VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

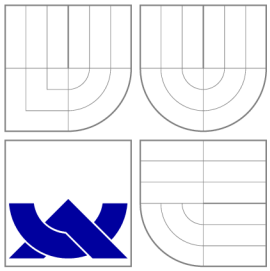
ROBOTICKÝ NÁKUPNÍ KOŠÍK

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

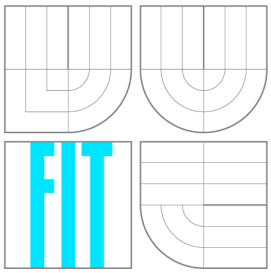
AUTOR PRÁCE
AUTHOR

JURAJ KRZEMINSKÝ

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROBOTICKÝ NÁKUPNÍ KOŠÍK

ROBOTIC SHOPPING CART

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JURAJ KRZEMINSKÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. ZDENĚK MATERNA
SUPERVISOR

BRNO 2015

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementaci řídicí jednotky pro robotický nákupní košík. Ten má za úkol sledovat vybranou osobu po nákupním středisku a plnit roli nosiče tašek. V textu jsou popsány základní principy detekce a sledování lidské postavy v obraze a také způsob snímaní obrazu RGB-D kamerou Kinect. Zbytek textu se věnuje detailnímu popisu hotové aplikace a implementaci v ní použitých vybraných metod a algoritmů.

Abstract

This Bachelor thesis deals with design and implementation of control unit for robotic shopping cart. Its main task is to follow a chosen person through a mall and serve as bag carriage. The text describes basic principles of human detection and tracking in images and also the method of capturing them with RGB-D camera kinect. Rest of this text contain detailed description of created application and implementation of used methods and algorithms.

Klíčová slova

Detekce postavy v obraze, RGB-D senzor, Kinect, hlubkova mapa, ROS, OpenCV, HOG

Keywords

Human Detection in Images, RGB-D Sensor, Kinect, Depth Map, ROS, OpenCV, HOG

Citace

Juraj Krzeminský: Robotický nákupní košík, bakalářská práce, Brno, FIT VUT v Brně, 2015

Robotický nákupní košík

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Zdeňka Materny. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Juraj Krzeminský
18. května 2015

Poděkování

Na tomto místě chci poděkovat vedoucímu mé bakalářské práce, panovi Ing. Zdeňku Maternovi za věnovaný čas a ochotu při konzultacích.

Obsah

Kapitola 1

Úvod

Ľudia si už od dávnych dôb snažili zjednodušiť namáhavú a repetitívnu činnost. Spočiatku si pomáhali rôznymi nástrojmi alebo nechali prácu vykonávať niekoho iného. Neskôr, s pokrokom vedy a techniky, a s príchodom prvej priemyselnej revolúcie, sa začali objavovať prvé stroje, ktoré zaň začali vykonávať väčšinu fyzickej práce. Tieto stroje však stále vyžadovali pomerne značné riadenie a kontrolu. Prvé samostatne pracujúce zariadenia vznikli až s príchodom robotiky a kybernetiky koncom dvadsiatého storočia. Odvtedy sa tieto odvetvia vyvíjajú astronomickým tempom. V roku 2014 sa zvýšil dopyt po priemyselných robotoch o 15% a predpokladá sa, že tento nárast bude ešte dlho pokračovať [1]. Tento vzostup spôsobil zavedenie robotov aj do každodenného života bežných ľudí. Vznikli prístroje ako sú robotické vysávače alebo inteligentné pračky, úlohou ktorých je pomáhať s drobnými domácimi prácami. Cieľom tejto práce je podobné zariadenie, konktrétne program pre riadenie robota, určeného k noseniu nákupných tašiek. Dôležitým prvkom takéhoto pomocníka je schopnosť orientovať sa v reálnom prostredí, konkrétne rozpoznať a bezpečne následovať správneho človeka. Efektívne riešenie tohto problému a jeho praktické využitie je hlavnou náplňou tejto práce.

Aplikácia bola vytvorená vo forme balíku pre robotický operačný systém ROS, kokrétne verziu *ROS Hydro*. K ďalším softvérovým požiadavkám patrí operačný systém linux, ktorý je potrebný k behu ROS-a a knižnica openCV, ktorá bola použitá na detekciu postavy v obraze. Program bol vyvíjaný pre robota typu *Pioneer P3-AT* patriacému Fakulte informatiky VUTBR, avšak je možné ním riadiť širokú škálu iných zariadení, prakticky jedinými požiadavkami je prítomnosť Kinectu a schopnosť horizontálneho pohybu. S menšími úpravami by ho dokonca bolo možné použiť aj na riadenie pohybu po vertikálnej osi.

Táto práca sa v ďalších kapitolách venuje podrobnejšiemu opisu problematiky počítačového videnia, zvlášť detekcie postavy v obraze, popisu funkcie Kinectu a hlavným princípom systému ROS. Predposledná kapitola sa potom venuje návrhu a implementácií samotnej aplikácie, popisuje použité, ako aj dostupné metódy a ich následné využitie v kóde. V závere sa text venuje známym nedostatkom a možným rozšíreniam do budúcna.

Kapitola 2

Analýza zadania

Hlavným problémom pri návrhu programu pre ovládanie automatizovaného nákupného košíka je detekcia a sledovanie postavy v reálnom prostredí. Okolné prostredie je možné analyzovať a pretransformovať na dáta pochopiteľné robotom veľkým počtom spôsobov, z ktorých najviac informácií zachytávajú tie, ktoré sú založené na vizuálnom popise dát. Hlavnú rolu teda bude hrať kamera a algoritmy určené na rozpoznanie a rozdelenie objektov v obraze. Tejto problematike sa venuje obor výpočtovej techniky s názvom počítačové videnie, ktorého princípy ako aj niektoré konkrétne algoritmy boli použité pri realizácií výslednej aplikácie. Pre zvýšenie informačnej hodnoty popisu okolia je v mojom riešení použitá 3D kamera Kinect. Pridanie tretieho rozmeru výrazne napomáha odstráneniu problémov naskytajúcich sa pri pohybe robota v skutočnom prostredí.

Kapitola 3

Používané platformy a zariadenia

Táto kapitola sa venuje popisu použitého hardvéru a softvéru dôležitého pre vytvorenie a fungovanie našej aplikácie.

V úvodnej časti bude predstavený robot typu Pioneer, na ktorom bola výsledná aplikácia vyvíjaná a testovaná. V krátkosti bude opísaný aj jeho operačný systém s názvom ROS. Cieľom je objasniť jeho štruktúru, princípy práce v ňom a vysvetliť niektoré základné pojmy.

Ďalej sa potom zaoberá detailnejším popisom RGB-D senzoru Kinect, ktorý robot používa na získanie dát o svojom okolí. Dôraz je kladený na vysvetlenie metód, ktoré Kinect využíva na vytvorenie hĺbkovej mapy.

V úplnom závere bude čitateľ zoznámený s knižnicou openCV, ktorá sa zaoberá počítačovým videním, a ktorú vo veľkej miere používa aj moja aplikácia.

3.1 Robot typu Pioneer

Táto kapitola sa bližšie venuje popisu robota typu Pioneer P3-AT, na ktorom bola aplikácia vyvíjaná a testovaná. Aj napriek tomu, že aplikácia nevyžaduje žiaden špeciálny hardvér a teoretický by mala fungovať na ľubovoľnom type robota, nikdy nebola skúšaná na inom zariadení, a preto budem považovať tento typ za nevyhnutnú súčasť riešenia. Následujúci text sa venuje popisu technických parametrov a obsiahnutých zariadení na školskom robotovi s názvom *Toad*.

3.1.1 Toad

Toad je robot typu Pioneer P3-AT patriaci Fakulte Informačných technológií VUT v Brně. Pioneer je rada dvoj a štvorkolesových robotov určených do vonkajšieho prostredia. Obsahujú všetky základné časti potrebné pre samostatnú navigáciu a pohyb po vonkajšom prostredí. Každý robot tohto typu je vyrobený z tvrdenej hliníkovej konštrukcie a je vybavený výkonným motorom na jednosmerné napájanie, vybalancovaným podvozkom s pohonom na dve kolesá, zdrojom energie v podobe batérie a riadiacim mikrokontrolérom typu Hitachi H8S. Mikrokontrolér je vybavený operačným systémom PSOS (Pioneer Server Operating System). Robot obsahje aj vlastné vývojové prostriedky vo forme voľne šíriteľného vývojového prostredia ARIA (Advanced Robotics Interface for Applications), avšak k riešeniu tejto práce bol použitý výhradne systém ROS.

Tento typ robota bol vybraný k práci vďaka jeho dobrým fyzikálnym vlastnostiam. Toad dokáže odniesť až 40 Kg dodatoçnej záťaže a váži len 14 Kg, takže je ho ľahké preniesť aj

jedinou osobou. To z neho robí skvelé zariadenie na prevážanie menších nákladov ako sú nákupy, alebo ľahší stavebný materiál [3].

Toad je vybavený 3D senzorom Velodyne HDL-32E a RGB-D kamerou Kinect [5]. Kinect bude podrobne rozoberaný v ďalšej podkapitole.

Obrázek 3.1: Toad – robot typu Pioneer P3-AT [4].

Toad je ovládaný pomocou robotického operačného systému s názvom ROS, ktorý pri realizácií tejto práce zohrával kľúčovú úlohu, keďže slúži nielen na riadenie jednotlivých častí robota, ale aj ako vývojové prostredie pre písanie preň určených aplikácií.

3.2 Robot Operating System

Roboty sú pomerne rôznorodé zariadenie. Rôzne typy, vykonávajú odlišné funkcie, dôsledkom čoho je široké spektrum používaného hardvéru. To do značnej miery komplikuje písanie riadiacého softvéru. Pre každé zariadenie existuje samostatný ovladač a ich vzájomná komunikácia býva problematická. Tým výrazne trpí možnosť znovupoužitia kódu. Tieto, a množstvo ďalších problémov rieši robotický operačný systém.

ROS nie je operačný systém v tradičnom zmysle, ide o tzv. meta-operačný systém. Je to vrstva nad operačným sytémom, ktorej funkciami sú okrem tradičných úloh OS ako je abstrakcia hardvéru, kontrola zariadení na najnižšej vrstve alebo manažovanie processov, aj zabezpečenie prostriedkov a knižníc pre návrh, implementáciu, preklad a beh zdrojového kódu. Ide vlastne o akúsi kombináciu vývojového prostredia pre aplikácie na ovládanie robotov a ich operačného systému [27].

Výhodou ROS-u oproti iným takýmto vývojovým prostrediam, ako je napríklad YARP, Orca alebo Microsoft RObotics Studio, je jeho dostupnosť. ROS je distribuovaný pod BSD open-source licenciou.

ROS momentálne beží len na unixových systémoch, ako je Ubuntu alebo OS X. Z toho dôvodu je ako základý OS pre výslednú aplikáciu použitá linuxová distribúcia Ubuntu, konkrétne verzia 12.04 Precise.

3.2.1 Základné pojmy

Ná pochopenie štruktúry a spôsob práce systému ROS je potrebné najprv vysvetliť niekoľko základných pojmov.

Nasledujúce informácie sú prevzaté z oficiálnej webovej stránky ROS-u [9].

Uzol

Uzlom sa nazývajú procesy, ktoré vykonávajú výpočty. Každý uzol funguje ako samostatná jednotka, čo napomáha maximálnej modularite systému. Pri zvyčajnom behu robota je aktívne obrovské množstvo navzájom komunikujúcich uzlov. Jeden uzol môže napríklad riadiť kameru, ďalší ovládať motor podvozku, tretí vykresľovať videnie robota na obrazovku a podobne. Uzol je napísaný v niektorom z bežných programovacích jazykov za pomoci ROS knižníc roscpp alebo rospy.

Správa

Uzly navzájom komunikujú zasielaním správ. Správa je v podstate ľubovoľná dátová štruktúra, vytvorená jedným uzlom, a prečítaná ďalšími. ROS umožňuje užívateľovi nadefinovať a používať vlastné formáty správ. Správa môže obsahovať primitívne dátové typy ako sú ingeter alebo boolean, ale aj polia alebo iné vnorené štruktúry.

Téma

Správy nie sú v ROS-e posielané priamo, ale pomocou integrovaného transportného systému. Uzol publikuje správu, ktorú chce poslať, na príslušnú tému a každý uzol, ktorý to potrebuje, si túto správu môže prečítať tým, že sa prihlási k danej téme. Týmto spôsobom komunikácia medzi uzlami nie je nijako vynútená, a prijímajúci uzol si správu môže prečítať nezávisle od času, kedy bola poslaná. Každý uzol je schopný publikovať aj prijímať správy z viacerých tém zároveň. Vo všeobecnosti platí, že komunikujúce uzly navzájom o sebe nevedia. Uzol, ktorý vytvoril správu netuší, kedy a kto túto správu prečítal. Hlavnou myšlienkou takéhoto návrhu je oddeliť vytvorenie informácie od jej spracovania.

Každá téma má unikátny názov, ktorý slúži ako jej identifikátor, a formát správ, ktoré sa naň dajú publikovať.

Služba

Model posielania správ prostredníctvom tém, je v mnohých prípadoch efektívne riešenie, avšak nie je vhodné pre klasický request/reply prístup. Tento typ komunikácie je v ROS-e riešený použitím služieb. Služba je definovaná svojím unikátnym menom, a dvojicou správ, jednou pre požiadavku a druhou pre odpoveď. Zabudované knižnice ROS-u sa vo väčšine prípadov správajú k tomuto spôsobu komunikácie akoby šlo o vzdialené volanie procedúr.

Master

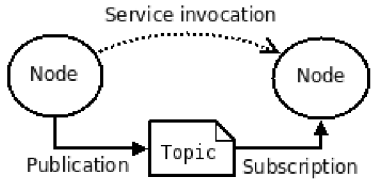
ROS Master rieši registráciu mien a vzťahy medzi jednotlivými časťami systému. Bez neho by uzli neboli schopné navzájom sa nájsť, vymieňať si správy alebo využívať služby. Master obsahuje referencie na všetky uzly, témy a služby v systéme. Každý uzol okamžite po svojom vytvorení kontaktuje ROS Master, ktorý ho registruje. Následnou komunikáciou s Master uzlom dokážu získať inormácie o ďalších prvkoch v systéme a nadviazať s nimi spojenie.

Bag

Bag je formát, ktorý slúži pre ukladanie a opätovné prehrávanie súborov publikovaných správ. Umožňujú jednoduché zozbieranie a uloženie testovacích dát, ako sú napríklad streamy zo senzorov alebo kamier, ktoré by sa inak získavali len veľmi obtiažne. Bag súbor je možné vytvoriť zo všetkých tém, ktoré sú momentálne na robotovi aktívne, alebo si vybrať len niektoré, ktoré sú dôležité pre daný účel. Takýto špecializovaný bag súbor sa potom nazýva subset.

3.2.2 Princíp funkcie

Ako je z predošlých riadkov zrejmé, ROS je založený na autonómnosti jeho jednotlivých prvkov. Každý uzol existuje ako samostatná jednotka a navzájom komunikujú pomocou Master uzla, ktorý slúži ako menný server. Jeden uzol môže vykonávať časť jednej alebo aj viacero kompletných úloh. Komunikácia prebieha buď prostredníctvom tém, kde sa uplatňuje hromadné posielanie s neobmedzeným počtom odosielateľov a príjemcov, alebo prostredníctvom služieb, ktoré fungujú na základe tradičnej komunikácie typu požiadavok/odpoveď. Komunikáciu medzi jednotlivými uzlami názorne zobrazuje následujúci obrázok 3.2.



Obrázek 3.2: Znázornenie komunikácie dvoch uzlov.

Súborový systém ROS-u je založený, tak ako aj zvyšok jeho konštrukcie, na maximálnej modularite. Hlavnou časťou sú balíky, ktoré obsahujú ľubovoľný počet samostatne spustiteľných uzlov, knižníc, dátových zdrojov, konfiguračných súborov alebo čokoľvek iné, čo je potrebné k spusteniu uzlu. Okrem tejto základnej stavebnej jednotky existujú ešte takzvané metabalíky, ktoré slúžia ako reprezentácia združenia niekoľkých samostatných balíkov, a manifesty. Manifest poskytuje metadata o vytvorených balíkoch, ako sú napríklad meno, verzia alebo závislosti. Balíky môžu byť ďalej usporiadané do tzv. repozitárov, ktoré slúžia na zjednodušenie návrhu a údržby rozsiahlých projektov. ROS umožňuje písať aj vlastné formáty správ a služieb. Takto vytvorené správy sú uložené v priečinku */msg* a služby v priečinku */srv* vo vnútri vyvíjaného balíku [9].

Riadiaci program, ktorý bol vyvinutý v rámci riešenia tejto bakalárskej práce sa nachádza v mnou vytvorenom balíku *person_following_bp* a obsahuje jediný uzol *detection_modul*.

3.3 RGB-D kamera

Aby robot mohol pracovať v reálnom prostredí, vyžaduje schopnosť vnímať svoje okolie. K tomuto účelu slúžia rôzne snímače, či uz zvuku, dotyku alebo najčastejšie obrazu [30].

Dnešné roboty na snímanie obrazu prevažne používajú RGB-D kamery, ktoré okrem bežného farebného obrazu ponúkajú aj istú formu priestorového videnia prostredníctvom hĺbkovej mapy.

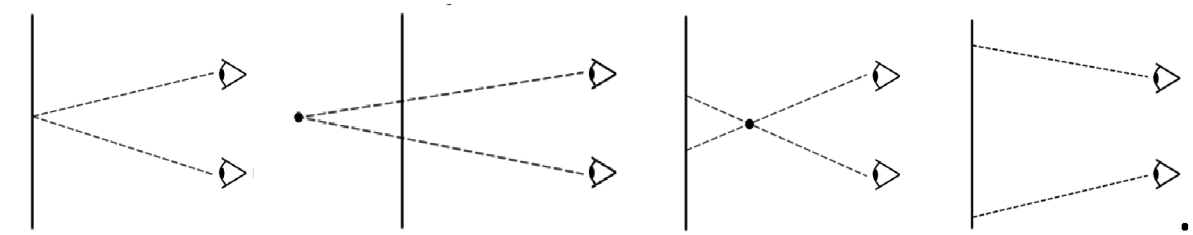
Nasleduje stručný prehľad najpoužívanejších bezkontaktných metód na meranie vzdialenosti objektov.

Stereoskopické videnie

Táto, momentálne asi najpoužívanejšia metóda, funguje na rovnakom princípe ako ľudské priestorové videnie, porovnáva rozdiel v obrazoch nasnímaných dvoma zhodnými 2D kamerami umiestnených od seba v známej vzdialenosti. Na základe použitého osvetlenia existuje pasívna alebo aktívna stereoskopia. Pasívna využíva len okolité svetlo, zatiaľ čo pri aktívnej snímacie zariadenie disponuje vlastným zdrojom osvetlenia. Metóda stereoskopického videnia využíva pasívnu stereoskopiu [26].

Kamery musia byť pri tejto metóde umiestnené tak, aby ich pozorovacie osi boli navzájom rovnobežné. Tým sa dosiahne mierne odlišný pohľad sledovanej scény, obrazy budú horizontálne posunuté. Výsledná hĺbka je potom získaná analýzovaním vzťahov medzi stredom jednotlivých kamier a pozorovaným objektom. Dôležitým faktorom je tzv. paralaxa, veľkosť posunu pozorovaného objektu v snímkoch z jednotlivých kamier.
Paralaxa môže byť [31]:

- **Nulová** – nulový posun medzi zobrazením objektu kamerami. Pri zobrazení ľavou kamerou je objekt umiestnený na rovnakom mieste spoločnej časti záberu ako pri zobrazení pravou kamerou.
- **Pozitívna** – Na obraze ľavej kamery je snímaný objekt v ľavej časti, zatiaľ čo na snímku pravej kamery sa nachádza v jeho pravej časti.
- **Negatívna** – Opak pozitívnej paralaxy, pravá kamera objekt zobrazuje v ľavej časti obrazu, zatiaľ čo ľavá kamera v pravej časti.
- **Divergentná** – Ide o špeciálny prípad pozitívnej paralaxy, kde je posun medzi zobrazením objektu väčší ako rozostup kamier. Zobrazovaný objekt by musel byť umiestnený za kamerami. V reálnom svete tento typ neexistuje.



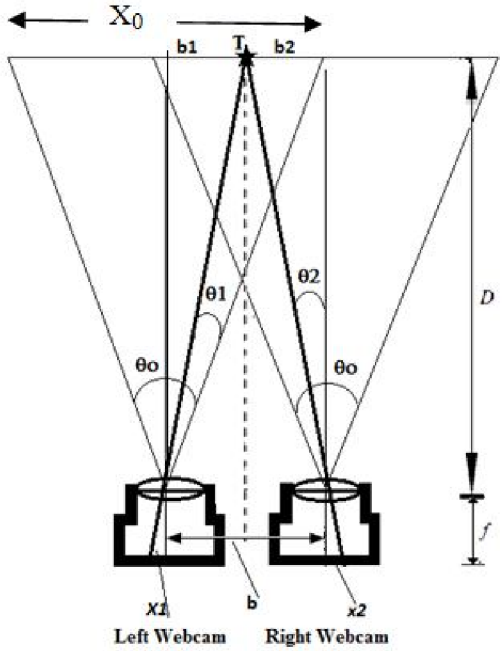
Obrázek 3.3: Zľava doprava: nulová paralax, pozitívna paralax, negatívna paralax, divergentná paralax [11].

Veľkosť paralaxy p sa vypočíta ako $x1 - x2$, kde $x1$ je posun x-ovej súradnice objektu v obraze kamery číslo jedna oproti stredu a $x2$ predstavuje ten istý údaj, ale získany z druhej kamery. Výsledný vzťah pre výpočet vyzerá potom takto :

$$D = f * \frac{b}{p} \tag{3.1}$$

kde D predstavuje hľadanú vzdialenosť, p vypočítanú paralaxu, f ohniskovú vzdialenosť a b rozostup kamier. Tento vzťah je odvodený použitím pasívnej triangulácie, názornu ukážku je možné nájsť v [23].

So zvyšovaním rozostupu kamier b narastá aj presnosť detekcie vzdialenosti D, ale znižuje sa priestor, ktorý snímajú obidve kamery. Ďalšou nevýhodou je tiež nutnosť synchronizácie kamier. Výhodou tohto prístupu je zase nízka cena a nenáročnosť technológie.



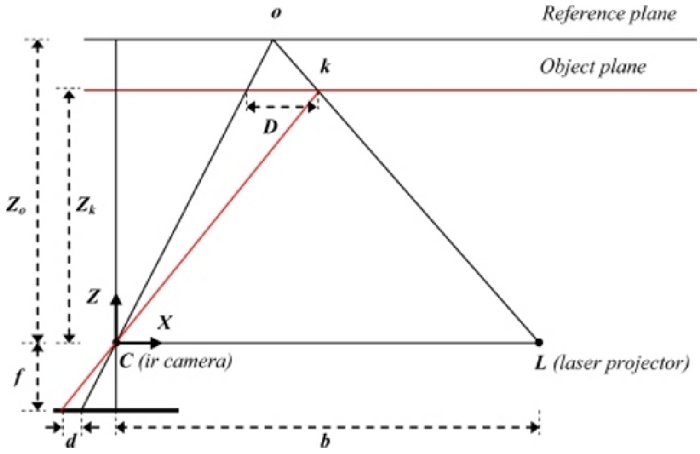
Obrázek 3.4: Názorný obrázok výpočtu vzdialenosti pomocou stereoskopie [23].

Metóda štrukturovaného svetla

Tento spôsob pracuje na podobnom princípe ako prvá metóda, ale miesto dvojice kamier sa skladá zo zdroja svetelného alebo infračerveného lúča a prijímača. Tieto dve časti sú, podobne ako v predošlom prípade kamery, umiestnené od seba v známej fixnej vzdialenosti. Priamy výpočet vzdialenosti takisto prebieha pomocou triangulácie, avšak miesto porovnávania dvoch obrazov sa posudzuje pozícia odrazeného lúča v zachytenom snímku. Čím bližšie sa tento lúč zachytený na snímku nachádza k svojmu zroju, tým kratšia je aj vzdialenosť k meranému objektu. Tento postup sa nazýva aktivná triangulácia. Výsledný vzorec má potom tvar [21]:

$$Z_k = \frac{Z_0}{1 + \frac{Z_0}{f*b} * d} \tag{3.2}$$

Význam jednotlivých premenných v rovnici je jasný z obrazku 3.5.



Obrázek 3.5: Aktivná triangulácie používaná pri metóde štruktúrovaného svetla. Obrázok je prevzaný z [21].

Podľa typu vysielača možme zariadenia tohto typu rozdeliť na [26]:

- **Lasérové skenery** – zvyčajne používajú laser a na objekt premietajú čiaru, ktorá je snímaná v nespojitých úsekoch.
- **Skenery s pevným vzorom** – disponujú laserom alebo LED vysielačom a na pozorovaný predmet vysielajú pevne daný vzor. Narozdiel od predošlého typu, snímač zachytáva všetky odrazené vzory naraz.
- **Skenery z programovateľným vzorom** – Fungujú rovnako ako skenery s pevným vzorom, ale naviac obsahujú svetelný modulátor umožňujúci dynamický meniť vysielaný vzor v závislosti na optických vlastnostiach skúmaného telesa.

Výhodou je nezávislosť metódy na vonkajšom osvetlení a pomerne vysoká presnosť. Problémom však bývajú objekty z lesklých materiálov ako aj fakt, že rýchlosť skenovania objektu je limitovaná výkonom snímača, kde použitie kvalitnejšieho hardvéru ma výrazny vplyv na výslednú cenu zariadenia. Túto metódu snímania hĺbky objektu používa aj zariadenie Kinect, ktoré bolo použité pri riešení tejto práce.

Metóda času odrazu

Tento spôsob je založený na meraní času alebo fázového posunu medzi vysielaným a prijímaným svetelným signálom. Zvyčajne sa meranie vykonáva pre každý zobrazený pixel v obraze.

Keďže na meranie vzdialenosti sa používa, rovnako ako v predchádzajúcej metóde, laser alebo iný zdroj infračerveného svetla, táto metóda trpí rovnakými nedostatkami. Jej výhodou je na druhej strane možnosť merania vzdialenosti všetkých pixelov scény zároveň ako aj vysoká presnosť a vysoký dosah [26].

3.3.1 Kinect

Kinect je zariadenie na ovládanie hernej konzoly Xbox pomocou pohybových gest a hlasových príkazov vyvinutý firmou Microsoft v roku 2010. Hlavnou časťou je 3D kamera.

Základný princíp snímania hĺbky Kinectu je založený na vysielaní infračervených vzorov a následnom zaznamenaní infračerveného obrazu pomocou CMOS snímača. Z toho je jasné, že používa metódu štruktúrovaného svetla. Jej princípy sú vysvetlené v príslušnom odstavci.

Tabuľka 3.1 obsahuje základné technické parametre 3D kamery Kinectu:

Parameter	Hodnota
Uhlové zorné pole	57° horz., 43° vert.
Framerate	cca. 30Hz
Rozlíšenie	640x480
Dosah detekcie hĺbky	0.8 m - 5 m

Tabulka 3.1: Technické parametre kamery Kinectu[15]

Obrázek 3.6: Kinect a jeho hlavné časti [29].

3.4 OpenCV

OpenCV je knižnica vydaná pod licenciou BSD zaoberajúca sa počítačovým videním. Má podporu pre široké spektrum operačných systémov a programovacích jazykov. Konkrétne ide o systémy Linux, Windows, Mac OS, iOS a Android, a programovacie jazyky C++, C, Python a Java. Táto knižnica ponúka množstvo algoritmov zaoberajúcich sa rôznymi oblasťami počítačového videnia.

V tejto práci bolo OpenCV použité k detekcií postavy metódou histogramu orientovaných gradientov. Hlavným dôvodom pre použitie tejto knižnice bola jej dostupnosť, nakoľko OpenCV je voľne dostupná a je ju možné zadarmo stiahnúť z ich oficiálnej stránky.

Kapitola 4

Počítačové videnie

Pojmom počítačové videnie sa označuje vedný obor zaoberajúci sa spracovaním, analýzou a pochopením informácií obsiahnutých v obrázkoch s cieľom transformácie dát z reálneho sveta do číselnej podoby. Aplikácia poznatkov získaných jeho štúdiom bola kľúčová pre zdarné dokončenie môjho programu.

Počítačové videnie je možné rozdeliť na základe vykonávaných úloh do nasledujúcich kategórií [17]:

- **Rozpoznávanie obrazu** – túto kategóriu je možné ďalej rozdeliť na detekciu, identifikáciu a rozpoznávanie objektov. Detekcia sa zaoberá vyhľadaním istého známeho vzoru v obrázku, príkladom je detekcia ľudskej postavy, ktorá je aj hlavnou témou tejto práce. Pri identifikácií sa rozpoznáva nejaký konkrétny vzor, napríklad rozpoznávanie ručne písaných číslic. Rozpoznávanie objektov sa zase venuje analýze jednotlivých premetov zo scény, do tejto podkategórie spadajú aplikácie založené na segmentácií obrazu.

- **Analýza pohybu** – zaoberá sa sledovaním trajektórie alebo meraním rýchlosti istého objektu, príkladom je detekcia gest.

- **Rekonštrukcia scény** – cieľom je vytvoriť 3D dátový model zobrazenej scény. Zvyčajne je k dispozícií viacero obrazov danej oblasti zachytených z rôznych uhlov.

- **Rekonštrukcia obrazu** – slúži na odstránenie šumu a skreslenia. Hlavným princípom je použitie rôznych filtrov a algoritmov na dopočítanie zdeformovanej časti obrazu

Z týchto kategórií je pre túto prácu významné len rozpoznávanie obrazu a do istej miery analýza pohybu, preto sa ostatnými nebudem ďalej zaoberať.

Väčšina programov vykonavajúcich nejakú funkciu založenú na počítačovom videní sa skladá z týchto častí [18]:

1 **Získavanie obrazových dát** – výsledný digitálny obraz môže byť zložený z dát pochadzajúcich z jednej alebo viacerých kamier a môže obsahovať tak informácie o farbe ako aj o iných parametroch okolia, ako je napríklad hĺbka alebo teplota.

2 **Predzpracovanie** – pred aplikovaním samotných algoritmov na detekciu a identifikáciu obsahu je často nutné obraz upraviť do vhodnejšej podoby. Často je výhodné zmeniť veľkosť, orientáciu alebo kontrast obrazu či aplikovať filtre na redukciu šumu a skreslenia.

3 **Extrakcia čŕt** – v tomto kroku sa z obrazu extrahujú vybrané črty, ktoré nás zaujímajú, ako sú napríklad čiary, obrysy, hrany, rohy alebo zhluky bodov. Takisto je možné oddeli objekty scény od pozadia.

4 **Segmentácia** – snímaný obraz sa rozdelí na menšie regióny v závislosti na prítomnosti nejakého spoločného rysu extrahovaného v predošlom kroku. Cieľom je odstrániť všetky dáta, ktoré nesúvisia s konkrétnym skúmaným objektom, ako sú pozadie a ostatné predmety. Výsledným produktom segmentácie je malá vzorka dát, väčšinou predstavujúca jeden jediný objekt.

5 **Detekcia** – pri detekcií je konkrétny segment porovnávaný na existenciu známych čŕt spoločných pre nejakú triedu objektov. Na základe tohto porovnania je potom zaradený do príslušnej kategórie.

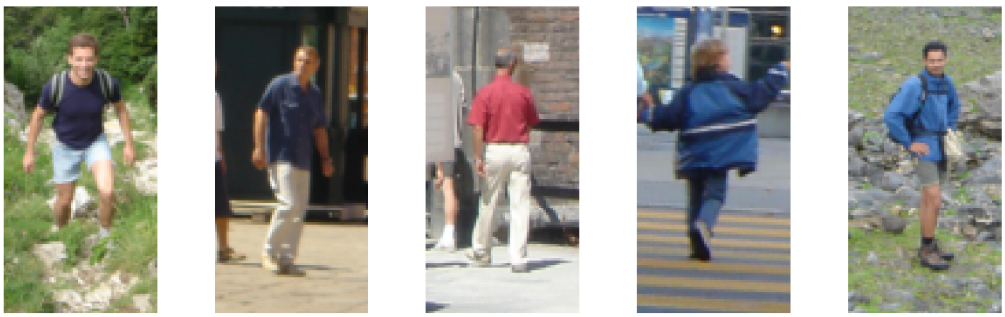
6 **Vyhodnotenie** – výsledné rozhodnutie vyplyvajúce z analýzy obrazu. Často ide o pravdivostnú informáciu, či sa nachádza daný objekt v príslušnom obraze.

Podľa jejto štruktúry je napísaná aj výsledná aplikácia, ktorej je venovaná kapitola 5.

4.1 Detekcia postavy v obraze

K detekcií postavy sa prístupuje podobne ako k detekcií ľubovoľného objektu v obraze. Obraz je rozdelený na menšie časti alebo sú z neho vybraté jednotlivé objekty pomocou segmentácie, ktoré sú potom porovnávané na zhodnosť čŕt so známymi obrázkami. Avšak rôznorodosť ľudí tento proces výrazne zťažuje. Pri detekcí bežných objektov, ako sú stromy alebo cesty, sa často používa ako jeden z rozhodovacích faktorov farba objektu, ľudská postava je však oblečená v odeve vpodstate ľubovoľnej farby. Okrem farebnosti sa nosené odevy líšia aj typom a tvarom, šaty majú iný obrys ako napríklad smoking. Rôznorodosť obrysov zvyšuje aj množstvo póz, ktorých sú ľudia schopný. človek môže sedieť, stáť rozpažený, s jednou rukou predsebou alebo nadobúdať množstvo iných póz, pri detekcií automobilu stačí dbať len na rôznu orientáciu.

Detekcia postavy sa zvyčajne vykonáva za účelom zistenia prítomnosťi, počtu, lokácie, trajektórie alebo identity osôb na obrázku.



Obrázek 4.1: Rozličné ľudské postavy. Obrázok prevzatý z [25].

Existujúce metódy sa dajú rozdeliť na základe rysu použitého k detekcií na [12]:

- **Detekcia celej postavy** – obraz sa analyzuje po častiach, použitím tzv. lokalného vyhľadávacieho okna. Na prítomnosť ľudskej postavy sa porovnáva každé okno a postava sa detekuje ako celok. V prípade detekcie postav rôznej veľkosti je nutné použiť

viacero veľkosti okien, čo značne zvyšuje počet porovnávaných vzorkov a negatívne ovplyvňuje časovú náročnosť metódy. Porovnávané vlastnosti môžu byť globálne, ako napríklad obrysové vzory, alebo lokálne. Príkldom lokálnej vlastnosti je aj nami použitá metóda histogramu orientovaných gradientov.

- **Detekcia po častiach** – postavy sú modelované ako kolekcia menších objektov. Najprv sa obraz preskúma na ľahko detekovateľné lokálne črty popisujúce jednotlivé časti človeka, ako je napríklad tvár alebo hlava a plecia. Výsledky analýzy obrazu na jednotlivé vlastnosti sa potom spoja dokopy a ak niektorá časť obrázku obsahuje všetky potrebné atribúty, je možné považovať ju za detekovanú ľudskú postavu. Na rozpoznanie jednotlivých rysov je použitá štandardná husto vzorkovaná pyramída, hľadajúca požadované vlastnosti vo vyhľadávacích oknách všetkých veľkostí, pokryvajúcich celý obraz. Nevýhodou tejto metódy je vysoká časová a výpočetná náročnosť.

- **Detekcia implicitných tvarov** – tento prístup kombinuje detekciu so segmentáciou. Extrahované lokálne črty sú použité na porovnávanie s referenčnými obrázkami a každá zhoda sa ráta za jeden bod pre pravdepodobnosť výskytu postavy v danom mieste. Výhodou tohto prístupu je pomerne malý počet trénovacích snímkov.

- **Detekcia pohybu** – túto metódu je možné použiť len pre video záznamy. Obraz sa rozdelí na statické pozadie a na objekty, pri ktorých bola zaznamenaná pohybová činnosť. Týmto sa vyčlenia siluety objektov v popredí a následná detekcia sa vykonáva len pre časti obrazu, ktoré ich obsahujú. Celkový počet porovnaní je omnoho menší ako v predošlých metódach. Nevýhodou tohto prístupu je obmedzené použitie pre obrazy nasnímané pohyblivými kamerami.

- **Detekcia viacerými kamerami** – pri tejto metóde je na zachytnie scény použitých viacero kamier. Cieľom je vytvoriť mapu scény rozdelenú na polia, pre ktoré sú vypočítané pravdepodobnosti výskytu človeka, vznikne tzv. mapa pravdepodobnosti výskytu (Probability Occupancy Map). Touto metódou je možné sledovať pozíciu až šiestich postáv súčašne.

Nasleduje popis najpoužívanejších metód používaných pri rozpoznávaní objektov.

4.1.1 RANSAC

RANSAC (Random Sample Consensus) je iteratívny algoritmus na odhadnutie parametrov a vytvorenie matematického modelu z pozorovaných dát. Algoritmus je produkuje správne výsledky len s určitou pravdepodobnosťou, čím viac iterácií sa vykoná, tým je táto pravdepodobnosť vyššia. Základnou myšlienkou je, že dáta sa skladajú z validných dát, ktorých distribúciu je možné opísať istým súborom parametrov modelu a z extrémov, ktoré do tohto opisu nesedia. Extrémy pochádzajú zo šumu, skreslenia alebo z nesprávnej interpretácie popisovaných dát (na obrázku je iný objekt, ako sa predpokladá). Ďalším predpokladom je, že z pomerne malého vzorku validných dát je možné odvodiť parametre modelu, ktorý optimálne popisuje ich rozloženie [13].

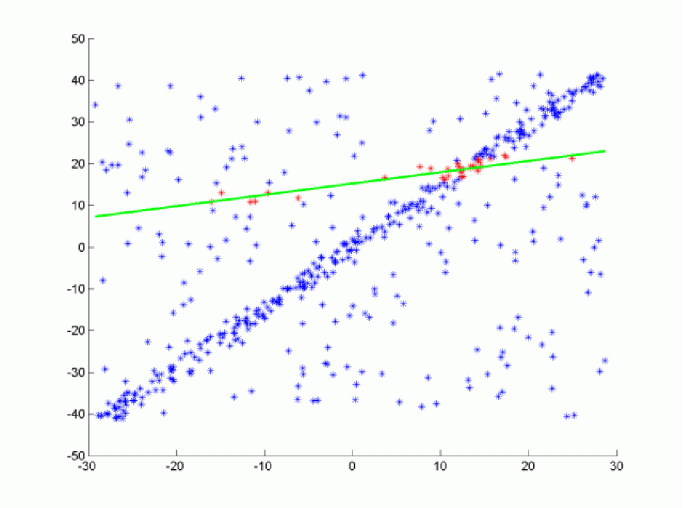
Algoritmus pracuje v dvoch krokoch, ktoré sa potom opakujú až do uspokojivého výsledku. Na začiatku máme k dispozícií parametre predpokladaného objektu [24].

1. V prvom kroku sa náhodne vyberie vzorka dát z obrázku. Parametre modelu sú prepočítané len na základe tejto vzorky dát.

2. V tomto kroku algoritmus zistí, ktoré časti celého obrazu súhlasia s parametrami modelu vytvoreného v prvom kroku. Dáta, ktoré tomuto predpokladanému modelu nevyhovujú sú označené za extrémy a ignorované. Ak ich celkový počet presiahne nejakú maximálnu hodnotu chybovosti, použitý model sa považuje za nesprávny.

Set vzorky dát, ktorá v druhom kroku vyhovovala predpokladanému modelu, sa nazýva *consensus*. Consensus predstavuje mieru správnosti parametrov modelu. Každou iteráciou sa upravuje popis modelu na základe novej vzorky dát, a na základe porovnania jeho veľkosti sa na ďalšie výpočty buď použije nový alebo predošlý model. Model sa považuje za absolutne spravny, keď consensus obsahuje dostatočný počet dát.

Výhodou algoritmu je dobrá presnosť aj pri vysokom skreslení, nevýhodou je možná neobmedzená časová náročnosť. V prípade limitovaného počtu iterácií zase nemusí nájdené riešenie byť optimálne alebo dokoca ani správne.



Obrázek 4.2: Princíp funkcie algortmu RANSAC. Obrázok prevzatý z [13].

4.1.2 Haar-wavelet Cascade Detector

Základnou myšlienkou tohto prístupu je, že nesprávnych vzoriek je omnoho viac ako správnych a väčšina z nich je ľahko vylúčiteľná. Kaskádové detektory preto obsahujú sériu testov, z ktorých každý ďalší je o čosi komplexnejší, ale aj časovo a výpočetne náročnejší, ako predošlé. Tým sa výrazne zníži potrebný čas a prostriedky na vylúčenie nesprávnych vzoriek.

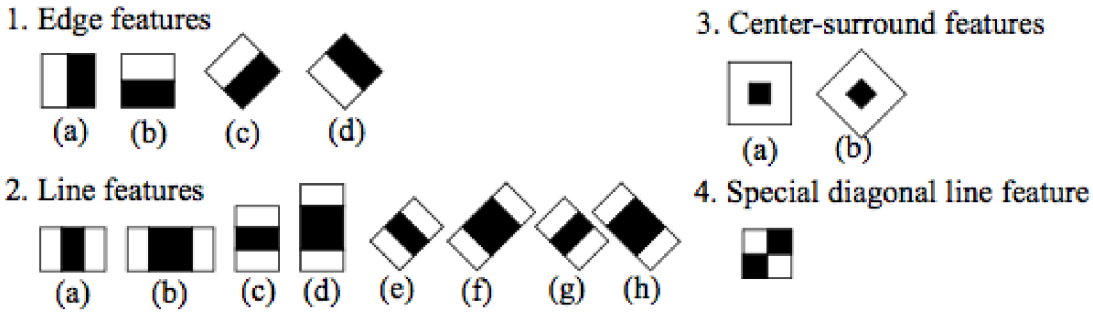
Haar-wavelet používa ako jednotlivé klasifikátory (testy) Haarove klasifikátory. Haarov klasifikátor obsahuje jedinú Haarovu črtu a jej prah dovolených hodnôt. Vzorka sa potom klasifikuje na základe črtu, porovná sa s prahom a test sa vyhodnotí ako úspešný, alebo v prípade nevhodnosti vzorky ako neúspešný.

Haarova črta je založená na fakte, že rôzne obrázky toho istého objektu majú vo všeobecnosti tmavšie a svetlejšie miesta nachádzajúce sa na rovnakých miestach. Napríklad pri detekcií tváre sú zväčša regióny oči a pier tmavšie, zatiaľ čo líca a nos svetlejšie.

Konkrétne prevedenie Haarových klasifikátorov je umiestnenie viacerých príľahlých obdĺžnikov do detekčného okna a následné spočítanie hodnoty všetkých pixelov nachádzajúcich sa v ich vnútri. Tieto sumy sa potom porovnajú navzájom a keď tento rozdiel je v hodnota rámci prahu pre daný klasifikátor, test je považovaný za úspešný.

Tento kasifier je pomerne slabý, a preto sa používa vo veľkých množstvách umiestnených v kaskádach. Hlavnou výhodou je rýchlosť detekcie.

Tieto informácie boli čerpané z [28].



Obrázek 4.3: Prehľad Haarovych čŕt. Obrázok prevzatý z [16].

4.1.3 SIFT

SIFT (Scale-invariant Feature Transform) je algoritmus používaný na detekciu a popis lokálnych čŕt v obraze. Je ho možné použiť na širokú škálu úkonov, od rozpoznávania objektov, cez sledovanie pohybu, až po vytváranie 3d scény.

Prvým krokom detekcie pomocou SIFT algoritmu je nájdenie a vypočítanie zmene veľkosti odolných čŕt v obrázku. Tieto vlastnosti sú získané z celého snímku naraz v týchto štyroch krokoch:

1 **Detekcia extrémov pri zmene veľkosti** – k nájdeniu extrémov odolných zmene veľkosti obrázka sa používa prístup nazvaný scale-space. Jej základom je popis obrázku pri rôznej zmene veľkosti pomocou Gaussovej funkcie $G(x, y, \sigma)$. K spoľahlivej detekcií stabilných vrcholov v snímku je urobený rozdiel výsledných Gaussových funkcií popisujúcich obraz susediacich veľkostí $D(x, y, \sigma)$. Z týchto rozdielov sa potom náhodne vyberajú vzorky, ktoré sa porovnávajú so svojimi susedmi, ôsmimi v snímku ich veľkosti a s deviatimi v snímkach o úroveň zmenšenom a zväčšenom. Vzorka je za extrém považovaná len v prípade, že jej hodnota je najväčšia alebo najmenšia spomedzi všetkých susedných bodov.

2 **Lokalizácie bodov** – v tomto kroku sa získané extrémy porovnávajú so svojím okolím a určuje sa ich skutočná poloha a veľkosť, čím sa vylúčia body so slabým kontrastom a extrémy vzniknuté kvoli šumu. K tomu sa používa 3D kvadratická funkcia taylorovej rady. Ak je nejaký extrém bližšie k inej vzorke ako k tej, z ktorej bol získaný, urobí sa interpolácia a pozícia extrému sa preráta s použitím novej vzorky. Extrémy s nízkym kontrastom sa jednoducho vynechajú na základe nízkej hodnoty pixelu oproti okoliu v oblasti jeho výskytu. Ďalším krokom je vyfiltrovať body označené za maximá nachadzajúce sa na hranách (skutočné maximá sú vrcholy alebo objekty v popredí). To je dosiahnuté použitím Hessianovej matice.

3 **Získanie orientácie** – orientácia vrcholov sa získa použitím histogramu orientovaných gradientov. Pre každý detekovaný extrém sa vytvorí vlastný histogram z gradientov bodov v jeho okolí. Najvyššia hodnota z každého histogramu sa použije na

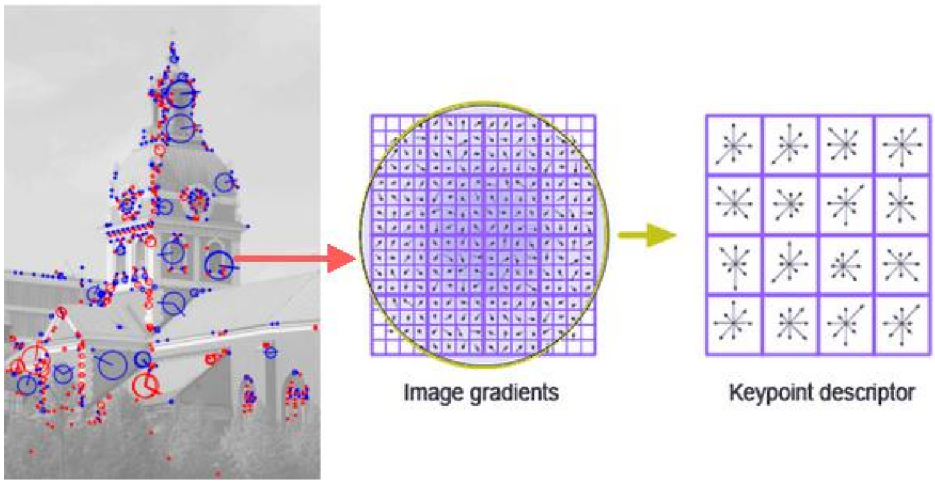
vytvorenie bodu s príslušnou orientáciou v danom extréme. Ak niektorý z histogramov obsahuje aj iné orientácie s veľkosťou aspoň 80 % najvyššej hodnoty, tak sa v danom mieste vytvorí viacero bodov s odlišnou orientáciou.

4 **Vytvorenie deskriptorov** – z predošlých krokov máme zistené extrémy, ich lokáciu, veľkosť a orientáciu. Z týchto parametrov je možné vytvoriť pomerne jednoznačný popis jednotlivých častí obrazu. Tento popis je vytvorený pomocou 128 rozmerného vektoru. Ten je získaný ako suma šestnástich osemsmerných histogramov, vytvorených z poľa o veľkosti 16*16 vzoriek. Tieto vektory sú potom normalizované, aby sa dosiahlo odolnosti voči zmene osvetlenia.

Takto získané popisné črty sú potom porovnávané s tými, ktoré máme v databáze priradené k známym objektom, pomocou algoritmu best-bin-first, založeného na metóde vyhľadania najbližšieho suseda pomocou Euklidovskej vzdialenosti. K ďalšiemu vyčleneniu nesprávne detekovaných objektov slúžia Houghove transformácie. Tieto odhalia zhluky bodov s črtami popisujúcimi ten istý objekt v tej istej polohe a orientácií. Ak zhluk obsahuje aspoň tri takéto body, je označený za možného kandidáta. Pre každého kandidáta je potom vykonávaná metóda najmenších štvorcov, pomocou ktorej sa vylúčia body s nesprávnymi parametrami. Metódu najmenších štvorcov prestaneme vykonávať, keď už nedochádza k odstráneniu žiadných bodov. Ak aj po tomto kroku v zhluku ostala aspoň trojica bodov, dané miesto obsahuje detekovaný objekt.

Najväčšou výhodou SIFT algoritmu je odolnosť voči zmenám veľkosti, orientácie a čiastočne aj voči zmene osvetlenia.

Informácie o tomto algoritme boli čerpané z [22].



Obrázek 4.4: Názorná ukážka výpočtu výsledného vektoru z gradientov v okolí vrcholu. Žltý kruh určuje váhu jednotlivým gradientom a predstavuje veľkosť (plochu) extrému získanú aplikáciou Gaussovej funkcie. Zdroj : [22].

4.1.4 SURF

Algoritmus SURF (Speeded Up Robust Features) je určený na detekciu a popis bodov záujmu v obraze. Jeho hlavné princípy vychádzajú z metódy SIFT. Algoritmus je použiteľný

na širokú škálu úkonov. Je ním možné vyhľadávať a rozpoznávať objekty, sledovať ich trajektóriu, vytvárať 3D scénu alebo len vytvoriť popis bodov záujmu v obrázku.

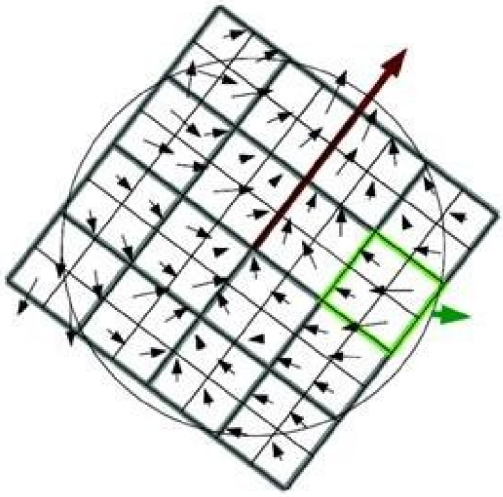
Princíp detekcie extrémov je podobný ako v metóde SIFT, ale v každom kroku používa odlišné metódy. K detekcií extrémov nepoužíva rozdiel Gaussovych funkcií, ale tzv. integrálne obrazy . Tie vzniknú ako funkcia hodnôt všetkých pixelov v časti obrázku ohraničenou obdĺžnikovým konvolučným filtrom. Následne sa na tieto obrázky použijú Hessianove matice, ktoré sú použité na detekciu kvapkovitých (blob-like) štruktúr. Tie sa nachádzajú v mieste, kde je determinant výsledných matíc maximálny. Tieto body je však potrebné detekovať aj v iných veľkostiach vstupného snímku. K tomu slúži prístup scale-space použitý aj v metóde SIFT, ale narozdiel od nej, SURF nepoužíva pyramídu rôznych veľkostí obrazov, ale využíva použité konvolučné filtre a integrálne obrazy. Miesto zmeny veľkosti celého snímku sa mení len použitý konvolučný filter. Na zistenie konkrétnej polohy sa používa rovnaká Taylorová rada, použitá aj v algoritme SIFT.

K popisu extrémov sa nepoužíva histogram orientovaných gradientov, ale vlastná metóda založená na princípe Haar-wavelet. V prvom kroku sa vypočíta orientácia bodu v kruhovom rádiuse v okolí zisteného extrému. To sa vykoná použitím klasických Haar-wavelet štvorcov aplikovaných na integrálne obrázky. Výsledky sa pretransformujú na lokálny orientačný vektor. Tento vektor slúži na zachovanie nezávislosti deskriptoru na zmene orientácie obrazu. V ďalšom kroku sa okolo extrému vytvorí štvorec rozdelený na ďalších 16 menšich štvorcov. V každom z týchto regiónov sa potom vypočíta smerový vektor rovnakým spôsobom ako v prvom kroku. Tieto jednotlivé vektory sú potom zapísané formou výsledného vektoru o veľkosti šestdesiatštyri položiek.

Posledný krok je porovnávanie regiónov s referenčnými obrázkami v databáze. Na zrýchlenie tohoto procesu sa používa indexácia na základe znamienka Laplaciana. Znamienko Laplaciana potom rozlišuje tmavé bloby na bielom pozadí od opačnej situácie. Pri samotnom porovaní sa potom porovnáva len kontrast popisných čŕt.

Výhodou tohto algoritmu oproti metóde SIFT je mnohonásobne vyššia rýchlosť detekcie a vyššia odlonosť voči transformáciam obrazu.

Informácie o tejto metóde boli čerpané z [20].



Obrázek 4.5: Ukážka deskriptoru SURF. Vektory v každom štvorci sú získakné pomocou metódy Haar-wavelet. Zdroj : [20].

4.1.5 FAST

FAST (Features from Accelerated Segment Test) je technika určená na vyhľadávanie rohov v obraze. Tie potom slúžia ako popisné črty obrázku, na základe ktorých je možné vykonať detekciu objektov.

Funguje na princípe porovnania intenzity kruhu pixelov v okolí vybraného bodu. Použitý kruh má veľkosť šestnásť pixelov a jeho polomer sú tri pixely. Daný bod je považovný za rohový, ak existuje počet susediacich pixelov v kruhu n, ktoré sú všetky tmavšie ako súčet intezity skúmaného bodu a hodnoty prahu, alebo sú všetky svetlejšie ako tento rozdiel. V praxi sa ukázalo výhodné použiť pre n hodnotu dvanásť. Potom je možné použiť na vyfiltrovanie bodov, ktoré nemôžu byť rohom, zrýchlený test, kedy sa porovnávajú len body z kruhu označené 1, 5, 9, 13. Aby mohol byť vybraný bod považovaný za roh, musia byť svetlejšie alebo tmavšie aspoň tri z nich. Tento rýchly test však nefunguje, ak je $n < 12$.

Na vyriešenie tohto problému sa používa metóda strojového učenia. Algoritmus najprv otestuje všetky body vo všetkých testovacích snímkach. Pixely z kruhového okolia sú rozdelené podľa rozdielu ich intezity oproti bodu, ku ktorému daný kruh patrí, do troch kategórií, tmavšie, svetlejšie a podobné. Následne je použitý algoritmus rozhodovacieho stromu ID3, aby sa zistilo, ktorá pozícia pixelu z kruhového okolia bodu obsahuje najviac informácií potrebných k rozhodnutiu, či skúmaný bod je rohový (nakoľko ide o testovacie obrázky, rohy sú dopredu známe). Tento proces výberu pozície pokračuje rekurzívnym aplikovaním na vzniknuté podstromy, až kým podstromy neobsahujú samé rohy, alebo naopak, žiadne rohy. Týmto nám potom vznikne rozhodovací strom, ktorý je možné použiť na detekciu rôznych typov rohov.

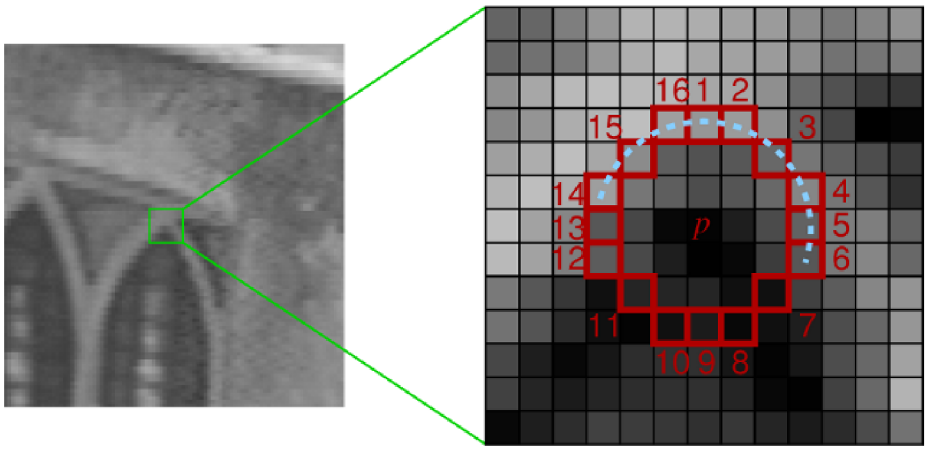
Posledným krokom v algoritme je odstránenie prípadov, kedy je viacero susedných bodov označených za ten istý roh. K tomu sa používa hodnotiaca funkcia, ktorá hodnotí body na základe nasledujúcich kritérií:

1. Maximálna hodnota počtu bodov z kruhu n, pri ktorej je bod stále považovaný za roh
2. Maximálna hodnota prahu, pri ktorej je bod stále považovaný za roh.
3. Suma rozdielov medzi jednotlivými pixelmi z kruhu a rohovým bodom.

Na základe tejto hodnotiacej funkcie sa potom z hluku rohových bodov vyberie ten s najväčšou hodnotou.

Hlavnou výhodou algoritmu FAST je omnoho vyššia rýchlosť ako u väčšiny ostatných prístupov.

Informácie boli čerpané z [19].



Obrázek 4.6: Ukážka detekcie rohu pomocou metódy FAST. Zdroj : [19].

4.1.6 HOG

HOG (Histogram of Oriented Gradients) je momentálne najpoužívanejší detektor ľudskej postavy. Funguje na podobnom princípe ako metóda SIFT, ale líši sa v tom, že nehľadá v obrázku extrémy, ale histogramy počíta pre regióny rovnakej veľkosti pokrývajúce celý snímok. HOG je oproti ostatným metódam výnimočný tým, že hľadaný objekt nepopisuje ako súbor lokálnych čŕt, ale na základe jedinej globálnej črty popisujúcej jeho tvar.

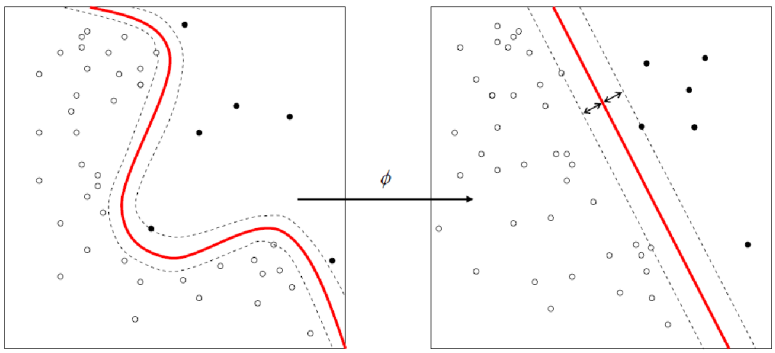
Na vytvorenie jednotlivých histogramov popisujúcich prekrývajúce sa časti obrazu sa používa posuvné detekčné okno. Pre detekciu ľudskej postavy sa používa okno o veľkosti 64*128 pixelov. Toto okno je ďalej rozdelené na regióny o veľkosti 8x8 pixelov. V každom z týchto regiónov sa vytvorí histogram orientovaných gradientov s deviatimi pinmi, ktorý obsahuje gradienty jednotlivých pixelov. Je možné použiť histogram pokrývajúci celých 360°, ale v pôvodnom algoritme bol použitý histogram pokrývajúci len 180°. Na jeden pin potom pripadlo 20°. Hodnotu vektorov, ktorých orientácia nebola hodnotou žiadného pinu, rozdelíme medzi dva najbližšie, a zachováme príslušný pomer. Ak je napríklad vektor s orientáciou 75°, tak 1/4 jeho hodnoty pridelíme pinu s hodnotou 70° a zvysok pinu 90°.

Deskriptor potrebuje byť odolný zmene osvetlenia a kontrastu. K tomu sa používa normalizácia, ale miesto normalizácie každého histogramu sa používa normalizácia bloku. Blok je tvorený štvoricou regiónov, takže obsahuje štyri histogramy. Tie sa konkatenuju a vytvorí sa z nich vektor. Tento vektor sa potom vydelí jeho hodnotou, čím sa normalizuje. Jednotlivé bloky sa z 50 % prekrývajú, čím sa dosiahne, že vo výslednom deskriptore, sa histogram každého regiónu objaví dvakrát. Deskriptor má podobu vektoru zloženého z vektorov jednotlivých blokov v okne. Obsahuje 3780 hodnôt [25].

Obrázek 4.7: Prekrývanie blokov obsahujúcich štvoricu regiónov. Zdroj : [25].

Posledným krokom k detekcií objektu je porovnávanie deskriptoru pre danú oblasť s deskriptormi popisujúcimi referenčné obrázky. To sa robí pomocou trénovaného SVM (support-vector machine).

SVM je binárny klasifikátor, čo znamená, že vstupy rozdeľuje na dva výstupy, v našom prípade, rozhoduje o tom, či je na zobrazenom obrázku človek alebo tam nie je. Princíp binárneho klasifikátoru spočíva v tom, že na základe istého počtu vstupných vzoriek, ktorých zaradenie do tried poznáme, sa postupne prerátavá lineárna funkcia, pomocou ktorej je možné rovinu rozdeliť na dve časti. V našom prípade jedna časť obsahuje objekty klasifikované ako ľudské postavy, druhá ostatné objekty. Ak dané skupiny nie je možné oddeliť lineárnou funkciou, použije sa podporný vektor, ktorý dané body preusporiada, často do viacerých rozmerov [14].



Obrázek 4.8: Ukážka aplikácie podporných vektorov. Zdroj : [14].

Výhodou HOG oproti ostatným metódam je, že bola vyvýjaná s dôrazom na detekciu postavy, takže pri plnení tejto úlohy dosahuje najvyššiu presnosť a zároveň je dostatočne rýchla, aby mohla byť použitá v aplikáciach pracujúcich v reálnom čase.

Kapitola 5

Návrh a implementácia

Táto kapitola sa venuje bližšiemu popisu vytvoreného riešenia. V prvej časti je popísaný návrh aplikácie s dôrazom na použité algoritmy a riešenia problémov, ktoré sa pri vývoji aplikácie vyskytli. Druhá časť sa potom venuje ich samotnej realizácií.

5.1 Návrh

V úvode sú popísané existujúce aplikácie, zhrnuté ich výhody a nevýhody, ako aj dôvod, prečo ich nie je vhodné použiť miesto mojej aplikácie. Hlavná časť sa venuje použitým algoritmom na úpravu obrazu ako aj samotnú detekciu, a navrhnutým vzorcom pre výpočet rýchlosti pohybu robota. Posledná časť potom popisuje vyskytnuté problémy a to, ako som ich v konečnom dôsledku vyriešil.

5.1.1 Existujúce riešenia

Prvým krokom v návrhu riešenia bolo vyhľadať a preskúmať existujúce riešenia daného problému, ktoré by mohli poslúžiť ako inšpirácia. Nasleduje prehľad najlepších a najkonkrétnejších aplikácií, ktoré som bližšie študoval.

Kinect Skeletal Tracking

Medzi prvými spôsobmi detekcie a sledovania postavy, ktoré som preskúmal, bola aplikácia Kinectu Kinect Skeletal Tracking. Táto aplikácia slúži len na detekciu osôb a následné sledovanie ich pózy, nezaoberá sa riadením pohybu. Aj napriek tomu som si však aplikáciu podrobnejšie naštudoval, za účelom použitia jej voľne distribuovateľnej verzie Openni tracker obsiahnutú v ROS-ovi [6] ako detekčný modul. Jej prístup sa však neukázal ako príliš vhodný. Hlavným nedostatkom bola nutnosť sledovanej osoby stáť čelom oproti senzoru, malý dosah spoľahlivej detekcie, ako aj to, že aplikácia nebola vyvýjaná pre použitie na mobilnej základni, kedy by mohlo dochádzať k pomiešaniu jednotlivých osôb na obrazovke. Informácie o aplikácií Kinectu boli čerpané z [10].

Turtlebot Follower

Turtlebot follower je balík vyvynutý pre systém ROS, ktorý vytvára riadiaci program pre robota turtlebot, za účelom sledovania ľudí a robotov. Toto riešenie sa spočiatku javilo ako vhodné pre použitie v našej aplikácií, avšak aj ono bolo rýchlo zavrhnuté. Jeho najväčšou

nevýhodou je jeho univerzálnosť. To, že aplikácia je určená na sledovanie ľudí, ako aj robotov, ktorý môžu mať prakticky ľubovoľný tvar, malo výrazný vplyv na spôsobe, akým bola samotná detekcia implementovaná. Jej princíp je založený na jednoduchom sledovaní centroidu zhluku bodov pred robotom, takže robot sa nestará, čo sleduje. To nám neprišlo ako príliš vhodné správanie a od tejto implementácie som upustil [8].

Person Following Robot

Táto aplikácia najlepšie spĺňa požiadavky môjho zadania. Detekcia je v nej riešená pomocou mnou odmietnutého balíka Openni tracker, ku ktorému bol vyvinutý pomocný uzol, odstraňujúci hlavné nedostatky tohto prístupu. Nakoľko som sa rozhodli balík Openni tracker nepoužiť, a aj hlavne preto, že túto aplikáciu som objavili až neskôr, v čase keď už detekcia v mojom riešení spoľahlivo fungovala, bolo aj toto riešenie odmietnuté [7].

Naše riešenie

Nakoľko sa mi nepodarilo nájsť vyhovujúce existujúce riešenie, rozhodol som sa pristúpiť k vlastnému návrhu. Ten spočíval v rozdelení aplikácie do dvoch logických častí. Prvá sa zaoberá detekciou postavy použitím niektorého z na to určených algoritmov a vypočítaním reálnej pozície človeka v priestore a druhá má za úlohu na základe týchto dát vypočítať uhlovú a lineárnu rýchlosť pohybu robota a riadiť nečakané stavy. Tieto dve logické časti sú implementované v jedinom uzle systému ROS. Výhodou takéhoto prístupu je, že odpadá nutnosť vzájomnej komunikácie pomocou správ, miesto nich je možné ovládať chod programu čiste pomocou stavových premenných.

5.1.2 Použité platformy

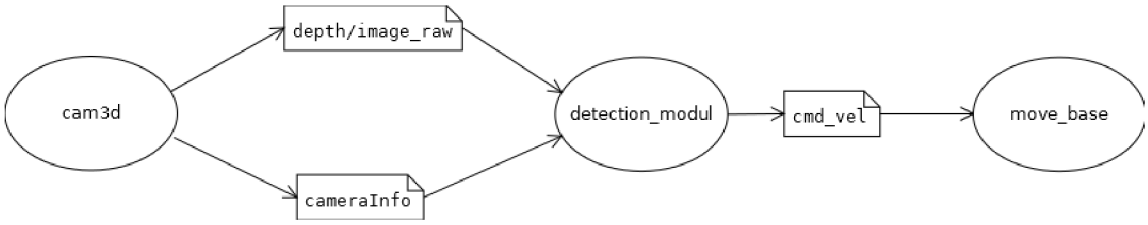
Aplikácia bola vyvýjaná pre meta-operačný systém ROS verzie Hydro (pre bližšie informácie viď kapitolu 3.2). Ten zároveň poslúžil aj ako vývojové prostredie. Hlavným dôvodom jeho použitia bola jeho dostupnosť, keďže je vyvíjaný ako open-source softvér, ako aj fakt, že je použitý na všetkých roboch, ktorých som mal k dispozícií. Z tých bol vybratý robot typu Pioneer P3-AT s názvom Toad, ktorý mal vhodné rozmery, aby bol ľahko prenositeľný a zároveň aby zvládol odniesť náklad o váhe naplnenej nákupnej tašky . Ďalšie informácie o tomto robotovi sú obsiahnuté v kapitole 3.1. Ako zobrazovacie zariadenie bola použitá RGB-D kamera Kinect, ktorá už bola na robotovi umiestnená a nakalibrovaná.

Balíčky pre systém ROS je možné písať len v programovací jazykoch C++, Python, Lisp a do istej miery v Jave alebo pomocou Lua skriptov. Java a Lua sú v ROS-e implementované zatiaľ len v experimentálnej verzií a z ostavajúcich možností som sa rozhodol použiť jazyk C++, nakoľko som s ním už v minulosti viackrát pracoval.

5.1.3 Podrobný návrh a použité algoritmy

Aplikácia je rozdelená na dve hlavné časti. Prvá sa zaoberá spracovaním obrazu a detekciou postavy a druhá riadením pohybu robota. Okrem týchto hlavných a pre správny beh aplikácie nevyhnutných častí, program obsahuje aj podporný modul pre nastavenie niektorých parametrov a vizualizáciu. Tá prebieha hlavne prostredníctvom príkazového riadku.

V diagrame 5.1 je popísaná komunikácia môjho programu s inými používanými uzlami. Bližší popis jednotlivých vzťahov je obsiahnutý v odstavcoch popisujúcich návrh konkrétnych častí.



Obrázek 5.1: Diagram popisujúci komunikáciu môjho programu s ostatnými uzlami.

Detekčný modul

Detekčný modul spracováva vstupy z kamery, upravuje obraz do vhodného tvaru a vykonáva samotnú detekciu postáv. Výstupom tejto časti je štruktúra popisujúca 3D súradnice sledovanej postavy v reálnom priestore.

Samotný detekčný modul dodržuje tradičnú štruktúru programu pre počítačové videnie (viď stranu 12).

- **Získavanie obrazu** – obraz je získany z 3D kamery Kinect, konkrétne z ROS uzlu cam3d. Ten publikuje veľké množstvo rôznych typov obrazu. Medzi inými farebné rgb obrazy, hĺbkové mapy ako aj mračná bodov. Moja aplikácia pracuje len s hĺbkovými snímkami, nakoľko farba nehrá pri mojom spôsobe detekcie takmer žiadnu rolu, a mračná bodov sú zbytočne príliš priestorovo náročné.

- **Predzpracovanie** – v tomto kroku sa získana hĺbková mapa pretransformuje do vhodného formátu na ďalšie spracovanie. Za účelom vysokej rýchlosti aplikácie boli vynechané operácie pre úpravu obrazu, ako je napríklad vyhladzovanie, ktoré by program výrazne spomaľovali.

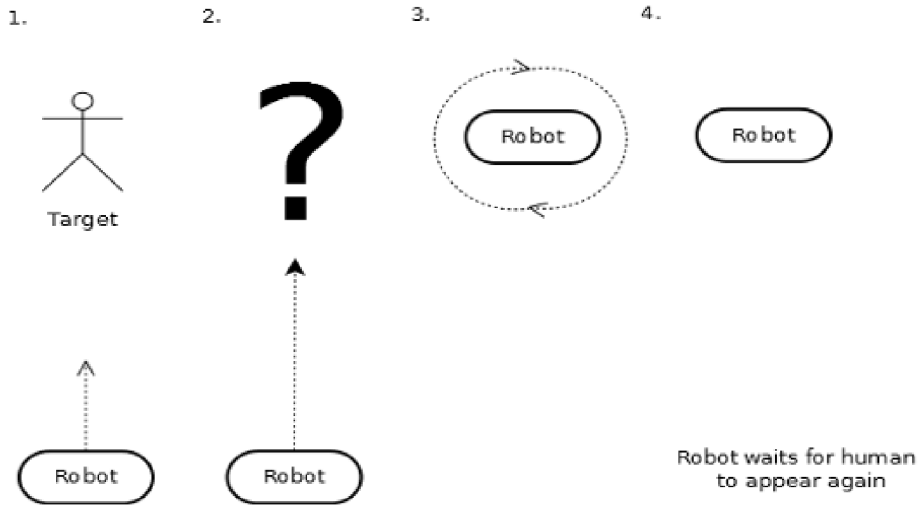
- **Extrakcia čŕt, segmentáci a detekcia** – tieto kroky sú postupne vykonávané použitím funkcie z knižnice OpenCV. Ako detekčný algoritmus bol zvolený prístup HOG, ktorý je momentálne najpoužívanejší a najlepší voľne dostupný detektor ľudskej postavy. Jeho veľkou výhodou je aj to, že je voľne dostupný, narozdiel od algoritmov SIFT a SURF, ktoré sú patentované a ich použitie je obmedzené. V tejto časti je riešené aj sledovanie detekovanej postavy. To bolo navrhnuté jednoduchým posúvaním okna, kde očakávame, že sa sledovaná postava objaví, po obrazovke. K posunu má dôjsť vždy tak, aby stred nájdenej postavy bol stredom okna. Ak bol nájdený človek označený za nami hľadaného, musí ležať nový stred niekde v oblasti prekrytej predchádzajúcim oknom. Výhodou tohto prístupu je, že danú postavu nestratíme ani pri náhlej zmene smeru jej pohybu.

- **Vyhodnotenie** – na základe údajov z detekcie sú vyrátané reálne súradnice detekovanej postavy v priestore v metroch. Za počiatok súradnicového systému je považovaná základňa robota. Vzdialenosť postavy od robota je získana z hĺbkovej mapy ako najnižšia hodnota z náhodného výberu bodov pokrývajúcich miesto obrazu, kde bola detekovaná. K vypočítaniu týchto súradníc je potreba poznať parametre kalibrácie kamery. To je v obrázku 5.1 znázornené témou cameraInfo.

Modul riadenia pohybu

Tento modul sa zaoberá vypočítaním rýchlosti a smeru pohybu robota tak, aby následoval detekovanú postavu v určitej vzdialenosti. Získané hodnoty sa následne upravia do vhodného intervalu a predajú sa ďalej uzlu, ktorý ma na starosti riadenie pohybu podvozku. Z bezpečnostných dôvodov, ako aj preto, aby sa na robota dal umiestniť náklad, sa robot nikdy nepohybuje smerom vzad.

Okrem základného riadenia smeru a rýchlosti sa tento modul venuje aj riešeniu neštandardných situácií, ako je napríklad stratenie sledovaného človeka napríklad v dôsledku zájdenia za roh. To je riešené pomocou jednoduchého algoritmu, ktorý riadi následujúci pohyb robota tak, aby sa robot dostal na posledné známe miesto sledovanej osoby. Po príchode naň robot vykoná pomalú otočku o 360° s cieľom nájsť stratenú osobu. Aby netrebalo na robota čakať, smer rotácie je zhodný s posledným smerom pohybu stratenej osoby. Ak robot detekuje postavu kedykoľvek počas tohto cyclu, vráti sa okamžite späť do bežnej činnosti sledovania. Týmto prístupom však vzniká problém, ak chce človek na robota niečo umiestniť. Ak sa priblíži dostatčne blízko, robot ho nezachytí celého a nerozpozná ho ako ľudskú postavu. Tým sa aktivuje algoritmus na znovunájdenie cieľa a robot sa priblíži k človeku a začne rotovať. K zrážke síce nedôjde, vďaka implementovanému bezpečnostnému opatreniu popísanému nižšie, avšak rotovanie robota výrazne znepríjemní umiestňovanie nákupu. Tento problem je vyriešený tak, že robot po stratení cieľa odmeria vzdialenosť objektov nachádzajúcich sa v časti obrazu, kde bola naposledy detekovaná sledovaná postava. Robot začne postavu hľadať len v prípade, ak je táto vzdialenosť nižšia ako posledná známa vzdialenosť k cieľu, inak ostane nečinne stáť. Tento spôsob je však ešte len vo fáze vývoja a nie je úplne spoľahlivý.



Obrázek 5.2: Správanie robota pri strate sledovaného cieľa.

Aby sa predišlo možnosti havárie a poškodeniu hardvéru, obsahuje tento modul aj jednoduchý bezpečnostný mechanizmus. Ten funguje na princípe zistenia najnižšej vzdialenosti objektov priamo pred robotom. Ak je príliš nízka, robot okamžite zastane. K tomuto testovaniu dochádza vždy pred posielaním vypočítanej rýchlosti, tzn. raz pre každý prijatý snímok z kamery.

Modul pre vizualizáciu a spracovanie parametrov

Posledná časť aplikácie nie je nevyhnutná pre jej beh, ale slúži len ako rozhranie pre nastavenie a testovanie. Plní dve hlavné funkcie, nastavenie riadiacich parametrov a vizualizáciu behu programu.

Užívateľ má možnosť nastaviť hodnotu niekoľkých základných parametrov. Z nich najdôležitejším je hodnota vzdialenosti, v ktorej sa robot drží od sledovaného cieľa. Ďalej je možné nastaviť ešte hodnoty časov, ktoré robot strávi pohybom a rotáciou v prípade straty následovanej postavy.

Čo sa týka vizualizácie, program podáva hlásenia o svojom stave najmä kontrolnými výpismi do konzoly. Okrem nich však ešte zobrazuje dáta získané z kamery, v ktorých je zvýraznený práve sledovaný objekt.

5.2 Implementácia

Výsledná aplikácia bola realizovaná vo forme balíčka pre meta-operačný systém ROS. Ten je nazvaný *person_following_bp* a obsahuje jediný uzol s názvom *detection_modul*.

Výsledný uzol je napísaný v jazyku C++ s využitím triedne orientovaného prístupu a obsahuje jedinu triedu rovnakého názvu (detectionModul) a funkciu main, v ktorej sa vytvorí jedna instancia tejto triedy, inicializuje sa ROS uzol, spracujú sa vstupné parametre a invokuje sa volanie funkcie `ros::spin()`. Tá slúži na sledovanie odoberaných tém a kedykoľvek sa na nich objaví nová správa, funkcia zavolá príslušnú časť programu určenú na jej obsluhu. To je opakované v nekonečnom cykle až do ukončenia existencie uzlu.

5.2.1 Trieda detectionModul

Táto trieda je hlavnou časťou môjho programu. Medzi jej atribúty patrí veľké množstvo premenných, niektoré z nich služia na úpravu obrazu, detekciu a sledovanie postavy alebo na vypočítanie jej pozície v realnom čase, či na ovládanie odoberania a publikovania správ na jednotlivé používané témy. Ostatné potom plnia úlohu riadiacich premenných. Výhodou pevne vytvorených premenných je možnosť vrátiť sa k poslednej použitej hodnote. Toho sa využíva napríklad pre zistenie poslednej pozície človeka pri jeho stratení. Nevýhodou tohto prístupu je nutnosť nastaviť predvolenú hodnotu týchto atribútov alebo inak zabezpečiť, aby nebolo možné k premenným pristúpiť skôr ako obsahujú nejakú hodnotu.

Následujúci popis implementácie jednotlivých častí návrhu a príslušných metód je rozdelený a zoradený na základe použitia pri štandardnom behu programu.

Inicializovanie instancie triedy

V konštruktore sa nastavia defaultné hodnoty pre existujúce atribúty, ktoré to vyžadujú, a vytvoria sa príslušné vzťahy s ostatnými používanými uzlami. Uzol sa prihlási ako zdroj správ na tému riadiacu pohyb robota *vel_msg* a ako príjemca správ prichádzajúcich na tému *cameraInfo*. K odberu obrazových dát z kamery dôjde až neskôr, po získaní informácií o kamere.

Načítanie parametrov z príkazovej riadky

Po inicializácií vytvorenej instancie triedy dôjde k načítaniu vstupných parametrov. Príslušná metóda je volaná funkciou main ešte pred vytvorením hlavného cyklu programu.

Parsovaniu jednotlivých parametrov sa venuje metóda `parameterParser`. Jej vstupom sú vstupné parametre programu prevzaté z funkcie main. Výstupom je v prípade parametru -h výpis nápovedy na obrazovku a nastavenie príslušných premenných v programe na základe zadaných hodnôt.

Program rozpoznáva následujúce hodnoty parametrov :

- **-h** - zobrazí nápovedu k ovládaniu programu
- **-pffd** - nastaví vzdialenosť, v akej robot následuje sledovanú postavu. Vyžaduje práve jeden argument, ktorý obsahuje požadovanú hodnotu zadanú v metroch.
- **-pftt** - nastaví čas, ktorý robot strávi rotáciou pri pokuse nájsť stratenú postavu. Vyžaduje práve jeden argument, ktorý obsahuje požadovanú hodnotu v sekundách.
- **-pfmt** - nastaví čas, ktorý robot strávi pohybom vpred pri pokuse nájsť stratenú postavu. Vyžaduje práve jeden argument, ktorý obsahuje požadovanú hodnotu v sekundách.

Za každým zadaným parametrom je nutné uviesť príslušnú hodnotu. Pokiaľ sa tomu tak nestane, príslušná premenná bude nastavená na prednastavenú hodnotu a do konzoly sa vypíše chybové hlásenie poukazujúce na chybne zadaný parameter. Akceptované sú len nezáporné hodnoty a smú obsahovať desatinnú čiarku.

Táto metóda využíva k svojej funkcií aj metódu `fakeStod`, ktorá slúži na prevádzanie číselnej hodnoty zo stringu do formátu double. Ide o vlastnú verziu štandardnej funkcie jazyka c++ `stod`, ktorá je navrhnutá tak, aby lepšie vyhovala mojím potrebám.

Prednastavené hodnoty premenných sú navrhnuté tak, aby robot došiel na poslednú známu pozíciu človeka, urobil práve jednú otočku a sledoval svoj cieľ v najmenšej vzdialenosti, pri ktorej je ešte možné vidieť celú postavu. Spúšťať program s parametrami pozmeňujúcimi niektorú z týchto hodnôt sa preto nedoporučuje. Program poskytuje túto možnosť hlavne z dôvodov testovania na iných typoch robotov.

Inicializácia parametrov kamery

Na začiatku behu hlavného cyklu programu je nutné získať informácie o kalibrácií a parametroch kamery. K tomu slúži metóda `cameraInfoInit`, ktorá je volaná po príchode novej správy na tému *cameraInfo*. K prihláseniu na odber správ z tejto témy došlo v konštruktore triedy.

Parametre kamery sú dôležité z dvoch dôvodov. Prvým z nich je zistenie rozlíšenia, potrebného najmä k vytvoreniu sledovacieho boxu na správnom mieste v obraze a pre správnu funkciu algoritmu vypočítavajúceho najnižšiu hodnotu hĺbky v danej časti snímku. Druhým dôvodom je potom použitie funkcie `projectPixelto3dRay` zo vstavanej knižnice ROS-a s názvom *pinhole_camera_model*. Tá slúži pre vytvorenie 3D bodu znázorňujúcom jediný pixel z obrazu zachyteného kamerou v reálnom priestore. K tomuto účelu vyžaduje okrem údajov o kamere ako sú napríklad rozlíšenie, zorné pole či ohnisková vzdialenosť aj hodnotu transformácie počiatku súradnicového systému kamery oproti základni robota. ROS využíva na popis tohto posunu tzv. transformy. Transform obsahuje údaje o zmene posunu a orientácie daného súradnicového systému oproti inému. ROS si jednotlivé transformy ukladá do stromovej štruktúry, ktorej základom je súradnicový systém základne robota.

Nakoľko nie je veľmi pravdepodobné, aby počas behu programu došlo k zmene parametrov kamery, uzol zruší svojú účasť na odoberaní správ z príslušnej témy po prvej prijatej

správe. Následne sa pripraví pre prijímanie obrazových dát z kamery a prihlási sa na tému *depth\image_raw*.

Po vykonaní tejto metódy sa skončí fáza počiatočnej inicializácie a program prejde do stavu detekcie a sledovania.

Predzpracovanie obrazu

Ako už bolo spomenuté, vo funkcií main sa po inicializácií programu donekonečná volá funkcia `ros::spin()`. Tá načúva na všetkých témach, na ktoré je program prihlásený, a po prijatí novej správy na ľubovoľné z nich zavolá príslušnú obslužnú rutinu. Po príchode nového snímku z kamery na tému *depth\image_raw* sa zavolá metóda s názvom callback, ktorá u nás predstavuje telo hlavného cyklu aplikácie. Jediným parametrom tejto metódy je práve získaný snímok z kamery.

Ten je vo formáte *sensor_msgs\Image*. Ide o vlastný formát systému ROS, ktorý je optimalizovaný pre prenos medzi jednotlivými uzlami, avšak jeho nevýhodou je, že ho nie je možné jednoducho spracovať pomocou nástrojov tretej strany. Preto je treba najprv každý získaný snímok pretransformovať do niektorého z bežných formátov. K tomu v ROS-ovi slúži balík *cv_bridge*. Jeho použitím sa získaný obraz z kamery pretransformuje do formátu CV_32FC1, ktorý už je možné v openCV používať.

CV_32FC1 je formát pre zobrazenie dát z hĺbkovej mapy, kde jednotlivé hodnoty pixelov nadobúdajú hodnoty ich vzdialenosti od roviny kamery v milimetroch. Metóda HOG, ktorú používame na detekciu postavy, je však v openCV zatiaľ implementovaná len pre použitie s formátmi CV_8UC1 (obraz v odtieňoch šedej) a CV_8UC4 (farebný obraz) a je nutné previesť ďalšiu konverziu. Nakoľko z hĺbkovej mapy nevieme zistiť farbu, a ani ju k detekcii nepotrebujeme, snímok je konvertovaný do formátu CV_8UC1.

Posledným krokom predzpracovania obrazu je jeho zmenšenie. Pokusmi som zistil, že aj keď je metóda HOG používaná pre detekciu v reálnom čase, stále je pomerne dosť časovo náročná. Pri originálnej veľkosti snímku trvala jedna iterácia hlavného cyklu programu približne jednu sekundu. Kinect zaznamenáva obraz rýchlosťou tridsať snímkov za sekundu, pričom robot bol nastavený tak, aby posielal len každý tretí. Z toho vyplýva, že obraz bolo nutné zmenšiť aspoň na jednú desatinu pôvodnej veľkosti. Najvhodnejším riešením preto bolo štyrikrát zmenšiť všetky jeho strany, čím sme dostali výsledný obrázok o veľkosti jednej šestnástiny originálneho snímku. Tým nastane určité skreslenie, ale jeho miera nie je postačujúca na to, aby ovplyvnilo výsledky detekcie.

Počas vývoja aplikácie bolo zvažované aj použitie Gaussoveho filtra na vyhladenie obrazu, avšak vo výslednej verzií nie je použitý, pretože funkcia knižnice openCV na detekciu metódou HOG používa svoj vlastný filter. Vo výsledku by teda jeho použitie nemalo žiaden efekt.

Obrázek 5.3: Výsledok predzpracovania obrazu.

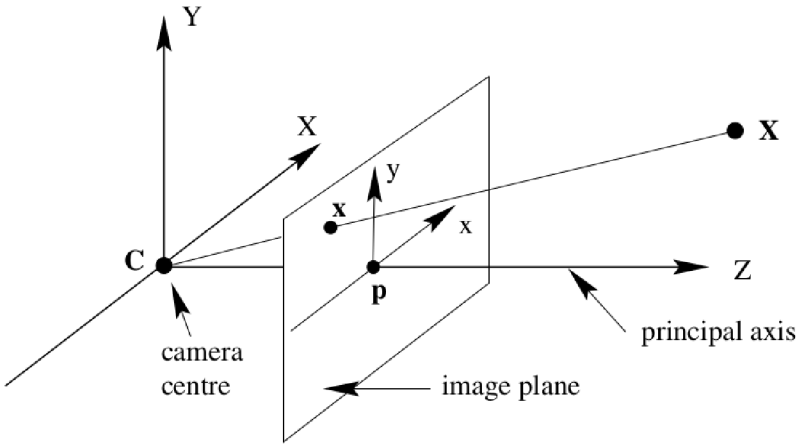
Detekcia postavy

K detekcií postavy sa používa funkcia knižnice openCV `hog.detectMultiScale`. Ide o aplikáciu detekčného algoritmu HOG s použitím trénovaného SVM. Multiscale znamená, že sa snímok postupne prehľadáva vyhľadavacími oknami stúpajúcich veľkostí. Tento prístup je časovo náročnejší, avšak jeho použitie je nevyhnutné, keďže nedokážeme spoľahlivo odhadnúť veľkosť sledovanej postavy. Do určitej miery je možné tento odhad vykonať na základe detekovanej vzdialenosti, čo by ale vyžadovalo náročnejšie predzpracovanie snímku vrátane použitia segmentácie.

Výsledkom detekcie je skupina štvorcov ohraničujúcich jednotlivé časti obrazu obsahujúce postavu. Tie sa ďalej kontrolujú až kým niektorý z nich nezahŕňa aj pixel na ktorom predpokladám výskyt sledovanej postavy. Ak takýto štvorec neexistuje, detekciu označím za neúspešnú, v opačnom prípde označím vyhovujúci štvorec za sledovaný cieľ a kontrolu okamžite ukončím (ak ich existuje viacero, vyberieme prvý). V prípade úspešnej detekcie sa vyráta nová predpokladaná pozícia postavy ako stred vyhovujúceho štvorca a ten sa zakreslí do obrázku slúžiaceho k vizualizácií.

Získanie reálnych súradníc

Na prevod pixelu zo snímku z kamery do 3D bodu v realnom priestore sa používa funkcia `projectPixelTo3dRay` z balíku ROS-a *image_geometry::PinholeCameraModel*. K správnemu výpočtu je potrebné poznať parametre kamery a pozíciu pixelu v obraze. Samotný výpočet prebieha podľa vzorcov získaných z modelu kamery znázorneného na obrázku 5.4, s tým rodzielom, že výsledný bod je ešte nutné pretransformovať z jedného súradnicového systému do druhého (zo ss kamery do ss základne robota). To sa dosiahne vykonaním príslušných transformácií. Všetky tieto prevody sú však aplikované v rámci volania funkcie `projectPixelTo3dRay`, a ja som ich nijako nemusel meniť alebo nastavovať. V mojom programe prevádzam vždy stredový pixel boxu obsahujúceho sledovanú postavu.



Obrázek 5.4: Pinhole camera model. Obrázok prevzatý z [2].

Aby sa dospelo k reálnym hodnotám v metroch, potrebujeme prenásobiť získané súradnice reálnou hĺbkou prevedeného pixelu. Keďže detekovaná postava sa nemusí vždy nachádzať na rovnakom mieste v jej štvorci, nie je dostačujúce odmerať hodnotu jedného pixelu. Miesto toho je zisťovaná hodnotu stredového pixelu a troch skupín štyroch bodov umiestnených približne na uhlopriečkach. Každý ďalší bod je od predošlého vzdialený o štvrtinu polovice šírky štvorca po horizontálnej osi a o pätinu polovice výšky štvorca v smere vertikálnom. Cieľom tejto nerovnosti je netestovať body, ktoré by sa reálne mohli nachádzať pod detekovanou postavou, keďže za výslednú hodnotu vyberám najmenšiu zistenú a podlaha pred postavou je bližšie ako ona samotná.

Obrázek 5.5: Približné umiestnenie bodov.

Výpočet rýchlosti a smeru pohybu

Kinect meria vzdialenosť bodov k zobrazovacej pláni, my však potrebujeme vedieť vzdialenosť bodu k základni robota. Tú je možné vypočítať jednoduchým použitím pytagorovej vety:

$$z = \sqrt[2]{z_k^2 + x^2} \tag{5.1}$$

kde z označuje vzdialenosť k robotovi, z_k vzdialenosť k zobrazovacej pláni a x je x-ová súradnica bodu.

Na základe hodnoty z určíme potrebnú rýchlosť robota. Tá od tejto nej závisí lineárne s maximom pri 10 m a minimom pri hodnotách nižších ako je nastavená sledovacia vzdialenosť. Výsledný vzorec pre riadenie rýchlosti potom vyzerá takto:

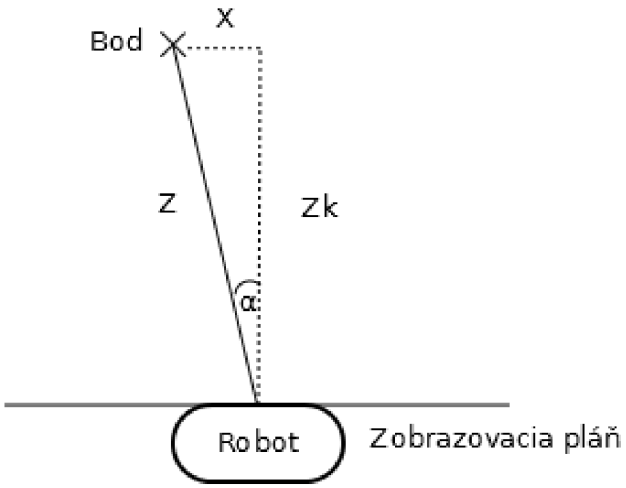
$$z = \frac{\sqrt[2]{z_k^2 + x^2} - fd}{6.25} \tag{5.2}$$

fd značí nastavenú hodnotu sledovacej vzdialenosti a hodnota 6.25 je získaná ako pomer požadovanej maximálnej vzdialenosti 10 a najvyššej povolenej hodnoty pre rýchlosť 1.6.

Uhlovú rýchlosť robota získame podobným spôsobom, tentokrát použitím základnej trigonometrie:

$$\alpha = \arctan \frac{x}{z_k} \tag{5.3}$$

Výsledné hodnoty rýchlostí je následne vhodné upraviť pre ďalšie spracovanie. Použitím vzťahu 5.2 nám môže výjsť záporna hodnota. Keďže robot je navrhnutý tak, aby sa nikdy nepohyboval smerom dozadu, je nutné všetky takéto hodnoty nahradiť nulou. Pri uhlovej rýchlosti je zase potrebné ošetriť možnosť nekonečnej aproximácie. Najvhodnejším spôsobom je vytvorenie istej tolerancie, kedy sa robot už nebude pokúšať otočiť za sledovaným cieľom. V mojom programe som si zvolil hodnotu približne $\pm 10°$.



Obrázek 5.6: Obrázok popisujúci vzťahy pre výpočet uhlu a vzdialenosti medzi robotom a bodom.

Hľadanie stratenej postavy

K výpočtu reálnych súradníc a rýchlostí dôjde len v prípade úspešnej detekcie, inak sa spustí časť programu určená na znovunájdenie stratenej postavy. Jej algoritmus pracuje v niekoľkých krokoch, z ktorých každý trvá inú dobu a jeho celé vykonanie trvá viacero prechodov hlavným cyklom. Na sledovanie stavu, v ktorom sa práve daný algoritmus nachádza,

slúži premenná `operation_tag`, ktorá je výpočtového typu *operation* a môže nadobúdať následujúcich hodnôt:

- **start_move** – táto hodnota sa nastaví vždy pri úspešnej detekcií a značí, že v prípade nepodarenej detekcie sa má robot začať pohybovať na posledné známe miesto sledovaného cieľa. Rýchlosť tohto pohybu je závislá od jeho poslednej známej vzdialenosti. Ešte pred tým sa však odmeria hĺbka v štvorci, v ktorom sa naposledy postava nachádzala a ak je menšia ako posledná zistená, algoritmus okamžite prejde do stavu none. To by malo vyriešiť prípad, keď chce užívateľ niečo vložiť do košíka a priblížil sa už dostatočne blízko nato, aby sa nevošiel celý do obrazu kamery, čím sa znemožní jeho úspešné rozpoznanie. V opačnom prípade sa nastaví príznak moving.

- **moving** – táto hodnota sa nastaví v kroku start_move a znamená, že robot má pokračovať v pohybe. Tento krok sa opakuje až kým neuplynie nastavená hodnota príslušného času. Vtedy sa do premennej `operation_tag` uloží hodnota start_turn.

- **start_turn** – táto hodnota sa nastaví po uplynutí času stráveného pohybom a robot sa v danej vetve pripravuje na rotáciu okolo svojej osi. Predpokladané umiestnenie postavy sa nastaví do stredu obrazovky a začne sa rotácia. Jej rýchlosť je konštantná a je volená dostatočne nízka, a jej smer závisí od poslednej známej x-ovej pozície sledovanej postavy. To urýchli proces jej znovunájdenia v prípade jednoduchého zájdenia za roh. Príznak sa nastaví na hodnotu turning.

- **turning** – táto hodnota sa nastaví v kroku start_turn a znamená, že robot sa má ďalej venovať rotácií okolo svojej osi. Po uplynutí prednastaveného času sa nastaví príznak none.

- **none** – tento príznak značí, že robot má nečinne čakať, kým sa postava znovu sama neobjaví. Je nastavený v prípade neúspešného znovunájdenia postavy alebo ak je pravdepodobné, že sa užívateľ pokúša položiť na robota nákup. Takisto je táto hodnota nastavená pri prvotnej inicializácií triedy.

Publikovanie rýchlosti na tému cmd_vel

Posledným krokom, ktorý je nutné vykonať nezávislé od výsledku detekcie, je publikovanie nastavených rýchlostí na tému *cmd_vel*. Ešte pred tým je však vhodné implementovať bezpečnostný mechanizmus, ktorého úlohou je zabrániť zrážke robota s ľubovoľným objektom. V mojom prípade ide o jednoduchú detekciu najnižšej hĺbky v obraze rovnakým princípom, akým je zisťovaná vzdialenosť detekovanej postavy, s tým rozdielom, že tentokrát body pokrývajú väčšinu plochy snímku z kamery. Ak je odmeraná vzdialenosť menej ako jeden meter, robot okamžite zastaví.

Kapitola 6

Záver

Výsledkom tejto práce jeť program riadiaci robota tak, aby plnil úlohu samohybného inteligentného nákupného košíka. Tá spočíva v rozumnom následovaní svojho majiteľa po obchodnom centre. Keďže obchod nie je tvorený otvorenou plochou, ale veľkým množstvom nepriehľadných regálov, bolo dôležité navrhnúť riadenie pohybu tak, aby robot bol schopný sledovať užívateľa aj počas chíľkovej straty vizuálneho kontaktu, spôsobenej jeho zájdením za roh. Výsledná aplikácia nájde uplatnenie aj v iných odvetviach, prakticky všade tam, kde je potreba prenášať nejaké predmety. Napríklad je ju možné využiť ako robotický detský kočík alebo na nosenie stavebného materiálu.

K úspešnému vytvoreniu aplikácie bolo nutné najprv naštudovať širokú škálu používaných nástrojov a postupov. Šlo najmä o získanie poznatkov potrebných k práci so systémom ROS a k výberu vhodného detekčného algoritmu, ktorý by bol voľne dostupný, no zároveň aj dostatočne presný a rýchly. Nemenej dôležité boli nadobudnuté vedomosti o spôsobe, akým Kinect zaznamenáva hĺbku objektov.

Prístup k detekcií pomocou Kinectu a knižnice openCV sa osvedčil ako celkom spoľahlivý, problém nastáva len v prípade príliš osvetlenej scény, kedy má kamera problém určiť hĺbku objektov.

Možné rozšírenia by sa dali rozdeliť do dvoch skupín. Prvá skupina sú rozšírenia vylepšujúce rýchlosť a presnosť detekcie, medzi ktoré by sa dalo zaradiť použitie segmentácie. Tá slúži na separáciu jednotlivých objektov v obrázku, ktoré je potom možné ďalej roztriediť na základe ľahko dostupných parametrov ako je výška v spojení s hĺbkou a tým určiť, či to môže byť ľudská postava, ešte pred aplikovaním samotného detekčného algoritmu. Týmto spôsobom nie je nutné používať metódu HOG na celý obraz, ale len na pár jeho predom vybraných častí a aj to len vo forme jedinej veľkosti detekčného okna. To by malo za následok mnohonásobné zrýchlenie detekcie. Druhá skupina je tvorená vylepšeniami pridávajucími nejakú novú funkcionalitu. Napríklad pridanie metódy určenej na obídenie prekážky stojacej robotovi v ceste alebo implementácia lepšieho sledovacieho algoritmu, ktorý by dokázala sledovaný objekt rozpoznať aj v hustom dave ľudí. To by mohlo byť dosiahnuté pridaním ďalších sledovacích kritérií ako je napríklad farba alebo materiál odevu, poprípade výška postavy. Ďalším rozširením by mohlo byť načítanie parametrov z konfiguračného súboru. Tým by bolo možné predom navrhnúť ideálne hodnoty premenných pre rôzne použité typy robotov.

Práca na projekte bola zaujímavá a vedel by som si predstaviť, že by som sa niečomu podobnému venoval aj v budúcnosti.

Literatura

[1] Accelerating demand for industrial robots will continue. 2014-9-30 [cit. 2015-5-4].
URL http://www.ifr.org/news/ifr-press-release/accelerating-demand-for-industrial-robots-will-continue-658/

[2] A C++ code to compute OpenGL 4x4 GL_MODELVIEW_MATRIX from 2D3D points homography. [cit. 2015-5-4].
URL https://braintrekking.wordpress.com/2013/06/02/a-c-code-to-compute-opengl-4x4-gl_modelview_matrix-from-2d-3d-points-homography/

[3] Pioneer 3 Operations Manual. [cit. 2015-5-4].
URL http://www.ist.tugraz.at/_attach/Publish/Kmr06/pioneer-robot.pdf

[4] RoboLab. [cit. 2015-5-4].
URL http://merlin.fit.vutbr.cz/wiki/index.php/RoboLab

[5] RoboLab Toad. [cit. 2015-5-4].
URL http://merlin.fit.vutbr.cz/wiki/index.php/RoboLab_Toad

[6] ROS Wiki: Openni Tracker. [cit. 2015-5-4].
URL http://wiki.ros.org/openni_tracker

[7] ROS Wiki: Skeleton Tracker Teleoperation Package for Mobile Robot. [cit. 2015-5-4].
URL http://wiki.ros.org/openni/Contests/ROS3D/SkeletonTrackerTeleoperationPackageforMobileRobot

[8] ROS Wiki: Turtlebot Follower. [cit. 2015-5-4].
URL http://wiki.ros.org/turtlebot_follower

[9] ROSWiki - Concepts. [cit. 2015-5-4].
URL http://wiki.ros.org/ROS/Concepts/

[10] Skeletal Tracking. [cit. 2015-5-4].
URL https://msdn.microsoft.com/en-us/library/hh973074.aspx

[11] Stereoscopic Parallax. [cit. 2015-5-4].
URL http://www.3d-forums.com/threads/stereoscopic-parallax.4/

[12] Wikipedia: Pedestrian Detection. [cit. 2015-5-4].
URL http://en.wikipedia.org/wiki/Pedestrian_detection

[13] Wikipedia: RANSAC-Alogrithmus. [cit. 2015-5-4].
URL http://de.wikipedia.org/wiki/RANSAC-Algorithmus

[14] Wikipedia: Support vector machine. [cit. 2015-5-4].
URL http://en.wikipedia.org/wiki/Support_vector_machine

[15] M.R. Andersen, T. Jensen, P. Lisouski, A.K. Mortensen, M.K. Hansen, T. Gregersen, P. Ahrendt : Kinect Depth Sensor Evaluation for Computer Vision Applications. 2012 [cit. 2015-5-4].
URL http://eng.au.dk/fileadmin/DJF/ENG/PDF-filer/Tekniske_rapporter/Technical_Report_ECE-TR-6-samlet.pdf

[16] de Campos, F. M. S.: Detection and tracking faces in videos - How to detect faces in videos ? 2011-2-21 [cit. 2015-5-4].
URL
http://www.bitabit.eng.br/2011/02/21/como-detectar-faces-em-videos/

[17] Dana H. Ballard, C. M. B.: *Computer Vision*. Prentice-Hall Inc., New Jersey, 1982 [cit. 2015-5-4], ISBN Â0-13-165316-4.

[18] Davis, E. R.: *Computer Vision*. Elsevier Inc., třetí vydání, 2005 [cit. 2015-5-4], ISBN 978-0-12-206093-9.

[19] Edward Rosten, T. D.: Machine learning for high-speed corner detection. [cit. 2015-5-4].
URL http://www.edwardrosten.com/work/rosten_2006_machine.pdf

[20] Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool: SURF: Speeded Up Robust Features, Computer Vision and Image Understanding. *Computer Vision and Image Understanding*, ročník 110, č. 3, 2008 [cit. 2015-5-4]: s. 346–359.

[21] Kourosh Khoshelham, S. O. E.: Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications. [cit. 2015-5-4].
URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3304120/

[22] Lowe, D. G.: Distinctive Image Features from Scale-Invariant Keypoints. 2004-1-5 [cit. 2015-5-4].
URL http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

[23] Manaf A. Mahammed, F. A. K., Amera I. Melhum: Object Distance Measurement by Stereo VISION. *International Journal of Science and Applied Information Technology (IJSAIT)*, ročník 2, č. 2, 2013-3-12 [cit. 2015-5-4]: s. 5–8, ISSN 2278-3038.

[24] Martin A. Fischler, R. C. B.: Random Sample Consenus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, ročník 24, č. 6, 1981-6-1 [cit. 2015-5-4]: s. 381–395.

[25] McCormick, C.: HOG Person Detector Tutorial. 2013-5-9 [cit. 2015-5-4].
URL https://chrisjmccormick.wordpress.com/2013/05/09/hog-person-detector-tutorial/

[26] Michael Brading, Kenneth Salsman, Manjunath Somayaji, Tim Droz, DaniĂŤl Van Nieuwenhove, Pedro Gelabert, Brian Dipert: Using 3D sensors to bring depth discernment to embedded vision apps. [cit. 2015-5-4].
URL http://www.embedded.com/design/real-world-applications/4411991/1/Using-3D-sensors-to-bring-depth-discernment-to-embedded-vision-apps

[27] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, Andrew Ng: ROS: an open-source Robot Operating System. [cit. 2015-5-4].
URL http://www.robotics.stanford.edu/~ang/papers/icraoss09-ROS.pdf/

[28] Peng Wang, B. Z., Jianmin Li: A Real World Detection System. [cit. 2015-5-4].
URL http://people.csail.mit.edu/wangpeng/visapp12.pdf/

[29] Sanford, K.: Smoothing Kinect Depth Frames in Real-Time. 2012-1-24 [cit. 2015-5-4].
URL http://www.codeproject.com/Articles/317974/KinectDepthSmoothing

[30] Sergei L. Pyshkin, J. M. B. (editor): *Optoelectronics – Advanced Materials and Devices*. InTech, první vydání, 2013-1-16 [cit. 2015-5-4], ISBN 978-953-51-0922-8.

[31] Vinkler, M.: *Využití pohybového snímače Kinect ve virtuální realitě*. Dizertační práce, Fakulta informatiky, Masarykova Univerzita, Brno, 2012.

Příloha A

Obsah CD

- **/node** – balík systému ROS obsahujúci vytvorenú aplikáciu
- **/manual** – návod na inštaláciu a používanie aplikácie
- **/tz** – táto práca vo formáte pdf
- **/tz/tex** – zdrojové súbory tejto práce v Latex-u
- **/plagat** – plagát prezentujúci vytvorenú aplikáciu