VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

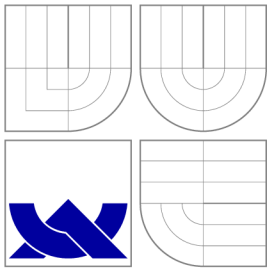
VYHLEDÁVÁNÍ KORESPONDUJÍCÍCH OBJEKTŮ VE DVOJICI SNÍMKŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

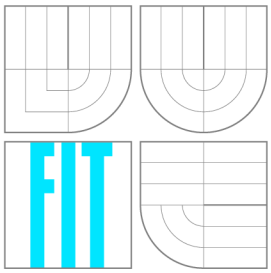
AUTOR PRÁCE
AUTHOR

ANDREJ VRBENSKÝ

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VYHLEDÁVÁNÍ KORESPONDUJÍCÍCH OBJEKTŮ VE DVOJICI SNÍMKŮ

SEARCH OF CORRESPONDING OBJECTS IN A PAIR OF IMAGES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ANDREJ VRBENSKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. FILIP ORSÁG, Ph.D.

BRNO 2011

Abstrakt

Práce shrnuje část problematiky stereoskopie a popisuje nejrozšířenejší metody na hledání korespondujících objeků ve dvojici stereo snímků. Zaměřuje se hlavně na metody hledání vzoru, založených na porovnávání intenzit. Tyto metody se snaží vylepšit za pomoci SIMD SSE instukcí. Výsledek je pak otestován na testovacích snímcích a vyhodnocen. Metody jsou implementovány v jazyce C++ a také v jazyce symbolických instrukcí.

Abstract

This thesis is aimed on area of stereoscopy. At first, there is some space dedicated to stereoscopic theory, primarily to seaching of corresponding objects in a stereo image pair. Main attention is given to template matching methods, which are based on intesity comparison. Then we try to optimize these methods with SIMD SSE instructions and run some tests with image examples. These methods are implemented in C++ and also in assembly language.

Klíčová slova

hledání objektů, stereoskopie, rektifikace, disparita, intensity-based metody, hledání vzorů, cross-correlation, SIMD, OpenCV, počítačové vidění

Keywords

object searching, stereoscopy, rectification, disparity, intensity-based methods, template matching, cross-correlation, SIMD, OpenCV, computer vision

Citace

Andrej Vrbenský: Vyhledávání korespondujících objektů ve dvojici snímků, bakalářská práce, Brno, FIT VUT v Brně, 2011

Vyhledávání korespondujících objektů ve dvojici snímků

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Filipa Orsága, PhD.

.....
Andrej Vrbenský
16. května 2011

Poděkování

Chtěl bych poděkovat svému vedoucímu Ing. Filipu Orságovi, PhD. za rady a odbornou pomoc pri řešení této bakalářské práce.

Obsah

Kapitola 1

Úvod

Od polovice 20. storočia sa z informatiky postupne stal jeden z najrýchlejšie rozvíjajúcich sa odborov. V tej dobe boli počítače obrovské stoje, no postupne ako rástla technologická vyspelosť, tak sa počítače miniaturizovali a zároveň sa zvyšoval ich výkon. Počítačové systémy sú v dnešnej dobe takmer všade okolo nás. Desktopy a laptopy zjednodušujú ľuďom život, poskytujú informácie a zábavu, superpočítače pomáhajú vedcom s náročnými výpočtami a urýchľujú tak vývoj nových technológií, mikropočítače sú súčasťou skoro každého elektronického zariadenia. Počítače boli vyvinuté aby ľuďom pomáhali, prípadne ich nahradili v náročných a nebezpečných činnostiach a podmienkach. Aby mohli fungovať, musia byť tieto počítače, buď ovládané človekom, alebo musia vedieť reagovať na vonkajšie vplyvy autonómne. Tu môžeme nájsť paralelu medzi zmyslami u človeka a rôznymi detektormi, ktoré používajú stroje. Napríklad zrak. Svetlo dopadá na sietnicu oka a v mozgu sa prijatý signál spracuje. Pri počítači svetlo dopadá na pole fotoprvkov. Kamera ho zaznamená ako obrazové dáta a následne ich procesor spracuje. Spracovanie obrazových dát spadá pod oblasť počítačovej grafiky a využíva sa v rôznych inteligentných systémoch, napríklad: detektory pohybu, navigácia robota, sledovanie objektov.

1.1 Ciele práce

Táto práca sa týka oblasti počítačovej grafiky, konkrétne spracovaním obrazu a počítačového videnia. Cieľom je zoznámenie sa s problematikou stereoskopie a s princípmi v nej používanými. Na základe získaných znalostí si vyberieme jeden z algoritmov používaných pre vyhľadanie objektu vo dvojici snímkov(v jednom stereo snímku) a implementujeme ho. Nakoniec na základe jeho výsledkov určíme disparitu. Našim cieľom bude vybrať dostatočne presný a zároveň rýchly algoritmus. Preto sa v práci budeme venovať aj optimalizácii algoritmu. Takto implementovaný algoritmus vhodne otestujeme a na testovacích snímkoch prevedieme experimenty. V nich sa zameriame aj na porovnanie s inými algoritmami. Výsledky experimentov vhodným spôsobom spracujeme a na ich základe zhodnotíme kvalitu algoritmov. Pod pojmom kvalita pritom myslíme hlavne ich presnosť a rýchlosť.

1.2 Obsah práce

V nasledujúcej 2 kapitole predstavíme obraz a jeho farebné modely, na ktoré môžeme pri spracovaní obrazu naraziť. Ukážeme si techniky, ktoré budú v práci použité, a rovnako tam nájdeme popisy najčastejšie používaných algoritmov. 3 kapitola pojednáva o oblasti

stereoskopie a snaží sa čitatela oboznámiť so základnými princípmi v stereoskopii. O naprogramovanej testovacej aplikácii sa dočítame v kapitole 4, ktorá popisuje jej návrh a implementáciu. Kapitola 5 obsahuje testy a experimenty, ktoré boli vykonané, ich popis a výsledky. Na konci nájdeme zhodnotenie práce a dosiahnutých výsledkov.

Kapitola 2

Spracovanie obrazu

Predtým ako môžeme s obrazom začať pracovať, mali by sme vedieť čo to obraz vlastne je a ako môže byť reprezentovaný. Ďalej sa bližšie pozrieme na niektoré základné pojmy a techniky používané pri spracovaní obrazu. Tieto princípy sú všeobecne veľmi využívané, rovnako ako aj v tejto práci, preto im treba venovať pozornosť. Väčšina týchto algoritmov je implementovaná aj v knižnici OpenCV[opencv]. Informácie v tejto kapitole boli čerpané z [16][5][17][6][9][8].

2.1 Digitálny obraz

Digitálny obraz dnes získame jednoducho. Stačí zobrať fotoaparát či kameru a začať snímať. Všetko automaticky spravia tieto prístroje, no funguje to zložitejšie. Za optikou kamery sa nachádza svetelný senzor, ktorý je dnes poväčšine vyrobený technológiou CMOS alebo CCD. Senzor je konštruovaný ako pole prvkov – pixelov, kde každý pixel obsahuje fotodetektor. Na fotodetektory jednotlivých pixelov dopadajú fotóny, ktoré produkujú tzv. fotoprúd. Ten sa pomocou príslušných obvodov prevedie na elektrické napätie a nakoniec na číselný údaj[5]. Potom sa často prevedie kompresia niektorým z používaných kodekov ako napr. JPEG. Výsledkom je teda digitálny obraz, inak nazývaný aj digitálny rastrový obraz. Rastrovým obrazom sa rozumie množina pixelov usporiadaná do tvaru matice, kde pixel nesie informáciu o svojej farbe a má svoju presnú pozíciu v obraze. Pixely sú usporiadané do dvoj dimenzionálneho pola – matice.

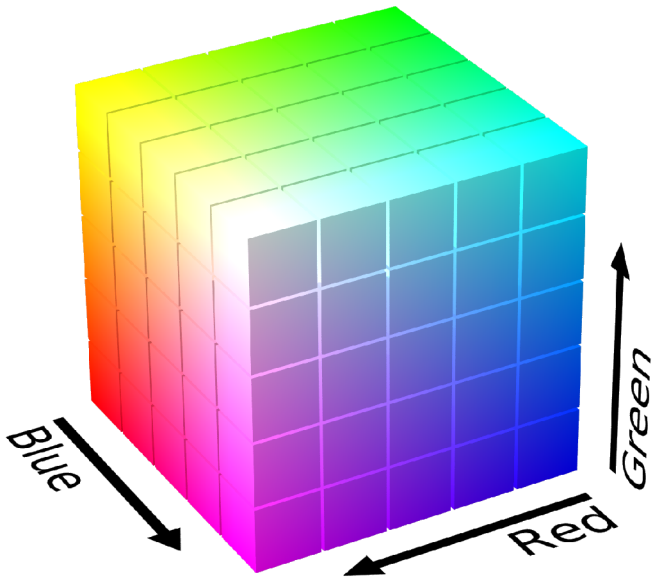
2.2 Farebné modely

Farba má v spracovaní obrazu svoje pevné miesto. Jej využitie ako deskriptoru, niekedy značne zjednodušuje identifikáciu objektov. Hlavne vtedy, ak sa jedná o spracovanie, pri ktorom sa aktívne zapája do procesu aj človek. Ten so svojím vnímaním farieb bude môcť lepšie rozlišovať scénu na farebnom obrázku, ako na obrázku v odtieňoch šedej. Preto sa spracovanie obrazu z pohľadu farby delí na dve oblasti: full-color processing a pseudo-color processing. Pri full-color sú obrázky spracovávané vo farebnom prevedení. Pri pseudo-color sú väčšinou priradené farbám určité intenzity – pracuje sa s obrázkami v odtieňoch šedej. Farebné modely sú matematickými modelmi, ktoré pomocou matematických štruktúr definujú farbu.

2.2.1 Model RGB

Predstavuje lineárny farebný priestor, kde základné farby sú červená, zelená a modrá o vlnových dĺžkach[8] $645,16\,nm$; $526,32\,nm$; $444,44\,nm$. Ďalšie farby sa získavajú aditívnym miešaním týchto základných farieb. RGB model sa používa vo väčšine vstupno-výstupných rozhraní, ktoré pracujú s video signálom, ďalej v monitoroch, televíziách, počítačovej grafike a mnohých ďalších prístrojoch. Výnimkou sú napríklad tlačiarne, kde sa používa model CMYK ktorý používa subtraktívne miešanie farieb. RGB model sa stal rozšíreným vďaka svojej jednoduchosti, a aj tomu, že je podobný so systémom vnímania farieb u človeka.

Jednotlivé pixely majú priradený troj dimenzionálny vektor tvaru (r, g, b), ktorý určuje intenzitu jednotlivých farebných zložiek a tým definuje farbu pixelu. Vektor $(0, 0, 0)$ bude vyjadrovať čiernu farbu a vektor (k, k, k) bielu farbu. Hodnota k určuje úroveň kvantizácie farebných zložiek. Využiteľný farebný priestor(angl. color space) je teda k^3 rôznych farieb. Pri 24-bit farebnom obrázku to znamená, že $k = 28$ (8-bit na každý farebný kanál), čo predstavuje $256^3 = 16777216$ využiteľných farieb. Tieto farby sú väčšinou zobrazované v tvare RGB kocky – obrázok 2.1.

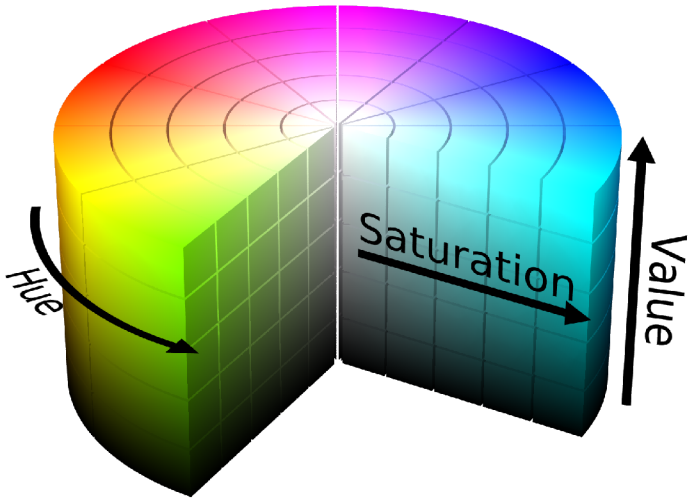


Obrázok 2.1: RGB farebný priestor[22]

2.2.2 Model HSV

Model HSV pracuje na inom princípe. Farba je definovaná pomocu troch hodnôt H – hue, S – saturation a V – value. Value je achromatická zložka, teda bez farby a jej jedinou vlastnosťou je intenzita. Hue je atribút spojený s dominantnou farbou a jej miešaním so svetlom, a teda reprezentuje dominantnú farbu tak, ako ju vidí pozorovateľ. Saturation je zase atribút, ktorý prestavuje hodnotu bieleho svetla, ktoré je zmiešané s atribútom hue. Čisté farby ako červená sú tzv. plne saturované a farby ako ružová (červená+biela) sú saturované menej. Hue a saturation sú spolu nazývané ako chromatické, a teda určujú farbu, zatiaľ čo value určuje intenzitu tejto farby. To, že je oddelená intenzita od farby umožňuje použiť niektoré

algoritmy, ktoré buď nieje možné, alebo je ťažké aplikovať na RGB model. Tento model je niekedy nazývaný aj HSB, kde B znamená brightness a značí to isté ako value.



Obrázok 2.2: HSV farebný priestor[20]

2.2.3 Model Grayscale

Alebo obraz v odtieňoch šedej, by sa dal charakterizovať ako pseudo-farebný model. Pseudo-farebné modely boli využívané viacej v minulosti. No s rozšírením plnofarebných kamier, fotoaparátov a monitorov sa z bežného života vytráca. Ale v spracovaní obrazu má stále svoje silné miesto, pretože existuje mnoho algoritmov, ktoré ho využívajú. Šedotónový obraz môžeme získať, buď použitím monochromatickej kamery alebo pomocou prevodu z RGB modelu, podľa jednoduchého vzťahu [11]:

$$I \ = 0.299R + 0.587G + 0.114B \tag{2.1}$$

Vidíme, že jedinou vlastnosťou pixelu je jeho intenzita. Tento prevod je vlastne redukciou farebného priestoru. Používa sa hlavne v takých prípadoch, keď nepotrebujeme informáciu o farbe, výstupné zariadenie nedisponuje podporou RGB farebného priestoru, alebo chceme použiť algoritmy na šedotónových obrazoch. Výhodou je, že takto prevedený obraz má menšiu veľkosť dát. Vo väčšine prípadov sa používa vzorkovanie na 8 bitov, čo nám dáva $2^8 = 256$ úrovní šedej farby.

2.3 Predspracovanie

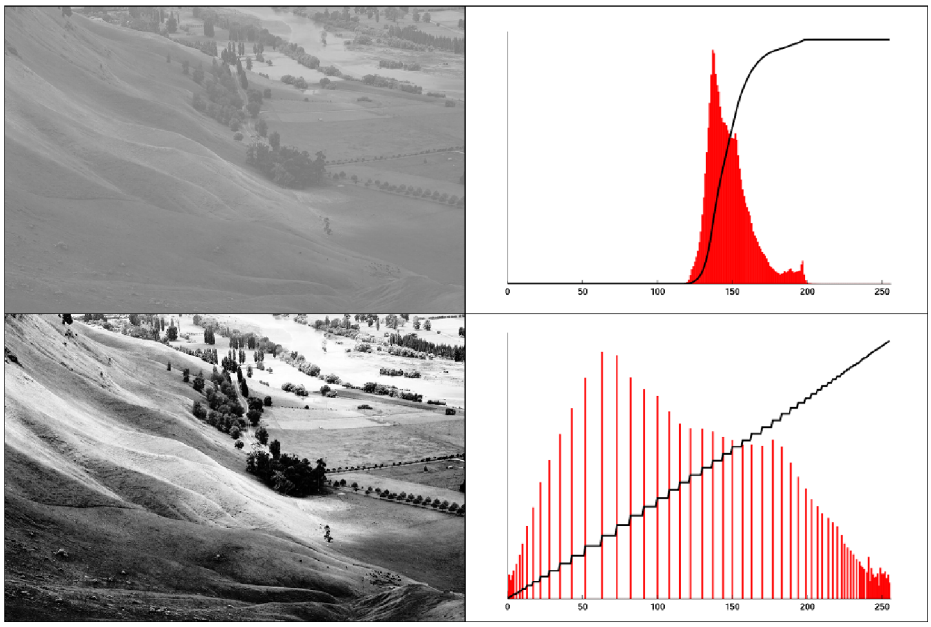
2.3.1 Histogram

Určenie histogramu vlastne nieje operácia s obrazom. Ale vyjadruje štatistiku výskytu šedotónových hodnôt v obraze. Histogram obrazu s X hodnotami šedej bude reprezentovaný jedno-dimenzionálnym poľom o veľkosti X prvkov, kde prvok je počet výskytov odpovedajúcej intenzíty v obraze[6]. Tu je treba si uvedomiť, že jeden histogram môže odpovedať

viacerým obrazom. Histogram sa využíva napríklad pri operáciach ako ekvalizácia histogramu(angl. histogram equalization), pri rôznych metódach segmentácie obrazu, pri metóde histogram matching, alebo hľadaní optimálnych svetelných podmienok. Histogram býva väčšinou zobrazovaný graficky, ako na obrázku 2.3. Vľavo sa nachádzajú obrázky a napravo ich histogramy.

2.3.2 Ekvalizácia histogramu

Máme histogram, ktorý obsahuje veľa výskytov blízkych intenzít, a ostatné intenzity sú slabo využité alebo vôbec. Potom obrázok, ktorý tento histogram reprezentuje bude mať malý kontrast. Pre metódy detekcie objektov je výhodnejší väčší kontrast, napríklad pre detekciu hrán, rohov. Preto použijeme operáciu ekvalizácia histogramu. Táto operácia zvyšuje globálny kontrast obrázku. Intenzity sú efektívne rozdistribuované pomocou vhodnej funkcie po celom histograme, pričom sa prejavuje aj vedľajší negatívny efekt – niektoré úrovne šedej sa vôbec nevyužijú. Ekvalizácia sa dá previesť aj na farebných obrázkoch – vtedy ju treba vykonať pre každý farebný kanál zvlášť. Nasledujúci obrázok 2.3 zobrazuje príklad ekvalizácie. Hore sa nachádza obrázok pred ekvalizáciou spolu s jeho histogramom. Dole je obrázok s histogramom už po ekvalizácii. Už na prvý pohľad je vidieť vyšší kontrast. Kým na prvom obrázku sa takmer nedajú rozoznať žiadne detaily, tak na druhom je ich už plno. Je to spôsobené tým, že v pôvodnom obrázku neboli vôbec zastúpené intenzity pod hodnotou 120, ani nad hodnotou 200, zato v novom sú zastúpené hodnoty z celého rozsahu intenzít.



Obrázok 2.3: Equalizácia histogramu[19]

2.3.3 Normalizácia histogramu

Výstupy niektorých použitých algoritmov nie sú v tvare, ktorý by sa dal zobraziť na obrazovku (napr. záporné hodnoty pixelov). Vtedy prichádza na radu normalizácia obrazu.

Normalizácia teda slúži na namapovanie súčasných hodnôt pixelov, na nové hodnoty, ktoré spadajú do určeného rozsahu. Prevod sa vykonáva nasledujúcim vzťahom [2]:

$$dest(x,y) = [src(x,y) - src_{min}]\frac{dest_{max} - dest_{min}}{src_{max} - src_{min}} + dest_{min} \tag{2.2}$$

Kde src je intenzita pixelu na pozícii (x,y), src_{max} a src_{min} je rozsah intenzít v pôvodnom obrázku, $dest_{max}$ a $dest_{min}$ je požadovaný nový rozsah. Pracuje to nasledujúcim spôsobom: aktuálnu spodnú hranicu posunieme na nulu pomocou $src(x,y) - src_{min}$, potom súčinom s výrazom $\frac{dest_{max}-dest_{min}}{src_{max}-src_{min}}$ zmeníme rozsah intenzít, a pripočítaním $dest_{min}$ posunieme do požadovaného rozsahu.

2.3.4 Filtrovanie

Slovo filter je prebrané z oblasti spracovania signálu, kde filtre slúžia na modelovanie (upravovanie) vstupného signálu. Napríklad horná priepusť neprepúšťa signál nízkych frekvencií. Na podobné účely(úpravu) slúžia filtre aj v 2D priestorovom spracovaní obrazu. Výstupy z kamier môžu byť zašumené, obraz rozostrený. Pomocou filtrov dokážeme tieto problémy riešiť.

Proces filtrácie môže byť podľa [9] vyjadrený výrazom:

$$g(x,y) = T[f(x,y)] \tag{2.3}$$

kde $f(x,y)$ je obrázok na vstupe, $g(x,y)$ obrázok na výstupe, T je operácia nad obrazom f definovaným okolím pixelu na pozícii (x,y). Filter je teda určený okolím pixelu – neighborhood, ktoré býva väčšinou štvoruholníkového tvaru, a filtrovacou operáciou, ktorú vykonáva nad pixelmi obrazu určenými týmto okolím. Výsledkom filtrovania je nový obraz, ktorého hodnoty pixelov sú vypočítané filtrovacou operáciou, a ich súradnice odpovedajú stredu okolia filtra nad obrazom. Tento stred sa pritom postupným posúvaním dostane nad každý pixel obrazu.

Pri filtrovaní je treba spomenúť dve dôležité operácie a to konvolúciu a koreláciu (tabuľka 2.1). Aj tieto operácie pochádzajú z oblasti spracovania signálu, kde patria k jedným z najpoužívanejších. Ich definíciu popisujú vzťahy v tabuľke(ref na tabuľku). Niektoré známe filtre ako Gaussian, Laplacian of Gaussian či Sobel-ov operátor používajú masku(tzv. konvolučná maska), ktorú konvolujú s filtrovaným obrázkom. Využitie korelácie zase nachádzame napríklad medzi algoritmami na hľadanie vzoru, kde korelácia určuje hodnotu podobnosti dvoch obrazov.

Konvolúcia	$f(x,y) * h(x,y) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(m,n)g(x-m,y-n)$
Korelácia	$f(x,y) \star h(x,y) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(m,n)g(x+m,y+n)$

Tabuľka 2.1: Konvolúcia a korelácia

2.3.5 Zmenšenie obrazu

Zmenšenie obrázku(angl. downsampling) by v tejto práci mohla byť využitá pri vysokých rozlíšeniach vstupných obrázkov. Obrázok transformujeme na menšie rozlíšenie, čím urýchlime prácu niektorým algoritmom. Na výber je niekoľko algoritmov, kde niektoré sú rýchlejšie ako iné, a niektoré produkujú kvalitnejší výstup ako iné. Závisí len na vhodnosti použitia, ktorý si z nich vyberieme. Spomeňme napríklad interpoláciu, alebo Lanczosov algoritmus. Zmenšením obrazu však môžeme stratiť niektoré detaily scény. A ako uvidíme pri popise algoritmov, detaily sú dôležité pre lepšie a pre presnejšie hľadanie objektov.

2.3.6 Diskrétna Fouriérova transformácia (DFT)

Fourierova transformácia sa používa dnes takmer všade. Vo fyzike, elektrotechnike, informatike, atd. My sa budeme zaoberať jej využitím v informatike, konkrétnejšie v oblasti spracovania obrazu. Tu sa používa jej diskrétna varianta, keďže pracujeme s diskrétnymi hodnotami – pixelmi v rastrovom obraze. Táto sekcia čerpá z [9].

Pri priamej implementácii DFT (tabuľka 2.2) je algoritmus veľmi pomalý. Spotrebuje až $(MN)^2$ sčítaní a násobení, takže by nemalo význam vôbec DFT používať. Našťastie sú dostupné algoritmy, ktoré časovú zložitosť transformácie znižujú. Napríklad algoritmus Rýchlej Fourierovej transformácie (Fast Fourier Transform) od pánov Cooley a Tuckey, má zložitosť už len $MN \log_2 MN$ operácií. Práve tento algoritmus spôsobil rýchly rozvoj využívania Fourierovej transformácie v praxi.

1D DFT	$F_m = \sum_{n=0}^{M-1} f_n e^{-j2\pi mn/M} \quad m = 0, 1, 2, \ldots, M-1$
1D IDFT	$f_n = \frac{1}{M} \sum_{m=0}^{M-1} F_m e^{j2\pi mn/M} \quad n = 0, 1, 2, \ldots, M-1$
2D DFT	$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi(ux-M+vy-N)}$
2D IDFT	$f(x,y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) e^{j2\pi(ux-M+vy-N)}$

Tabuľka 2.2: Vzťahy pre diskrétnu Fourierovu transformáciu

2D DFT pre obrázok, môže byť počítaná priamo 2D transformačným vzťahom, alebo je možné pre jej výpočet použiť 1D DFT pre každý riadok obrázka, a následne 1D DFT pre stĺpce výsledkov prvej transformácie. Transformáciu využijeme pri filtrovaní, kde obrázok a filter najprv transformujeme do frekvenčnej domény. Tu prevedieme požadovanú operáciu a obrázok prevedieme naspať z frekvenčnej domény do priestorovej pomocou inverznej diskrétnej Fourierovej transformácie. Operácia, ktorá by bola vykonaná v priestorovej doméne, pritom nie je rovnaká ako vo frekvenčnej. Existujú takzvané DFT transformačné páry[9], ktoré definujú aká operácia v priestorovej doméne odpovedá akej operácii vo frekvenčnej doméne. Tento proces je výhodný preto, lebo niektoré operácie sú menej časovo náročné vo frekvenčnej doméne. My využijeme tzv. korelačný teorém,

$$f(x,y) \star h(x,y) \quad \Leftrightarrow F^*(u,v)H(u,v) \tag{2.4}$$

$$f^*(x,y)h(x,y) \quad \Leftrightarrow F(u,v) \star H(u,v) \tag{2.5}$$

z ktorého vzťahu môžeme vidieť, že korelácia je nahradená násobením. A keďže korelácia samotná obsahuje násobenia a sčítania, tak algoritmy využívajúce koreláciu budú rýchlejšie s použitím DFT. Platí to však len pri náročných úlohách, kde bude čas strávený samotnou transformáciou (DFT + IDFT) oproti výpočtom zanedbateľný. Inak by sa mohlo stať, že algoritmus s transformáciou bude paradoxne pomalší.

2.4 Algoritmy pre vyhľadávanie objektov

Momentálne existuje veľa algoritmov na detekciu objektov. Niektoré sú len vylepšeniami starších, iné sa zakladajú na úplne nových princípoch. My sa budeme zaoberať oblasťou hľadania vzoru – template matching. Vzor je niečo ako model, tvar, v našom prípade to bude výrez obrazu objektu. Hľadaním sa rozumie operácia, pri ktorej budeme porovnávať vzor na jeho zhodu, či nezhodu. Vzor sa porovnáva s celým snímkom, v ktorom ho chceme nájsť a jeho pravdepodobnú pozíciu určuje miesto s najväčšou mierou zhody, prípadne najmenšou mierou nezhody. V tejto oblasti hľadania vzoru sú známe dva prístupy k hľadaniu, a to: princíp založený na intenzite – intesity-based a princíp založený na vlastnostiach/rysoch – feature-based.

2.4.1 Intensity-based

Niekedy nazývané aj window-based, alebo lokálne algoritmy, využívajú hodnotu intenzity jednotlivých pixelov. Vždy sa pracuje so všetkými pixelmi vzoru aj porovnávaného obrázka, preto sú časovo náročné. Rýchlosť algoritmov môžeme zvýšiť buď zmenšením rozlíšenia obrázku, ale zmenšením prehľadávanej oblasti v obrázku. Základný princíp týchto algoritmov je nasledujúci: Vzor postupne posúvame po celom obrázku, kde pre každú jeho pozíciu vypočítame pomocou jedného z algoritmov mieru jeho zhody alebo mieru jeho nezhody. Táto hodnota určuje podobnosť medzi vzorom a časťou obrázku nad ktorou sa práve vzor nachádza. Potom z jednotlivých hodnôt vyberieme najvyššiu alebo najnižšiu, podľa toho, aký algoritmus použijeme. Môžeme usúdiť, že tieto algoritmy požívajú princíp Winner Take All(WTA), a teda zhoda nastáva vždy iba na jednej pozícii v obrázku. Viac zhodných pozícií v obrázku nemôže existovať. Tieto algoritmy fungujú najlepšie vtedy, keď sú vyhľadávané objekty čo najviac texturované, a naopak zlyhávajú pri objektoch so súvislými plochami bez textúr. Taktiež treba povedať, že nie sú invariantné voči geometrickým transformáciám ako zväčšenie/zmenšenie či rotácia. Tým pádom môžeme povedať, že najlepšie výsledky budú podávať, keď obrázky budú snímané v rovnakej orientácii oboch kamier, ktoré budú blízko seba. Ďalej nasleduje prehľad najpoužívanejších algoritmov. Tieto algoritmy vychádzajú z funkcie vzdialenosti (distance function).

Sum of absolute differences Vychádza z Euklidovej vzdialenosti, kde je vzdialenosť v jedno-dimenzionálnom priestore definovaná ako absolútna hodnota rozdielu hodnôt dvoch bodov:

$$\sqrt{(p-q)^2} = |p-q| \tag{2.6}$$

V našom prípade sú týmito hodnotami intenzity pixelov. Tento algoritmus počíta mieru nezhody obrázku so vzorom. Je definovaný vzťahom[10]:

$$SAD(x,y) = \sum_{i=0}^{T_{rows}} \sum_{j=0}^{T_{cols}} |T(i,j) - I(x+i, y+j)| \tag{2.7}$$

V ktorom I predstavuje intenzitu pixelu na pozícii (x, y) v prehľadávanom obrázku a T intenzitu pixelu na pozícii (i, j) vo vzore. Vzťah počíta súčet absolútnych hodnôt rozdielov intenzít pre jednotlivé odpovedajúce si pixely obrázku a vzoru. Tento súčet je vykonaný pre jednotlivé riadky, ktoré sú potom spočítané dohromady. Keďže algoritmus určuje mieru nezhody, tak potom najnižšia hodnota SAD určuje pozíciu najpravdepodobnejšej zhody vzoru v obrázku.

Sum of squared differences Aj tento algoritmus počíta mieru nezhody obrázku so vzorom. Má väčšiu výpočetnú náročnosť, nakoľko obsahuje násobenie. Je podobný algoritmu SAD, ale namiesto absolútnej hodnoty obsahuje druhú mocninu. Tá spôsobuje, že intenzity od seba vzdialené viacej budú ohodnotené väčšou váhou. Algoritmus je definovaný vzťahom:

$$SSD(x,y) = \sum_{i=0}^{T_{rows}} \sum_{j=0}^{T_{cols}} T(i,j) - I(x+i, y+j)^2 \tag{2.8}$$

$I(x, y)$ je intenzita pixelu v prehľadávanom obrázku a $T(i, j)$ je intenzita pixelu vo vzore. Vzťah počíta súčet druhých mocnín rozdielov intenzít pre jednotlivé odpovedajúce si pixely obrázku a vzoru pre každý riadok. Súčty riadkov sú potom spočítané dohromady. Keďže algoritmus určuje mieru nezhody, tak najnižšia hodnota SSD určuje pozíciu najpravdepodobnejšej zhody vzoru v obrázku.

Cross Correlation Metóda korelácie v hľadaní vzoru vychádza z Euklidovej vzdialenosti podľa [3]:

$$\int (f-g)^2 = \int f^2 + \int g^2 - 2\int (f \cdot g \tag{2.9}$$

Kde výraz $\int g^2$ predstavuje energiu vzoru a je koštantný, pretože sa nemení – vzor je celý čas rovnaký. Výraz $\int f^2$ je energiou porovnávanej oblasti, a so zmenou polohy sa mení len minimálne. Preto ho považujeme tiež za konštantu. Zostávajúci výraz je cross korelácia, ktorá vyjadruje mieru zhody vzoru s obrázkom. V diskrétnej forme:

$$CORR(x,y) = \sum_{i=0}^{T_{rows}} \sum_{j=0}^{T_{cols}} T(i,j) I(x+i, y+j) \tag{2.10}$$

Kde $I(x, y)$ a $T(i, j)$ sú hodnoty intenzít pixelov na zadaných pozíciách v obrázku a vo vzore. A keďže vyjadruje mieru zhody, tak najvyššia hodnota $CORR$ určuje pozíciu najpravdepodobnejšej zhody vzoru v obrázku.

Je tu možnosť využitia korelačného teorému DFT. Najskôr je obrázok aj vzor prevedený pomocou Fourierovej transoformácie do frekvenčnej domény viď 2.2. Potom je na základe

korelačného teorému vykonaná operácia a obrázok je prevedený naspäť z frekvenčnej do priestorovej domény za pomoci inverznej fourierovej transformácie[9]:

$$CORR(x, y) = IDFT[DFT(T)DFT^*(I)] \tag{2.11}$$

Normalized cross correlation U jednoduchej korelácie nastáva problém pri porovnaní s miestami, ktoré obsahujú súvislé plochy vysokých hodnôt intenzít, napr. biele flaky, odlesky svetla. Takéto miesta môžu získať lepšie ohodnotenie, ako je skutočná pozícia vzoru v obrázku, čo vedie k chybnej detekcii. Tieto problémy rieši normalizácia [8]:

$$CORR(x, y) = \frac{\sum_{i=0}^{T_{rows}} \sum_{j=0}^{T_{cols}} T(i, j)I(x + i, y + j)}{\sum_{i=0}^{T_{rows}} \sum_{j=0}^{T_{cols}} I(x + i, y + j)^2} \tag{2.12}$$

Vidíme, že numerátor obsahuje koreláciu. Teda tento algoritmus rovnako ako nenormalizovaná korelácia určuje mieru zhody. I je intenzita pixelu na pozícii (x, y) v prehľadávanom obrázku. T je intezita pixelu na pozícii (i, j) vo vzore. Najvyššia hodnota NCC určuje pozíciu najpravdepodobnejšej zhody vzoru v obrázku. Môžeme použiť variantu, kde koreláciu v numerátore vypočítame vo frekvenčnej doméne.

Normalized correlation coefficient Je ďalšia modifikácia korelačného algoritmu, pričom sa snaží riešiť problémy, ktoré vzikajú pri rozličnom nasvietení scény, odrazoch svetla na jednotlivých porovnávaných obrázkoch. Používa nato tzv. korelačný koeficient, ktorý normalizuje vektory vzoru a obrázku[12]:

$$COEF(x, y) = \frac{\sum_{i=0}^{T_{rows}} \sum_{j=0}^{T_{cols}} [T(i, j) - \overline{T}][I(x + i, y + j) - \overline{I}]}{\sum_{i=0}^{T_{rows}} \sum_{j=0}^{T_{cols}} [I(x + i, y + j) - \overline{I}]^2 \sum_{i=0}^{T_{rows}} \sum_{j=0}^{T_{cols}} [T(i, j) - \overline{T}]^2} \tag{2.13}$$

Vidíme, že numerátor obsahuje koreláciu. Takže algoritmus určuje rovnako ako korelácia mieru zhody. $\overline{T}$ je priemer vzoru, $\overline{I}$ je priemer oblasti obrázku, nad ktorou sa nachádza vzor. I je intenzita pixelu na pozícii (x, y) v prehľadávanom obrázku. T je intezita pixelu na pozícii (i, j) vo vzore. Najvyššia hodnota $COEF$ určuje pozíciu najpravdepodobnejšej zhody vzoru v obrázku.

Vidíme, že oproti ostatných predstaveným algoritmom, bude tento časovo náročnejší, kedže obsahuje viacej operácií. Ale existuje možnosť, ako ho zrýchliť. Priemer vzoru sa vypočíta iba raz, a potom pracujeme len s jeho uloženou hodnotou. Výraz v numerátore, teda korelácia môže byť vypočítaná vo frekvenčnej doméne a výraz v denominátore môže byť predpočítaný pomocou running sum tables. Viac môžete nájsť v [12].

2.4.2 Feature-based

Tento prístup sa spolieha na silné vlastnosti obrázku, ako sú napríklad rohy a hrany. Rôzne algoritmy používajú rôzne vlastnosti. Každému takémuto bodu je priradený jeho deskriptor.

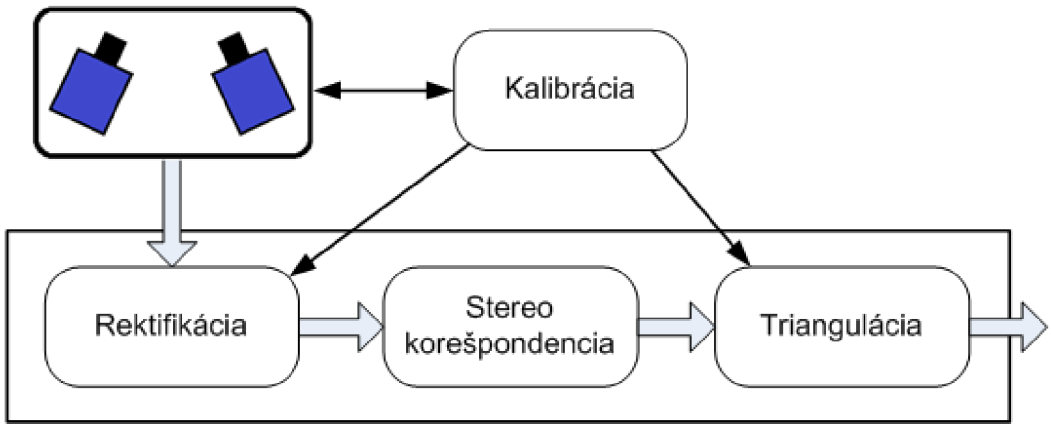
Opäť každý algoritmus využíva svoj vlastný deskriptor. K množine deskriptorov sa potom hľadá odpovedajúca množina deskriptorov v druhom obrázku. Niektoré z týchto algoritmov majú výborné výsledky aj vtedy, keď sú objekty na dvoch snímkoch značne posunuté, skosené, otočené alebo aj zväčšené či zmenšené. Hovoríme im, že sú invariantné ku týmto geometrickým transformáciám. Patria sem napríklad algoritmy SIFT a SURF[1]. Týmto algoritmom sa v tejto práci nebudeme venovať, no sú spomenuté preto, aby sme poskytli kompletné informácie.

Kapitola 3

Stereoskopia

Ľudský zrak nám umožňuje vnímať hĺbku priestoru. Je to možné vďaka tomu, že máme dve oči. Mozog spracuje obrazy z jednotlivých očí ako signál, a pomocou rozdielu (disparity) v nich interpretuje informácie o hĺbke. Podobne to funguje aj v stereoskopii. Keby sme mali k dispozícii len jeden snímok, je možné z neho dostať mnoho užitočných informácií, ale predstavu o hĺbke scény nám neposkytne. Je to rovnaké, ako keby sme si zakryli jedno oko – stratíme vnímanie hĺbky. Nato, aby sme mohli určiť hĺbku scény, sú potrebné minimálne dva snímky, nazývané aj stereosnímok. Hĺbka sa potom určí pomocou triangulácie, pre ktorú je potreba určiť disparitu medzi korešpondujúcimi bodmi jednotlivých snímkov. Zistenie korešpondujúcich bodov je pritom jedna z najnáročnejších otázok stereoskopie. V anglickej literatúre to nájdeme pod pojmom Correspondence Problem. Existuje niekoľko algoritmov riešiacich tento problém, pričom každý má určité výhody aj nevýhody. Táto kapitola nás oboznámi, čím sa zaoberá stereoskopia a aké problémy musí riešiť. Ako zdroje pre túto kapitolu boli použité [13][17][14][8].

3.1 Stereo systém



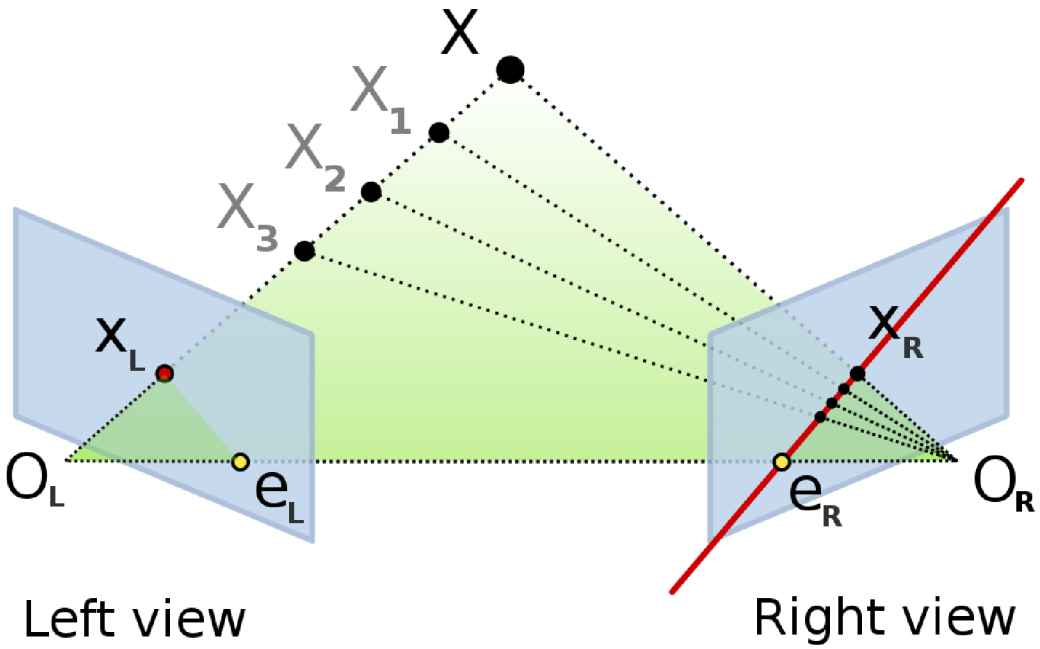
Obrázok 3.1: Stereo systém

Stereo systém sa skladá z niekoľkých procesov, ktoré môžeme vidieť na obrázku 3.1. Predtým ako začneme snímať, musíme previesť kalibráciu dvoch kamier. Tu sa väčšinou používa šachovnicová doska, pretože sa pri nej ľahko identifikujú korešpondujúce body

a kalibračný softvér. Nástroje na kalibráciu poskytujú napr. aj program Matlab a knižnica OpenCV. Po tom ako sú kamery skalibrované, môžeme začať snímať. Snímky potom prechádzajú rektifikačným procesom, pri ktorom sa využijú extrinsic a intrinsic parametre získané pri procese kalibrácie. Po rektifikácii pokračujeme hľadaním korešpondujúcich bodov a s výpočtom disparity. Nakoniec pomocou triangulácioU určíme hĺbku.

3.2 Geometria stereo kamier

Nasledujúci obrázok 3.2 zobrazuje geometriu dvoch stereo kamier, nazývanú aj epipolárna geometria. Ľavý obrázok (left view) je referenčný snímok, v ktorom označíme bod, ktorého korešpondenciu chceme vyhľadať. Pravý obrázok (right view) je cieľový snímok, v ktorom túto korešpondenciu hľadáme. O_R a O_T sú optické stredy týchto snímkov. Úsečka spájajúca optické stredy sa nazýva báza. X, $X1$, $X2$, $X3$ sú body v priestore snímanom kamerami. Vidíme, že všetky body sú z pohľadu ľavej kamery v priestore presne za sebou. To znamená, že sa premietnu v ľavom obrázku na totožnú pozíciu označenú X_L. Tá je určená priesečníkom plochy obrázku a priamkou vedenou z optického stredu O_R cez body X, $X1$, $X2$, $X3$. Môžeme pozorovať, že v druhom obrázku sa body všetky body X, $X1$, $X2$, $X3$ premietnu na jednu priamku. Tú nazývame epipolárna línia.



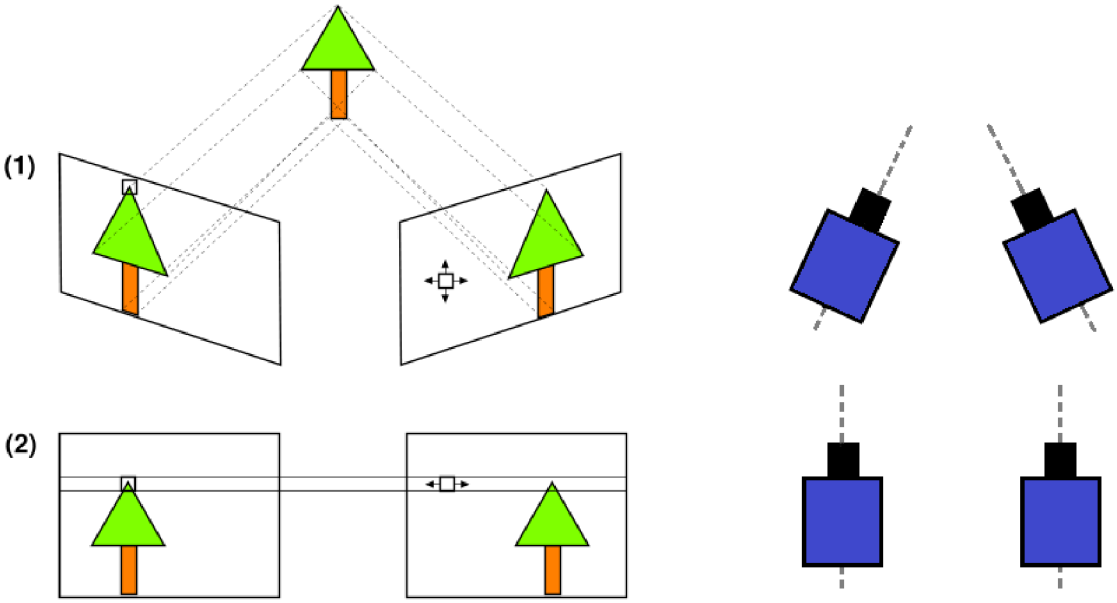
Obrázok 3.2: Epipolárna geometria[18]

Tento princíp je známy ako epipolárne obmedzenie. Epipolárne obmedzenie nám teda vyjadruje, že korešpondujúci bod k bodu z ľavého obrázka, sa nachádza na epipolárnej línii v pravom obrázku. Je zjavné, že pomocou tejto metodiky môžeme rapídne zmenšiť oblasť vyhľadávania korešpondujúcich bodov. Z prehľadávanej oblasti celého obrázka, nám stačí prehľadať iba oblasť definovanú epipolárnou líniou. Zelená oblasť sa nazýva epipolárna plocha. Je určená bázou spájajúcou optické stredy a úsečkami spájajúcimi jednotlivé optické

stredy s hľadaným bodom v priestore. Naproti tomu epipolárne línie sú určené priesečníkom epipolárnej plochy a plochy obrázku.

3.3 Rektifikácia stereo snímku

Keď vieme, ako pomocou epipolárneho obmedzenia zmenšiť prehľadávanú oblasť z 2D priestoru (celý obrázok) na 1D priestor (epipolárna línia), pomohlo by nám keby stereo obrázky boli vo vhodnejšej forme. Najlepšie by bolo, keby jednotlivé epipolárne línie predstavovali riadky v obrázku, a aby boli takto odpovedajúce riadky jednotlivých obrázkov zarovnané. Tohto dosiahneme, ak použijeme proces nazývaný rektfikácia[szeliski]. Tento proces nám odstráni geometrické a fotometrické deformácie vznikajúce rôznymi polohami kamier a ich natočením, alebo distorziou spôsobenom objektívom kamier. Rektifikácia pretransformuje stereo obrázky do spoločnej plochy, paralelnej k báze systému. Táto transformácia sa prejaví tak, že obrázky budú vyzerať ako by sme kamery nastavili do štandardnej formy (obrázok 3.3 vpravo dole). Potreba rektifikácie sa dá minimalizovať správnym nastavením hardvéru. Napríklad kamery môžeme zarovnať do štandardnej formy ručne čo najpresnejšie a silno ich ukotviť, aby sa zabránilo ich posunutiu. Na rektifikáciu sa väčšinou (keď sú dostupné) používajú tzv. extrinsic a intrinsic informácie získané pri kalibrácii kamier. Ak bol kalibračný krok vynechaný, je to zložitejšie. Musí sa použiť odlišný algoritmus. Napríklad algoritmus uvedený v [fusiello] potrebuje namiesto kalibračných nastavení poznať určitý počet korešpondujúcich bodov medzi obrázkami a na základe tých si vypočíta potrebné transformačné vzťahy. Rektifikačný algoritmus pracujúci s kalibračnými informáciami môžete nájsť napríklad v [4].



Obrázok 3.3: Rektifikácia[21]

3.4 Stereo korešpondencia

Ako už bolo spomenuté určenie korešpondencií je jedných z hlavných problémov stereoskopie. Je to zapríčinené hlavne tým, že algoritmy spotrebujú veľké množstvo výpočetného času a niektoré systémy potrebujú pracovať real-time. Ďalej sa tu pridávajú ďalšie problémy spôsobené rozdielnou polohou kamier a svetlom dopadajúcim v rozdielnych uhloch. Medzi ne patria rôzne distorzie, nesúmernosť, šum, odlesky či problémy s priehľadnými premetmi[13].

K problému korešpondencie poznáme dva prístupy, tzv. sparse correspondence a dense correspondence. Sparse correspondence sa k tomu stavia takto: najskôr sa v obrázku určia silné vlastnosti, ako napr. rohy, hrany, čo značí feature-based princíp. Každá takáto vlastnosť je popísaná deskriptorom. K nim sa potom vyhľadávajú korešpondencie v druhom obrázku. Buď sa vyhľadáva podľa porovnávania deskriptorov, alebo je použitý niektorý z lokálnych algoritmov. Následne pre 3D rekonštrukciu celej scény sa použijú špecializované interpolačné algoritmy, ktoré sú schopné dopočítať niektoré chýbajúce body. Je zjavné, že obrázok musí mať dostatok silných vlastností, inak bude vyhľadávanie korešpondujúceho objektu neúspešné, prípadne veľmi nepresné. Pri dense correspondence sa hľadá korešpondujúci bod v druhom obrázku, pre každý pixel obrázku prvého. Používajú sa hlavne lokálne window-based algoritmy[17]. Tie fungujú najlepšie vtedy, keď sú vyhľadávané objekty čo najviac texturované a kamery sú v blízkej pozícii. Vtedy sa jednotlivé obrázky veľmi nelíšia, a hľadanie vzoru podáva veľmi dobré výsledky.

3.5 Disparita

Po tom ako máme učenú polohu jednotlivých korešpondujúcich bodov, môžeme prejsť k výpočtu disparity. Disparita môže byť horizontálna alebo vertikálna. Nás zaujíma iba horizontálna, pretože už z postavenia kamier a následnej rektifikácie je jasné, že poloha objektu sa bude odlišovať iba v horizontálnej polohe. Disparita sa teda vypočíta jednoducho ako rozdiel v horizontálnych polohách korešpodujúcich bodov:

$$d = x_L - x_R \tag{3.1}$$

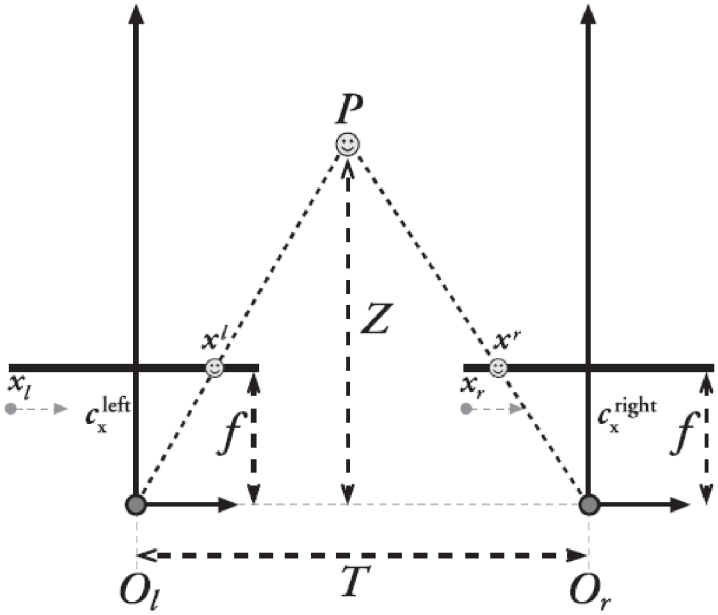
Výborne je to znázornené na 3.4. Pri niektorých aplikáciách ako napríklad rendering, je potrebná vysoká presnosť disparity. Vtedy sa v rámci pospracovania obrazu používajú metódy na určenie disparity so sub-pixelovou presnosťou.

3.6 Triangulácia

Disparita úzko súvisí s trianguláciou. Triangulácia je proces, pri ktorom pomocou disparity v stereo snímkoch určíme hĺbku, v ktorej sa nachádza korešpondujúci objekt. Z (obrázku) je jasné, že môžeme odvodiť vzťah medzi hĺbkou a disparitou, pomocou metódy podobnosti trojuholníkov[4]:

$$\frac{T - d}{Z - f} = \frac{T}{Z} \tag{3.2}$$

A z toho vyjadrením hĺbky (vzdialenosti) dostaneme:



Obrázok 3.4: Disparita a hĺbka scény[4]

$$Z = \frac{f \cdot T}{d} \tag{3.3}$$

Kde Z je hĺbka, v ktorej objekt leží v priestore. f je ohnisková vzdialenosť, T je veľkosť bázy a d je disparita určená podľa 3.4. Vidíme, že čím je disparita väčšia, tým sa objekt nachádza bližšie k pozorovateľovi a naopak. Disparita je teda inverzná k hĺbke. Po vypočítaní hĺbky pre všetky body snímku dostaneme hĺbkovú mapu – depth map. Tá znázorňuje hĺbku pre jenotlivé body obrázka a používa sa ďalej napríklad pri vyhodnocovaní pohybu autonómnych robotov v priestore, či pre rekonštrukciu 3D scény, objektu. Ak sa snažíme určiť len polohu určitého objektu, stačí nám určiť len jedinú disparitu, a to práve pre tento objekt. Potom z kalibračných informácií a disparity môžeme určiť jeho vzdialenosť.

Kapitola 4

Návrh a implementácia

Táto práca sa zaoberá vyhľadávaním objektov vo dvojici snímkov, teda budeme implementovať aplikáciu, ktorá rieši korešpondenčný problém. To je proces na obrázku stereo systému 3.1. Budeme teda predpokladať, že máme k dispozícii stereo páry obrázkov, ktoré boli nasnímané pomocou skalibrovaných kamier, a ktoré už prešli rektifikačným procesom. Na týchto obrázkoch vyskúšame vybrané implementované algoritmy – tie určia korešpondujúce objekty. A potom na základe ich výsledku určíme disparitu. Aby sme overili funkčnosť a rýchlosť jednotlivých implementácií algoritmov, tak prevedieme rôzne testy a experimenty.

4.1 Použité programové prostriedky

Programová časť práce, teda samotná aplikácia na testovanie, je napísaná v jazyku *C/C++*. Využité sú hlavne rysy jazyka C++, ktorý umožňuje objektovo orientovaný prístup k písaniu aplikácií. Programovali sme vo vývojovom prostredí Microsoft Visual Studio 2010 s následným prekladom pod Visual C++ prekladačom. V tomto prostredí bol vygenerovaný aj diagram tried zobrazený nižšie.

Ďalej bola použitá open-source knižnica *OpenCV*[4] vo verzii 2.1. Je to knižnica pre počítačové videnie a spracovanie obrazu, ktorú pôvodne vyvíjala firma Intel. Dnes je však voľne dostupná pod licenciou BSD. Je dostupná pre niekoľko programovacích jazykov a je multiplatformná. Nájdeme v nej obrovské množstvo implementovaných algoritmov. Ak by bola potreba vysokého výkonu za každú cenu, tak je možné si od firmy Intelu prikúpiť tzv. Intel Integrated Performance Primitives (IPP). To je knižnica obsahujúca vysoko optimalizované algoritmy podporujúce viac-jadrové procesory a úzko spolupracujúca s OpenCV. OpenCV obsahuje aj štruktúry a objekty pre reprezentáciu a načítanie/ukladanie videa a obrázkov, či knižnicu highgui pre tvorbu jednoduchých užívateľských rozhraní. Vďaka tomu ak nechceme, tak nemusíme použiť žiadnu ďalšiu externú knižnicu.

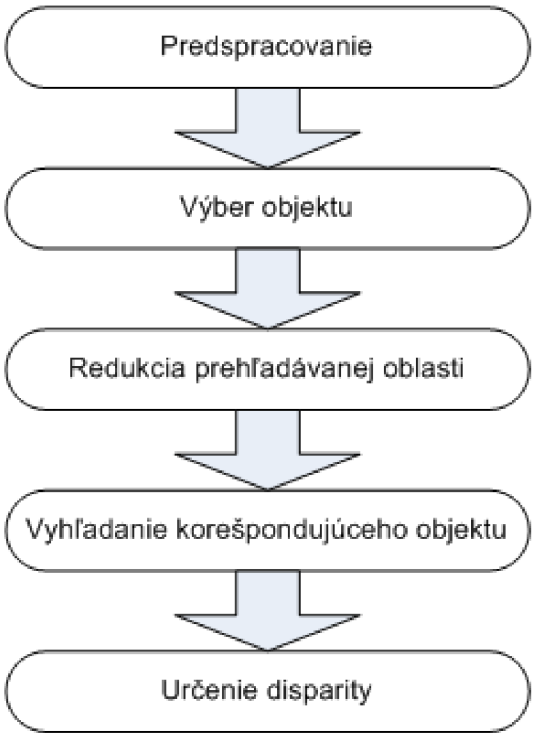
Pri optimalizácii bol použitý aj *jazyk symbolických inštrukcií ASM*. Je to nízkoúrovňový programovací jazyk, ktorý je špecifický pre konkrétnu platformu. S ním boli použité aj SIMD inštrukcie *MMX/SSE*. SIMD (single instruction multiple data) inštrukcie sú špeciálne inštrukcie procesoru, ktoré dokážu vrámci jednej inštrukcie spracovať väčšie množstvo dát naraz. Viac k SIMD inštukciám môžeme nájsť v Intel manuáloch[7]. Algoritmus implementovaný v ASM bol preložený do knižnice typu dll. Pre preklad poslúžil prekladač NASM a linker ALINK. Bol použitý aj súbor makier GENERAL.MAC od doktora Filipa Orsága.

Na testovacie účely boli použité obrázky z Middlebury Stereo Datasets[15]. Obrázky

sú snímané zo 7 pozíc, za použitia 3 rozdielnych osvetlení. Tieto stereo obrázky sú už rektifikované, a sú knim dostupné kalibračné informácie a mapy disparít.

4.2 Návrh

Návrh sa zakladá na metóde zdola-nahor. Celý problém sme rozdelili na viacej malých podproblémov (jednotlivé algoritmy), ktoré sme postupne pospájali do väčších celkov. Ďalej návrh využíva princípy objektovo orientovaného programovania. Jednotlivé obrázky budú reprezentované objektom typu *cv::Mat*, čo je trieda z knižnice OpenCV pre reprezentáciu matice. A keďže obrázok je vlastne matica pixelov, tak slúži aj pre reprezentáciu obrázkov. Táto trieda nahradila v novom objektovom rozhraní knižnice štruktúru IplImage. Je však možné používať aj túto staršiu štruktúru, pretože prevod je jednoduchý. Hrubý návrh práce aplikácie môžeme vidieť na nasledujúcom obrázku 4.1.



Obrázok 4.1: Princíp práce algoritmu

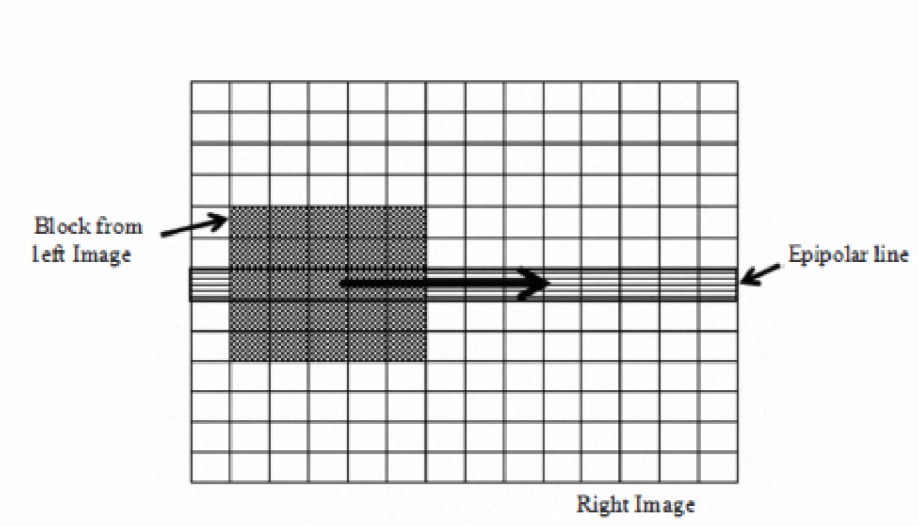
Vidíme, že aplikácia pracuje v nasledujúcich krokov: Ako prvý krok, budú v rámci predspracovania obrazu použité techniky filtrovania. Použijeme jeden z filtrov na odstránenie šumu, a tiež jeden z filtrov na zvýraznenie rysov v obrázku. Farebné obrázky prevedieme do odtieňov šedej a obrázky vo veľkom rozlíšení zmenšíme. Druhým krokom je výber objektu. Objekt bude reprezentovaný štvoruholníkovým výrezom z ľavého obrázka, ktorý treba označiť pomocou počítačovej myšky. Tretím krokom je redukcia prehľadávanej oblasti v obrázku, vďaka ktorej budú algoritmy rýchlejšie, presnejšie a budú mať menej falošných detekcií. Štvrtým krokom je už hľadanie korešpondujúceho objektu. Tu použijeme zvolený algoritmus, ktorý sa aplikuje na oblasť určenú predchádzajúcim krokom. Súčasťou tohto kroku je aj lokalizácia najlepšie ohodnotenej pozície. Pre algoritmy, ktoré určujú mieru nezhody budeme hľadať minimálnu hodnotu, a pre tie, ktoré určujú mieru zhody budeme

hľadať maximálnu hodnotu. Posledným piatym krokom je určenie disparity na základe pozície korešpondujúceho objektu.

Teraz sa pozrieme na návrh aplikácie z pohľadu užívateľa. Na začiatku musí užívateľ vybrať obrázky stereo páru. Musí si vybrať jeden, alebo aj viac algoritmov, ktoré chce práve použiť pre hľadanie objektu. Ďalej môže nastaviť voliteľné parametre, ako zapnúť redukciu prehľadávanej oblasti, definovať vlastnú oblasť, nastaviť parametre potrebné pre triangu-láciu. Potom, keď chce začať hľadať, označí objekt na ľavom obrázku. Označený objekt sa začne automaticky vyhľadávať v pravom obrázku za použitia určených algoritmov. Po skončení vyhľadávania sa vypočíta disparita ako rozdiel polôh objektu v dvoch obrázkoch. Užívateľ je informovaný o výsledkoch hľadania výpisom.

4.3 Redukcia prehľadávanej oblasti

Všetky lokálne algoritmy budú pracovať na princípe template-matchingu. Vzor, ktorý si určí užívateľ aplikácie budeme postupne posúvať po celom obrázku. Je dôležité sním prejsť celým obrázkom, pretože hľadaný objekt sa môže nachádzať kdekoľvek. My však máme jednu obrovskú výhodu. Poznáme polohy jednotlivých kamier, ich natočenie a parametre. Spolu s kalibráciou teda snímky môžu byť rektifikované, čo využijeme rapídnym zmenšením prehľadávanej oblasti v obrázku. Predtým sme museli prehľadať celý 2D priestor pravého obrázka, no teraz už nám stačí prehľadať len 1D oblasť definovanú epipolárnou líniou (obrázok 4.2).



Obrázok 4.2: Hľadanie po epipoláre[10]

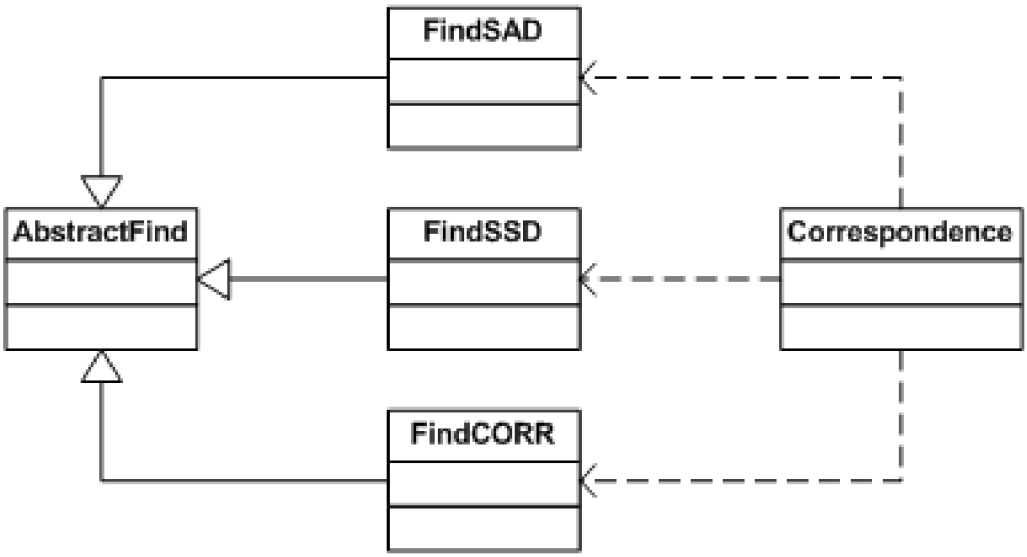
Prehľadávanú oblasť však môžeme zmenšiť ešte viacej. Tentokrát použijeme skutočnosť, že poradie objektov na obidvoch snímkoch musí byť rovnaké. Spolu stým, že poznáme postavenie kamier, môžeme obmedziť oblasť hľadania len na jeden smer. Teda ak objekt označíme na ľavom obrázku, tak jeho pozíciu v pravom obrázku budeme hľadať smerom doľava od pozície na akej sa nachádzal v ľavom obrázku. A platí to aj opačne, teda ak objekt označíme v pravom obrázku, tak v ľavom ho budeme hľadať iba smerom doprava. To ale ešte nieje všetko, prehľadávanú oblasť môžeme ešte zmenšiť. Tentokrát použijeme skutočnosť, že vieme, že objekty ktoré sa nachádzajú ďalej od kamier majú menšiu disparitu, ako tie, ktoré

sú bližšie. Môžeme teda určiť hodnotu maximálnej disparity. Napríklad náš systém by mal merať vzdialenosť objektov pomocou výpočtu disparity a následnou triangulációu. Určíme si od akej minimálnej vzdialenosti budeme chcieť merať. A v tejto vzdialenosti priradíme disparitu, ktorá bude maximálna možná. Bližšie objekty teda nezmeriame, ale algoritmus bude pracovať rýchlejšie vzhľadom na ďalšie zmenšenie oblasti hľadania.

Zmenšenie oblasti má okrem zrýchlenia práce algoritmov za následok aj zvýšenie ich presnosti, a obmedzenie falošnej detekcie objektu. Je to spôsobené tým, že v miestach, ktoré sme vyradili z vyhľadávanej oblasti, sa mohli nachádzať potencionálne lepšie ohodnotené miesta ako má byť skutočná pozícia vzoru. Pri experimentovaní si to overíme. Rovnako sa pozrieme aj na rýchlosť algoritmov pred a po zmenšení prehľadávanej oblasti.

4.4 Implementácia v triedach

Samotné algoritmy budeme implementovať dvomi spôsobmi: pomocou triviálneho algoritmu a potom pomocou optimalizovaného algoritmu. Prvý triviálny algoritmus bude počítať priamo jednotlivé vzťahy, v jazyku C/C++. Druhý bude optimalizovaný pomocou SIMD inštrukcií procesora *MMX/SEE*, ktoré budú použité v jazyku symbolických inštrukcií. Všetky algoritmy sú implementované vo vlastných triedach, ktoré sú derivované z abstraktnej triedy *AbstractFind*. Nasleduje popis implementácie jednotlivých tried.



Obrázok 4.3: Diagram tried

4.4.1 Trieda AbstractFind

Trieda AbstractFind je, ako jej názov napovedá, abstraktnou triedou. To znamená, že všetky triedy derivované z AbstractFind musia implementovať všetky jej metódy vlastným spôsobom. Rovnako kvôli tomu nieje možné túto triedu inštantizovať – vytvárať jej objekty. Trieda deklaruje dve metódy, ktoré sú deklarované ako čisto virtuálne. Tým určujeme, že je nutné ich v derivovanej triede prepísať. Sú to metódy *Trivial()* a *Optimal()*. Každá metóda má dva parametre. Prvým parametrom je obrázok predstavovaný objektom typu cv::Mat a druhým parametrom je vzor, ktorý budeme hľadať v obrázku, rovnako vo forme objektu

typu cv::Mat. Návratová hodnota obidvoch metód je objekt typu cv::Point a predstavuje pozíciu vzoru v obrázku.

4.4.2 Trieda FindSAD

Účelom tejto triedy je implementácia algoritmu Sum of Absolute Differences 2.7. Trieda FindSAD je derivovaná z triedy AbstractFind, takže implementuje všetky jej abstraktné metódy.

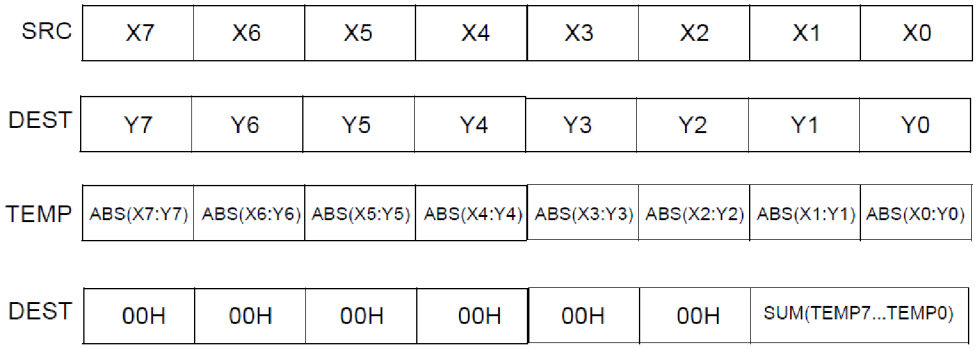
Metóda *Trivial()* definuje implementáciu algoritmu SAD triviálnym spôsobom: vzor posúvame po obrázku po stĺpcoch, čo zabezpečujú dva vonkajšie cykly for. Ďalšie dva vnútorné cykly zabezpečujú iteráciu obrázkom aj vzorom, pričom v každej jednej iterácii je vypočítaná hodnota SAD. Táto hodnota sa akumuluje až kým neprejdeme celým vzorom. Po tom ako ním prejdeme, je hodnota SAD porovnaná z doterajšou minimálnou hodnotou. Ak je SAD menšia, znamená to že aktuálna pozícia vzoru má lepšie ohodnotenie miery nezhody (menšia miera nezhody = lepšie ohodnotenie) a táto pozícia je presnejšia oproti predchádzajúcej. Po prejdení celým obrázkom je vrátený objekt triedy cv::Point, obsahujúci najlepšie ohodnotenú pozíciu.

Metóda *Optimized()* definuje implementáciu algoritmu SAD netriviálnym spôsobom – pomocou inštrukcií SIMD. Metóda dynamicky (za behu programu) načíta funkciu *findSAD()* z knižnice *findSAD.dll*, v ktorej je samotný algoritmus implementovaný. Kvôli optimalizácii sme algoritmus implementovali v jazyku symbolických inštrukcií. Funkcia findSAD() je volaná podľa konvencie __stdcall a dostáva osem parametrov v poradí: dáta obrázku, šírka obrázku, výška obrázku, dáta vzoru, šírka vzoru, výška vzoru, pozícia riadku, pozícia stĺpca. Aby bol algoritmus rýchly, využívame paralelné spracovanie viacerých pixelov. Pixely načítavame po riadkoch. Je to maximálne 16 pixelov(128 bitov) naraz, načítaných z pamäte priamo do xmm registra inštrukciou *MOVDQU*. Keď už v riadku neostáva 16 pixelov, ale je ich tam menej, opäť načítame 128 bitovú hodnotu z pamäte a pomocou inštrukcie *PSLLDQ* posúnieme xmm register toľkokrát, aby v ňom ostali len zvyšné pixely riadka. Potom pomocou inštrukcie *PSADBW* 4.4: spočítame súčet absolútnych rozdielov pre 16 pixelov obrázka a 16 pixelov vzoru súčasne. Tento súčet je spočítaný zvlášť pre 8 pixelov uložených v spodných a horných slabikách xmm registrov. Máme teda dva výsledky ktoré budú uložené: jeden v najnižšom slove a druhý v štvrtom slove xmm registra. Tieto dve hodnotu vytiahneme z registra inštrukciou *PEXTRW* a spočítame. Naraz tak máme spracovaných 16 pixelov, čo je oproti triviálnemu algoritmu teoreticky 16-násobné zrýchlenie. Takto to prebieha pre všetky riadky, pričom po každom riadku sa hodnota SAD akumuluje. Po prebehnutí všetkých riadkov je hodnota v SAD ohodnotením aktuálnej pozície a dôjde k porovnaniu s doteraz najlepšou hodnotou v bestSAD. Ak je aktuálna hodnota lepšia (teda menšia), tak sa stane najlepšou. Pozícia prislúchajúca tejto hodnote sa uloží do pamäte na adresy posledných dvoch predaných parametrov.

4.4.3 Trieda FindSSD

Táto trieda implementuje algoritmus Sum of Squared Differences 2.8. Rovnako ako predchádzajúca trieda je odvodená z triedy AbstractFind a implementuje jej abstraktné metódy *Trivial()* a *Optimized()*.

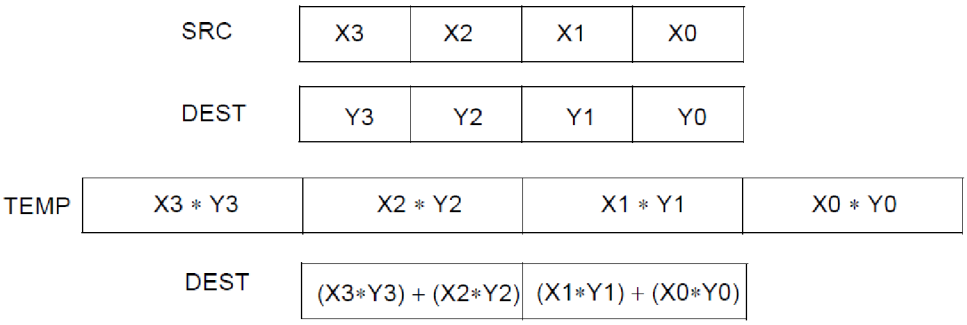
Metóda *Trivial()* definuje implementáciu tohto algoritmu, jednoduchým priamym spôsobom. Pracuje takmer na totožnom princípe ako algoritmus SAD. Dva cykly zabezpečujú prechod obrázkom, ďalšie dva prechod po pixeloch vzoru a obrázku. Pre každú dvojicu pixelov je vypočítaný ich rozdiel a následne tento rozdiel umocnený na druhú. Výsledok



Obrázok 4.4: Inštrukcia PSADBW[7]

akumulujeme pre každú pozíciu v prehľadávanom obrázku a porovnávame s predchádzajúcou najlepšou pozíciou, pričom menšia hodnota znamená lepší výsledok. Na konci nám zostane najlepšia hodnota, ktorá určuje pozíciu vzoru v obrázku.

Metóda *Optimized()* implementuje algoritmus Sum of Squared Differences netriviálnym spôsobom. Algoritmus bol naprogramovaný v jazyku symbolických inštrukcií, a skompilovaný do knižnice findSSD.dll. Tá je volaná konvenciou __stdcall, pričom sú jej predávané parametre: ukazovateľ na dáta obrázku, šírka a výška obrázku, ukazovateľ na dáta vzoru, jeho šírka a výška a ukazovatele na riadok a stĺpec zhody. Z pamäte si načítame pomocou *MOVDQU* dva 128 bitové bloky dát do dvoch xmm registrov, jeden z obrázka druhý zo vzoru. Každý z nich obsahuje 16 pixelov. To znamená, že máme 8 bitov na pixel a potrebujeme ich konvertovať na 16 bitov. Pomocou štyroch inštrukcií *PUNPCKHBW* ich teda prevedieme a výsledky prevodu budeme mať v štyroch xmm registroch. Medzi odpovedajúcimi registrami spravíme rozdiel inštrukciou PSUBW. Spravíme ich druhú mocninu a ich súčet pomocou *PMADDWD*4.5. Výsledky máme opäť v dvoch registrom, tak spravíme ich súčty (inštrukcia *PADDD*) a horizontálny súčet (inštrukcia *PHADDD*). Celkový súčet nakoniec akumulujeme v premennej currSSD. V nej sa po prejdení všetkých riadkov nachádza ohodnotenie pozície v obraze a je porovnaná s doteraz najlepšou pozíciou. Ak je menšia stáva sa najlepšou. Po ohodnotení všetkých pozícií, sa najlepšia pozícia zapíše do pamäte na miesto posledných dvoch predaných parametrov.



Obrázok 4.5: Inštrukcia PMADDWD[7]

4.4.4 Trieda FindCORR

Táto trieda implementuje algoritmus Cross-Correlation 2.11. Presne tak,ako predchádzajúce triedy, je odvodená z triedy AbstractFind, a teda musí implementovať abstraktné metódy tejto triedy.

Metóda *Trivial()* implementuje korelačný algoritmus triviálnym spôsobom. Vzor postupne posúvame po obrázku. O to sa starajú dva vonkajšie cykly for. V nich sa nachádzajú ďalšie dva vnútorné cykly for. Tie zabezpečujú iteráciu po pixeloch obrázku aj vzoru, pričom v každej jednej iterácii je vypočítaná hodnota korelácie ako súčet súčinov hodnôt odpovedajúcich si pixelov obrázku a vzoru. Hodnota korelácie sa agreguje až dovtedy, pokiaľ neprejdeme celým vzorom a časťou obrázku, ktorú vzor práve pokrýva. Po tom ako je dopočítaná konečná hodnota korelácie, porovnáme ju so zatiaľ najlepšou doterajšou hodnotou. Ak je nová hodnota väčšia, znamená to, že aktuálna pozícia vzoru má lepšie ohodnotenie, teda miera zhody je väčšia. Pozíciu, ktorej prislúcha toto ohodnotenie si uložíme. Po prejdení obrázkom vrátime objekt triedy *cv::Point*, ktorý reprezentuje najlepšie ohodnotenú pozíciu.

Metóda *Optimized()* implementuje algoritmus Cross-Correlation netriviálnym prístupom. Ako názov napovedá, opäť sú využité SSE inštrukcie. Samotný algoritmus je implementovaný v knižnici *findCORR.dll*. Je to dynamicky linkovaná knižnica, takže je načítaná za behu programu. Obsahuje rovnako pomenovanú funkciu *findCORR()*, ktorá je volaná podľa konvencie __stdcall a predávame jej osem parametrov: ukazovateľ na dáta obrázku uložené v pamäti, šírku a výšku obrázku, ukazovateľ na dáta vzoru, šírku a výšku vzoru, ukazovatele na pozíciu riadku a stĺpca zhody. Pre výpočet používame hlavne inštrukcie, ktoré dokážu spracovať viac pixelov naraz. Z pamäte načítavame 128 bitový blok dát inštrukciou *MOVDQU*, ktorý obsahuje 16 pixelov (8 bitov na pixel). Keďže nemáme žiadnu SIMD inštrukciu súčinu, ktorá by pracovala so slabikami, musíme si hodnoty pixelov skonvertovať zo slabík na slová. Najprv si zduplikujeme načítané pixely, a použijeme dvojicu inštrukcií *PUNPCKLBW* a *PUNPCKHBW*. Prvá nám prevedie osem 8 dolných a druhá 8 horných slabík na slová. Potom môžeme začať počítať koreláciu. Hlavné miesto tu zaujíma inštrukcia *PMADDWD*4.5. Tá spočíta súčin odpovedajúcich si slov dvoch xmm registrov, pričom súčin už má veľkosť dvojslova. Následne sú dva tieto súčiny sčítané a uložené do odpovedajúceho dvojslova cieľového xmm registru. Takto naplnené registre budeme mať dva, aby sme obsiahli všetkých 16 načítaných pixelov. Spočítame ich dohromady inštrukciou *PADDD* a hneď nato inštrukciou *PHADDD*. Celý postup sa opakuje pre všetky pixely vzoru a časti obrázka pokrytého vzorom, a pritom sa hodnota korelácie akumuluje. Po spracovaní posledného pixelu je hodnota korelácie porovnaná s doteraz najlepšou hodnotou a ak je väčšia (pri korelácii určujeme mieru zhody, nie nezhody ako u predchádzajúcich algoritmoch), nahradí ju.

4.4.5 Trieda Correspondence

Táto trieda zapúzdruje všetky implementované algoritmy, a pridáva vlastné metódy pre predspracovanie, redukciu oblasti, trianguláciu, vykreslenie zhodnej oblasti.

Metóda *Preprocess()* má za úlohu vykonať dve operácie. Najskôr z testovacích obrázkov odfiltrujeme šum pomocou jedného z filtrov. Vybrali sme si nato mediánový filter. Následne prevedieme ekvalizáciu histogramu už odfiltrovaných obrázkov. Tieto dve operácie by nám mali poskytnúť lepšie podmienky pre samotné hľadanie objektov.

Metóda *SearchSpaceReduct()* zmenšuje prehľadávanú oblasť v pravom obrázku. Využíva možnosť epipólarneho obmedzenia a maximálnej disparity. Robí to na základe veľkosti vybraného objektu, kde jeho výška určí výšku redukovanej oblasti. Zatiaľ čo jeho šírka, spolu

s maximálnou disparitou, a vyhľadávaním len jedným smerom určujú šírku redukovanej oblasti. Veľkosť maximálnej disparity musí byť zadaná užívateľom. Ak ju nezadá, tak sa použije iba omedzenie oblasti založené na epipoláre. Samotná implementácia využíva možnosti knižnice OpenCV. Pomocou tzv. oblasti záujmu (region of interest – ROI) vyberieme oblasť, ktorú chceme prehľadávať, a potom už sa algoritmy aplikujú iba na túto oblasť.

Metóda *Triangulate()* vypočíta vzdialenosť objektu v priestore na základe vzťahu 3.4. Využíva k tomu údaje zadané užívateľom. Ak ich užívateľ neposkytne, potom je triangulácia zablokovaná.

Metóda *DisparityEstimate()* určí disparitu na základe pozície objektu v pravom a ľavom snímku.

Metódy *FindBySAD(), FindBySSD(), FindByCORR()* používajú na vyhľadávanie objekty tried, v ktorých sú algoritmy implementované.

Metódy *FindByNCC(), FindByCOEFF()* používajú funkciu *cv::matchTemplate()* z knižnice OpenCV. Táto funkcia implementuje algoritmy normalizovanej korelácie a korelačného koeficientu s využitím Fourierovej transformácie.

Kapitola 5

Testovanie a experimenty

V tejto kapitole budeme venovať testovaniu aplikácie a neskôr experimentom na testovacích obrázkoch. Na to aby sme doložili, že naše optimalizované algoritmy pracujú matematicky korektne, bude stačiť ich porovnanie s triviálnym algoritmom. Potom podrobíme všetky algoritmy experimentom na testovacích snímkoch, kde sa ukážu ich schopnosti. To či pozície označené algoritmami ako zhodné, skutočne hľadaným objektom zodpovedajú, budeme musieť nejakým spôsobom overiť. Buď to budeme robiť manuálne, alebo môžeme použiť testovacie stereo obrázky, pre ktoré už bola určená skutočná disparita všetkých bodov obrázku. Porovnáme ich presnosť a rýchlosť, na základe výsledkov týchto experimentov algoritmy vyhodnotíme.

5.1 Testovanie aplikácie

Pri testovaní funkčnosti jednotlivých algoritmov sa objavili problémy, ktoré sme museli vyriešiť. Pre algoritmy implementované v ASM, sme dostávali niekedy totožné (správne) výsledky a niekedy rozdielne výsledky, ako podávali ich triviálne ekvivalenty. Nakoniec po dlhom debuggovaní sme prišili nato, že OpenCV niekedy pridáva za jednotlivé riadky obrázkov výplň – tzv. padding. Pravdepodobne preto, aby boli dáta v pamäti zarovnané. Toto sme obišli tým, že ak obrázok nieje uložený v pamäti kontinuálne, použijme metódu *clone()* objektu triedy cv::Mat. Tá obrázok naklonuje, a to tým spôsobom, že riadky budú uložené v pamäti kontinuálne. Preto treba zdôrazniť, že pre algoritmy implementované v ASM musíme použiť obrázky, ktorých dáta sú uložené v pamäti po riadkoch ihneď za sebou. Po vyriešení tohto problému sme algoritmy opäť porovnali. Výsledok môžete vidieť v tabuľke 5.1,

Pokus č.	1	2	3	4	5
SAD	(782,916)	(665,490)	(434,431)	(521,646)	(278,563)
SAD optimal	(782,916)	(665,490)	(434,431)	(521,646)	(278,563)
SSD	(782,916)	(666,490)	(434,431)	(521,646)	(261,563)
SSD optimal	(782,916)	(666,490)	(434,431)	(521,646)	(261,563)
CORR	(734,916)	(834,490)	(432,431)	(407,646)	(141,563)
CORR optimal	(734,916)	(834,490)	(432,431)	(407,646)	(141,563)

Tabuľka 5.1: Korešpondujúce body detekované algoritmami

kde je vidieť, že triviálny a optimalizovaný algoritmus podávajú úplne totožné výsledky(totožné polohy). Overili sme teda správnu funkčnosť ich implementácie. Z tabuľky sa ďalej dá vyčítať, že algoritmy SAD a SSD podávajú skoro totožné výsledky. Je to spôsobené tým, že SSD funkcia pridelí vzdialenejším intenzitám väčšie váhy, inak pracujú rovnakým princípom.

5.2 Experimenty

Experimenty sú založené na zmene rôznych parametrov, po ktorých sú znova určené korešpondencie. Medzi tieto zmeny patrilo hľadanie objektu s rôznymi veľkosťami a rôznymi algoritmami. Taktiež sme zmerali nárast rýchlosti optimalizovaných variánt algoritmov implementovaných v ASM, oproti ich triviálnym implementáciám v C/C++. Medzi ďalšie zmenené parametre patria zmeny veľkosti prehľadávanej oblasti. V tejto kapitole zverejníme len niektoré výsledky experimentov, ale v prílohe [ref] môžete nájsť výsledky ďalších.

Vďaka optimalizácii s inštrukciami SSE/MMX by mali byť algoritmy rýchlejšie, ako keď sú implementované čisto v C/C++. Nasledujúca tabuľka 5.2 zobrazuje porovnanie rýchlostí všetkých implementovaných algoritmov.

Veľkosť objektu[px]	10x10	40x40	80x80	160x160	300x300	500x500
SAD	7.7	49.7	198.1	787.7	2731.9	7653.4
SAD optimal	< 1	< 1	3	6.1	18.6	50
SSD	3	9.6	20	90.3	316.6	883
SSD optimal	< 1	< 1	< 1	9.5	20.6	84.2
CORR	1.6	7.9	23.3	87.2	296.2	830.2
CORR optimal	< 1	< 1	< 1	10.9	28	75

Tabuľka 5.2: Porovnanie rýchlosti algoritmov v milisekundách – redukovaná oblasť

Pokusy boli prevedené na obrázkoch zo setu Rocks2, pri zapnutej redukcii prehľadávanej oblasti. Veľkosť označeného objektu je uvedená v tabuľke. Veľkosť celého obrázka, v ktorom sa vyhľadávalo bola 1278x1110 pixelov. Bolo prevedených 10 pokusov, z ktorých sme spravili priemer a dosadili do tabuľky. Z výsledkov je vidieť, že optimalizované algoritmy sú oveľa rýchlejšie, ako ich jednoduché implementácie. Je to spôsobené paralelným spracovaním pixelov. Rozdiel medzi optimalizovaným SAD a ďalsími dvoma optimalizovanými algoritmami bude ten, že SAD spracováva maximálne 16 pixelov paralelne a ostatné dva len 8. Dokonca pri obrázku veľkosti 500 na 500 pixelov sa stále držíme pod 100 milisekúnd. Bez problémov by sme ich mohli používať aj pri real-time aplikáciách. Najväčšiu zásluhu má natom redukovaná oblasť hľadania, ktorá je hlavne v prípade malých objektov zmenšená niekoľkonásobne.

V ďalšej tabuľke 5.3 prevedieme rovnaký experiment, ale oblasť hľadania nebudeme redukovať. Zrovnáme len optimalizované algoritmy, pretože triviálne sú pri takej veľkej oblasti hľadania veľmi pomalé. Z rovnakého dôvodu bolo hľadané len do veľkosti objektu 40x40 pixelov. Vidíme, že oproti redukovanej oblasti sú algoritmy oveľa pomalšie. Výhoda redukovanej oblasti je zjavná a výplýva z toho, že bez nej sú intensity-based algoritmy nepoužitelné.

Veľkosť objektu[px]	10x10	40x40
SAD optimal	296.8	2917.4
SSD optimal	402.8	2939.4
CORR optimal	302.6	2733.4

Tabuľka 5.3: Porovnanie rýchlosti algoritmov v milisekundách – plná oblasť

Asi najdôležitejším parametrom detekčného algoritmu je jeho presnosť. Tabuľka 5.4 zobrazuje porovnanie disparít vypočítaných jednotlivými algoritmami.To či sú disparity korektne určené, sme overovali manuálne. K nami implementovaným algoritmom boli pridané aj ďalšie metódy template matchingu z knižnice OpenCV.

Pokus č.	1	2	3	4	5	6	7	8	9	10
SAD	160	85	144	134	167	81	85	106	144	107
SSD	160	85	144	134	167	81	85	106	144	107
CORR	300	85	114	36	254	300	14	106	90	300
NCC	160	85	144	165	167	81	85	106	144	107
COEF	159	85	144	164	167	81	85	106	144	107

Tabuľka 5.4: Porovnanie vypočítanej disparity v pixeloch – redukovaná oblasť

Z väčšej časti sa výsledky jednotlivých algoritmov zhodujú. Na prvý pohľad vidíme, že na algoritmus cross-correlation sa nemôžeme spoľahnúť. Niekedy podáva korektné výsledky, ale akonáhle sa v prehľadávanej oblasti objaví miesto z vysokými intenzitami, tak dojde k falošnej detekcii. Ostatné algoritmy podávajú rovnaké výsledky, iba niekedy sa odlišujú o cca pixel. Za to vďačíme rekudovanej oblasti. Keď prevedieme rovnaký experiment v plnej oblasti (tabuľka 5.5), tak žiadny z našich implemetovaných algoritmov, nedokáže byť taký presný a počet falošných detekcií narastá. Algoritmus cross-correlation je už úplne nepoužiteľný. SAD a SSD fungujú dobre, keď sú objekty texturované, ale keď niesú, tak ich presnosť klesá. Algoritmy normalized cross-corelation a normalized corelation coefficient z knižnice OpenCV dopadli zo všetkých najlepšie.

Pokus č.	1	2	3	4	5	6	7	8	9	10
SAD	-96	160	6	867	145	169	171	342	-189	85
SSD	-96	160	6	80	145	169	171	80	256	86
CORR	-98	-77	-30	34	-196	157	110	-33	-11	-367
NCC	106	160	80	80	145	169	171	80	165	86
COEF	106	160	80	80	145	169	171	80	163	86

Tabuľka 5.5: Porovnanie vypočítanej disparity v pixeloch – plná oblasť

Kapitola 6

Záver

Cieľom tejto bakalárskej práce bolo oboznámenie sa s problematikou stereoskopie a navrhnutie riešenia problému stereo korešpondencie, pre vyhľadanie objektu v stereo snímku. Pre splnenie týchto bodov sme implementovali algoritmy, pomocou ktorých sme vykonali experimenty na testovacích snímkoch. Z experimentov vyplýva, že implementované algoritmy v ich triviálnej forme niesú v praxi použiteľné. Naopak algoritmy, ktoré sme optimalizovali pomocou inštrukcií SSE, vykazujú mnohonásobne vyššiu rýchlosť. Keď si spojíme ich vysokú rýchlosť s možnostou rapídneho zmenšenia prehľadávanej oblasti, tak sú predurčené k použitiu napríklad v real-time aplikáciách. Zistili sme, že použité algoritmy sú presné iba vtedy, keď je obmedzená oblasť hľadania. Inak vykazujú vyššiu mieru falošnej detekcie. Ukázali sme aj jednu z oblastí aplikácie takýchto algoritmov, konkrétne jednoduchého výpočtu vzdialenosti na základe určenej disparity.

Ako pozitívum a prínos tejto práce vidíme v tom, že sme sa mohli bližšie zoznámiť s oblasťou stereoskopie v čase, keď sa táto zaujímavá problematika dostáva medzi bežných ludí. Príkladom môže byť systém Kinect od spoločnosti Microsoft, ktorý si získal veľkú popularitu. Ďalšie plus je v tom, že sme sa dostali do kontaktu s knižnicou OpenCV, ktorú v budúcnosti ešte určite použijeme.

Do budúcnosti by sme mohli urobiť porovnanie aj s inými algoritmami. Rovnako by sme mohli výkon algoritmov ešte viac zvýšit. Otvárajú sa k tomu napríklad aj možnosti paralelizmu, za pomoci technológií CUDA respektíve OpenCL na grafických kartách. Ďalšou možnosťou zvýšenia paralelizmu môžu byť programovateľné hradlové polia. Takýto algoritmus by sme mohli potom použiť pri zložitejšej úlohe, ako napríklad – spolu s dvoma kamerami, by sme mohli sledovať určitý cieľ, určovať jeho vzdialenosť a počítať jeho GPS súradnice.

Literatúra

[1] Bay, H.; Tuytelaars, T.; Van Gool, L.: SURF: Speeded Up Robust Features. In *Computer Vision ECCV 2006*, editace A. Leonardis; H. Bischof; A. Pinz, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2006, s. 404–417.

[2] Blázsovits, G.: Digital Image Processing, Interaktívna učebnica spracovania obrazu. 2006.
URL http://dip.sccg.sk/predspra/predspra.htm

[3] Bock, R. K.: The Data Analysis BriefBook. 1998.
URL http://rkb.home.cern.ch/rkb/AN16pp/node283.html

[4] Bradski, G.; Kaehler, A.: *Learning OpenCV : Computer Vision with the OpenCV Library*. O'Reilly Media, 2008, ISBN 9780596516130.

[5] Brunelli, R.: *Template Matching Techniques in Computer Vision: Theory and Practice*. Wiley Publishing, 2009, ISBN 0470517069, 9780470517062.

[6] Bräunl, T.; Feyrer, S.; Rapf, W.; aj.: *Parallel Image Processing*. Springer, 2001, ISBN 9783540674009.

[7] Corporation, I.: Intel®64 and IA-32 Architectures Software Developer's Manual.
URL http://www.intel.com/Assets/PDF/manual/253667.pdf

[8] Forsyth, D. A.; Ponce, J.: *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002, ISBN 0130851981.

[9] Gonzalez, R. C.; Woods, R. E.: *Digital Image Processing (3rd Edition)*. Prentice Hall, 2007, ISBN 9780131687288.

[10] Hamzah, R.; Rahim, R.; Noh, Z.: Sum of Absolute Differences algorithm in stereo correspondence problem for stereo matching in computer vision application. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, ročník 1, july 2010, s. 652 –657, doi:10.1109/ICCSIT.2010.5565062.

[11] Kršek, P.: Základy počítačové grafiky, Studijní opora.
URL https://www.fit.vutbr.cz/study/courses/index.php.cs?id=6839

[12] Lewis, J. P.: Fast Normalized Cross-Correlation. 1995.
URL http://scribblethink.org/Work/nvisionInterface/nip.pdf

[13] Mattoccia, S.: Stereo Vision, Algorithms and Applications. 2010.
URL www.vision.deis.unibo.it/smatt

[14] Moallem, P.; Faez, K.: Fast edge-based stereo matching algorithm based on search space reduction. In *Neural Networks for Signal Processing, 2002. Proceedings of the 2002 12th IEEE Workshop*, 2002, s. 587 – 596, doi:10.1109/NNSP.2002.1030070.

[15] Scharstein, D.; Pal, C.: Learning conditional random fields for stereo. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007.
URL http://vision.middlebury.edu/stereo/data/

[16] Sonka, M.; Hlavac, V.; Boyle, R.: *Image Processing: Analysis and Machine Vision*. CL-Engineering, 1998, ISBN 053495393X.

[17] Szeliski, R.: Computer Vision : Algorithms and Applications. *Computer*, ročník 5, 2010: str. 979.
URL http://research.microsoft.com/en-us/um/people/szeliski/book/drafts/szelski_20080330am_draft.pdf

[18] Wikipedia: Epipolar geometry — Wikipedia, The Free Encyclopedia. 2011.
URL http://en.wikipedia.org/wiki/Epipolar_geometry

[19] Wikipedia: Histogram equalization — Wikipedia, The Free Encyclopedia. 2011.
URL http://en.wikipedia.org/wiki/Histogram_equalization

[20] Wikipedia: HSV Color Space — Wikipedia, The Free Encyclopedia. 2011.
URL http://commons.wikimedia.org/wiki/User:SharkD

[21] Wikipedia: Image rectification — Wikipedia, The Free Encyclopedia. 2011.
URL http://en.wikipedia.org/wiki/Image_rectification

[22] Wikipedia: RGB Color Space — Wikipedia, The Free Encyclopedia. 2011.
URL http://commons.wikimedia.org/wiki/User:SharkD

Príloha A

Obsah DVD

\bin	binárne súbory
\examples	testovacie snímky
\src	zdrojový kód
\text	text práce