Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

Diplomová práce

Metody odhadu polohy pohybujícího se objektu na základě obrazové informace

PLZEŇ, 2015 Lukáš Kiráľ

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: **2014/2015**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Lukáš KIRÁĽ**
Osobní číslo:	**A13N0162P**
Studijní program:	**N3918 Aplikované vědy a informatika**
Studijní obor:	**Kybernetika a řídicí technika**
Název tématu:	**Metody odhadu polohy pohybujícího se objektu na základě obrazové informace**
Zadávající katedra:	**Katedra kybernetiky**

Zásady pro vypracování:

1. Seznamte se se základními metodami odhadu stavu.
2. Použijte tyto metody pro odhad polohy objektu na základě obrazové informace.
3. Seznamte se prostředky pro obecné výpočty na GPU (GPGPU).

Rozsah grafických prací: **dle potřeby**
Rozsah pracovní zprávy: **40-50 stránek A4**
Forma zpracování diplomové práce: **tištěná**
Seznam odborné literatury:

Dodá vedoucí diplomové práce.

Vedoucí diplomové práce: **Ing. Miroslav Flídr, Ph.D.**
Katedra kybernetiky

Datum zadání diplomové práce: **1. října 2014**
Termín odevzdání diplomové práce: **15. května 2015**

Doc. RNDr. Miroslav Lávička, Ph.D.
děkan

L.S.

Západočeská univerzita v Plzni
Fakulta aplikovaných věd

Prof. Ing. Josef Psutka, CSc.
vedoucí katedry

P R O H L Á Š E N Í

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

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

V Plzni dne 25. 8. 2015

……

vlastnoruční podpis

Poděkování

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

Anotace

Tato práce se zabývá odhadem polohy pohybujícího se objektu v prostoru pomocí Kalmanova filtru, kde jsou jako měření použity údaje získané na základě obrazových dat a následné triangulace. V první části je uveden přehled možných metod detekce pohybu v obraze. Dále jsou využity dvě kamery pro triangulaci a získání polohy objektu v prostoru. Tyto souřadnice jsou potom použity jako měření pro Kalmanův filtr a rozšířený Kalmanův filtr. Nakonec jsou zhodnoceny výsledky navrhovaného postupu.

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

Abstract

This work deals with the estimation of a moving object position in space using the Kalman filter, where data based on image and subsequent triangulation are used as measurement. In first part, there is an overview of the possible methods of motion detection in the image. Then two cameras are used to triangulate and obtain the position of an object in space. These coordinates are used as measurement for the Kalman filter and the extended 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 diplomové práce je vytvořit systém pro lokalizaci robotů v místnosti. Konkrétně bude sledován robot s automatickým řídícím systémem. Tento systém bude mít k dispozici informaci o přibližné poloze robotu z jeho senzorů. Úkolem této práce je přidat objektivní informaci vnějšího pozorovatele. Výstupem tak budou souřadnice sledovaného robotu v rovině a případně také rychlost jeho pohybu.

Asi nejjednodušším řešením by bylo využití GPS. To 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 informací z kamerového systému.

Výsledný odhad polohy robotu musí splňovat určité požadavky. Nejdůležitější je přesnost odhadu. Pro navigaci v místnosti bude zapotřebí informaci o poloze s přesností na centimetry. Dalším požadavkem je rychlost běhu programu. Pro sledování pohybujícího se objektu v reálném čase bude potřeba zpracovat dostatečné množství snímků za sekundu. V případě pomalého běhu programu by byla informace o poloze robota příliš zastaralá. Vzhledem k tomu, že se z velké části jedná o úlohu zpracování obrazu, bude možné využít paralelního zpracování pomocí GPU. To bude také jedním z cílů práce, stejně jako otevřenost a multiplatformnost navrhovaného postupu.

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.

První část práce se bude zabývat výběrem hardwaru a softwaru. V další kapitole se otestují různé metody na detekci pohybu v obraze a jejich implementace. Ve čtvrté kapitole bude provedena kalibrace systému a triangulace. Tím budou získány prostorové souřadnice sledovaného objektu. Pátá kapitola bude zaměřena na filtraci naměřených údajů.

Kapitola 2

Použitý hardware a software

Ke sledování polohy pohybujícího se robotu lze využít mnoho různých přístupů. Tato část práce se bude zabývat výběrem vhodných prostředků. Ty lze rozdělit do 2 skupin. První skupinu tvoří hardware - kamery a další vybavení. Druhou částí je software - knihovny a ostatní programové prostředky.

2.1 Hardware

Použitý hardware lze opět rozdělit na 2 části. První část je tvořena senzory. Zde byly použity kamery. Druhou částí je výpočetní hardware. Základem bude samozřejmě PC. Kromě toho bude pro výpočty možné využít také řídicí jednotku Beaglebone.

Kamery

K určení souřadnic určitého objektu v prostoru je zapotřebí informace z více zdrojů, proto bude využito více kamer rozmístěných v místnosti. Konkrétně budou k dispozici 3 webkamery Logitech c525. Ty mohou pracovat až v rozlišení 720p při rychlosti snímání 30 snímků za sekundu, ale v této práci je použité rozlišení pouze 640x480 z důvodu rychlejšího zpracování. 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 bude zabývat jedna z následujících kapitol.

Řídicí jednotka

Vzhledem k tomu, že důležitým aspektem je také cena, budou otestovány případné možnosti využití řídicí jednotky Beaglebone pro účely této práce. V podstatě jde o alternativu ke konkurenčnímu Raspberry Pi s výkonnějším procesorem [1]. Konkrétně bude k dispozici BeagleBone Black rev C. Jedná se o levný, ale celkem výkonný malý počítač, který navíc disponuje GPU s podporou OpenGL ES 2.0. Kompletní specifikace je uvedena na stránkách výrobce [2]. Na Beaglebone je možné provo-

zovat operační systém na bázi linuxu, konkrétně byl zvolen systém Debian. V době psaní této práce byla aktuální verze 8 (Jessie).

2.2 Software

Ačkoliv je možné celou aplikaci začít psát od nuly, existuje celá řada již napsaných knihoven, které mohou programování velice usnadnit. Například Processing [3] nebo OpenCV [4], přičemž právě OpenCV je nejrozsáhlejší otevřenou knihovnou pro podporu počítačového vidění. Tato část práce představí vhodné programové vybavení, se kterým se bude dále pracovat.

OpenCV

Asi nejrozšířenější knihovnou v oblasti počítačového vidění je knihovna OpenCV. 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 účely této práce:

- core - The Core Functionality
  - obsahuje základní funkce pro manipulaci s obrazovými daty
- imgproc - Image Processing
  - obsahuje metody pro zpracování obrazových dat
- highgui - High Level GUI and Media
  - umožňuje ukládat, načítat a zobrazovat obrázky a video
- calib3d - Camera calibration and 3D reconstruction
  - obrahuje metody potřebné pro kalibraci kamerového systému a triangulaci
- feature2d - 2D Features framework
  - obsahuje některé detektory a deskriptory

Pro OpenCV je k dispozici také řada přídavných balíčků. Např. Intel © Threading Building Blocks (TBB) pro využití více-jádrových procesorů. Knihovna OpenCV tedy poskytuje vhodný základ pro vytvoření požadovaného systému pro sledování polohy pohybujícího se robotu v místnosti.

OpenGL

Knihovna OpenCV sice obsahuje také modul gpu, který umožňuje akceleraci výpočtů využitím grafické karty. Modul gpu je však zaměřen pouze na grafické karty podporující technologii CUDA. Proto bude ohledně akcelerace výpočtů pomocí GPU

využito jiné řešení. Konkrétně se bude jednat o OpenGL Shadery a jazyk GLSL (OpenGL Shading Language) [5]. Toto programové vybavení je sice primárně určeno pro zobrazování 3D grafiky na monitoru počítače, ale lze jej využít i pro jiné účely. Vzhledem k co největší volnosti nasazení výsledného systému se bude pracovat pouze se specifikací OpenGL ES 2.0 (OpenGL for Embedded Systems). Ta by měla pro účely této práce bohatě stačit a eventuálně by potom bylo možné konečné řešení provazovat i na mobilních zařízeních, podporujících OpenGL ES ve verzi 2.0 a vyšší.

Kapitola 3

Detekce objektu v obraze

Před vlastním odhadem polohy robotu 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. Pro sledování pohybujícího se objektu je vhodné hledat v obraze pohyb. Ten se projevuje změnami 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. Oba přístupy byly vyzkoušeny ke sledování vláčku na obrázku 3.1.

Obrázek 3.1: Sledovaný objekt

3.1 Rozdíl snímků

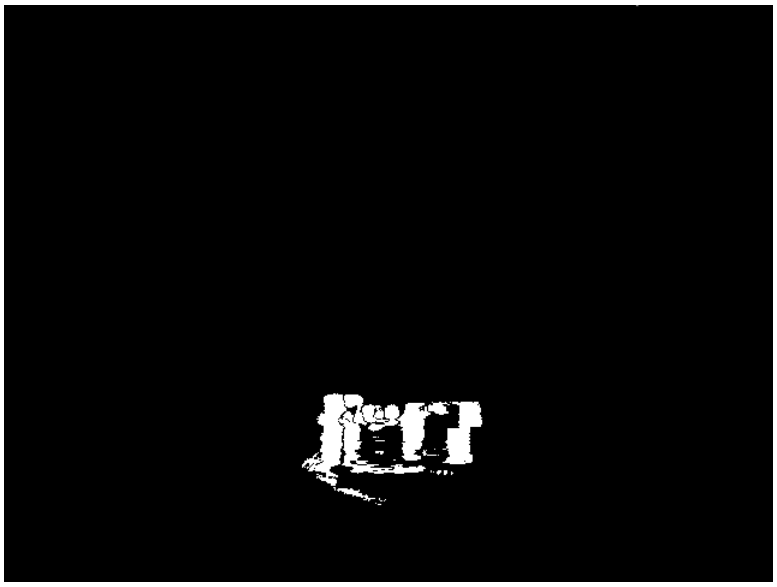
Nejjednodušší způsob detekce pohybu v obrazové sekvenci je porovnání dvou sousedních snímků. Případné rozdíly mezi nimi pravděpodobně indikují pohyb. Samozřejmě se může jednat také o změnu osvětlení nebo automatickou korekci kamery, ale tyto faktory se objevují spíše sporadicky a nemají dlouhodobější 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 je potřeba určit práh a pixely s menší hodnotou než práh obarvit černě a pixely s větší hodnotou obarvit bíle. Tento přístup je popsán rovnicí 3.1:

$$|P[F(t)] - P[F(t-1)]| > T, \tag{3.1}$$

kde $P[F(t)]$ je intenzita pixelu v aktuálním snímku, $P[F(t-1)]$ je intenzita pixelu v předchozím snímku a T je zvolený práh.

Ještě je vhodné použít erozní filtr na odstranění šumu. Výsledek je vidět na obrázku 3.2.



Obrázek 3.2: 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 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 omezení vznikne spíše kvůli snímkovací frekvenci 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 není detekován žádný objekt, budou se za aktuální polohu hledaného robotu považovat poslední známé souřadnice.

3.2 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 by bylo možné na základě rozdílu aktuálního a referenčního snímku lokalizovat případné pohybující se objekty podle rovnice 3.2:

$$|P[F(t)] - P[B(t)]| > T, \tag{3.2}$$

kde $P[F(t)]$ je intenzita pixelu v aktuálním snímku, $P[B(t)]$ je intenzita pixelu v pozadí a T je zvolený práh.

Model pozadí je potřeba v čase aktualizovat, protože nelze očekávat, že se celá scéna nebude během celého sledování vůbec měnit. Pro aktualizaci pozadí bylo vypracováno velké množství metod. Některé jsou jednoduché, zatímco jiné jsou poněkud složitější. K otestování různých metod výborně poslouží knihovna BGSLibrary [6], která má implementováno více než 20 různých algoritmů na modelování pozadí. Pro testování bude použita jednoduchá scéna na obrázku 3.1, kde jediný pohybující se předmět bude sledovaný objekt.

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, takže je tvořeno průměrem posledních snímků podle rovnice 3.3:

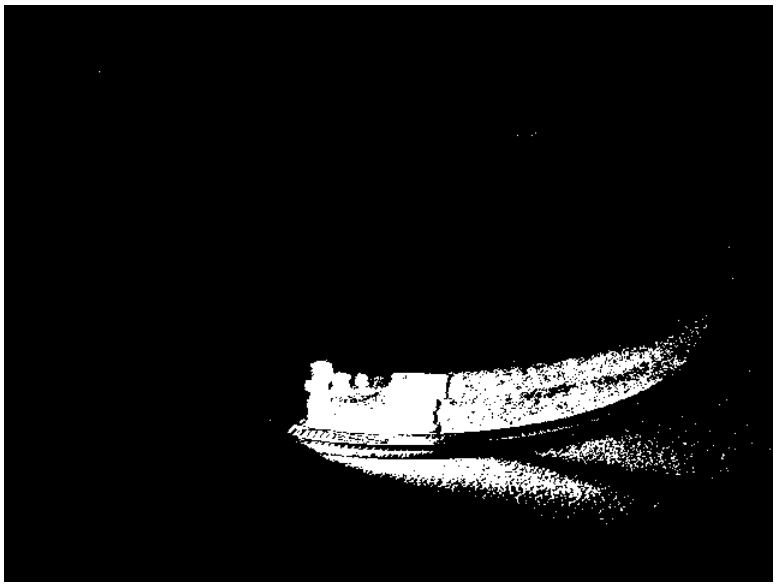
$$B(t) = \frac{1}{N} \sum_{i=1}^{N} F(t-i), \tag{3.3}$$

kde $B(t)$ je model pozadí, $F(t)$ je aktuální snímek a N je počet snímků, ze kterých se pozadí tvoří.

Do pozadí se tak částečně dostane i sledovaný objekt. Výsledek tohoto přístupu zobrazuje obrázek 3.3.

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 tím, že pozadí samotné obsahuje stopu objektu. 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 3.3: Metoda adaptivního pozadí

Dalším přístupem je metoda modelování pozadí pomocí směsi gausiánů. Zde existuje mnoho různých algoritmů a modifikací. Tato metoda je popsána například v [7]. Princip je založen na tom, že hodnota intenzity každého pixelu je reprezentována jako směs několika Gaussových rozdělení podle rovnice 3.4:

$$P(X_t) = \sum_{i=1}^{K} \omega_{i,t} \eta(X_t, \mu, \sigma), \tag{3.4}$$

kde $P(X_t)$ je intenzita pixelu, K je počet směsí gausiánů, přičemž se vzhledem k výpočetním nárokům používá 3 až 5 těchto směsí, $\omega_{i,t}$ je váha dané směsi a $\eta(X_t, \mu, \sigma)$ je distribuční funkce normálního rozdělení pravděpodobnosti.

Pozadí je potom určeno prvními B rozděleními, které přesáhnout práh podle rovnice 3.5:

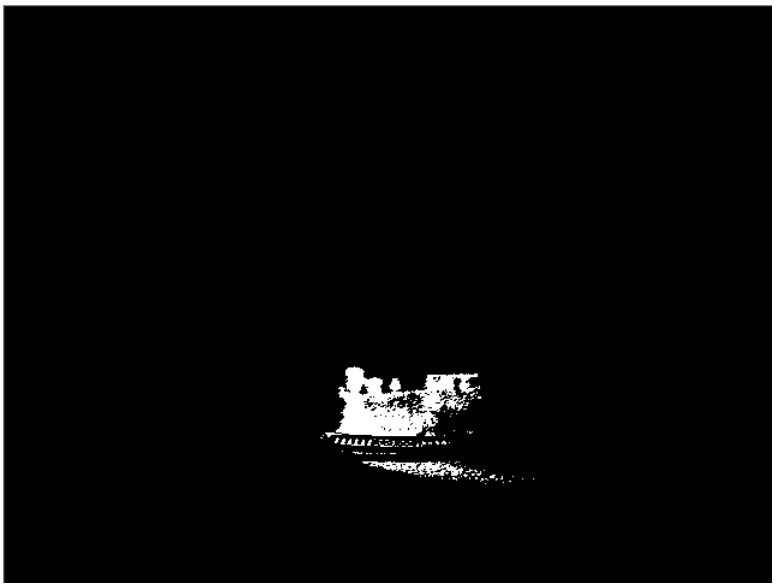
$$B = \underset{b}{\operatorname{argmin}}(\sum_{i=1}^{b} \omega_{i,t} > T), \tag{3.5}$$

Ostatní rozdělení reprezentují popředí.

Tato metoda je implementována také v knihovně OpenCV a to hned ve dvou variantách. Obě ale mají podobné výsledky, proto je uvedena pouze jedna.

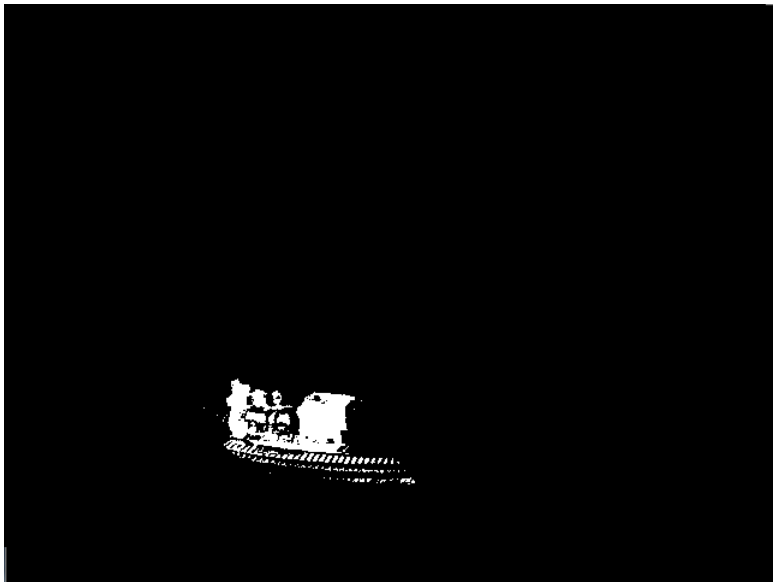
Výsledek je vidět na obrázku 3.4. Sledovaný objekt je detekován téměř celý. Není rozmazaný a nezobrazuje se ani stopa. Stejné výsledky jsou dosaženy nezávisle na

rychlosti pohybu. Robot je také detekován ještě několik sekund po zastavení. Nevýhodou tohoto přístupu je vyšší výpočetní náročnost. 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 3.4: Metoda směsi gausiánů

V této práci je použita metoda která vychází z [8]. Tento přístup počítá pro každý pixel vlastní hodnotu rychlosti učení. Ta je určena podle dvou parametrů. První parametr závisí na rozdílu mezi aktuálním snímkem a pozadím. Druhý parametr závisí na době, kterou je pixel detekován v pozadí. Tímto způsobem lze eliminovat případné opakující se pohyby v pozadí (např. pohyb způsobený větrem, blikání signalizačních diod apod.) Tato metoda byla ještě modifikována přidáním třetího parametru, který pracuje s dobou, jakou je pixel detekován v popředí. To je z toho důvodu, že původní metoda neaktualizuje daný pixel, pokud byl detekován jako pohybující se objekt. V případě správné detekce je to v pořádku. Pokud byla ale detekována pouze změna v pozadí, původní metoda by zde pořád detekovala pohyb. Pokud je tedy určitý pixel detekován v popředí delší dobu, lze předpokládat špatnou detekci a pixel bude převeden do pozadí. Výsledek je vidět na obrázku 3.5. Opět je detekován téměř celý objekt. Výhodou této metody je také relativně rychlý běh a možnost snadné implementace paralelního zpracování. Algoritmus i jeho implementace bude podrobněji vysvětlena v následující části.



Obrázek 3.5: Metoda s různou rychlostí učení

3.3 Algoritmus detekce pohybu

V této práci je tedy pro vytvoření modelu pozadí použitý poslední zmíněný přístup. Ten je dobře popsán v [8]. V celém algoritmu se pracuje s obrazem ve stupních šedi. Kamera sice poskytuje barevný RGB obraz, ale pro účely této práce je vhodné jej nejprve přebarvit na šedotónový obraz. Potom bude každý pixel reprezentován pouze hodnotou jasu.

Inicializace modelu pozadí

Před vlastním sledováním pohybu je potřeba vytvořit model pozadí. To je celkem jednoduché. Stačí vzít prvních N snímků v sekvenci a prostým průměrováním lze získat počáteční model pozadí. Scéna samozřejmě nesmí obsahovat objekt, který potom bude sledován. Počáteční model lze tedy získat takto:

$$BM_N(x,y) = \frac{\sum_{m=1}^{N} I_m(x,y)}{N} \tag{3.6}$$

kde:

- BM_N je jasová hodnota pixelu ve výsledném modelu pozadí
- N je počet snímků, ze kterých se vytváří pozadí (vhodnou volbou je $N = 100$)
- I_m je jasová hodnota pixelu v m-tém snímku

Odečítání pozadí

Když je k dispozici pozadí, je potřeba určit kritérium, podle kterého lze rozhodovat, jestli patří daný pixel aktuálního snímku do pozadí nebo do popředí. Nejdříve se musí vyčíslit hodnota rozdílu mezi aktuálním snímkem a modelem pozadí:

$$D_t(x,y) = |I_t(x,y) - BM_{(t-1)}(x,y)| \tag{3.7}$$

kde:

- D_t je výsledný rozdíl intenzit
- I_t je intenzita pixelu v aktuálním snímku
- BM_{t-1} je intenzita pixelu v posledním modelu pozadí

Výsledek potom bude porovnán se zvoleným prahem T. Pokud bude hodnota D_t větší než práh T, bude pixel detekován jako objekt v popředí. V opačném případě bude zasazen do pozadí. Hodnotu prahu T lze určit například experimentálně tak, aby v popředí zůstal pouze sledovaný objekt.

Aktualizace pozadí

Protože se scéna časem mění, model pozadí se musí s každým snímkem aktualizovat. Toto je nejsložitější a také výpočetně nejnáročnější část celého algoritmu. Nové pozadí je vždy složeno ze starého modelu a aktuálního snímku podle tohoto vztahu:

$$BM_t(x,y) = \alpha_{ad,t}(x,y)I_t(x,y) + (1 - \alpha_{ad,t}(x,y))BM_{t-1}(x,y) \tag{3.8}$$

kde:

- BM je pozadí
- I je aktuální snímek
- α je rychlost učení modelu pozadí

Hodnota rychlosti, jakou se aktuální snímek přenáší do pozadí se zjišťuje pro každý pixel samostatně. Výsledná hodnota závisí na dvou vážených parametrech:

$$\alpha_{ad,t}(x,y) = w_1\alpha_1 + w_2\alpha_2 \tag{3.9}$$

kde w_1 a w_2 jsou váhy a platí: $w_1 + w_2 \leq 1$

První parametr α_1 závisí na rozdílu mezi aktuálním snímkem a posledním modelem pozadí podle vztahu:

$$\alpha_1 = e^{-\frac{1}{2}\frac{D_t(x,y)^2}{\sigma_1^2}} \tag{3.10}$$

kde σ_1 je $\frac{T}{5}$.

Tento vztah platí pokud $D_t(x, y) < T$. V opačném případě bude $\alpha_1 = 0$. Tím je zaručeno, že se pozorovaný objekt nedostane do pozadí. Pro menší změny scény, kde se nejspíš nejedná o pohyb bude tento parametr větší a pozadí se tak bude měnit rychleji.

Druhý parametr α_2 závisí na době pixelu v pozadí C_{bg} a je dán vztahem:

$$\alpha_2 = e^{-\frac{1}{2}\frac{(\xi_{max} - C'_{bg})^2}{\sigma_2^2}} \tag{3.11}$$

kde:

- ξ_{max} je 150
- σ_2 je 15
- C'_{bg} je $min(\xi_{max}, C_{bg})$

Tyto hodnoty byly experimentálně zjištěny. Tento vztah platí pro $C_{bg} > 30$. Jinak bude hodnota $\alpha_2 = 0$. To nám zajistí, že pixely, které se často mění mezi pozadím a popředím budou spíše přeneseny do pozadí.

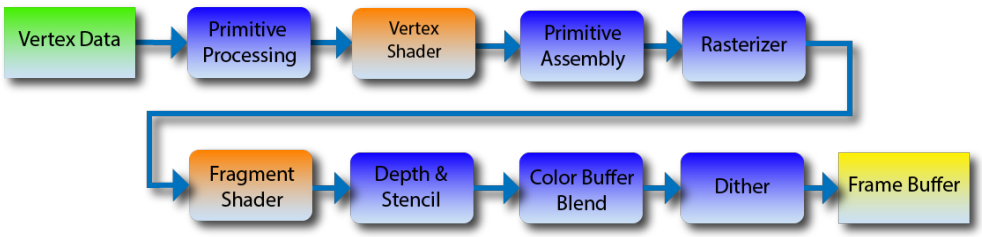
Tento algoritmus aktualizuje pouze ty části pozadí, ve kterých nebyl detekován pohyb. Pokud ale dojde ke špatné detekci, nepřenese se tato změna nikdy do pozadí. Proto je přidána ještě jedna podmínku. Pokud bude doba detekce pixelu v popředí C_{fg} větší než 100, pixel se přenese do pozadí.

3.4 Implementace pomocí GLSL

Použitý algoritmus počítá při aktualizaci pozadí novou hodnotu pro každý pixel nezávisle na ostatních. Z toho důvodu je možné urychlit výpočet tohoto procesu paralelním zpracováním. V této části bude nastíněn postup, jakým lze použitý algoritmus implementovat pomocí OpenGL s využitím jazyka GLSL. Nejprve se načte obraz z kamer pomocí knihovny OpenCV. Jazyk GLSL pracuje s texturami, takže potom se načtený obrázek převede na texturu. OpenCV a GLSL používají obrácené indexování, takže je potřeba obraz ještě přetočit. Tyto akce provádí následující kód:

```
1 cv::VideoCapture cap(0);
2 cap >> image;
3 cv::flip(image, image, 0);
4 glBindTexture(GL_TEXTURE_2D, textures[0]);
5 glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image.cols,
6   image.rows, 0, GL_BGR, GL_UNSIGNED_BYTE, image.ptr());
```

Ostatní zpracování se provede pomocí shaderů. Hlavní funkcí OpenGL je vykreslování obrazu do framebufferu. Předtím ale musí každý obraz projít různými fázemi. Schéma tohoto procesu je na obrázku 3.6. K dispozici jsou dva programovatelné shadery, oba jsou povinné. Prvním je vertex shader a druhým fragment shader.



Obrázek 3.6: Schéma OpenGL

Vertex shader

Vertex shader primárně slouží k prostorové transformaci vrcholů při tvorbě 3D grafiky. Protože v této úloze se pracuje pouze s dvourozměrným obrazem, který není potřeba nijak transformovat, nebude se v tomto shaderu v podstatě nic dělat a pouze se předají dál vstupní informace:

```
1  layout (location = 0) in vec3 Position;
2  layout (location = 1) in vec2 TexCoord;
3
4  out vec2 TexCoord0;
5
6  void main()
7  {
8      gl_Position = vec4(Position, 1.0);
9      TexCoord0 = TexCoord;
10 }
```

Fragment shader

Veškeré výpočty se v této úloze provádějí právě ve fragment shaderu. Výstupem toho shaderu je pouze barva pro každý pixel v obraze. Přesto může být využit jak pro inicializaci modelu pozadí, tak pro odečítání a aktualizaci pozadí. Stačí vždy připojit správný buffer, do kterého se bude renderovat. Tímto způsobem lze také využít texturu pro ukládání proměnných jako je doba pixelu v pozadí. Aby mohl být tento shader použit pro různé funkce, musí obsahovat větvení:

- určení popředí - v této části odpovídá výsledná barva bílé, pokud je pixel detekován v popředí a černé, pokud v pozadí
- aktualizace doby - výstupní barva pixelu zde odpovídá dané době - čím vyšší je intenzita pixelu, tím delší je doba detekce, přičemž rozsah je od nuly (černá barva) do 1 (bílá barva)
- aktualizace pozadí - zde je výstup složený z aktuálního snímku a posledního modelu pozadí podle vztahů 3.8 - 3.11

Takto vypadá výsledný fragment shader:

```
#version 330

in vec2 TexCoord0;
out vec4 FragColor;

float finalColor;
float gray1, gray2;
vec4 color1, color2;
float time, alfa1, alfa2, alfa;
float threshold = 0.1;

uniform sampler2D current, background,timeBack;
uniform bool init, diff, timeInBack;

void main()
{
   color1 = texture2D(current, TexCoord0.xy);
   color2 = texture2D(background, TexCoord0.xy);
   gray1 = dot(color1.rgb, vec3(0.2126, 0.7152, 0.0722));
   gray2 = dot(color2.rgb, vec3(0.2126, 0.7152, 0.0722));

   /* Cast pro urceni popredi */

   if(diff)
   {
     if(abs(gray1 - gray2) > threshold && texture2D(timeBack,
    TexCoord0.xy)[0] > 0.01)
     {
       FragColor = vec4(1,1,1,1);
     }
     else
     {
       FragColor = vec4(0,0,0,0);
     }
   }

   /* Cast pro aktualizaci doby */

   else if(timeInBack)
   {
     if(abs(gray1 - gray2) > threshold)
     {
       FragColor = vec4(texture2D(timeBack, TexCoord0.xy)[0]+0.01,
    0, 0, 1);
     }
     else
     {
       FragColor = vec4(0, texture2D(timeBack, TexCoord0.xy)
    [1]+0.01, 0, 1);
     }
   }

```

```
50    /* Cast aktualizace pozadi */
51
52    else
53    {
54      if(init)
55      {
56        finalColor = (gray1 + gray2*50)/51.f;
57        FragColor = vec4(finalColor, finalColor, finalColor, 1);
58      }
59      else
60      {
61        if(abs(gray1 - gray2) < threshold)
62        {
63          alfa1 = exp((-1/2)*(gray1-gray2)*(gray1-gray2)/0.0025);
64          alfa2 = exp((-1/2)*(1-texture2D(timeBack, TexCoord0.xy)[1])
     /0.1);
65          alfa = (alfa1+alfa2)/2;
66          finalColor = alfa*gray1 + (1-alfa)*gray2;
67        }
68        else
69        {
70          time = texture2D(timeBack, TexCoord0.xy)[0];
71          if(time == 1)
72          {
73            finalColor = gray1;
74          }
75          else
76          {
77            finalColor = gray2;
78          }
79        }
80        FragColor = vec4(finalColor, finalColor, finalColor, 1);
81      }
82    }
83  }
```

3.5 Shrnutí

Bylo otestováno několik přístupů k detekci pohybujících se objektů v obraze. Zvolena byla metoda pomocí modelování pozadí, především z důvodu větší robustnosti. Hlavní nevýhodu oproti porovnání následujících snímků, kterou je pomalejší běh lze navíc odstranit pomocí paralelního zpracování. Dále byla vybrána metoda s adaptivní rychlostí učení. Ta dosahuje podobných výsledků jako model pomocí směsi gausiánů, ale při nižších výpočetních nárocích.

Kapitola 4

Kalibrace systému a triangulace

Po nalezení hledaného robota v obraze bude dalším krokem výpočet jeho polohy v prostoru. K tomu bude zapotřebí informace minimálně ze dvou nezávislých zdrojů, protože kamery 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 je možné určit hloubku. Nejprve je ale třeba systém kamer nakalibrovat.

4.1 Stereo kalibrace

Vztah, kterým se transformují souřadnice reálného světa do obrazu z kamer popisuje rovnice (4.1).

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

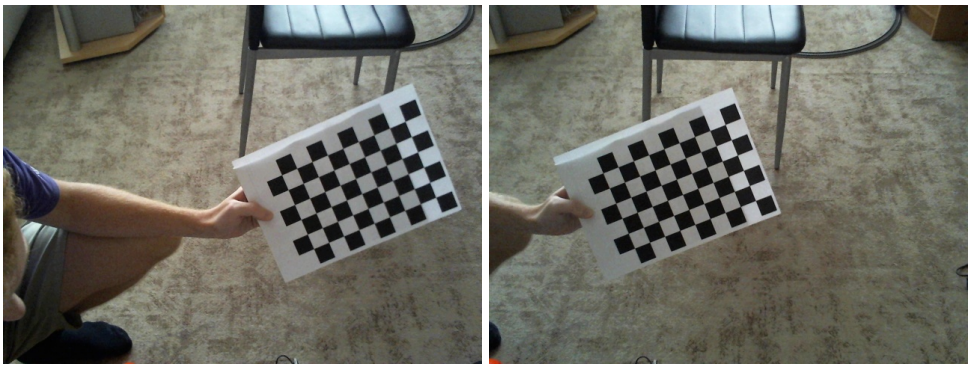
kde:

- 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 souřadnicového systému vzhledem ke kameře.

K získání těchto matic opět velmi dobře poslouží knihovna OpenCV. Všechny metody jsou popsány zde [9]. První je metoda `calibrateCamera`. Tato metoda provede odhad vnitřních i vnějších parametrů. K tomu je zapotřebí několik párů 3D souřadnic bodu v prostoru s korespondující 2D projekcí v obraze z kamery. Tyto souřadnice lze získat pomocí objektu se známou geometrií a snadno detekovatelnými významnými body. V OpenCV je zabudovaná podpora pro detekci šachovnice na obrázku 4.1. K tomuto účelu slouží funkce `findChessboardCorners`.

Algoritmus nejdříve odhadne vnitřní parametry kamery. Poté se odhaduje poloha kamery vzhledem ke snímanému objektu. Nakonec se pomocí Levenberg-Marquardtova algoritmu minimalizuje odchylka mezi naměřenými souřadnicemi z kamery a souřadnicemi vypočítaným pomocí dosud odhadnutých parametrů.



Obrázek 4.1: Kalibrační vzor

Kamery budou umístěny 25 cm od sebe. Čím větší je tato vzdálenost, tím méně náchylná na chyby bude následná triangulace. Na druhou stranu je potřeba zajistit, aby se obrazy z kamer dostatečně překrývaly. Vždy se tedy bude jednat o kompromis a záleží na tom, jak velkou plochu je potřeba sledovat. Rozmístění kamer je vidět na obrázku 4.2.

Pro zjištění vzájemné polohy kamer slouží metoda `stereoCalibrate`. Ta funguje podobně jako funkce `calibrateCamera`. Dokáže tedy 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. Výstupem funkce `stereoCalibrate` je rotační matice R a translační matice T, popisující vzájemnou polohu kamer. Předpokladem jsou stejná data, jako u metody `calibrateCamera`. Je opět potřeba získat obrazy kalibrační šachovnice z obou kamer. Pro správnou kalibraci je potřeba 10-20 párů. Důležitá je také jejich různorodost. Šachovnici je třeba umístit různě natočenou do různých částí obrazu. Musí ale být vždy vidět celá v obou snímcích.

Obrázek 4.2: Rozmístění kamer

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),

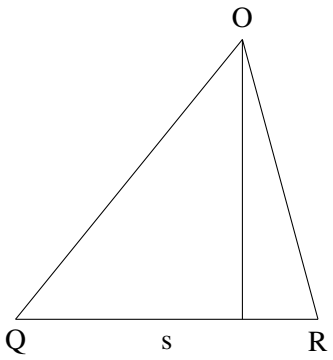
$$P1 = \begin{bmatrix} f & 0 & cx_1 & 0 \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} P2 = \begin{bmatrix} f & 0 & cx_2 & T_x * f \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.2}$$

kde T_x je horizontální posun mezi kamerami.

Po nakalibrování kamerového systému se již nesmí 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. Je zobrazena na obrázku 4.3.



Obrázek 4.3: Triangulace

Máme pozorovaný objekt O s prostorovými souřadnicemi x, y, z. Dále máme dva pozorovatele Q, R, jejichž poloha a vzájemná vzdálenost s je známá a je obsažena v projekčních maticích P_Q a P_R. Pozorovatelé ale vidí souřadnice objektu O pouze ve 2D jako u_1, v_1, resp. u_2, v_2. Lze tvrdit, že pozorovaný objekt leží na epipolární přímce určené souřadnicemi u_1, v_1 resp. u_2, v_2 a polohou daného pozorovatele. V případě dvou pozorovatelů/kamer máme tedy dvě přímky. Hledaný objekt musí ležet na každé z nich. Z toho vyplývá, že 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 vzniklou chybu.

Ačkoliv knihovna OpenCV obsahuje také metodu pro triangulaci, její přesnost není příliš dobrá. Proto bude využita jiná metoda, konkrétně iterativní lineární triangulace. Ta je popsána v [10]. Tato metoda nejprve vychází z lineární triangulace. Začneme rovnicí (4.3) pro pozorovatele Q,

$$w(u_1, v_1, 1)^T = P_Q x \tag{4.3}$$

kde w je neznámý faktor měřítka, $(u_1, v_1, 1)^T$ jsou homogenní souřadnice bodu v obraze, P_Q je projekční matice pozorovatele Q a x jsou souřadnice v reálném prostoru. Nyní pokud označíme $i-$tý řádek matice P_Q jako P_{Qi}, získáme rovnice (4.4).

$$wu_1 = P_{Q1}^T x, \qquad wv_1 = P_{Q2}^T x, \qquad w = P_{Q3}^T x \tag{4.4}$$

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

$$u_1 P_{Q3}^T x = P_{Q1}^T x, \qquad v_1 P_{Q3}^T x = P_{Q2}^T x \tag{4.5}$$

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 lze pomocí metody nejmenších čtverců nalézt nejlepší řešení.

Lineární triangulace samostatně není příliš přesná. Chybu je možné 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).

$$\epsilon = u_1 P_{Q3}^T x - P_{Q1}^T x \tag{4.6}$$

Cílem je ale minimalizovat odchylku změřené souřadnice u_1 a projekce x, která má tvar (4.7).

$$\epsilon' = u_1 - P_{Q1}^T x / P_{Q3}^T x \tag{4.7}$$

Vydělením pravé strany rovnice (4.6) výrazem $P_{Q3}^T x$ lze přejít k rovnici (4.7). Protože $P_{Q3}^T x = w$, budou všechny rovnice váženy výrazem $1/w$. Samozřejmě to nelze hned provést, 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

Odhad polohy

Vzhledem k tomu, že v průběhu celého procesu získávání prostorových souřadnic sledovaného objektu vznikají chyby, výsledek tak odpovídá realitě pouze přibližně. Výstup triangulace je navíc pouze statické měření, které nevyužívá informaci o dynamice systému. Proto je ještě potřeba provést filtraci. Pro objasnění filtrace vyjdeme z [11]. Úloha filtrace bývá také nazývána úlohou odhadu (estimace) systému, protože se snažíme odhadnout, jak se systém bude chovat. Potom je možné jej efektivněji sledovat. Základem pro většinu algoritmů sledování polohy je model systému. V tomto případě to znamená model pohybu sledovaného objektu. Čím přesnější model bude pro sledování k dispozici, tím lepší výsledky lze obecně očekávat. V následující části bude uveden postup, jakým lze modelovat daný systém.

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. Přechod z jednoho stavu do následujícího lze popsat rovnicí (5.1),

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

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 $p(w_k)$. Navíc je známá také hustota pravděpodobnosti počátečního stavu $p(x_0)$. Dále měření může být popsáno rovnicí (5.2),

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

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.

V případě lineárního systému přejdou uvedené rovnice do tvaru (5.3), resp. (5.4)

$$x_{k+1} = Fx_k + w_k \tag{5.3}$$

$$z_k = Hx_k + v_k \tag{5.4}$$

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 ale výrazně ztížilo úlohu odhadu. Dalším požadavkem často bývá, 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 odhadu stavu je tedy na základě měření odhadnout stav systému x_k. K řešení tohoto problému existují různé přístupy. V této práci je použitý Bayesovský přístup, který bude popsán v další části.

5.2 Bayesovský přístup

Obecný problém odhadu je dán rovnicemi (5.1) a (5.2). Cílem Bayesovského přístupu je určení podmíněné hustoty pravděpodobnosti $p(x_k|z^l)$. Podle toho, do jakého času máme k dispozici měření 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

V této práci se budeme věnovat pouze filtraci a jednokrokové predikci. Cílem je získat aposteriorní hustotu pravděpodobnosti nebo také filtrační hustotu pravděpodobnosti. Ta je definována rekurzivním vztahem:

$$p(x_k|z^k) = \frac{p(z_k|x_k)p(x_k|z^{k-1})}{p(z_k|z^{k-1})} \tag{5.5}$$

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

Pro určení aposteriorní hustoty pravděpodobnosti je potřeba znát model měření, který je dán vztahem (5.2). Dále je potřeba prediktivní hustota pravděpodobnosti, která vychází z modelu dynamiky systému (5.1) a je určena vztahem:

$$p(x_k|z^{k-1}) = \int p(x_k|x_{k-1})p(x_{k-1}|z^{k-1})dx_{k-1} \tag{5.6}$$

5.3 Kalmanův filtr

Výše popsané vztahy vedou v případě lineárního gaussovského systému na vztahy popisující 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 [12]. 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):

$$\hat{x}_k^{'} = F\hat{x}_{k-1} \tag{5.7}$$

$$P_k^{'} = FP_{k-1}F^T + Q \tag{5.8}$$

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):

$$K_k = P_k^{'}H^T(HP_k^{'}H^T + R)^{-1} \tag{5.9}$$

$$\hat{x}_k = \hat{x}_k^{'} + K_k(z_k - H\hat{x}_k^{'}) \tag{5.10}$$

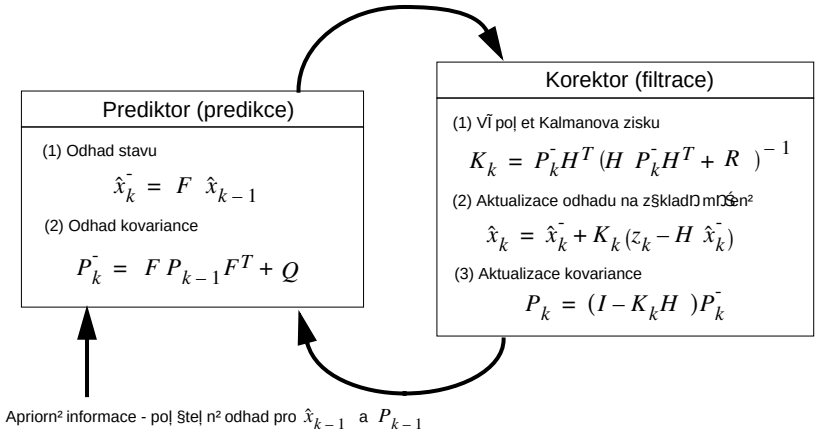
$$P_k = (I - K_kH)P_k^{'} \tag{5.11}$$

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

Realizace Kalmanova filtru

Předpokladem pro použití Kalmanova filtru je linearita a gaussovost systému. Pro nelineární systémy lze použít rozšířený Kalmanův filtr. Ten provádí v každém kroku linearizaci systému v daném pracovním bodě. Opět je možné využít knihovnu OpenCV. Ta má implementován pouze základní (lineární) filtr. Přesto je možné jej použít i jako rozšířený Kalmanův filtr. Stačí jen mírně modifikovat algoritmus výpočtu. Především je nutné dopředu znát jakobián přechodové funkce. Ten se potom v každém kroku aktualizuje a používá pro predikci místo matice přechodu. V této práci bude využit jak standardní Kalmanův filtr, tak i jeho rozšířená verze. Který filtr použít závisí na příslušném modelu pohybu. V další části je uveden přehled těchto modelů.



Obrázek 5.1: Diagram Kalmanova filtru

5.4 Modely Pohybu

Pro účely sledování polohy budou porovnány tři modely. Od jednoduchého lineárního modelu, přes model s konstantním poloměrem otáčení až po model pohybu po křivce.

Lineární model

Nejjednodušší model 2D pohybu je prostý lineární model. Uvažujeme konstantní směr a také rychlost pohybu. Případné manévry objektu jsou reprezentovány stavovým šumem modelu. Stav systému lze tedy 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 tento model bude mít matice přechodu systému mezi stavy tvar:

$$F = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.12}$$

kde T je perioda vzorkování. Měřit se bude pouze poloha, proto matice měření bude mít tvar:

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

Model s konstantním poloměrem otáčení

Další model je popsán například v [13]. Tento model předpokládá, že se cíl pohybuje konstantní rychlostí a také konstantní úhlovou rychlostí. Vektor stavu bude opět uspořádaná čtveřice hodnot $[x, y, v_x, v_y]$. Matice přechodu ale bude mít jiný tvar:

$$F = \begin{bmatrix} 1 & 0 & \frac{sin\ \omega T}{\omega} & -\frac{1 - cos\ \omega T}{\omega} \\ 0 & 1 & \frac{1 - cos\ \omega T}{\omega} & \frac{sin\ \omega T}{\omega} \\ 0 & 0 & cos\ \omega T & -sin\ \omega T \\ 0 & 0 & sin\ \omega T & cos\ \omega T \end{bmatrix} \tag{5.14}$$

Tuto matici je možné zjednodušeně aproximovat takto:

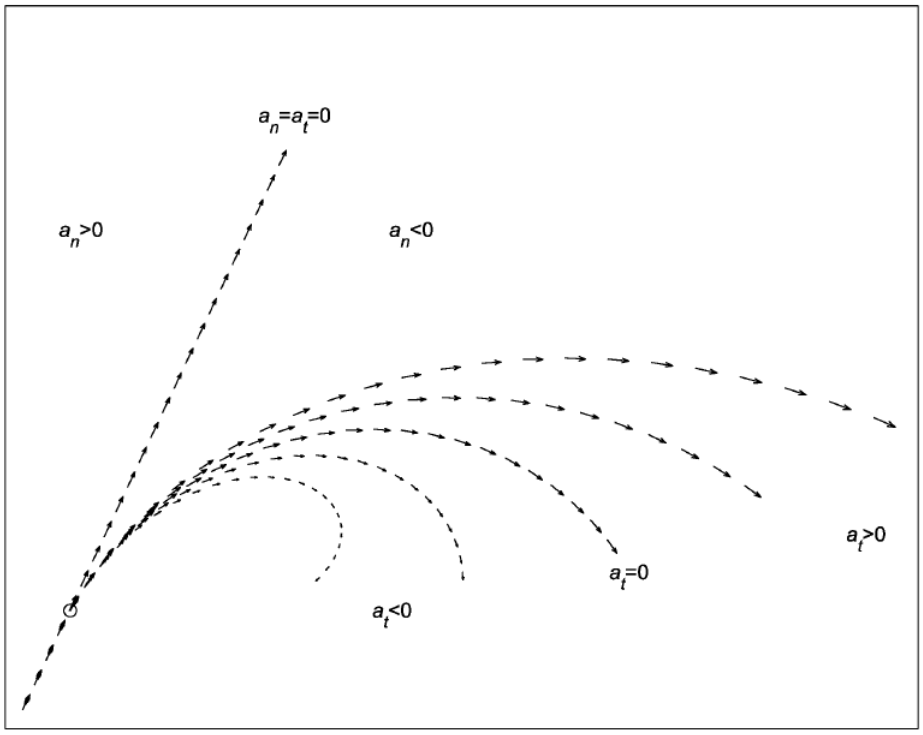
$$F = \begin{bmatrix} 1 & 0 & T & -\omega T^2/2 \\ 0 & 1 & \omega T^2/2 & T \\ 0 & 0 & 1\ -\ \omega T^2/2 & -\omega T \\ 0 & 0 & \omega T & 1\ -\ \omega T^2/2 \end{bmatrix} \tag{5.15}$$

Pořád se jedná o nelineární matici. Pokud je ale ω konstantní a známá, bude i celá matice konstantní. Potom je možné opět použít k filtraci standardní Kalmanův filtr. Matice měření bude mít stejný tvar jako v minulém případě, tedy:

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{5.16}$$

Model pohybu po křivce

Posledním testovaným modelem je model pohybu po křivce. Tento model je na obrázku 5.2 a je popsán v [14].



Obrázek 5.2: Model pohybu po křivce

Oproti předchozím modelům je obecnější. Stav je nyní reprezentován uspořádanou šesticí $[x, v_x, y, v_y, a_t, a_n]$, kde a_t je tečné zrychlení a a_n je normálové zrychlení. Pomocí těchto zrychlení je modelován tvar trajektorie. Model je popsán rovnicí:

$$\dot{x}(t) = f(x(t)) + w_x(t) \tag{5.17}$$

kde

$$x(t) = \begin{bmatrix} x(t) \\ v_x(t) \\ y(t) \\ v_y(t) \\ a_t(a) \\ a_n(t) \end{bmatrix}, \qquad f(x) = \begin{bmatrix} v_x(t) \\ a_t(t)f^{vx} + a_n(t)f^{vy} \\ v_y(t) \\ a_t(t)f^{vy} - a_n(t)f^{vx} \\ 0 \\ 0 \end{bmatrix},$$

$$f^{vx} = \frac{v_x(t)}{\sqrt{v_x^2(t)+v_y^2(t)}}, \qquad f^{vy} = \frac{v_y(t)}{\sqrt{v_x^2(t)+v_y^2(t)}}$$

Zde již bude potřeba použít rozšířený Kalmanův filtr. K tomu je zapotřebí znát jakobián přechodové funkce. Ten má tvar:

$$\frac{\partial}{\partial x}(f(x(t))) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & a_t(t)f_{vx}^{vx} + a_n(t)f_{vx}^{vy} & 0 & a_t(t)f_{vy}^{vx} + a_n(t)f_{vy}^{vy} & f^{vx} & f^{vy} \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & a_t(t)f_{vx}^{vy} - a_n(t)f_{vx}^{vx} & 0 & a_t(t)f_{vy}^{vy} - a_n(t)f_{vy}^{vx} & f^{vx} & -f^{vy} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.18}$$

kde

$$f_{vx}^{vx} = \frac{v_x^2(t)}{(v_x^2(t)+v_y^2(t))^{3/2}}, \qquad f_{vy}^{vx} = \frac{v_y(t)v_x(t)}{(v_x^2(t)+v_y^2(t))^{3/2}},$$

$$f_{vx}^{vy} = \frac{v_x(t)v_y(t)}{(v_x^2(t)+v_y^2(t))^{3/2}}, \qquad f_{vy}^{vy} = \frac{v_y^2(t)}{(v_x^2(t)+v_y^2(t))^{3/2}}$$

Matice měření bude opět podobná, jako v minulých případech. Změní se pouze její dimenze:

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{5.19}$$

Ještě je důležité poznamenat, že vzhledem k použití nelineárního filtru není zaručena konvergence odhadu.

Kapitola 6

Ověření navržených metod

V této části budou zhodnoceny dosažené výsledky navrhovaného přístupu ke sledování polohy pohybujícího se robotu. Nejprve bude zkoumána přesnost, přičemž budou porovnány použité modely pohybu na stejných datech. Dále bude vyhodnocena rychlost zpracování.

6.1 Přesnost odhadu

Každý model byl testován na 3 sadách měření. Kamery byly vždy umístěny asi 25 cm od sebe ve výšce 120 cm od země. Scéna je obrázku 6.1.

Obrázek 6.1: Rozmístění kamer

První měření bylo sledování vláčku o rozměrech 15x8x4 cm na kolejích, tvořících kružnici. To z toho důvodu, aby byla známá skutečná trajektorie pohybu. V tomto případě se jednalo o relativně pomalý pohyb. Celkem byl tento experiment opakován 10x pro každý model. V grafech je zobrazen pouze jeden běh. Ostatní pokusy sloužily pro ověření chyby odhadu. Výsledky jsou vždy uvedeny v tabulce. Další 2 měření tvoří pohyb florbalového míčku na provázku. Tento pohyb byl o něco rychlejší. Nejprve se míček pohyboval po spirále směrem ke středu. Nakonec byl vyzkoušen obecný pohyb, jehož trajektorie bude zřejmá z grafu.

Vzorkovací perioda byla ve všech případech $T = 0.05s$. Úhlová rychlost ω ve druhém modelu byla nastavena na $\omega = 0.6\ rad/s$.

Kovarianční matice stavového šumu a šumu měření byly pro první dva modely:

$$Q_F = \begin{bmatrix} 1e-3 & 0 & 0 & 0 \\ 0 & 1e-3 & 0 & 0 \\ 0 & 0 & 1e-3 & 0 \\ 0 & 0 & 0 & 1e-3 \end{bmatrix} Q_H = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

Pro třetí model měly tyto matice tvar:

$$Q_F = \begin{bmatrix} 1e-4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1e-2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1e-4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1e-2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1e-10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1e-10 \end{bmatrix}$$

$$Q_H = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 \end{bmatrix}$$

Všechny tyto hodnoty byly experimentálně nastaveny tak, aby byl odhad v dané úloze co nejlepší.

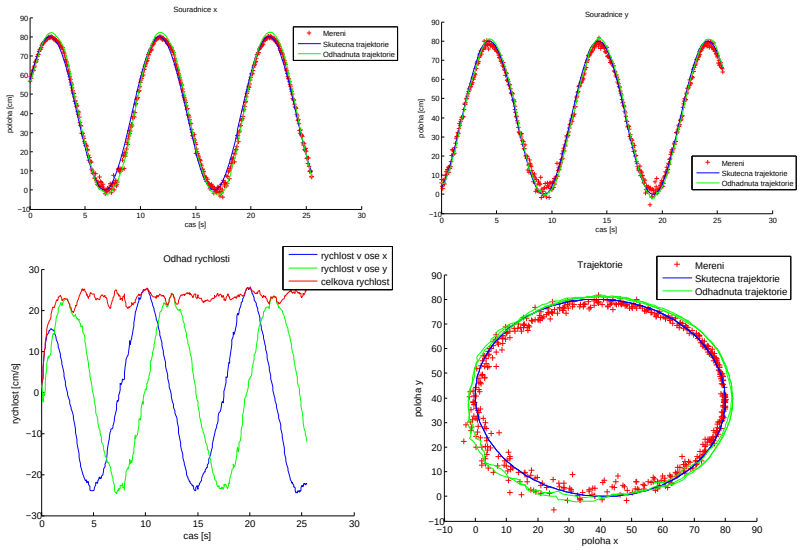
Aby bylo možné testovat modely na stejných datech, bylo nejprve provedeno pouze měření a teprve poté bylo toto měření použito pro úlohu filtrace. Za počáteční stav bylo zvoleno první měření v případě polohy a 0 pro ostatní stavové proměnné. Kovarianční matice odhadu byla zvolena ve tvaru $P_{k-1} = 0.1I$, kde I je identická matice.

Lineární model

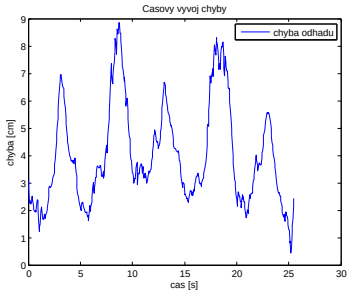
Prvním testovaným modelem byl jednoduchý lineární model pohybu. Jak je vidět z grafů 6.2, 6.3, 6.4, 6.5 jeho přesnost je docela dobrá. Odchylka odhadu od skutečné trajektorie je uvedena v následující tabulce:

Číslo měření	Střední hodnota chyby [cm]	Směrodatná odchylka chyby [cm]
1	4.2914	1.8126
2	4.2843	1.7958
3	4.0947	1.8071
4	4.3025	1.7965
5	4.2015	1.8124
6	4.2941	1.8048
7	4.0094	1.7994
8	3.9260	1.7967
9	3.9891	1.8142
10	4.008	1.8023
průměr	4.1401	1.8042

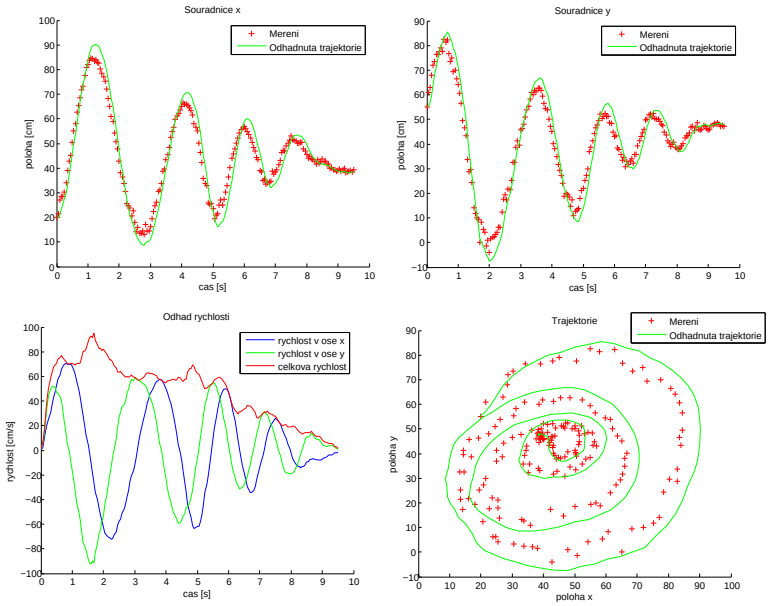
Je však zřejmá pomalejší reakce na změny pohybu. Nejvíce je to patrné při pohybu po spirále 6.4, kde leží odhadnutá trajektorie prakticky neustále vně měření. Totéž lze říci o pohybu po kružnici 6.2, kde je zase detekován pohyb po kružnici s větším poloměrem. Přesto pro pomalu se pohybující objekty, které nemění příliš prudce směr pohybu by byl i tento jednoduchý model použitelný.



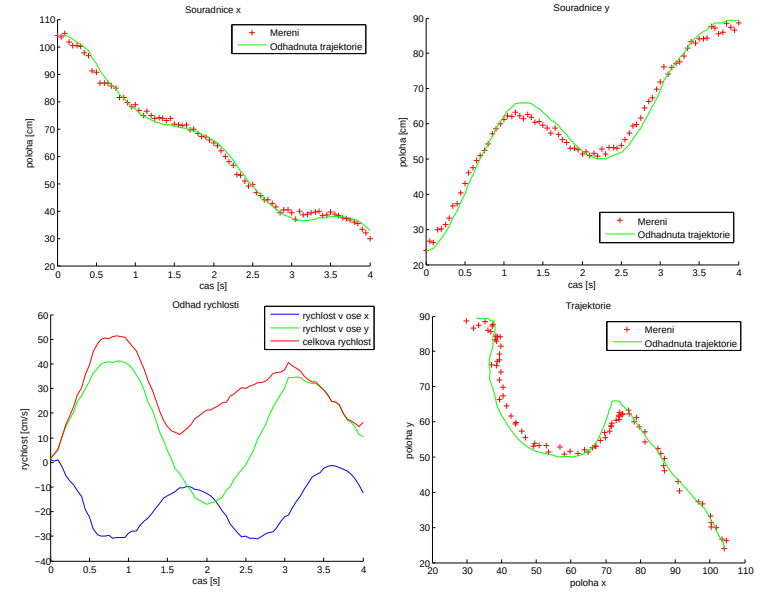
Obrázek 6.2: Pohyb po kružnici



Obrázek 6.3: Odchylka odhadu při pohybu po kružnici



Obrázek 6.4: Pohyb po spirále



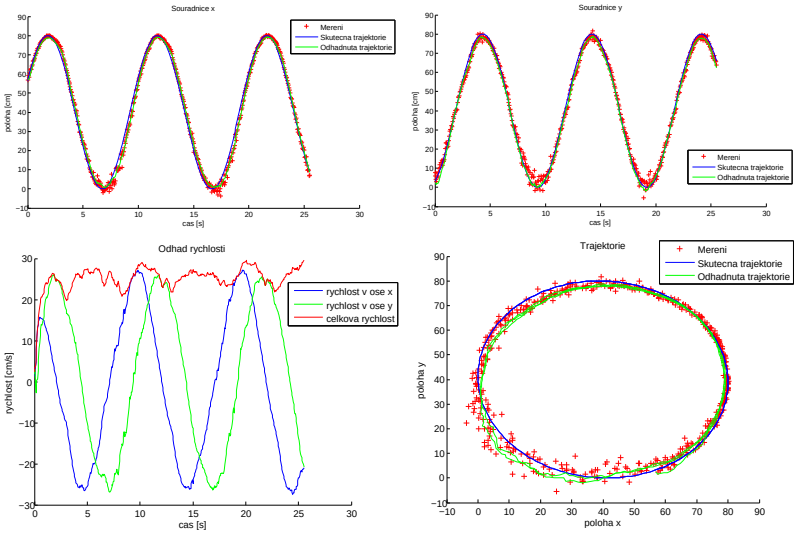
Obrázek 6.5: Obecný pohyb

Model s konstantním poloměrem otáčení

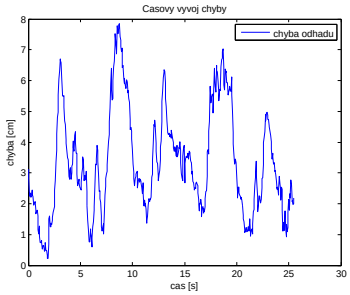
Podle grafů 6.6, 6.7 6.8, 6.9 poskytuje tento model přesnější odhad, než předchozí lineární model. To odpovídá očekávání, protože tento pohyb přesně odpovídá modelu. Odchylky při pohybu po kružnici jsou uveden v následující tabulce:

Číslo měření	Střední hodnota chyby [cm]	Směrodatná odchylka chyby [cm]
1	3.2547	1.7745
2	3.3421	1.7542
3	3.2842	1.7642
4	3.2485	1.7587
5	3.3286	1.7648
6	3.2212	1.7548
7	3.3101	1.7503
8	3.3001	1.7612
9	3.2785	1.7591
10	3.3214	1.7519
průměr	3.2889	1.7594

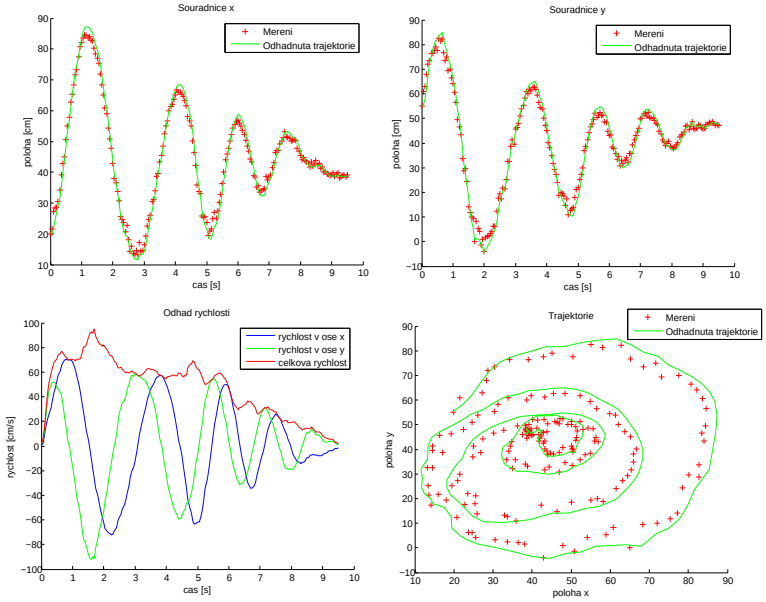
Ještě je potřeba zmínit, že na pravé straně kružnice je odhad lepší než na levé. To je dáno přesností dostupného měření. Na levé straně byly horší světelné podmínky. Vláček vrhal stín, který byl částečně detekován, proto má v těchto místech měření větší rozptyl. V dalších dvou případech již odhad není tak dobrý. Při pohybu po spirále 6.8 je pořád zřejmé, že se odhad nepřizpůsobuje dostatečně rychle. Podobně je tomu ve třetím případě 6.9.



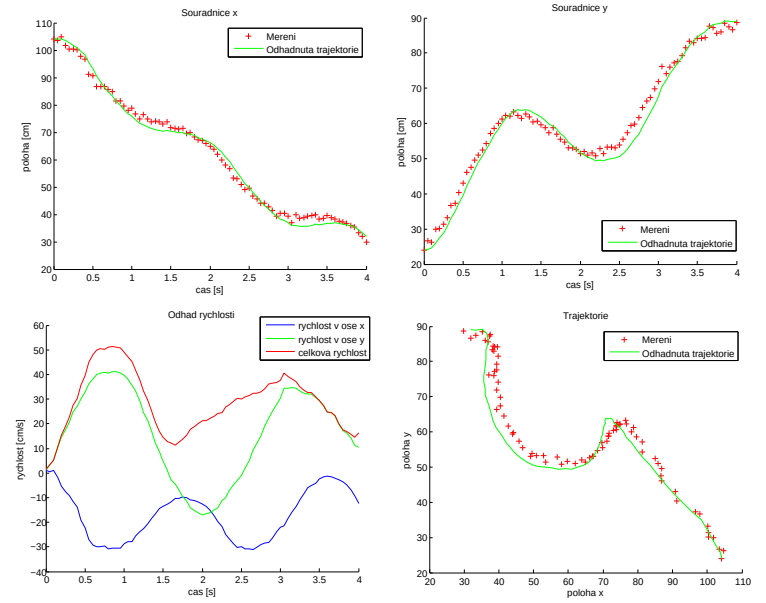
Obrázek 6.6: Pohyb po kružnici



Obrázek 6.7: Odchylka odhadu při pohybu po kružnici



Obrázek 6.8: Pohyb po spirále



Obrázek 6.9: Obecný pohyb

Model pohybu po křivce

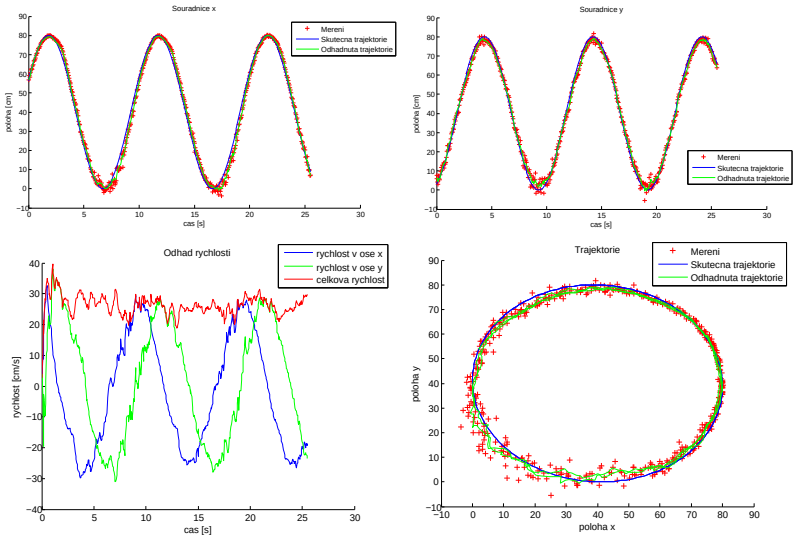
Jak je zřejmé z grafů 6.10, 6.11, 6.12, 6.13, odhad pomocí tohoto modelu je nejpřesnější. Odchylky při odhadu pohybu po kružnici jsou opět uvedeny v tabulce:

Číslo měření	Střední hodnota chyby [cm]	Směrodatná odchylka chyby [cm]
1	2.8014	1.6015
2	2.7942	1.6242
3	2.7895	1.5968
4	2.7549	1.5899
5	2.8267	1.6018
6	2.7813	1.6172
7	2.8495	1.5958
8	2.7659	1.6143
9	2.8049	1.6159
10	2.7941	1.6207
průměr	2.7962	1.6078

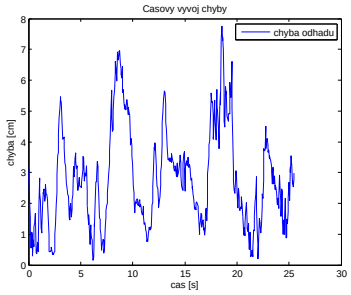
Ve všech třech testovaných případech byla odhadnutá trajektorie pohybu velmi podobná skutečnosti. Tento model je také schopný reagovat na rychlejší změny pohybu, než předchozí modely. Dále je nutné poznamenat, že výpočetní nároky jsou oproti jednodušším modelům vyšší jen minimálně a v porovnání se zbytkem programu je tento rozdíl zanedbatelný. Srovnání s předchozími modely je v následující tabulce:

Model	Střední hodnota chyby [cm]	Směrodatná odchylka chyby [cm]
1	4.1401	1.8042
2	3.2889	1.7594
3	2.7962	1.6078

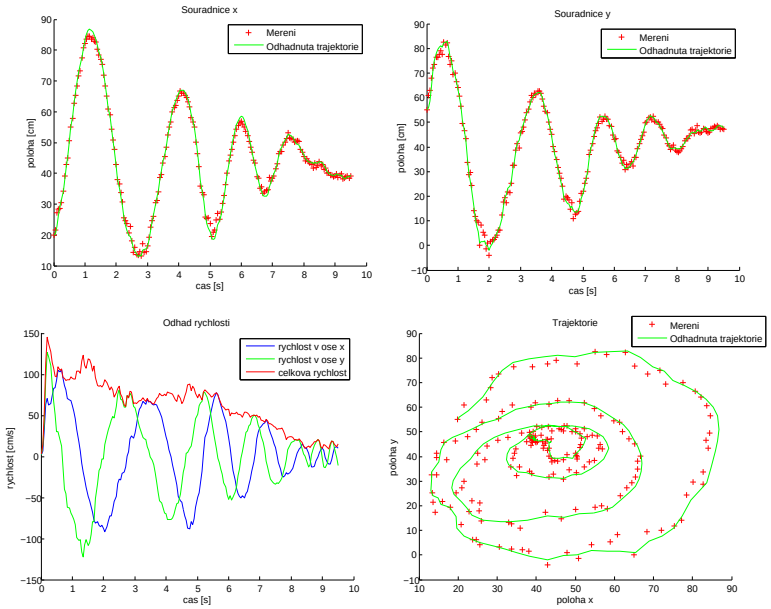
Přesnost odhadu je především v případě třetího modelu dostatečná pro sledování pohybujícího se objektu.



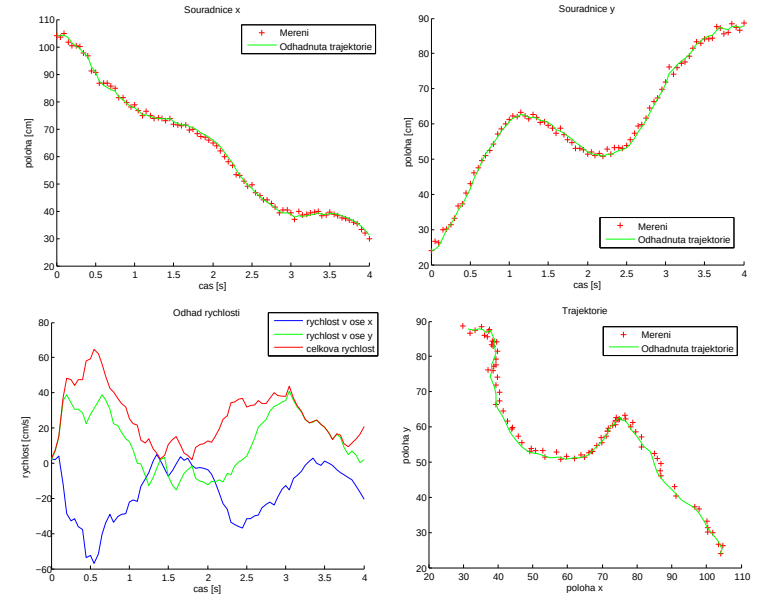
Obrázek 6.10: Pohyb po kružnici



Obrázek 6.11: Odchylka odhadu při pohybu po kružnici



Obrázek 6.12: Pohyb po spirále



Obrázek 6.13: Obecný pohyb

6.2 Rychlost běhu

Nakonec byla zhodnocena rychlost běhu programu. Zkoumán byl počet snímků za sekundu, který je schopný systém zpracovat. Testovací scéna zůstala stejná. Sledovaným objektem byl vláček na kolejích.

Algoritmus	Model pozadí [fps]	Celý program [fps]
Použitý postup s GLSL	24	20
Použitý postup bez GLSL	20	16
Použitý postup pomocí Beaglebone	4	4
OpenCV MOG	6	6
OpenCV MOG2	9	7

V tabulce jsou uvedeny rychlosti zpracování programu. Nejrychlejší je podle očekávání zpracování akcelerované pomocí GLSL. I bez pomoci grafické karty je ale rychlost programu celkem dobrá. Dále byly pro srovnání vyzkoušeny standardní metody modelování pozadí pomocí směsi gaussiánů, které jsou implementovány v OpenCV (MOG a MOG2). Oproti použitému algoritmu jsou výrazně pomalejší.

Zpracování pomocí Beaglebone také není příliš rychlé. Pro sledování robotu v reálném čase rozhodně není tato rychlost dostatečná. V tomto případě se jednalo o zpracování pouze pomocí procesoru, který není tak výkonný jako procesory u běžných PC. Jistou možností by bylo využití grafického jádra pro část zpracování obrazu. Tím by bylo možné získat nějaké snímky za sekundu navíc. Případně by bylo také možné využít dvě jednotky Beaglebone. Provádět na každé jednotce zpracování obrazu pouze z jedné kamery a výsledek potom posílat dále ke zpracování do PC.

Kapitola 7

Závěr

Navrhovaný algoritmus odhadu polohy pohybujících se objektů je založený na detekci pohybu v obraze na základě modelování pozadí. Triangulací jsou získány prostorové souřadnice, které jsou dále filtrovány pomocí Kalmanova filtru, případně pomocí rozšířeného Kalmanova filtru. Byly otestovány celkem 3 modely pohybu. Od jednoduchého lineárního modelu až po komplexnější nelineární model, přičemž druhý jmenovaný dosahuje podle očekávání lepších výsledků. Výsledná přesnost odhadu je velmi dobrá a rychlost zpracování je také dostatečná.

Tento postup je tedy použitelný při určitých omezeních. Hlavním omezením je schopnost sledovat pouze jeden objekt. Pokud se v obraze bude pohybovat více objektů, bude potřeba určit, který je hledaný robot, případně, jestli ve scéně vůbec je. Dalším omezením je rychlost zpracování 20 snímků za sekundu. Ta je dostatečná pro pomalu se pohybující objekty. Vzhledem ke snímkovací frekvenci kamer, která je 30 snímků za sekundu zde již není příliš prostoru pro vylepšení. Pokud by tedy bylo cílem sledovat velmi rychlý pohyb, toto řešení by nebylo vhodné. Jednou z možností by bylo například využití vysokorychlostních kamer a úprava algoritmu. Například by bylo vhodné rozdělit výpočty mezi více zařízení. Provádět zpracování obrazu pro každou kameru zvlášť na samostatném zařízení (např. Beaglebone s využitím GPU a OpenGL shaderů) a výsledek potom zpracovat centrálně v dalším zařízení.

Literatura

[1] Michael Leonard, *How to Choose the Right Platform: Raspberry Pi or Beagle-Bone Black*, 2014 [vid. 2015-2-20], Dostupné online:

<http://makezine.com/magazine/how-to-choose-the-right-platform-raspber

[2] *Beagleboard*, [vid. 2015-3-12], Dostupné online:

<http://beagleboard.org/black>

[3] Ben Fry, Casey Reas, *Processing*, [vid. 2015-3-25] Dostupné online:

<http://www.processing.org/>

[4] *OpenCV Open Source Computer Vision*, [vid. 2015-3-27], Dostupné online:

<http://opencv.org/>

[5] *OpenGL - The Industry's Foundation for High Performance Graphics*, [vid. 2015-4-14], Dostupné online:

<https://www.opengl.org/>

[6] Sobral, A.C., *BGSLibrary: A OpenCV C++ Background Subtraction Library*, 2013 [vid. 2015-3-29], Dostupné online:

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

[7] C. Stauffer, W. Grimson, *Adaptive background mixture models for real-time tracking*, 1999 [vid. 2015-3-14], Dostupné online:

<http://www.ai.mit.edu/projects/vsam/Publications/stauffer_cvpr98_track.pdf>

[8] Ka Ki Ng, Edward J. Delp, *Background substraction using a pixel-wise adaptive learning rate for object tracking initialization*, 2011

[9] *OpenCV Camera Calibration and 3D Reconstruction*, [vid. 2015-5-14], dostupné online:

<http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html>

[10] Richard I. Hartley, Peter Sturm, *Triangulation*, 1994 [vid. 2015-5-14], dostupné online:

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

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

[12] Greg Welch, Gary Bishop, *An Introduction to the Kalman Filter*, 2001 [vid. 2015-7-8], dostupné online:

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

[13] X. Rong Li, Vesselin P. Jilkov, *Survey of Maneuvering Target Tracking. Part I: Dynamic Models*, 2003

[14] Lokukaluge P. Perera, Carlos Guedes Soares, *Ocean Vessel Trajectory Estimation and Prediction Based on Extended Kalman Filter*, 2010