 Strukturální modelování

Ve strukturálním přístupu se využívá struktury systému, tedy vztahů mezi veličinami. Zavádí se proměnná x_k, která označuje stav systému v kroku k. V našem případě bude stav systému reprezentovat souřadnice robota v rovině a také rychlost jeho pohybu. Dimenze vektoru stavu tak bude čtyři. Přechod z jednoho stavu do druhého lze popsat rovnicí (5.1),

$$x_{k+1} = f(x_k) + w_k \tag{5.1}$$

kde stav v následujícím kroku je popsán funkcí stavu v aktuálním kroku f za přítomnosti šumu w_k. Dále měření může být popsáno rovnicí (5.2),

$$z_k = h(x_k) + v_k \tag{5.2}$$

kde z_k je vektor měření, h je známá vektorová funkce a v_k je šum měření.

V této práci se omezíme na problém lineární estimace stavu. Náš systém sice vykazuje jistou nelinearitu, ale i lineární estimátor by mohl být dostatečně přesný pro odhad polohy robota. V tomto zjednodušeném případě tak přejdou uvedené rovnice

do tvaru (5.3),

$$x_{k+1} = Fx_k + w_k \qquad z_k = Hx_k + v_k \tag{5.3}$$

kde F a H jsou matice příslušných dimenzí.

Dále je nutné poznamenat, že šum zde působí aditivně. Tento přístup by bylo možné zobecnit a zavést šum i do funkcí f a h. Toto zobecnění by nám ale výrazně ztížilo úlohu odhadu.

5.2 Pravděpodobnostní modelování

Pravděpodobnostní modelování se využívá pro specifikaci poruch, tedy šumu měření a stavového šumu. Bylo by by žádoucí, aby šum nevykazoval žádné možnosti predikce. Takový proces se nazývá bílý šum. Pokud budeme chápat stav v tradičním smyslu, tak stav x_k musí obsahovat veškerou informaci o minulosti systému do času t_k, která je zapotřebí k určení dalšího vývoje systému. Proto šum w_k nesmí vykazovat žádnou závislost do minulosti. Musí se tedy jednat o bílý šum.

Úlohou estimace stavu je tedy na základě měření odhadnout stav systému x_k. K řešení tohoto problému existují různé přístupy. My se nyní zaměříme na Bayesovský přístup.

5.3 Bayesovský přístup

Nejprve formulujeme obecné řešení problému odhadu. Nechť vektor stavu se vyvíjí podle následujícího vztahu:

$$x_{k+1} = f(x_k) + w_k \tag{5.4}$$

kde x_k je n_x dimenzionální vektor stavu systému v čase t_k a w_k je n_x dimenzionální stavový šum působící na sytém v čase t, kde $t_k \leq t < t_{k+1}$ a f_k je známá vektorová funkce příslušné dimenze. Náhodný proces w_k je bílý šum se známou hustotou pravděpodobnosti $p(w_k)$ a známe také hustotu pravděpodobnosti počátečního stavu $p(x_0)$. Stav systému je sledován pomocí měřených hodnot z_k, které jsou ve známém vztahu k x_k, ale obsahují také šum:

$$z_k = h(x_k) + v_k \tag{5.5}$$

kde z_k je n_z dimenzionální vektor známých měřených dat v čase t_k a v_k je n_z dimenzionální vektor šumu měření ovlivňující data v čase t_k. Náhodný proces v_k je opět bílý šum se známou hustotou pravděpodobnosti $p(v_k)$. Procesy w_k, v_k a náhodná veličina x_0 jsou navzájem nezávislé.

Pro použití Bayesovksých vztahů je nezbytné zavést čas, ve kterém bude probíhat odhad v závislosti na měření. Cílem je určení podmíněné hustoty pravděpodobnosti $p(x_k|z^l)$. Mohou nastat tři případy:

- pro $l > k$ se úloha nazývá vyhlazování

- pro $l = k$ se úloha nazývá filtrace
- pro $l < k$ se úloha nazývá predikce

My se budeme věnovat pouze filtraci a jednokrokové predikci. Potom lze užitím Bayesova pravidla získat aposteriorní hustotu pravděpodobnosti nebo také filtrační hustotu pravděpodobnosti rekurzivně:

$$p(x_k|z^k) = \frac{p(z_k|x_k)p(x_k|z^{k-1})}{p(z_k|z^{k-1})} \tag{5.6}$$

kde $p(z_k|z^{k-1}) = \int p(z_k|x_k)p(x_k|z^{k-1})dx_k$ je normalizační konstanta.

Prediktivní hustota pravděpodobnosti je určena vztahem:

$$p(x_k|z^{k-1}) = \int p(x_k|x_{k-1})p(x_{k-1}|z^{k-1})dx_{k-1} \tag{5.7}$$

5.4 Kalmanův filtr

Kalmanův filtr je chápán jako rekurzivní algoritmus generující lineární, nestranný odhad ve smyslu podmínění střední hodnoty (minimální variance) neznámého stavu dynamického systému ze zašuměných dat získávaných v diskrétních časových okamžicích. Princip fungování algoritmu je popsán například v [14]. Filtr odhadne stav systému v určitém čase a v dalším kroku opraví tento odhad podle naměřených (zašuměných) hodnot. Rovnice Kalmanova filtru lze rozdělit do dvou kroků. První jsou rovnice týkající se predikce na základě modelu (prediktivní rovnice):

$$\hat{x}_k^{'} = F\hat{x}_{k-1} \tag{5.8}$$

$$P_k^{'} = FP_{k-1}F^T + Q \tag{5.9}$$

kde $\hat{x}_k^{'}$ je střední hodnota apriorního odhadu, $P_k^{'}$ je kovariance apriorního odhadu a Q je kovariance stavového šumu. Předpokládáme, že střední hodnota šumu je nulová.

Druhý krok představují rovnice pro získání aposteriorního odhadu na základě měření (filtrační rovnice):

$$K_k = P_k^{'}H^T(HP_k^{'}H^T + R)^{-1} \tag{5.10}$$

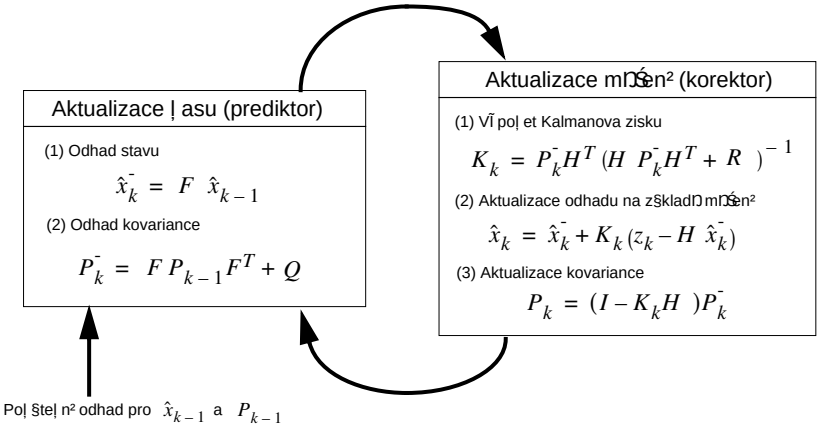
$$\hat{x}_k = \hat{x}_k^{'} + K_k(z_k - H\hat{x}_k^{'}) \tag{5.11}$$

$$P_k = (I - K_kH)P_k^{'} \tag{5.12}$$

kde koeficient K_k je Kalmanův zisk, R je kovariance šumu měření (opět předpokládáme nulovou střední hodnotu šumu), $\hat{x}_k$ je střední hodnota aposteriorního odhadu, P_k je kovariance aposteriorního odhadu a I je identická matice.

Prediktivní rovnice nám podávají apriorní informaci o stavu systému v následujícím kroku. Filtrační rovnice opravují apriorní odhad na základě naměřených hodnot a

poskytují nám vylepšený aposteriorní odhad. Na rovnice pro výpočet prediktivní hustoty pravděpodobnosti může být nahlíženo také jako na tzv. *prediktor* a na rovnice pro výpočet filtrační hustoty pravděpodobnosti zase jako tzv. *korektor*. Celý algoritmus tedy odpovídá struktuře *prediktor-korektor*. Iterativní princip fungování Kalmanova filtru demonstruje obrázek 5.1.



Obrázek 5.1: Diagram Kalmanova filtru

Předpokladem pro použití Kalmanova filtru je linearita a gaussovost systému. To náš systém splňuje pouze částečně. Pro nelineární systémy lze použít rozšířený Kalmanův filtr. My se však zatím omezíme pouze na klasický Kalmanův filtr.

Realizace Kalmanova filtru

K realizaci Kalmanova filtru můžeme opět využít knihovnu OpenCV. Ta má filtr implementován, takže nám stačí pouze zadat potřebné parametry modelu. Stav systému budeme definovat jako uspořádanou čtveřici hodnot $[x, y, v_x, v_y]$, kde x a y jsou souřadnice polohy a v_x a v_y jsou rychlosti pohybu ve směru x a y. Pro lineární model bude mít matice přechodu systému mezi stavy tvar:

$$F = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.13}$$

kde dt je perioda vzorkování. Měřit budeme pouze polohu, proto matice měření bude mít tvar:

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{5.14}$$

Kapitola 6

Výsledky

V této části práce zhodnotíme výsledky navrhované přístupu ke sledování pohybujících se robotů. Hodnotit budeme především přesnost odhadu polohy. Otestujeme také vliv změny rozlišení kamer nebo změny jejich vzájemné pozice. Pro každou konfiguraci budeme také zjišťovat, do jaké vzdálenosti je měření ještě přesné.

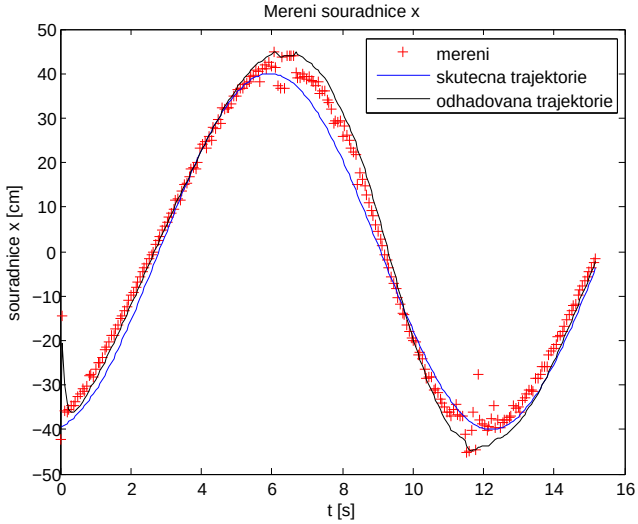
6.1 Měření přesnosti

Pro měření přesnosti systému je třeba co nejpřesněji znát skutečnou trajektorii sledovaného objektu. K tomuto účelu se výborně hodí například malý vláček na kolejích. Koleje mají předem známou konfiguraci, takže naměřené hodnoty budeme mít s čím porovnat. Pro naše měření použijeme kruhové koleje. Průměr kruhu bude 80 cm a střed umístíme do vzdálenosti přibližně 10 cm od kamer. Výslednou polohu vždy převedeme na souřadnicový systém vzhledem ke kolejím, kde střed kruhu bude považován za počátek soustavy souřadnic.

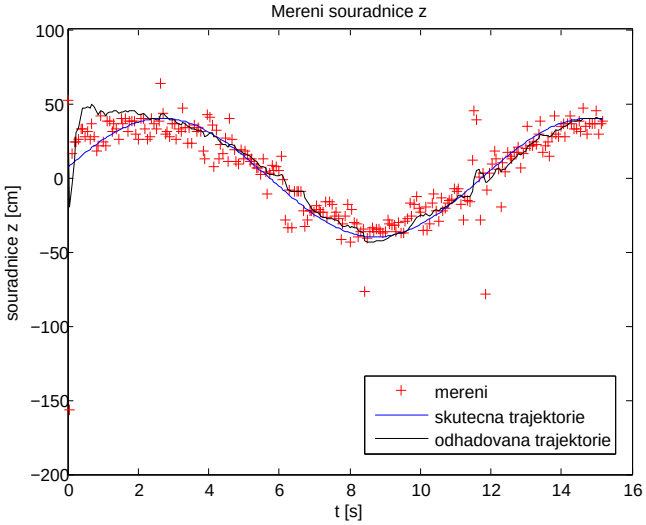
Toto měření provedeme pro čtyři různé konfigurace kamer. Budeme testovat rozlišení 640x480 bodů a také širokoúhlé 1280x720. Při obou rozlišeních vyzkoušíme případy, kdy kamery budou vzdálené 10 cm a pak také 30 cm od sebe.

Konfigurace 1

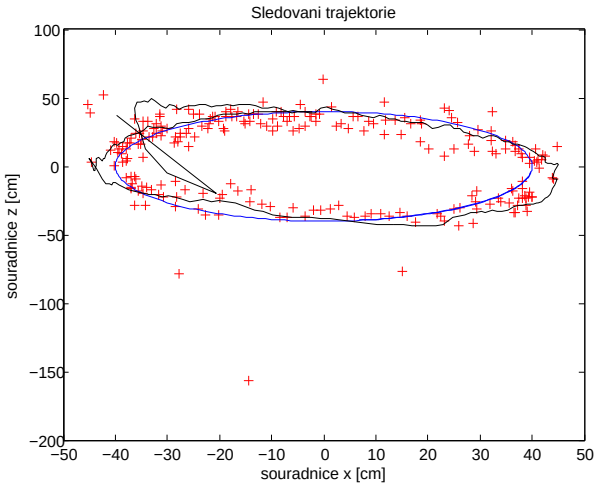
Nejprve budeme testovat kamery v rozlišení 640x480 umístěné 10 cm od sebe. Výsledky jsou vidět na obrázcích 6.1, 6.2 a 6.3. Na první pohled je zřejmé, že přesnější je měření souřadnice x. To je pochopitelné, protože chyby v detekci objektu a triangulaci se mnohem více přenášejí na měření vzdálenosti. Průměrná chyba pro x činí 3,6 cm a pro z je to 5.8 cm. Problémové oblasti jsou začátek sledování a potom momenty, kdy se pozorovaný objekt pohybuje směrem ke kamerám. V tomto případě ve snímcích nedochází k výrazným rozdílům a sledovaný objekt je detekován pouze částečně.



Obrázek 6.1: Měření souřadnice x při SD 10cm



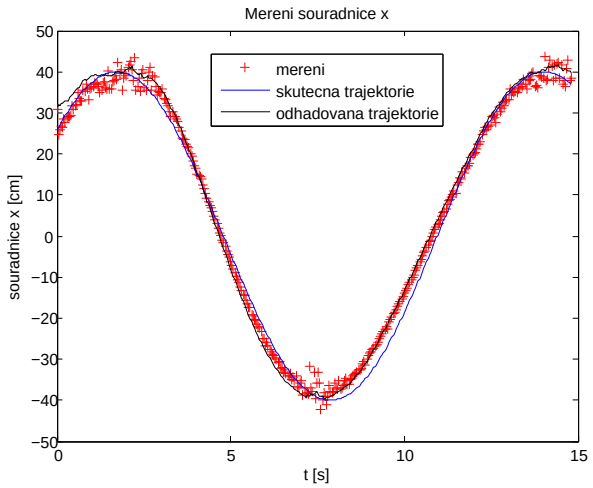
Obrázek 6.2: Měření souřadnice z při SD 10cm



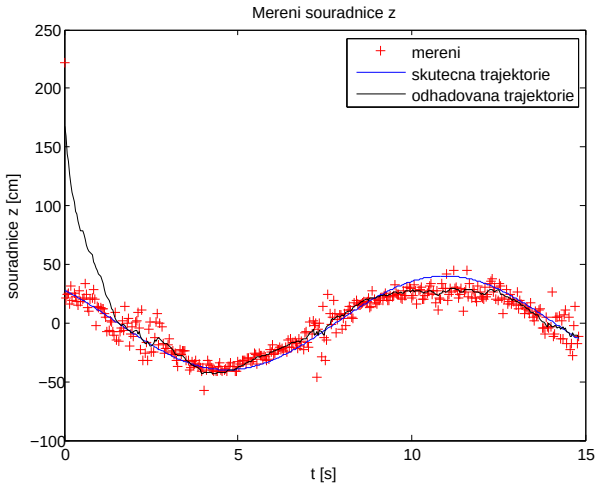
Obrázek 6.3: Výsledná trajektorie

Konfigurace 2

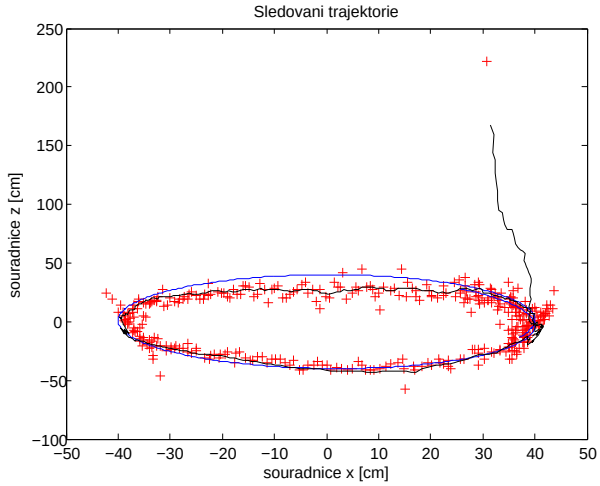
Jakou druhou vyzkoušíme změnu rozlišení na 1280x720, vzdálenost mezi kamerami zůstane 10 cm. Výsledky zobrazují obrázky 6.4, 6.5 a 6.6. Je zřejmé, že zvýšení rozlišení přispělo ke zvýšení přesnosti měření. Chyba pro x je přibližně 2.5 cm a pro z 5 cm. Problémové oblasti opět zůstávají stejné, ale při HD rozlišení je objekt detekován i při pohybu ke kameře skoro celý.



Obrázek 6.4: Měření souřadnice x při HD 10cm



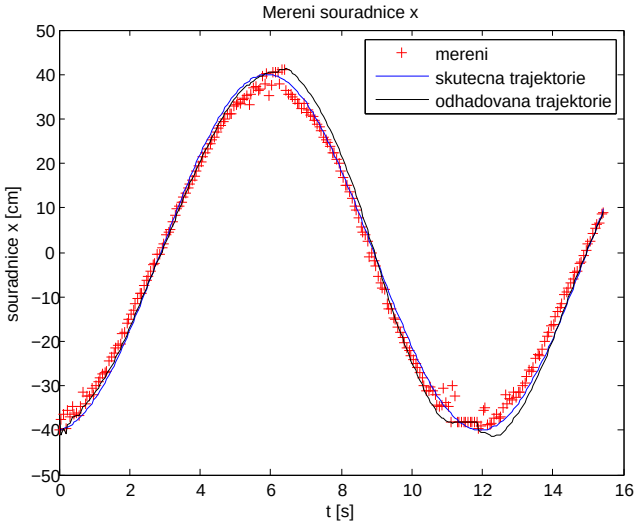
Obrázek 6.5: Měření souřadnice z při HD 10cm



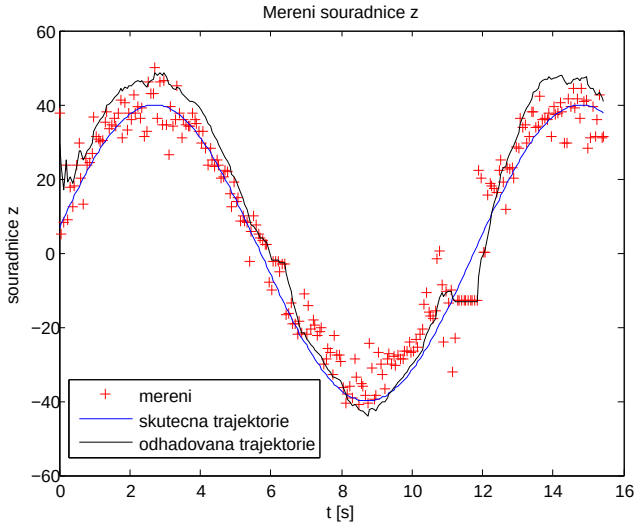
Obrázek 6.6: Výsledná trajektorie

Konfigurace 3

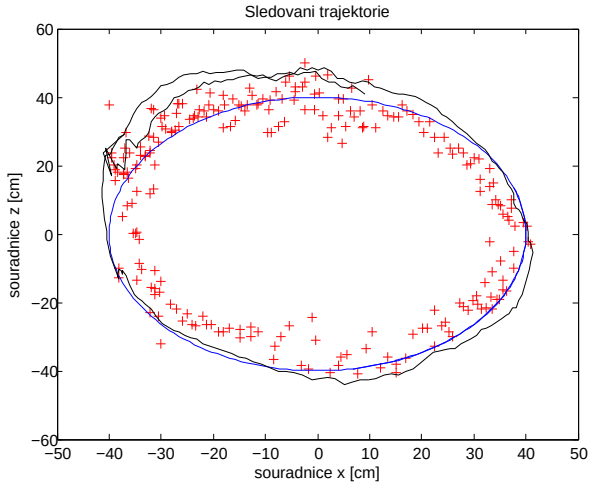
V této části otestujeme vliv změny vzdálenosti mezi kamerami. Rozlišení nastavíme na 640x480 a kamery umístíme do vzdálenosti 30 cm od sebe. Při této vzdálenosti by systém měl být více robustní při triangulaci. Výsledky zobrazují obrázky 6.7, 6.8 a 6.9. Opět nastalo jisté zlepšení. Při měření souřadnice z se již nevyskytují zcela chybné hodnoty. Průměrná chyba je v případě x 1,5 cm a pro z 4.6 cm.



Obrázek 6.7: Měření souřadnice x při SD 30cm



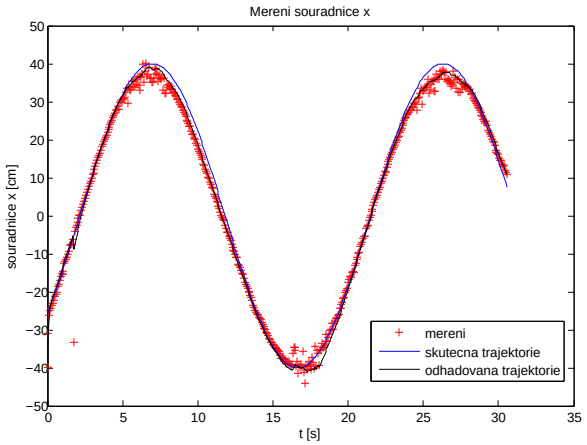
Obrázek 6.8: Měření souřadnice z při SD 30cm



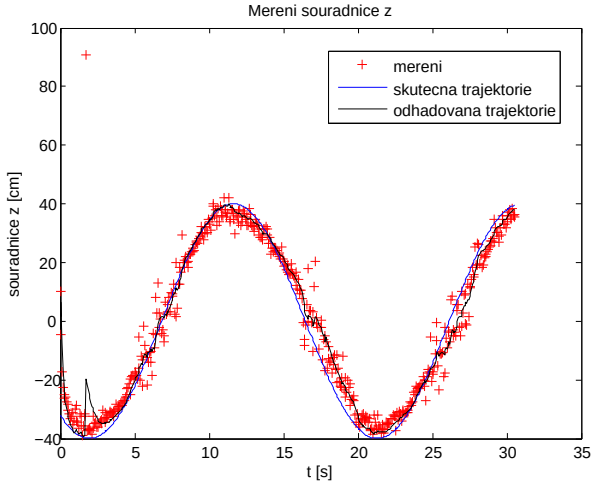
Obrázek 6.9: Výsledná trajektorie

Konfigurace 4

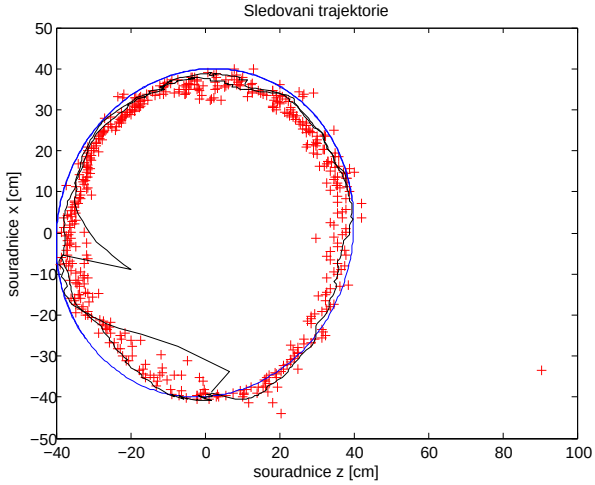
Posledními parametry bude rozlišení 1280x720 při vzdálenosti kamer 30 cm. Výsledky jsou na obrázcích 6.10, 6.11 a 6.12. Podle očekávání bylo dosaženo největší přesnosti. Pro souřadnici x je průměrná chyba 1,5 cm a pro z je to 3,2 cm. Jako problémové oblasti se opět projevuje začátek sledování a také pohyb objektu ke kamerám.



Obrázek 6.10: Měření souřadnice x při HD 30cm



Obrázek 6.11: Měření souřadnice z při HD 30cm



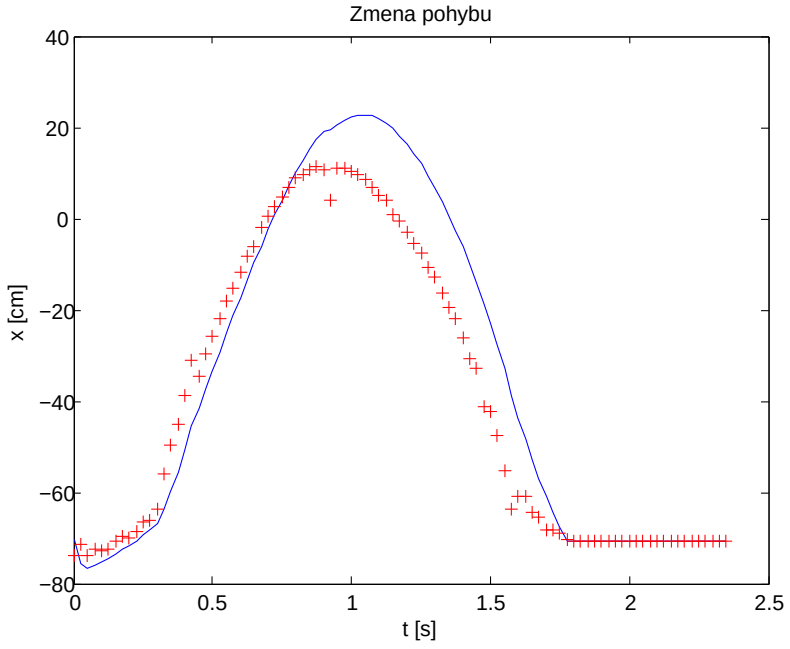
Obrázek 6.12: Výsledná trajektorie

6.2 Měření nelineárního pohybu

V této části se zaměříme na schopnost systému sledovat nelineární změny pohybu. Pro filtrování naměřených dat používáme Kalmanův filtr a systém máme popsaný lineárním modelem. Proto nejhorší výsledky můžeme očekávat v případě rychlých změn pohybu.

Rychlá změna směru

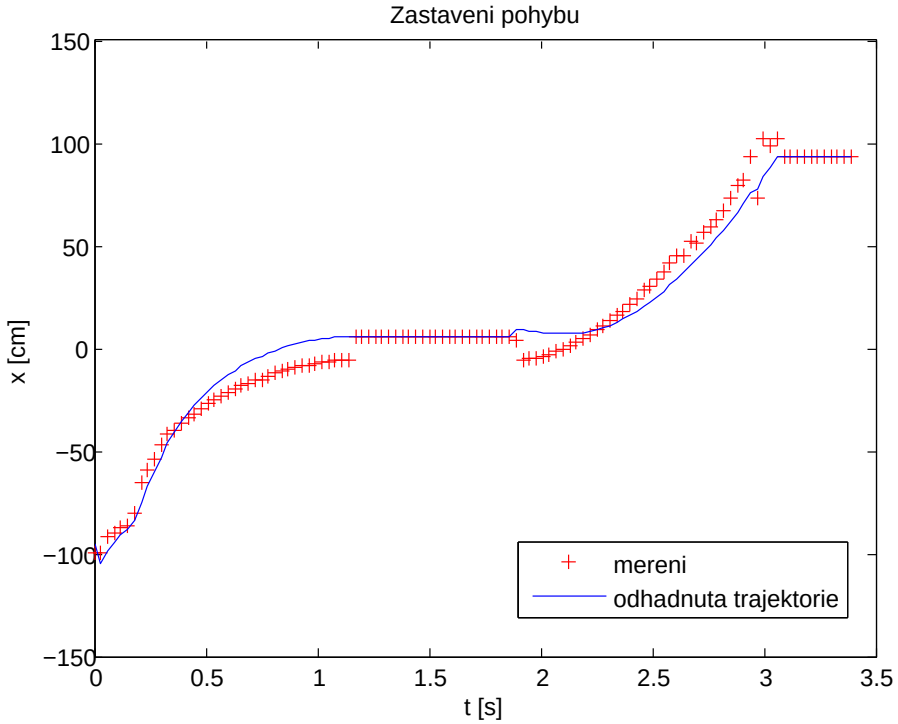
V tomto testu vyzkoušíme, jak je systém schopný reagovat na rychlé změny pohybu. Pro jednoduchost se omezíme pouze na souřadnici x. Sledovaný robot se bude pohybovat zleva doprava, před kamerou se zastaví a okamžitě se začne pohybovat zpět. Při tomto testu nemáme k dispozici skutečnou trajektorii, ale ta není potřeba. Zkoumáme totiž rozdíl naměřených hodnot a výsledku získaného filtrací. Výsledky zobrazuje obrázek 6.13. Z naměřených hodnot je patrné, kde bude ležet skutečná poloha. Program nám ale poskytuje poněkud jiné výsledky. To je podle očekávání, protože to odpovídá zadanému lineárnímu modelu. Maximální chyba je asi 10 cm. Navíc se tato chyba projevuje ještě relativně dlouhou dobu, i když v menší míře. Při rychlých změnách směru pohybu tedy není tento přístup vhodný.



Obrázek 6.13: Trajektorie při rychlé změně pohybu

Zastavení pohybu

Dalším problémem by mohlo být zastavení sledovaného objektu. Vzhledem ke způsobu detekce (rozdíl snímků) nebude mít systém v případě zastavení pohybu žádné údaje. V tomto případě budeme za polohu robota považovat poslední známou souřadnici. Opět se zaměříme pouze na souřadnici x. Sledovaný robot se znovu bude pohybovat zleva doprava, před kamerou se asi na jednu sekundu zastaví a potom bude pokračovat dál. Výsledky zobrazuje obrázek 6.14. Zde je vidět, že při zastavení pohybu je za změřenou souřadnici považována poslední známá poloha. Mohlo by se zdát, že bude výhodnější vzít do úvahy spíše poslední změřenou polohu, ale pokud bychom zrovna naměřili chybnou hodnotu, nastal by velký problém. Proto je jistější vycházet z filtrované hodnoty. Chyba byla v tomto případě necelých 7 cm, což při rozměrech robota 25cm x 20cm x 20cm je ještě přijatelné.



Obrázek 6.14: Trajektorie při zastavení pohybu

6.3 Měření dosahu

V této části budeme zkoumat maximální možnou vzdálenost od kamer. Testovat budeme opět stejné konfigurace kamer jako v první části. Naměřené vzdálenosti udává následující tabulka:

HD 30cm	
Změřená vzdálenost [cm]	Skutečná vzdálenost [cm]
290	300
320	350
360	400
HD 10cm	
Změřená vzdálenost [cm]	Skutečná vzdálenost [cm]
100	100
130	150
160	200
SD 30cm	
Změřená vzdálenost [cm]	Skutečná vzdálenost [cm]
100	100
140	150
160	200
SD 10cm	
Změřená vzdálenost [cm]	Skutečná vzdálenost [cm]
50	50
90	100
120	150

Pro každou konfiguraci lze v podstatě vymezit tři pásma. V blízkosti kamery jsou naměřené údaje přesné. Potom následuje pásmo zkreslených hodnot, které ale při nakalibrování systému můžeme ještě použít. Nakonec začnou hodnoty příliš kolísat a vzdálenost je prakticky neměřitelná.

Pro sledování pohybujících se robotů je z tohoto důvodu prakticky použitelná pouze kombinace HD rozlišení a vzdálenosti kamer 30 cm, která bezpečně funguje do 3 metrů.

6.4 Rychlost běhu

Velkou výhodou zkoumaného přístupu je jeho rychlost. Při rozlišení kamer 640x480 byla rychlost výpočtu v průměru 21 snímků za sekundu. Pro rozlišení 1280x720 byla rychlost o něco nižší, konkrétně 18 snímků za sekundu. Rozdíl v rychlosti tedy není příliš velký, přičemž i v rozlišení 720p je systém schopný sledovat robota v reálném čase.

Kapitola 7

Závěr

Navrhovaný algoritmus odhadu polohy pohybujících se objektů je založený na detekci pohybu v obraze na základě rozdílu následujících snímků. Triangulací jsou získány prostorové souřadnice, které jsou dále filtrovány pomocí Kalmanova filtru. Tento postup je vhodný pro sledování plynulejších pohybů, protože jedním z předpokladů pro použití Kalmanova filtru je linearita. V takovém případě je systém schopný sledovat daný objekt s chybou maximálně několik cm. Největším problémem se jeví rychlé změny směru pohybu, kdy může chyba být až 10 cm.

Tento postup je tedy použitelný při určitých omezeních. Sledovaný robot by neměl příliš často prudce měnit směr. Dalším omezením je schopnost sledovat pouze jeden objekt. Pokud se v obraze bude pohybovat více objektů, systém bude sledovat ten, který se nachází nejblíže u země. Dalším omezením je dosah měření pouze do vzdálenosti 3 metry. Odstraněním těchto omezení se bude zabývat navazující diplomová práce.

Literatura

[1] Riyad A. El-Iaithy, Jidong Huang, Michael Yeh, *Study on the use of Microsoft Kinect for robotics applications*, 2012

[2] Ben Fry, Casey Reas, *Processing*, Dostupné online:
<http://www.processing.org/>

[3] *OpenNI library for Processing*, Dostupné online:
<http://code.google.com/p/simple-openni/>

[4] *OpenCV Open Source Computer Vision*, 2013, Dostupné online:
<http://opencv.org/>

[5] D. J. Barnes, *Positive Builder*, 2011, Dostupné online:
<https://code.google.com/p/opencv-haar-cascade-positive-image-builder/>

[6] Matas et al, *Robust Wide Baseline Stereo from Maximally Stable Extremal Regions*, 2002, Dostupné na online:
<http://cmp.felk.cvut.cz/~matas/papers/matas-bmvc02.pdf>

[7] John F. Canny, *A Computational Approach to Edge Detection*, 1986

[8] Herbert Bay et al, *SURF: Speeded Up Robust Features*, 2006, Dostupné online:
<http://www.vision.ee.ethz.ch/~surf/eccv06.pdf>

[9] Edward Rosten and Tom Drummond , *Machine learning for high-speed corner detection*, 2006, Dostupné online:
<http://www.edwardrosten.com/work/rosten_2006_machine.pdf>

[10] Edward Rosten and Tom Drummond, *Fusing points and lines for high performance tracking*, 2005, Dostupné online:
<http://www.edwardrosten.com/work/rosten_2005_tracking.pdf>

[11] Sobral, A.C., *BGSLibrary: A OpenCV C++ Background Subtraction Library*, 2012, Dostupné online:

<http://code.google.com/p/bgslibrary/>

[12] Richard I. Hartley, Peter Sturm, *Triangulation*, 1994, dostupné online:

<http://users.cecs.anu.edu.au/~hartley/Papers/triangulation/triangulation.pdf>

[13] M. Šimandl, *Identifikace systémů a filtrace*, 1995

[14] Greg Welch a Gary Bishop, *An Introduction to the Kalman Filter*, 2001, dostupné online:

<http://www.cs.unc.edu/~tracker/media/pdf/SIGGRAPH2001_CoursePack_08.pdf>