UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

MOBILNÝ INTELIGENTNÝ ROBOT S RAMENOM A STEREOVIDENÍM

Diplomová práca

2015 **Bc. Marek Tučáni**

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

MOBILNÝ INTELIGENTNÝ ROBOT S RAMENOM A STEREOVIDENÍM

Diplomová práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	Aplikovaná informatika 2511
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Mgr. Pavel Petrovič, PhD.
Kód:	95f184bc-4cc9-45ea-aac8-81e33606b4b1

Bratislava 2015 **Bc. Marek Tučáni**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Marek Tučáni
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.9. aplikovaná informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský

Názov: Mobilný inteligentný robot s ramenom a stereovidením / *Mobile Intelligent Robot with Arm and Stereovision*

Cieľ: Práca nadväzuje na bakalársku prácu Petra Pukančíka [1], v ktorej bol zostrojený mobilný robot s ramenom s 5 stupňami voľnosti. Cieľom je pozdvihnúť túto platformu na platformu pripravenú na experimenty v umelej inteligencii a kognitívnej vede, rozšírením o systém stereovidenia. Študent preskúma algoritmy stereovidenia, zvolí a implementuje vhodný algoritmus. Súčasťou práce je vytvorenie a zdokumentovanie frameworku na riadenie robota a predvedenie funkčnosti platformy na konkrétnej aplikácii.

Literatúra:
[1] Peter Pukančík: Riadici systém s inverznou kinematikou pre mobilné robotické rameno, bakalárska práca, FMFI UK, 2012.
[2] E. R. Davies: Machine Vision: Theory, Algorithms, Practicalities, Elsevier, 2004.
[3] E. B. Corrochano: Geometric Computing for Perception Action Systems, Springer, 2001.

Kľúčové slová: mobilná robotika, stereovidenie, robotické rameno, inverzná kinematika

Vedúci: Mgr. Pavel Petrovič, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. PhDr. Ján Rybár, PhD.

Dátum zadania: 12.11.2013

Dátum schválenia: 14.11.2013

prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestné vyhlásenie

Vyhlasujem, že som záverečnú prácu vypracoval samostatne s použitím uvedenej literatúry.

Bratislava 2015 .....

Marek Tučáni

Poďakovanie

Ďakujem vedúcemu diplomovej práce Mgr. Pavlovi Petrovičovi, PhD., za jeho čas, odborné vedenie, podporu, poskytnuté konzultácie a dôležitú spätnú väzbu. Moje poďakovanie zároveň patrí všetkým, ktorí mi pomáhali s dostupnosťou či už knižných, alebo internetových zdrojov, ale aj tým, ktorí ma v mojom štúdiu a vytrvalej práci podporovali. Prácu bolo možné realizovať vďaka finančnému daru Humboldtovej nadácie pre prof. Ing. Igora Farkaša, Dr., z ktorého boli zakúpené mnohé súčiastky na konštrukciu robota.

Abstrakt

Modulárny autonómny robot je zostrojený z komponentov podľa samostatného návrhu. Modul stereovidenia zabezpečujú dve farebné kamery umiestnené na prednom paneli a prepojené s riadiacimi doskami SRV-1 od Surveyor Corporation. Obraz je prenášaný prostredníctvom bezdrôtového WiFi spojenia do počítačového modulu Gumstix, na ktorom beží distribúcia operačného systému Linux (Yocto Project). Pohyby ramena sú počítané inverznou kinematikou, pričom berieme do úvahy pracovný a konfiguračný priestor robotického ramena s 5 stupňami voľnosti. Vlastným prínosom práce je vytvorenie algoritmu navigácie robotického ramena na výpočet trajektórie pohybu: v prípade konfliktov v pracovnom priestore sa navrhuje alternatívna trajektória. Robot sa dokáže pohybovať v prostredí, vyhľadávať a pomocou robotického ramena Lynxmotion AL5D uchopovať predmety podľa určených kritérií. Práca rieši problém stereovidenia pomocou vybraných algoritmov z knižníc OpenCV, pričom porovnáva ich vlastnosti a kvalitu výstupu. Vďaka stereovideniu dokáže robot identifikovať polohu predmetov v jeho zornom poli. Uchopené predmety dokáže prevážať a doručiť na cieľové miesto. Na niekoľkých príkladoch aplikácií demonštrujeme použitie a vlastnosti vyvinutej platformy: vyhľadávanie a zbieranie predmetov stanovenej farby v prostredí, vyhľadávanie a ukladanie predmetov jeden na druhý a riešenie logickej manipulačnej úlohy s využitím všeobecného algoritmu plánovania postupnosti akcií. Výstupom práce je robotický systém pripravený na experimenty v kognitívnej vede a umelej inteligencii.

Kľúčové slová: mobilná robotika, stereovidenie, robotické rameno, inverzná kinematika

Abstract

Modular autonomous robot is constructed based on original design from various components. Two color cameras located on the front panel and connected to the control board SRV-1 produced by Surveyor Corporation provide the stereovision module. The image is transmitted through a wireless connection to the computer module Gumstix that runs a distribution of Linux operating system (Yocto Project). Robotic arm movements are calculated using inverse kinematics, while taking into account workspace and configuration space of a robotic arm with 5 degrees of freedom. The original contribution of this work consists of a navigation algorithm of robotic arm that calculates the trajectory for robot arm movement: in case of a conflict in the working area an alternative trajectory is constructed. The robot is able to move in the environment, search the environment and grip objects using the robotic arm Lynxmotion AL5D according to specific criteria. The work solves the problem of stereovision using selected algorithms from the OpenCV libraries. We compare their properties and the quality of the output. Thanks to stereovision robot is able to identify the location of objects in its field of vision. Grasped objects can be transported and delivered to a specific destination. In several example applications, we show the use and the properties of the robotic platform that has been developed: locating and collecting objects of a specified color in the environment, locating, picking and stacking objects on top of each other and solving a logical manipulation task using a general algorithm for planning action sequences. The output of this work is a robotic system that is ready for experiments in Cognitive Science and Artificial Intelligence.

Keywords: mobile robotics, stereovision, robotic arm, inverse kinematics

OBSAH

ÚVOD

Robotickú tému diplomovej práce som si zvolil z viacerých dôvodov. Hlavným dôvodom je moje nadšenie aplikovať informatiku v praxi. Robotickú tému považujem za ideálnu voľbu, nakoľko v sebe spája hardwarovú aj softvérovú časť. Vždy ma fascinovali mechanické zázraky modernej doby z pohľadu ich dômyselného fungovania. Aj napriek tomu, že som doteraz nemal takmer žiadne skúsenosti z oblasti robotiky, téma bola pre mňa mimoriadne príťažlivá. Predovšetkým ma zaujalo úzke spojenie troch veľmi zaujímavých oblastí – informatiky, elektroniky a mechaniky.

Stereovidenie je dnes jednou z najrýchlejšie sa rozvíjajúcich oblastí informatiky, nakoľko jeho využitie je široké. Princíp stereovidenia využil v robotike už v roku 1983 H. P. Movarec, no až v roku 2010 svet zaznamenal najväčší rozmach tejto technológie. Microsoft v novembri 2010 spustil predaj Kinectu, ktorý spôsobil technologickú revolúciu - čiastočnú v hernom priemysle, ale totálnu v robotike. Cenovo dostupný 3D snímač (senzor) totiž v robotickej komunite dlho chýbal. Kinect funguje na princípe ožarovania scény infrasvetlom, čo však spôsobuje, že vo vonkajšom prostredí na slnečnom svetle funguje iba s ťažkosťami. Stereovidenie je ale možné dosiahnuť aj pomocou bežných farebných kamier. To prináša isté výhody - má biologickú plausibilitu, takže môže byť vhodné na štúdium biologicky inšpirovaných systémov a je aj v princípe energeticky úspornejšie, keďže nevyžaduje osvetľovanie scény obrazom z projektora.

Roboty majú dnes široké využitie. Väčšina z nich je fixných a vykonávajú opakované operácie na výrobnej linke, kde majú omnoho jednoduchšiu úlohu, keďže sú v kontrolovanom a deterministickom prostredí, čo znamená, že sú často bez akýchkoľvek senzorov alebo spätnej väzby. Naopak, nasadenie robotov v záchranných, prieskumných, transportných či údržbových úlohách si vyžaduje manipuláciu v neznámom, nedeterministickom prostredí, a tiež využitie senzorov, počítačového videnia a inteligentných algoritmov. Tento aspekt problematiky stereovidenia približuje aj táto diplomová práca.

Hlavnou časťou diplomovej práce je robotické vnímanie prostredia pomocou sústavy dvoch kamier, pričom na základe týchto vnemov dokáže robot autonómne vykonávať akcie, na ktoré bol naprogramovaný. Akcie vykonáva za pomoci efektorov, pričom v našom prípade

sa jedná o robotické rameno. Pohyb ramena uskutočňuje robot pomocou inverznej kinematiky. Na pohyb v priestore slúžia štyri kolesá poháňané elektromotormi.

Diplomová práca je rozdelená na niekoľko častí. Prvá kapitola popisuje teoretické poznatky z oblasti robotiky a stereovidenia. Taktiež sa venuje inverznej kinematike, čím nadväzuje na bakalársku prácu študenta Petra Pukančíka (Riadiaci systém pre mobilné robotické rameno [7]). Druhá kapitola sa venuje implementačným možnostiam stereovidenia pomocou knižníc OpenCV, pričom bližšie popisuje algoritmy zakomponované v knižniciach. Tretia kapitola detailne popisuje mobilnú robotickú platformu, špecifikáciu jej komponentov, ich prepojenia a komunikačné protokoly. Uvádza všetky hardwarové modifikácie, ktoré bolo potrebné vykonať, a zároveň je praktickým príkladom využitia poznatkov popisovaných v teoretickej časti diplomovej práce. Následne sa práca venuje návrhu a implementácii samotného riadiaceho systému robota, pričom kladie dôraz na modulárnosť systému. Posledná kapitola popisuje výsledky nasadenia a testovania systému v praxi.

CIELE PRÁCE

Medzi hlavné ciele práce patrí:

- Porovnanie dostupných algoritmov stereovidenia. Demonštrovanie použitia vybraných algoritmov na reálnej robotickej platforme. Vyhodnotenie dosiahnutých výsledkov.

- Naprogramovanie pohybu existujúceho robotického ramena na základe vnemov z kamier. Definovanie pracovného priestoru ramena - pohyb ramena medzi jednotlivými bodmi v priestore bez kolízií s telom robota.

- Hardwarové zdokonalenie existujúcej robotickej platformy vytvorenej a popísanej v bakalárskej práci študenta Petra Pukančíka [7]. Obohatenie platformy o systém stereokamier a linuxovú platformu, na ktorej bude bežať navrhnutý a implementovaný robotický systém.

- Zostavenie a testovanie kompletného robotického systému, ktorý je schopný autonómneho pohybu a konania, využívajúc všetky hardwarové prostriedky robotickej platformy. Demonštrovanie jeho činnosti na modelových príkladoch.

- Vytvorenie knižnice pre jednoduchú prácu s robotickou platformou ako celkom.

1. VÝCHODISKÁ

Prvé počiatky mechanických výtvorov, ktoré označujeme pojmom robot, obsahovali iba jednoduché receptory na vnímanie prostredia, v ktorom sa nachádzali. Doba natoľko pokročila, že bolo potrebné, aby roboti čo najpresnejšie rozpoznávali objekty, ich farbu, tvar, rozmery, polohu a iné vlastnosti. Pre tento účel sa robotom pridáva „univerzálny“ senzor – kamera. Keďže kamera sama o sebe poskytuje iba dvojrozmernú informáciu o prostredí pred robotom, čoraz viac sa využívajú sústavy kamier (najčastejšie dvoch kamier). Princíp skladania obrazu z dvoch kamier, s cieľom získať trojrozmerné vnímanie prostredia, nazývame stereovidením.

1.1 Robot

Pojem robot sa dnes môže javiť ako bežne používaný výraz, no nie je tomu tak dávno, čo toto slovo neexistovalo. Ako autor pojmu robot sa najčastejšie označuje český spisovateľ Karel Čapek, ktorý ho použil vo svojom diele R.U.R. : Rossum's Universal Robots, ktoré bolo vydané v roku 1920. Dielo hovorí o možnom zneužití techniky a pojmom robot nazýva „umelých ľudí“ vyrábaných v továrňach. Týchto by sme v dnešnej dobe nazvali výstižnejším výrazom „android“.

Dnešné roboty možno považovať za kybernetické systémy, ktoré sa skladajú z troch základných subsystémov:

- senzorický subsystém
- riadiaci a rozhodovací subsystém
- motorický subsystém

Senzorický subsystém má za úlohu zbierať informácie z rôznych senzorov (kamera, tlakový senzor, sonar a pod.). Úlohou riadiaceho a rozhodovacieho subsystému je spracovávanie podnetov z prostredia, vyvodzovanie záverov, rozhodovanie a vysielanie inštrukcií do motorického subsystému. Tento je zodpovedný za pohyb robota v priestore alebo modifikáciu priestoru okolo neho.

1.2 Interakcia s prostredím

Pod interakciou robota s prostredím rozumieme akúkoľvek komunikáciu medzi robotom a prostredím. Rozlišujeme dva základné druhy komunikácie – získavanie informácií pomocou receptorov a vykonávanie akcií pomocou efektorov.

1.2.1 Receptory

Svetelný senzor je používaný na detekciu intenzity svetla, pričom je na túto funkciu vybavený najčastejšie fotorezistorom, prípadne fototranzistorom alebo fotodiódou. Fotorezistor je typ rezistora, ktorého odpor sa mení s meniacou sa intenzitou svetla. Väčšia intenzita svetla spôsobuje zníženie odporu rezistora. Fototranzistor je druh polovodičového fotodetektora, ktorý využíva rovnako ako fotodióda vnútorný fotoelektrický jav, vďaka ktorému sa generujú nosiče náboja pri osvetlení. Podstatná je generácia nosičov náboja v báze tranzistora, vďaka čomu nastáva zosilňovací jav rovnako ako u bežného použitia tranzistora. Citlivosť fototranzistora je pri rovnakej ploche väčšia ako u fotodiódy, avšak iné parametre (najmä rýchlosť, ale aj šumové parametre) sú horšie.

Zvukovými senzormi sú spravidla mikrofóny, ktoré detekujú zvuk a vracajú napätie úmerné hladine akustického tlaku. Jednoduchý robot môže byť navrhnutý tak, aby sa pohyboval podľa druhu zvuku ktorý zachytáva. Napríklad jedno tlesknutie môže znamenať otočenie doľava a dve tlesknutia by mohli znamenať otočenie doprava. Komplexnejší roboti môžu využívať zvukový senzor na rozpoznávanie reči a hlasu.

Ďalšími často používanými senzormi sú **senzory na detekciu objektov a prekážok** v priestore. Pomocou nich dokáže robot rozpoznávať vzdialenosť predmetov, ako aj ich tvar a veľkosť. Fungujú zvyčajne na niekoľkých základných princípoch:

- Infračervené senzory – fungujú na princípe odrazu infračerveného lúča. Obsahuje dve časti: emitor a detektor. Emitorom je IR LED dióda vyžarujúca svetlo s vlnovou dĺžkou napríklad 950N. Fototranzistor citlivý v tomto svetelnom spektre zastáva úlohu detektora. Samotná detekcia prekážok senzorom funguje nasledovne. Ak je pred senzorom nejaký predmet, odrazí sa vyslaný lúč späť do senzoru. Detektor (fototranzistor) dokáže nielen určiť, či sa lúč odrazil, ale

dokonca aj v akej intenzite. Čím viac svetla sa odrazí späť, tým väčší preteká prúd fototranzistorom, resp. tým menší je jeho odpor. Niektoré senzory využívajú modulovaný signál – kmitanie s nejakou frekvenciou, napr. 40KHz, vďaka čomu sú odolné voči meniacim sa svetelným podmienkam, pri ktorých by vyššie uvedený koncept nefungoval správne.

- Ultrazvukové senzory – vysielač vysiela ultrazvukový impulz a je zachytený prijímačom. Keďže rýchlosť zvuku vo vzduchu je takmer konštantná (približne 344 m/s), zodpovedá rozdiel medzi časom vyslania signálu a časom prijatia odrazeného signálu priamo úmerne vzdialenosti medzi robotom a prekážkou.
- Laserové senzory – laserové svetlo sa vysiela a odrazené svetlo je zachytené a analyzované. Vzdialenosť sa meria na základe výpočtu rýchlosti svetla, doby potrebnej pre odrazenie svetla a jeho zachytenie na prijímači. Tieto senzory sa používajú predovšetkým na meranie dlhších vzdialeností.
- Stereokamery – dve kamery umiestnené vedľa seba môžu poskytovať informáciu o hĺbke priestoru pred nimi vďaka stereovideniu. Spracovávanie dát zachytených kamerami je pomerne náročný proces pre robota s malou výpočtovou silou a pamäťou. Systém stereokamier je pre robota veľkým prínosom, nakoľko pomocou neho dokáže rozpoznávať tvar, veľkosť, ako aj vzdialenosť objektov.
- Hĺbkové senzory – do snímaného priestoru je premietaný špecifický vzor (napríklad mriežka), ktorý je snímaný pomocou kamier. Senzory analyzujú zobrazenie vzoru na predmetoch v priestore, za pomoci ktorého sú počítané vzdialenosti k nim.

Akcelerometer zaznamenáva gravitačné zrýchlenie, ktorého meraním dokážeme určiť, ako veľmi je robot naklonený. Toto meranie je užitočné v prípade, že robot musí neustále udržiavať rovnováhu alebo ak robot potrebuje informáciu o tom, či sa pohybuje na šikmej ploche alebo na rovine. Senzor tiež meria silu zrýchlenia, za pomoci čoho vieme určiť smer alebo aj rýchlosť pohybu.

Gyroskop je zariadenie na meranie zrýchlenia otáčavého pohybu. Prístroj môže pracovať na rôznych fyzikálnych princípoch, napríklad jeden typ využíva zákon zachovania momentu hybnosti, ktorý funguje podobne ako zotrvačník. Disponuje, na rozdiel od klasického zotrvačníka, ešte aj dvoma prstencami (jeden je vertikálny a druhý horizontálny).

Zotrvačník, ktorý je súčasťou gyroskopu, sa nazýva rotor. Využíva jav známy aj ako gyroskopická akcelerácia, ktorá zaručuje jeho stabilitu v akejkoľvek polohe. Vďaka tomuto javu môžeme pomocou zmeny polohy zotrvačníka vzhľadom k telu gyroskopu kalkulovať zmenu polohy celého zariadenia v troch osách. Takýto gyroskop označujeme tiež pojmom 3AXIS.

Hmatové **tlakové senzory** sú veľmi užitočné v robotike, najmä ak robot disponuje robotickou rukou. V efektore robotickej ruky sú zvyčajne umiestnené tlakové senzory, vďaka ktorým dokážeme detekovať uchopenie predmetu, prípadne aj silu uchopenia.

1.2.2 Efektory

Okrem pneumatických, hydraulických a lineárnych posuvných členov je vykonávanie akcií realizované najčastejšie pomocou elektromotorov, ktorých sa v robotike používa niekoľko základných druhov.

Krokový motor je typ viacpólového synchrónneho motora. Využíva sa najmä tam, kde je potrebné presné regulovanie otáčok, a tiež konkrétna poloha rotora. Princíp fungovania spočíva v tom, že prúd prechádzajúci cievkou statora vytvorí magnetické pole, ktoré pritiahne opačný pól magnetu rotora. Výhodou motora je, že sa dokáže zafixovať v jednej polohe. Vhodnou kombináciou zapojenia cievok vznikne rotujúce krokové magnetické pole, ktoré nielen otáča rotorom, ale zabezpečuje aj presnú polohu rotora voči statoru. Kvôli prechodovým magnetickým javom je rýchlosť otáčania motora pomerne limitovaná. Po prekročení istej hranice otáčok začne motor strácať schopnosť presného krokovania. Nevýhodou tohto typu motora je spomínaná nízka rýchlosť otáčania ale tiež vysoká spotreba elektrickej energie, nakoľko motor na udržiavanie presnej polohy spotrebúva veľké množstvo energie. Kvôli týmto nevýhodám nachádza v mobilnej robotike len malé uplatnenie.

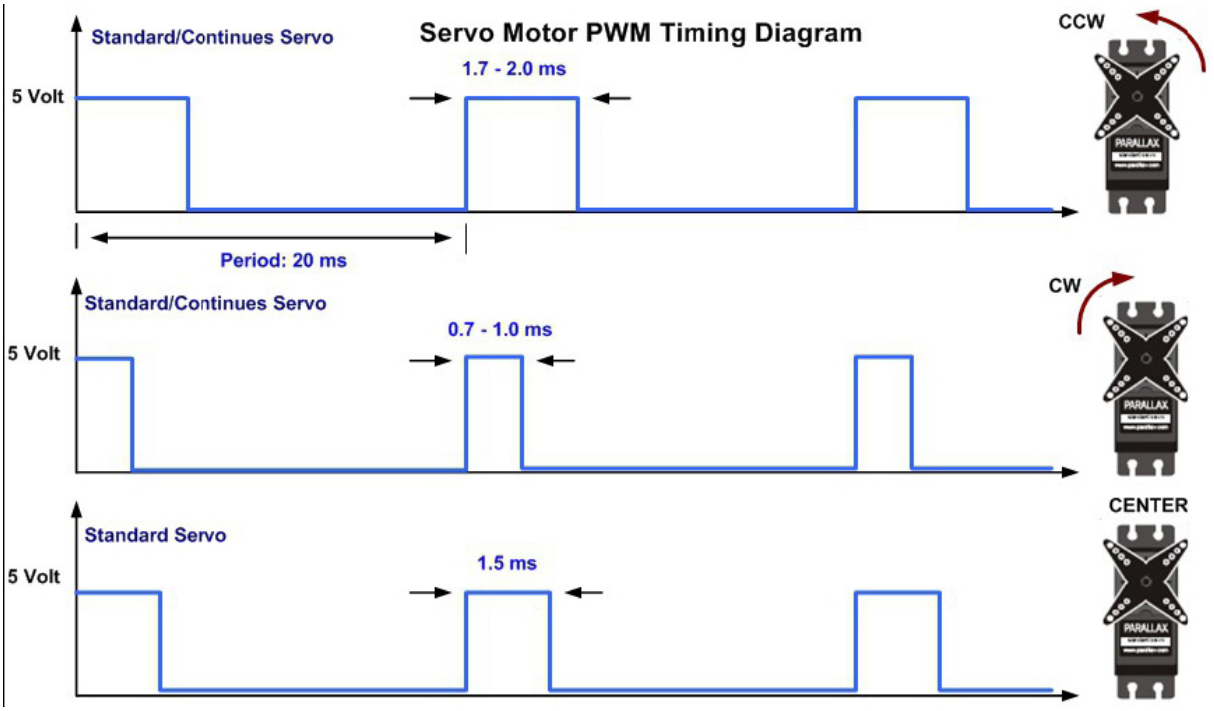
Elektrický **DC motor** na jednosmerný prúd (direct current) je zariadenie meniace jednosmerný elektrický prúd na mechanickú silu. DC motor sa využíva všade tam, kde je potrebný široký rozsah regulácie rýchlosti, a tiež tam kde, je odpor pri rozbiehaní motora väčší. Takýto typ motora je vhodný napríklad na poháňanie náprav mobilných robotov.

Ďalším typom motora využívaným v robotike je **servomotor**. Pod pojmom servomotor sa označuje servomechanizmus – zariadenie, ktoré na základe rozdielu spätnej väzby zo senzorov a referenčného vstupu upravuje výkon svojho mechanizmu. V robotike sa využívajú modelárske RC jednosmerné servomotory, ktoré sa väčšinou skladajú z nasledovných častí:

- elektrický DC motorček – samotný mechanizmus servomotora
- prevodové kolieska – prenášajú krútiaci moment z motorčeka
- potenciometer – zariadenie, ktoré slúži ako senzor aktuálnej polohy mechanizmu a umožní spätnú väzbu
- kontrolný obvod – elektronický obvod, ktorý vyhodnocuje rozdiel medzi referenčnou hodnotou (vstupom) a medzi aktuálnym nastavením mechanizmu, pričom posiela túto informáciu motorčeku

Jednou z techník riadenia motorov je impulzová šírková modulácia (**PWM - Pulse Width Modulation**). Impulzová šírková modulácia predstavuje techniku posielania informácie zakódovanej do šírky signálu v určenom rozsahu. Využíva sa či už na komunikáciu, prevod analógovej informácie do digitálnej, alebo na moduláciu elektrického výkonu dodávaného do zariadenia. Princíp fungovania spočíva v zmene dĺžky signálu posielaného v rozsahu danej periódy. Riadenie motorov pomocou PWM sa realizuje dvomi rôznymi spôsobmi:

1. riadenie servomotora - poloha servomotora sa odvíja od dĺžky signálu (šírky posielaného pulzu)
2. riadenie DC motora - rýchlosť motora závisí od dĺžky aktívneho stavu (označovaného ako „duty cycle“) voči dĺžke periódy a vyjadruje sa v percentách. 100% znamená, že celú periódu je signál „zapnutý“ a 0% znamená, že celú periódu je signál „vypnutý“.



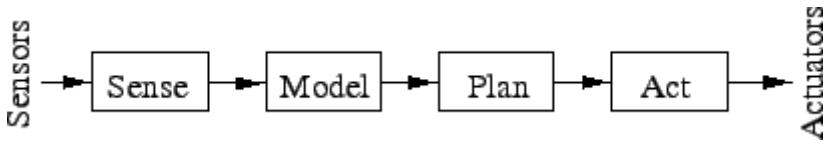
Obr. č. 1: Diagram popisujúci nastavenia RC servomotora PWM signálom [7]

1.3 Riadiace architektúry robotov

Počítačovú architektúru definoval Stone v roku 1980 nasledovne: „Computer architecture is the discipline devoted to the design of highly specific and individual computers from a collection of common building blocks.“ [9]. Robotická architektúra je v podstate to isté, s tým rozdielom, že v robotickom ponímaní sa pod výrazom architektúra rozumie skôr softvérová architektúra než hardvérová. Modifikáciou pôvodnej definície dostávame: „Robotic architecture is the discipline devoted to the design of highly specific and individual robots from a collection of common software building blocks.“ [9]. Pod robotickou architektúrou teda rozumieme množinu komponentov spolu s popisom interakcií medzi nimi.

1.3.1 Hierarchická architektúra

Tento typ architektúry sa používal už pri prvých robotoch, napríklad Shakey (SRI), CART (Standfort), Nasrem system (NASA) alebo ISAM (Isik) [11]. Správanie robota je založené na tzv. „sense-plan-act“ modeli.



Obr. č. 2: Sense-plan-act model [11]

Riešenie problému sa rozloží do jednotlivých sekvenčne organizovaných modulov. Najskôr perception modul zachytí podnety z prostredia, ktoré pošle modeling modulu. Modeling modul podľa nich aktualizuje informácie o prostredí (model sveta). Nasleduje plánovanie pomocou interného modelu sveta, a nakoniec execution modul vypočíta akciu a odošle príslušné príkazy pre aktuátory (efektory). Nevýhodou tohto centralistického a monolitického typu architektúry je problém „všetko alebo nič“, náročné postupné testovanie prototypu počas jeho vývoja, a tiež dlhý reakčný čas, resp. nedostatok oddelenia základných reflexných reakcií od komplexného modelu uvažovania.

1.3.2 Behavior-based architektúra

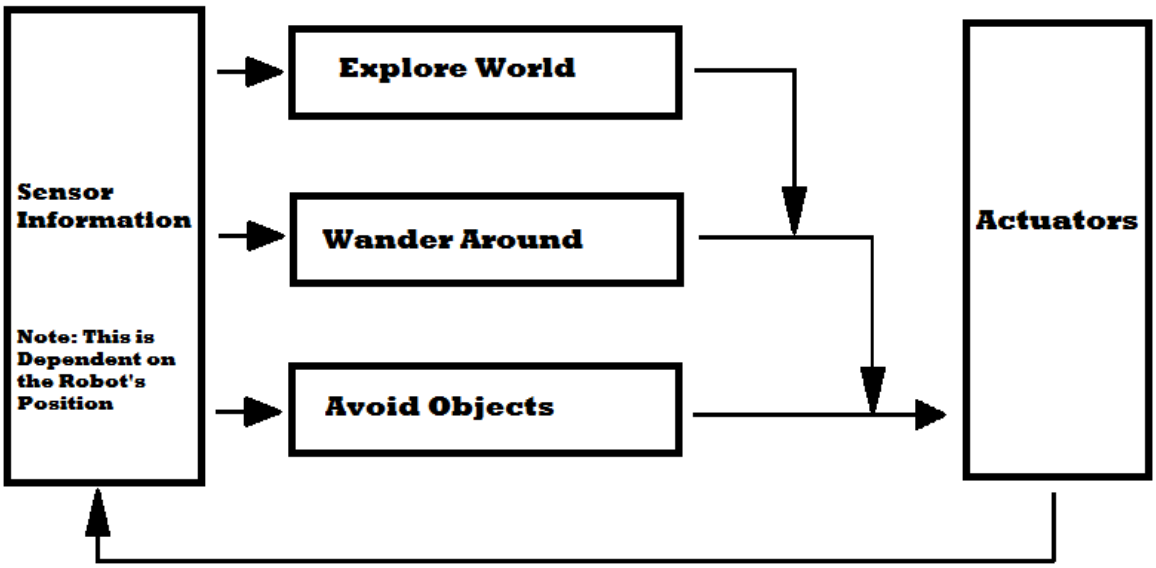
Behavior-based robotický systém [11] funguje najlepšie v prípade, ak reálny svet nedokážeme dostatočne presne charakterizovať. Toto však platí takmer v každom prípade. Nepresnosť, nepredvídateľnosť a šum sú typickými vlastnosťami prostredia, v ktorom inteligentné roboty riešia úlohy. Behavior-based systém je aspoň čiastočne reaktívny, čo znamená, že aspoň niektoré akcie volí ako priamu reakciu na konkrétnu situáciu zmeranú senzormi. Napríklad v prípade zosnímania prekážky na ňu okamžite zareaguje zastavením alebo obídením. Takéto správanie majú na starosti samostatné moduly, ktoré môžu mať priradené pevné alebo dynamicky sa meniace priority, pričom každý z nich má (po uplatnení priority) prístup k senzorickému i motorickému systému robota. Reaktívnosť spočíva napríklad v tom, že robota nie je potrebné naprogramovať tak, aby vedel ako vyzerá hľadaný predmet, alebo po akom povrchu sa robot pohybuje. Všetky potrebné informácie robot získa zo senzorov. Tieto informácie sú použité na postupné korigovanie jeho činnosti podľa prejavovaných zmien v jeho bezprostrednom okolí.

Subsumpčná architektúra [10] je typom reaktívnej architektúry. Princípom architektúry je rozdelenie správania robota do viacerých vrstiev. Vyššie vrstvy implementujú správanie pomocou nižších. Najnižšia vrstva by mohla mať napríklad za úlohu „avoid object“,

vrstva nad ňou „wander around“ a najvyššia vrstva by mohla mať za úlohu „explore world“. Vstupy (vnemy) z prostredia sú generované neustále a dáta postupujú cez jednotlivé vrstvy. Koordinácia signálov z jednotlivých vrstiev má dva hlavné mechanizmy:

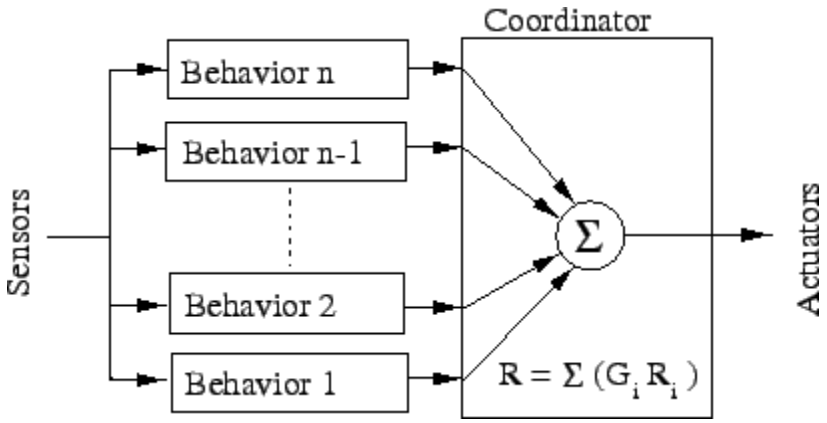
1. inhibícia - potlačenie signálu vyššou vrstvou, čo má za následok, že tok dát sa zastaví
2. supresia - nahradenie aktuálneho signálu nižšej vrstvy novým signálom z vyššej vrstvy

Nižšie vrstvy nevedia o existencii vyšších, a preto je možné bez akéhokoľvek zásahu do predošlých vrstiev pridávať nové vrstvy na vrch štruktúry.



Obr. č. 3: Príklad subsumčnej architektúry robota [10]

Architektúru **motor-schemas** [11] navrhol Ronald Arkin. Podobne ako v subsumpčnej architektúre, aj tu existuje viacero vrstiev správaní, ktoré akceptujú vstupy a vytvárajú akcie ako výstupy. Výstupom je vektor, pričom i-ty prvok vektora zodpovedá i-temu definovanému správaniu. Ak sa správanie týka napríklad pohybu robota, vektor bude dvojrozmerný (pozemný robot) alebo trojrozmerný (robot vo vzduchu/vode). Každý prvok vektora prispieva k výslednému pohybu robota, ktorý sa počíta ako suma jednotlivých prvkov (R_i) vektora, ktoré sú prenásobené koeficientom zosilnenia/zoslabenia (G_i).



Obr. č. 4: Príklad motor-schemas architektúry [11]

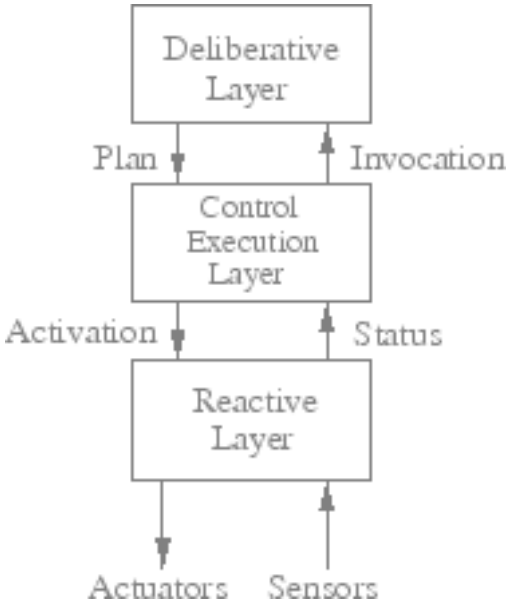
Tento princíp sa môže stretnúť s problémami v niektorých špecifických situáciách. Môže sa stať, že výsledný vektor pohybu, ktorý sme vypočítali ako lineárnu kombináciu vektorov pohybu jednotlivých správaní, sa rovná nule. Toto sa však dá jednoducho vyriešiť zavedením náhodného šumu, ktorý bude produkovať malý vektor náhodného smeru a bude zahrnutý do výpočtu výsledného vektoru. Príkladom je situácia keď sa robot nachádza presne pred prekážkou a cieľ cesty je presne za prekážkou. Ďalším problémom môže byť situácia, pri ktorej chce robot obísť prekážku a jedno správanie hovorí, aby šiel mierne doprava, druhé správanie aby šiel mierne doľava. Výsledný vektor by smeroval priamo do prekážky, čo rozhodne nie je ideálna voľba.

1.3.3 Hybridná architektúra

Napriek tomu, že behavior-based architektúra (najmä subsumpčná) bola popísaná ako veľmi efektívna pri použití v dynamickom a komplexnom prostredí, nemusí byť vždy vhodnou voľbou pre niektoré úlohy. Niekedy si úloha vyžaduje udržiavanie modelu prostredia a hlbšie uvažovanie. Behavior-based architektúra neobsahuje model prostredia, ktorý by sa priebežne updatoval. Obohatením behavior-based architektúry o prvky hierarchickej architektúry (model prostredia) dostávane tzv. hybridnú architektúru [11]. Zvyčajne sa rozdeľuje na tri časti:

1. Deliberative layer – vrstva zodpovedná za plánovanie úloh na najvyššej úrovni za účelom dosahovania cieľov.
2. Control execution layer – vrstva rozloží plán na menšie sub-tasky.

3. Reactive layer – vrstva zodpovedná za vykonávanie sub-taskov. Táto časť je implementovaná niektorou behavior-based architektúrou.



Obr. č. 5: Tri časti hybridnej architektúry [11]

1.4 Inverzná kinematika

Inverzná kinematika robotického ramena rieši problém ako dostať efektor robotického ramena do požadovaného bodu pod nejakým uhlom (orientáciou). Tento problém je pomerne komplexný. Pri inverznej kinematike v 3D priestore existuje niekoľko možných prístupov k riešeniu problému [7], ktoré môžeme rozdeliť na:

- analytické
- iteratívne (numerické, heuristické)

Štruktúra robotického ramena použitého v bakalárskej práci študenta Petra Pukančíka [7] je pomerne jednoduchá, výpočet uhlov v kĺboch ramena bol realizovaný prostredníctvom analytického prístupu, pričom sa zjednodušil sa na grafické (geometrické) riešenie. V analytickom prístupe k riešeniu problému inverznej kinematiky robotického ramena si rozdelíme celý systém kinematického zreťazenia na menšie jednoduchšie podsystémy, ktoré riešime samostatne. Tento prístup má význam len u menej zložitých kinematických zreťazení s malým počtom stupňov voľnosti DOF. V optimálnom prípade sa dá problém zjednodušiť až na súbor grafických riešení a výsledná konfigurácia sa vyráta priamo, bez potreby iteratívnych

aproximácií konfigurácie. Z povahy problému inverznej kinematiky ale vyplýva, že existuje viac možných konfigurácií na dosiahnutie danej pozície. Z tohto dôvodu je nutné zadať uhol zápästia, čiže uhol pod akým sa bude efektor približovať k hľadanej pozícií, čím zjednoznačníme riešenie inverznej kinematiky.

1.5 Stereovidenie

Stereovidenie je jednou z kľúčových tém vo výskume počítačového videnia. Cieľom stereovidenia je napodobiť ľudské videnie 3D sveta. Každé oko zachytáva iný obraz a evolúcia sa musela vysporiadať s problémom ako spojiť tieto dva obrazy do jedného obrazu, s ktorým by mohol mozog následne pracovať. Tento hypotetický obraz dostal meno po mýtickej kreatúre s jedným okom, Kyklopovi. Problémom pri vytváraní kyklopského obrazu je, že neexistuje žiaden priamy spôsob ako prekryť tieto obrazy. Väčšina predmetov sa nenachádza v rovnakých častiach týchto dvoch obrazov. To, že neexistuje jednoduché prekrytie oboch obrazov poskytlo evolúcii jedinečnú možnosť. Z rozdielu projekcií objektu sa s trochou znalosti geometrie dá vypočítať jeho vzdialenosť. Tento mechanizmus sa nazýva stereoskopické videnie. Prekvapivé je, že si tento fakt nikto neuvedomil až do roku 1838, kedy bolo stereovidenie oficiálne „objavené“ Charlesom Wheatstoneom.

Počítačové stereovidenie spočíva v zachytení dvoch obrázkov s mierne odlišným uhlom pohľadu na scénu v 3D svete. Porovnanie vzdialenosti objektu v ľavom a pravom zábere priamo závisí od vzdialenosti objektu od kamier. Porovnaním týchto dvoch záberov teda získavame informáciu o relatívnej hĺbke záberu. Výsledkom počítačového stereovidenia je tzv. hĺbková mapa (**disparity map**), čo je vlastne čiernobiely obrázok, ktorého každý pixel má istý odtieň sivej priamo zodpovedajúci relatívnej vzdialenosti od kamier. Objekty bližšie ku kamerám sa v hĺbkovej mape javia ako svetlejšie a objekty, ktoré sú vzdialenejšie ako tmavšie.

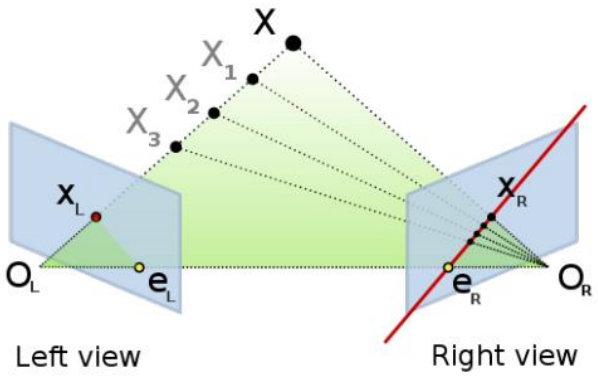
Hĺbkové mapy sú zaujímavé hneď z niekoľkých dôvodov a majú rôzne využitie v počítačovom videní:

- **3D modelovanie 2D obrázkov** - keď sa vezmú dva 2D obrázky a vypočíta sa hĺbková mapa, je možné jednoducho vytvoriť 3D model scény s použitím hĺbky ako tretieho rozmeru.

- **Sledovanie objektov** - v hĺbkovej mape je jednoduchšie sledovať objekt, pretože máme k dispozícii ďalšie možnosti segmentácie. Vytvorením segmentu pixlov na základe ich podobnosti informácie o hĺbke a pozícii v obrázku dostávame pomerne jednoducho rozoznateľný objekt na obrázku.
- **Rozoznávanie objektov v popredí** - podobne ako pri sledovaní objektov, ak vytvoríme segmenty podobných pixlov, vieme rozoznávať objekty umiestnené v popredí scény.
- **Informácia o prostredí pri plánovaní trasy** - hĺbkové mapy poskytujú aditívnu informáciu o snímanom prostredí, a v oblasti umelej inteligencie tak môžu byť využité k lepšiemu plánovaniu trasy.

1.5.1 Epipolárna geometria

Pojmom epipolárna geometria označujeme takú geometriu, pri ktorej premietacie roviny kamier neležia v jednej rovine.



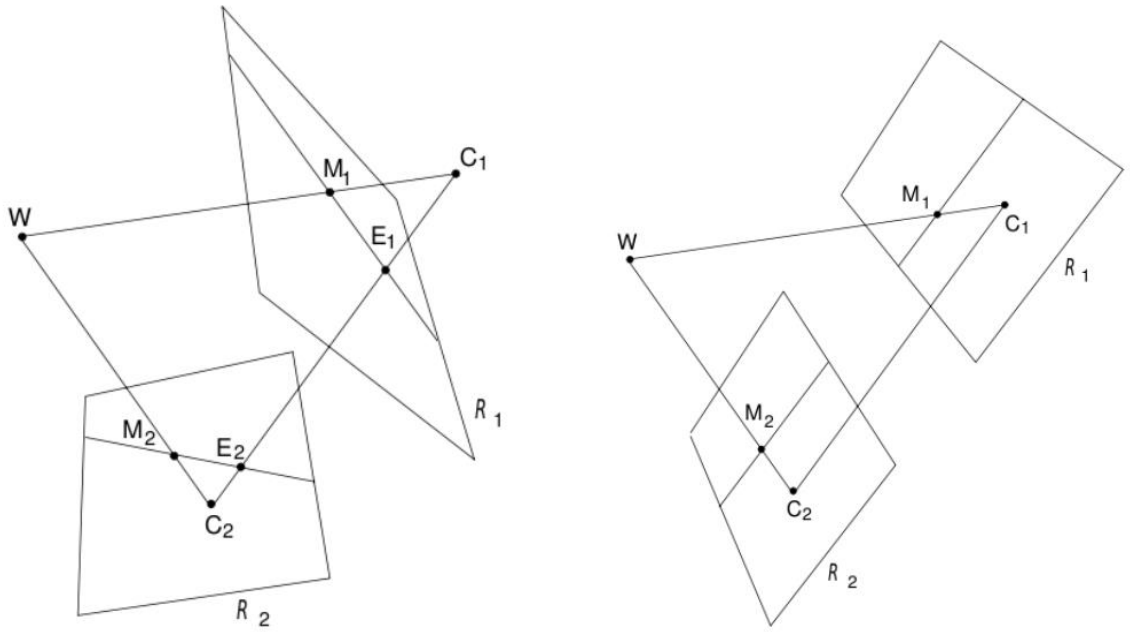
Obr. č. 6: Príklad epipolárnej geometrie [4]

Bod X v 3D priestore je premietnutý v ľavom pohľade, ktorý nazývame source image, ako bod X_L a nachádza sa na úsečke medzi ohniskom ľavej kamery O_L a bodom X v 3D priestore. Táto úsečka je premietnutá do pravého pohľadu, ktorý nazývame search image. Takto premietnutá úsečka, resp. priamka, sa nazýva epipolárna. Všetky epipolárne priamky sa pretínajú v jednom bode - epipole e_R v pravom obrázku (resp. e_L v ľavom obrázku). Pre každý pixel source image je teda potrebné vypočítať príslušnú epipolárnu priamku, v ktorej by sme hľadali snímaný pixel. Vhodnejšie by však bolo, keby každá epipolárna úsečka bola na tom istom riadku ako pixel, ktorému zodpovedá. Existuje spôsob ako transformovať obrázky

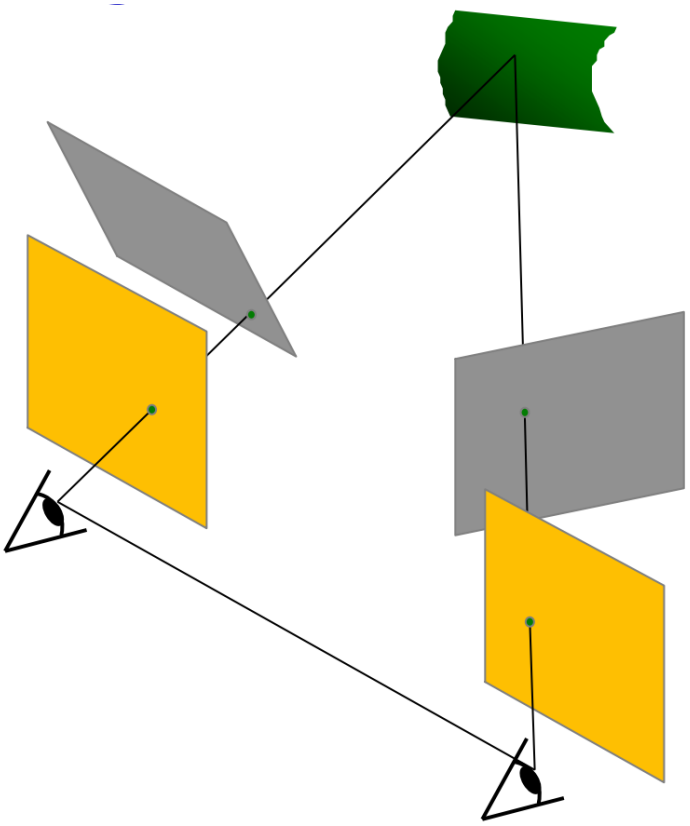
takým spôsobom, aby epipolárne úsečky boli paralelné a horizontálne orientované. Tento spôsob transformácie sa nazýva rektifikácia.

1.5.2 Rektifikácia obrazu

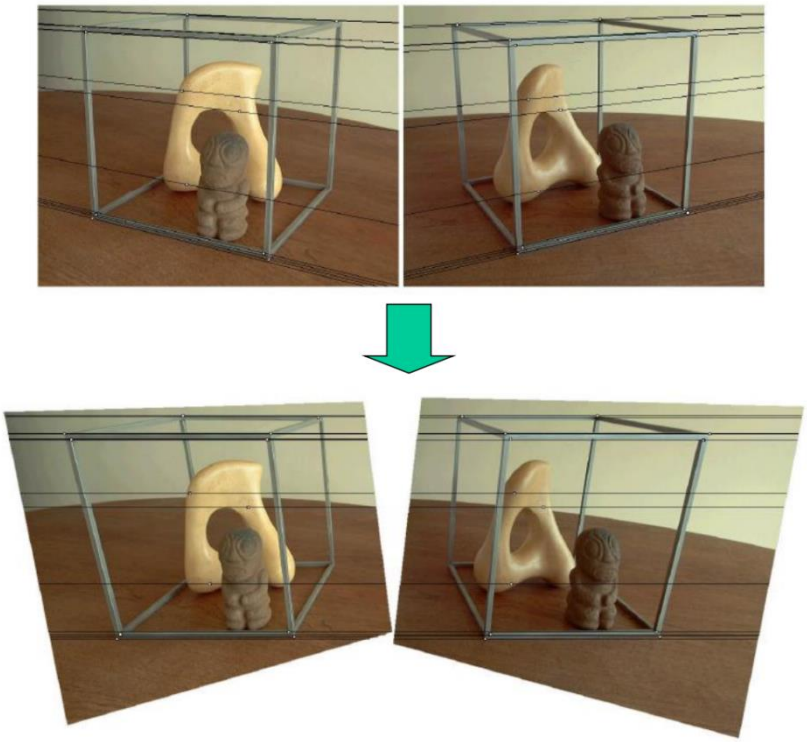
Pod pojmom rektifikácia obrazu rozumieme proces transformácie dvoch alebo viacerých obrázkov do spoločnej obrazovej roviny. Rektifikácia obrazu spočíva v nájdení a aplikovaní takej transformácie, aby sa jednotlivé epipolárne úsečky stali vodorovnými. Transformačná matica sa vypočíta osobitne pre ľavú a pravú kameru. Často sa tieto matice označujú pojmom „homography“. Po rektifikácii stačí hľadať korešpondencie bodov iba v príslušných riadkoch v oboch obrázkoch. Problém určovania vzdialenosti predmetov sa tak redukuje na pomerne jednoduché geometrické riešenie. Epipolárnu geometriu takto transformujeme na geometriu kanonickú.



Obr. č. 7: Grafické znázornenie rektifikácie obrazu epipolárnej geometrie (vľavo) na kanonickú geometriu (vpravo) [4]



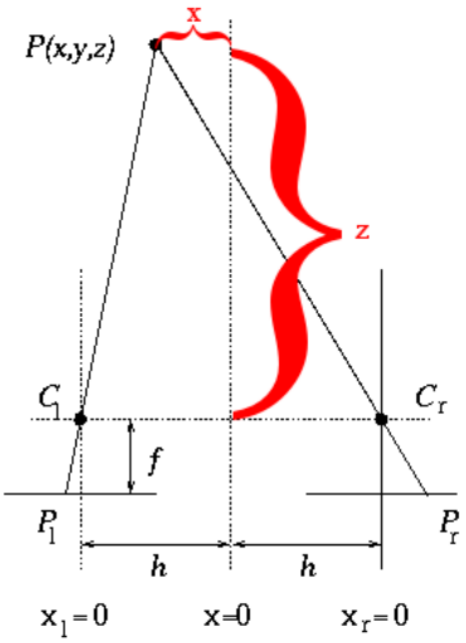
Obr. č. 8: Grafické znázornenie procesu rektifikácie. Sivou farbou sú znázornené reálne priemetne kamier, ktoré sa aplikáciou transformačných matíc dostávajú do jednej roviny (znázornené žltou farbou). [14]



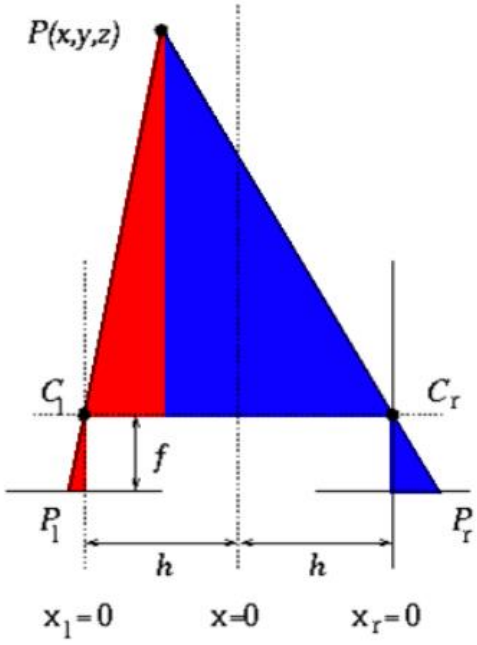
Obr. č. 9: Ukážka zachyteného obrazu pred rektifikáciou a po rektifikácii. [14]

1.5.3 Kanonická geometria

Kanonická geometria (kanonické stereo) je špeciálny prípad epipolárnej geometrie. Priemetne sú v tomto prípade rovnobežné, ležia v jednej rovine a epipoly sa nepretínajú. V takejto konfigurácii je problém hľadania hĺbky bodu podstatne jednoduchší.



Obr. č. 10: Pohľad zhora na scénu [6]



Obr. č. 11: Vyznačenie podobnosti trojuholníkov [6]

Poznáme vzdialenosť medzi kamerami $2h$, pozíciu ohnísk (C_l, C_r), ohniskovú vzdialenosť f a priemety bodu P (P_l, P_r). Cieľom je zistiť pozíciu bodu P, resp. iba jeho súradnicu z, ktorá vyjadruje hĺbku. Nech kamery majú súradnicu $z = 0$. Z obrázku je vidieť, že červené trojuholníky sú si navzájom podobné a rovnaká vlastnosť platí aj pre modré trojuholníky. Túto vlastnosť využijeme pri výpočte súradnice z bodu P:

$$tg(\alpha_l) = \frac{P_l}{f} = -\frac{(h+x)}{z}$$

$$tg(\alpha_r) = \frac{P_r}{f} = \frac{(h-x)}{z}$$

Pomocou premennej x vyjadríme rovnosť týchto dvoch rovníc, pričom výsledkom je vyjadrenie hĺbky z:

$$z(P_r - P_l) = 2hf$$

$$z = \frac{2hf}{(P_r - P_l)}$$

1.6 Plánovanie

Pod pojmom plánovanie rozumieme proces vytvárania plánu. Plán je sled akcií, ktoré je potrebné aplikovať tak, aby sa systém, pri dodržaní určených obmedzení, dostal z počiatočného stavu do stavu koncového. Jedným z najznámejších plánovacích systémov je STRIPS (Stanford Research Institute Problem Solver) [15]. STRIPS navrhli v roku 1971 Richard Fikes a Nils Nilsson. Pri plánovaní pracujeme s tzv. stavmi sveta (niekedy nazývané aj modely sveta), pričom každý stav sveta opisuje istú konfiguráciu pozorovaných premenných (rozloženia sveta) a spolu tvoria množinu situácií, ktoré môžu v uvažovanej úlohe nastať. Aby sme korektne zadefinovali problém, potrebujeme určiť počiatočný stav, koncový stav (prípadne množinu koncových stavov) a množinu akcií, ktoré možno vykonať. Pre každú akciu sú tiež definované preconditions a postconditions – teda podmienky, ktoré musia byť splnené pred vykonaním akcie a podmienky, ktoré nastanú po vykonaní akcie. Matematický zápis štandardného STRIPS problému je:

STRIPS je štvorica $\langle P, O, I, G \rangle$ kde platí že:

1. **P je množina podmienok** (boolovských premenných).
2. **O je množina operátorov** (akcií), pričom každý operátor je štvorica $\langle \alpha, \beta, \gamma, \delta \rangle$. Sú to množiny podmienok, ktoré musia byť true/false pred vykonaním akcie, a ktoré budú true/false po vykonaní akcie.
3. **I je počiatočný stav** sveta. Je to množina podmienok, ktoré sú true, ostatné sú implicitne false.
4. **G je koncový stav** sveta, reprezentovaný ako pár $\langle N, M \rangle$. Definuje množinu podmienok, ktoré musia byť false a ktoré musia byť true.

Prechody medzi jednotlivými stavmi sú modelované pomocou prechodových funkcií, ktoré možno formálne zadefinovať ako:

$$f : 2^P \times O \rightarrow 2^P$$

2^P je množina všetkých podmnožín P a teda množina všetkých možných stavov. Jednoduchý príklad definície problému zapísaného pomocou STRIPS je:

A monkey is at location A in a lab. There is a box in location C. The monkey wants the bananas that are hanging from the ceiling in location B, but it needs to move the box and climb onto it in order to reach them.

```
Initial state: At(A), Level(low), BoxAt(C), BananasAt(B)
Goal state:    Have(Bananas)
```

```
Actions:
               // move from X to Y
               _Move(X, Y)_
               Preconditions:  At(X), Level(low)
               Postconditions: not At(X), At(Y)

               // climb up on the box
               _ClimbUp(Location)_
               Preconditions:  At(Location), BoxAt(Location), Level(low)
               Postconditions: Level(high), not Level(low)

               // climb down from the box
               _ClimbDown(Location)_
               Preconditions:  At(Location), BoxAt(Location), Level(high)
               Postconditions: Level(low), not Level(high)

               // move monkey and box from X to Y
               _MoveBox(X, Y)_
               Preconditions:  At(X), BoxAt(X), Level(low)
               Postconditions: BoxAt(Y), not BoxAt(X), At(Y), not At(X)

               // take the bananas
               _TakeBananas(Location)_
               Preconditions:  At(Location), BananasAt(Location), Level(high)
               Postconditions: Have(bananas)
```

Riešenie STRIP problému sa často realizuje pomocou algoritmu **prehľadávania do šírky**. Prehľadávaný priestor je orientovaný graf, pričom každý vrchol je stav sveta a jeden vrchol grafu reprezentuje počiatočný stav. Graf rozvíjame po úrovniach (do šírky) až pokiaľ nenarazíme na žiadaný koncový stav. Rozvinutie vrcholu spočíva vo vykonaní všetkých možných akcií, a teda vo vytvorení všetkých možných synov (ďalších vrcholov) rozvíjaného vrcholu.

2. STEREOVIDENIE V OPENCV

OpenCV je knižnica pre počítačové videnie. Použitím tejto knižnice je možné úlohu spracovávania obrazu transformovať na integráciu rôznych častí OpenCV. Vďaka tomu je možné sa venovať problému na vyššej úrovni a oveľa rýchlejšie a efektívnejšie využiť čas vyhradený na dosiahnutie cieľa.

2.1 Kalibrácia kamier

OpenCV poskytuje metódu pre kalibráciu stereokamier, ktorej vstupom je zoznam súradníc bodov v ľavom obrázku, zoznam súradníc príslušných bodov v pravom obrázku a tiež prislúchajúci zoznam 3D súradníc, ktoré definujú kde tieto body ležia v reálnom svete. Ako vstupné body pre kalibráciu používame vnútorné vrcholy políčok šachovnice. Body v šachovnici sú rozpoznávané metódou OpenCV, ktorá vracia na výstupe zoznam 2D súradníc. Maticu pozícií bodov v reálnom svete volíme tak, že zohľadníme rozmery matice (počet vnútorných bodov horizontálne a vertikálne) a zachováme vzdialenosti medzi nimi (veľkosť hrany jedného štvorčeka šachovnice). Kalibračná funkcia vráti matice kamier, matice koeficientov skreslenia kamier, a maticu translácie/rotácie z jednej kamery do druhej. Matica kamery je veľkosti 3x4 a obsahuje informáciu o mapovaní 3D bodov do 2D obrazu. Nech X je bod v priestore (4-rozmerný vektor, homogénny) a bod Y je bod v obrázku (3-rozmerný, homogénny) a nech C je matica kamery (3 riadky, 4 stĺpce). Potom platí vzťah:

$$Y = C \cdot X$$

Matice koeficientov skreslenia slúžia na „vyrovnanie“ obrazu do jednej roviny (korekciu skreslenia objektívom), to znamená, že odstraňujú zakrivenie obrazu. Vďaka nim je priamka v reálnom svete priamkou aj v zábere a nie krivkou.

2.2 Algoritmy stereovidenia

Existuje viacero algoritmov na skladanie dvoch obrazov. Pod skladaním dvoch obrazov rozumieme skladanie dvoch pohľadov na scénu, ktoré sú snímané z dvoch rôznych uhlov. Cieľom procesu je vytváranie hĺbkovej mapy takým spôsobom, že ku každému pixlu v ľavom zábere nájdeme zodpovedajúci pixel v pravom zábere. Častým problémom je, že poradie pixlov jedného riadku bitmapy nekorešponduje s poradím pixlov v zodpovedajúcom

riadku druhej bitmapy. Ďalším problémom je, že ak predmet, alebo jeho časť, vidno v jednom zábere, neznamená to automaticky, že bude viditeľný aj v druhom zábere (interlineárna konzistencia). Mnohé stereo matching algoritmy sa snažia nájsť k pixlu z ľavého obrázku zodpovedajúci pixel v pravom obrázku na základe nejakej spoločnej vlastnosti - najčastejšie farby. Napriek tomu, že sa to zdá byť pomerne jasné riešenie, problém interlineárnej konzistencie stále pretrváva. V OpenCV sú využité dva základné princípy algoritmov - Graph Cut a Block Matching.

2.2.1 Graph Cut

Problematika algoritmu Graph Cut môže byť formálnejšie zadefinovaná nasledovným spôsobom [12]:

Nech L (resp. R) je množina pixlov ľavého obrázku (resp. pravého obrázku). Potom (p,q) je dvojica pixlov, v ktorej p označuje pozíciu pixlu v ľavom obrázku a q je pozícia pixlu v pravom obrázku. Predpokladáme, že obrázky sú rektifikované, a teda i-ty riadok ľavého obrázku naozaj zodpovedá i-temu riadku v pravom obrázku. Nech $A \subset L \times R$ je množina dvojíc pixlov, ktoré môžu potenciálne korešpondovať. Elementy množiny A nazývame tiež priradenia. Následne definujeme pre každé priradenie $a = (p,q)$ jeho rozdielnosť (disparity) $d(a) = d(p,q) := q - p$. Dve priradenia $a_1 = (p_1, q_1)$ a $a_2 = (p_2, q_2)$ sú susedné, zvyčajne zapisované ako $a_1 \sim a_2$, ak p_1 a p_2 sú susedné a $d(a_1) = d(a_2)$.

Konfiguráciou sa nazýva priradenie $f: A \to \{0, 1\}$. Predpokladajme priradenie $a = (p, q)$. Potom $f(a) = 1$ znamená, že p a q korešpondujú v rámci danej konfigurácie f. Také priradenia tiež nazývame ako aktívne. Ak $f(a) = 0$, znamená to, že priradenie je neaktívne. Konfigurácia sa nazýva unikátnou, ak pre každý pixel p (resp. q), existuje najviac jedno aktívne priradenie zahŕňajúce p (resp. q). Napríklad ak $f(p, q_1) = f(p, q_2) = 1$, potom vyplýva, že $q_1 = q_2$.

Ak je priradenie $a = (p, q)$ aktívne v unikátnej konfigurácii f, potom rozdielnosť (disparity) pixlu p je $df(p) := d(a) = q - p$ a rozdielnosť pixlu q je možné vyjadriť ako $df(q) := -d(a)$. Ak je každé priradenie $a = (p, q)$ zahŕňajúce p neaktívne, tak p

nekorešponduje so žiadnym pixlom v príslušnom riadku druhého obrázku. V takom prípade je p nazývané uzatvorené (occluded) v konfigurácii f.

Pre každú konfiguráciu f je možné vypočítať energiu, ktorá je daná nasledovným vzorcom [12]:

$$E(f) = E_{data}(f) + E_{occlusion}(f) + E_{smoothness}(f) + E_{uniqueness}(f).$$

Funkcia E pre výpočet energie konfigurácie f je tvorená súčtom štyroch zložiek. Každá zložka zohľadňuje isté požadované vlastnosti konfigurácie.

E_{data} - zohľadňuje ako dobre matchujú jednotlivé dvojice pixlov

$E_{occlusion}$ - minimalizuje prípady, kedy pixle neboli namatchované (occlusion)

$E_{smoothness}$ - penalizuje rôzne nerovnosti a nepravidelnosti konfigurácie

$E_{uniqueness}$ - posilňuje unikátnosť konfigurácie

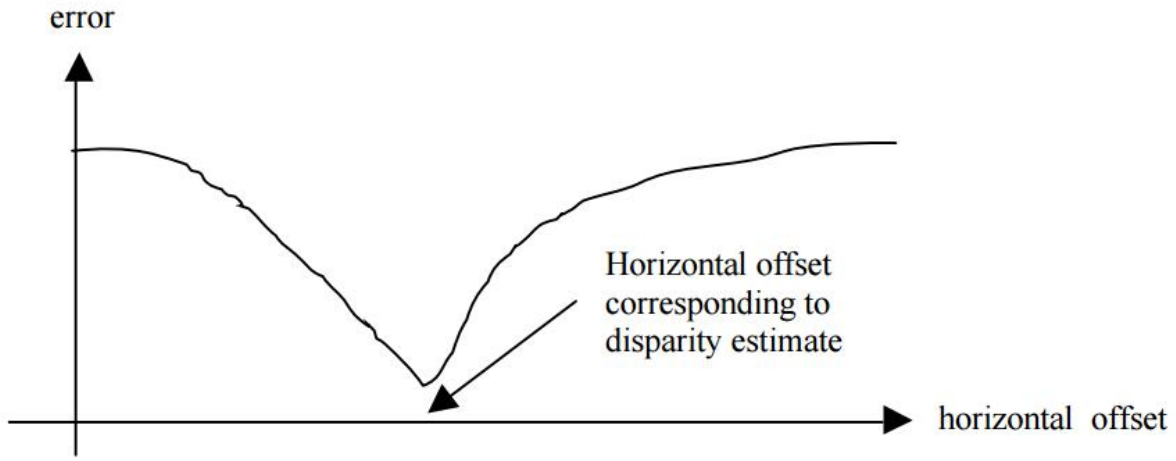
Výhodou popísaného prístupu je pomerne dobré zachovanie unikátnosti (jeden pixel v ľavom obrázku zodpovedá najviac jednému pixlu v pravom a naopak), a tiež rieši problém interlineárnej konzistencie. Hľadanie zodpovedajúcich pixlov je výpočtovo pomerne náročná operácia, a preto tento algoritmus nie je veľmi vhodný na real-time vyhodnocovanie priestoru [4]. V staršej verzii OpenCV bol tento algoritmus implementovaný v metóde `cvFindStereoCorrespondenceGC`, no v poslednej verzii (2.4.8) už chýba, resp. je deprecated.

2.2.2 Block Matching

Najčastejším prístupom pri vytváraní hĺbkovej mapy je snaha o namapovanie bloku pixlov (nazývaných tiež zhluky) z jedného obrázku na druhý obrázok. Tento prístup je známy ako Block Matching. Pre každú možnú pozíciu bloku pixlov je vypočítaná tzv. square-sense chyba, čo je chyba udávajúca vhodnosť umiestnenia bloku pixlov.

Hľadanie korešpondujúcich bodov pre každý pixel je výpočtovo náročná operácia. Uvažujme jednoduchú implementáciu: pre každý pixel v ľavom obrázku vystrihneme blok obklopujúcich pixlov (zvyčajne veľkosti napríklad 16x16 alebo 32x32 pixlov) a tento blok posúvame po zodpovedajúcom riadku v pravom obrázku. Predpokladáme, že obrázky prešli procesom rektifikácie. Na každej pozícii je vypočítaná square-sense chyba, ktorej algoritmus

nebýva úplne triviálny a zahŕňa množstvo výpočtov. Z toho dôvodu je potrebná optimalizácia (zredukovanie počtu volaní výpočtu chybovej funkcie). Miesto testovania všetkých pixlov v riadku je testovaná iba určitá podmnožina pixlov. Tie vyberáme na základe predpokladu, že susedné pixle budú mať podobnú hĺbku, nakoľko zodpovedajú rovnakému objektu. Iný prístup optimalizácie sa spolieha na to, že hodnoty chybovej funkcie tvoria často hladkú funkciu s jedným výrazným minimom (viď obr. č. 12).



Obr. č. 12: Hodnoty chybovej funkcie pri posúvaní bloku pixlov po riadku (horizontal offset) [13]

Hladkosť krivky často umožňuje nájsť minimum bez potreby prehľadávania celého riadku. Napríklad môžeme testovať iba každý tretí pixel v riadku a urobiť tak iba tretinu výpočtov. Z nich vieme potom pomerne presne určiť body, v okolí ktorých sa má presné minimum nachádzať. Následne je potrebné dopočítanie iba niekoľkých zvyšných hodnôt chybovej funkcie [13].

Veľkosť zhlukov je voliteľná. Vo všeobecnosti platí, že čím väčšie zhluky sa algoritmus snaží matchovať, tým sa jeho rýchlosť zvýši, no presnosť a detailnosť hĺbkovej mapy klesá. Väčšie zhluky sú často imúnne na „nedokonalosti“ reálneho sveta, ako napríklad odlesky alebo iné osvetlenie, no na druhú stranu nemusia dokázať zachytiť jemné detaily objektov (priehlbiny a pod.) [4]. OpenCV implementuje tento princíp v dvoch metódach - `StereoBM::operator` a `StereoSGBM::operator`.

2.3 Hĺbková mapa

Výstupom všetkých metód stereovidenia v OpenCV je hĺbková mapa alebo disparity map (popísaná v podkapitole 1.5). V práci budeme porovnávať algoritmy StereoBM a StereoSGBM, ktorým je potrebné zadať na vstupe niekoľko parametrov. Nájdenie ideálnych parametrov je predmetom testovania a nastavenia pre konkrétnu konfiguráciu kamier a rozsahu snímaného priestoru. Parametrami algoritmov sú:

`preFilterSize` - nastavuje veľkosť „pre-filtering“ okna.

`preFilterCap` - predspracovanie obrázkov pomocou „pre-filtering“. Nastavuje hranicu $\langle -preFilterCap,\ +preFilterCap \rangle$ podľa ktorej dochádza k miešaniu farieb (vyhladzovaniu plôch a hrán).

`SADWindowSize` - veľkosť matchovaného bloku pixlov. Musí to byť nepárne číslo väčšie alebo rovné 1. Zvyčajne sa volí v rozmedzí 3 až 11.

`minDisparity` - minimálna možná hodnota disparity. Zvyčajne je to nula, ale niekedy je potrebné túto hodnotu upraviť.

`numberOfDisp` - maximálna hodnota disparity po odčítaní minimálnej hodnoty disparity. Hodnota musí byť väčšia ako nula, pričom musí byť deliteľná číslom 16.

`textureThreshold` - parameter, podľa ktorého sa určuje minimálna hranica textúrovania objektov – disparity sa počíta iba pre objekty, ktoré prešli touto kontrolou.

`uniquenessRatio` - hranica v percentách, o ktorú najmenej by mala zvíťaziť najlepšia (minimálna) hodnota chybovej funkcie nad druhou najlepšou hodnotou chybovej funkcie, aby bola považovaná za korektné matchnutie. Zvyčajne je hodnota medzi 5-15 dostatočná.

`speckleWindowSize` - parameter, pomocou ktorého sa kontroluje „speckle filtering“ čo je filtrovanie zašumenia obrazu. Parameter nastavuje maximálnu veľkosť regiónu, ktorý môže byť označený ako šum. Nastavenie na hodnotu nula vypína filtrovanie. Zvyčajne sa však hodnota pohybuje v rozmedzí 50-200.

`speckleRange` - maximálna rôznorodosť disparity v rámci jedného komponentu (zhluku označeného ako šum). Ak je povolené filtrovanie,

parameter by mal byť kladné číslo, ktoré bude automaticky prenásobené 16-timi. Obvykle je nastavenie na hodnotu 1 alebo 2 postačujúce.

`P1` - prvý parameter kontrolujúci jemnosť (smoothness) disparity mapy. Zvyčajne sa nastavuje ako:

$$P1 = 8 * number\_of\_image\_channels * * SADWindowSize * SADWindowSize$$

`P2` - druhý parameter kontrolujúci jemnosť (smoothness) disparity mapy. Čím väčšia je hodnota `P2`, tým je disparity mapa jemnejšia. `P1` je hodnota penalizácie za rozdiel hodnôt disparity o 1 medzi susednými pixlami. `P2` je hodnota penalizácie za rozdiel hodnôt disparity o viac ako 1 medzi susednými pixlami. Algoritmus vyžaduje aby `P2 > P1` . Zvyčajne sa nastavuje ako:

$$P2 = 32 * number\_of\_image\_channels * * SADWindowSize * SADWindowSize$$

`disp12MaxDiff` - maximálny povolený rozdiel medzi disparity pri „left-right disparity“ kontrole. Pre vypnutie kontroly je potrebné nastaviť parameter na zápornú hodnotu alebo na hodnotu nula.

`fullDP` - pri nastavení parametru na true je pri každom vytvorení hĺbkovej mapy spustený „full-scale two-pass dynamic programming“ algoritmus. Výsledkom sú kvalitnejšie výstupy, no má veľkú zložitosť $O(W * H * numDisparities)$, ktorá sa prejaví už pri rozlíšení 640x480 px a vyšších. Defaultne je nastavený na false.

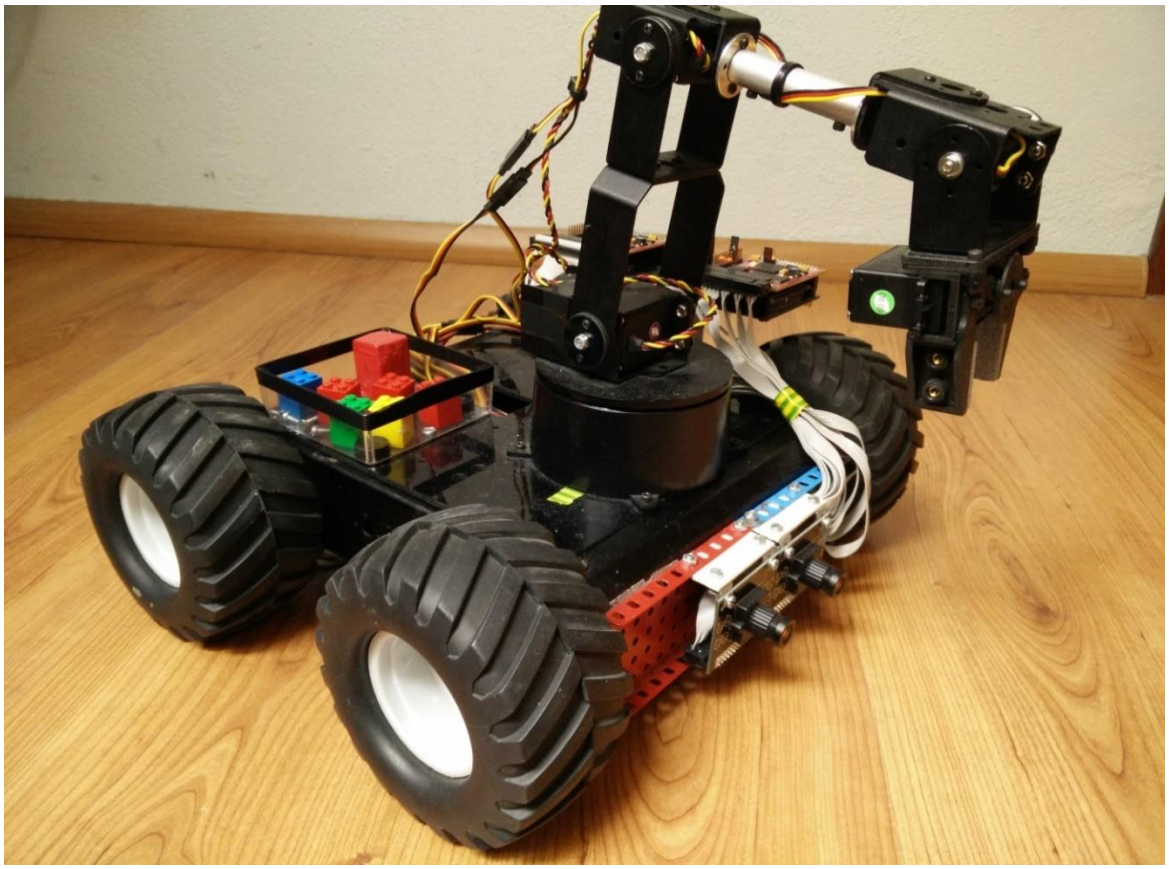
Výsledná hĺbková mapa je reprezentovaná maticou prvkov typu byte, nadobúdajúcich hodnoty 0-255. Hodnoty bodov nezodpovedajú priamo úmerne vzdialenosti ku kamerám. Pre tento výpočet je potrebné použiť metódu `reprojectImageTo3D()`, ktorej výstupom sú body obohatené o informáciu o ich vzdialenosti od kamery. Aby táto metóda dávala relevantné výsledky, musí byť najskôr vykonaná presná kalibrácia kamier.

3. MOBILNÝ ROBOT JANKO HRAŠKO

Práca na mobilnom robotovi bola založená na bakalárskej práci študenta P. Pukančíka - Riadiaci systém s inverznou kinematikou pre mobilné robotické rameno [7]. Robot bol v rozpracovanom stave, a v podstate sám od seba toho veľa vykonať nedokázal. Na druhú stranu bol hotový kompletný systém ovládania jeho štyroch DC motorov poháňajúcich robustné kolesá spolu so systémom ovládania servomotorov robotického ramena. Robotické rameno obsahuje celkovo päť servomotorov (päť stupňov voľnosti DOF), z ktorých jeden slúži ako efektor (gripper) a pomocou jedného sa realizuje pohyb ramena okolo svojej osi v rozsahu približne 200°. Zvyšné tri servomotory slúžia na pohyb ramena v jednej rovine. Vzhľadom na jednoduchosť ramena P. Pukančík implementoval inverznú kinematiku analytickou metódou, ktorá využíva geometrické riešenie problému, ako sme uviedli v podkapitole 1.4 vyššie.

Vzhľadom na zamýšľané použitie robota môžeme konštatovať, že bolo v značnej miere potrebné modifikovať existujúci hardvér a doinštalovať ďalšie komponenty. Za uvedeným účelom boli vykonané nasledujúce modifikácie:

- Úplné prerobenie napájania - výmena pôvodných NiMH batérii za dve nové vysokovýkonné LiPo batérie.
- Robotovi bol prirobený systém stereokamier, vďaka ktorým prijíma podnety z prostredia. Použitý bol systém Surveyor SVS pozostávajúci z 2 farebných kamier umiestnených na spoločnej doske od Surveyor Corporation s procesormi Blackfin. Obraz z kamier je poskytovaný prostredníctvom embedded web servera Lantronix Matchport, na ktorý sa pripája prostredníctvom Ad-hoc WiFi pripojenia.
- Pridanie počítačového modulu Gumstix, na ktorom beží linuxová distribúcia operačného systému (Yocto Project). Komunikácia s modulom SVS je riešená pomocou Ad-hoc WiFi prepojenia.



Obr. č. 13: Robot Janko Hraško

3.1 Technická špecifikácia robota Janko Hraško

Mobilný robot Janko Hraško obsahuje nasledovné hardvérové komponenty:

- Lynxmotion AL5D Arm Hardware-Only Kit
- Lynxmotion A4WD1 Robot Base
- 4x DC 12V motor
- servo motors for robot arm (3xHS-422, 1x HS475HB, 1x HS775HB)
- Surveyor Stereo Vision System
- Gumstix Overo Air + Summit Board
- Custom control board with two AVR microcontrollers ATmega88 for arm servo motors conotrol, Attiny2313 for driving 4 DC motors of robot base chassis
- 4-channel I2C Bi-directional Logic Level Converter BSS138 (Adafruit 757) - for serial connection betwenn 1.8V Gumstix and 5V control board
- SRF-08 ultrasonic sensor for obstacle detection
- 2pcs LiPo batteries: 3S (2600 mAh) DC motors power, and 2S (3300 mAh): gumstix, SVS, servo motors power

3.2 Hardvér

Telo robota tvorí Aluminum 4WD1 Rover Kit obsahujúci skrinku a štyri robustné kolesá. Tento kit neobsahoval žiadnu elektroniku. Vo vnútri tela sa nachádza **custom-made control board** obsahujúca dva mikroprocesory ATmega88 a ATtiny2313. Mikroprocesor ATmega88 zabezpečuje riadenie pohybu robotického ramena a mikroprocesor ATtiny2313 má za úlohu riadenie kolies. Vnútorná frekvencia mikročipov sa dá interne nastaviť na frekvencie, zväčša 4 či 8 MHz, ktoré nám ale nevyhovujú pre štandardnú komunikačnú rýchlosť prenosu 115200 bps, pretože majú pri tejto rýchlosti veľkú chybovosť. Preto bol pridaný externý kryštálový oscilátor, ktorý zmení frekvenciu mikroradiča na 11.0592MHz, čo je optimálna frekvencia pre danú rýchlosť prenosu. Z dosky je vyvedený štandardný sériový port, na ktorom po zapnutí počúvajú oba mikroprocesory.

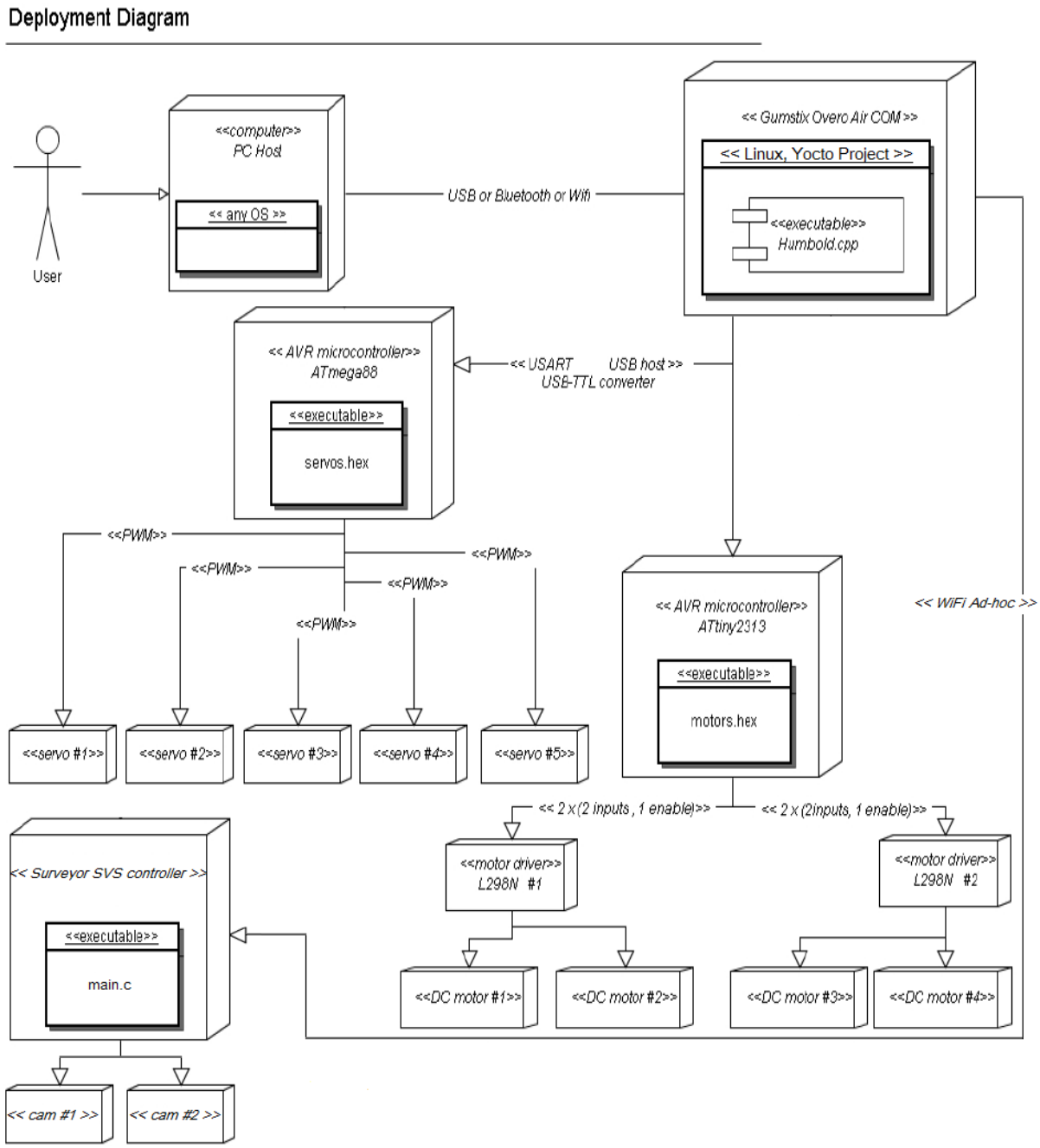
Ďalšou dôležitou časťou je **robotické rameno** Lynxmotion AL5B Arm Hardware-Only Kit. Rameno obsahuje 5 servomotorov. Skladá sa z otočnej základne, 3 rotačných kĺbov a jedného efektora. Spolu má teda 5 stupňov voľnosti DOF. Riadenie týchto servomotorov je zabezpečené pomocou piatich PWM signálov (ATmega88). Celé robotické rameno je napájané vysokovýkonnou Li-Po batériou s napätím 7,4V.

Kolesá robota sú hnané výkonnými prevodovými DC motormi GM37 (max 150 ot./min), ktoré sú napájané LiPo batériou s napätím 11,1V. Každý motor je ovládaný pomocou troch vstupov – 2x input a 1x enable. Inputy určujú smer otáčania a enable nastavuje rýchlosť otáčania cez PWM signál (popísaný v kapitole 1.2.2). DC motorom sa pomocou PWM signálu posiela zakódovaná informácia o rýchlosti otáčania. To znamená, že čím je šírka signálu v perióde väčšia, tým viac elektrického napätia preposiela radič motora, a tým rýchlejšie sa motor bude točiť. Každý z motorov môže byť v jednom zo štyroch módov:

- otáča sa v smere hodinových ručičiek
- otáča sa proti smeru hodinových ručičiek
- drží polohu (brzdí)
- je v režime voľnobehu

Robotické rameno je zapojené na hlavný vypínač. Počas behu robota je tento subsystém vždy aktívny. Mobilná platforma (servomotory kolies) má samostatný vypínač,

nakoľko pri práci s robotom tento subsystém často nie je potrebný (testovanie inverznej kinematiky a pod.).



Obr. č. 14: Grafické zobrazenie dekompozície systému [7]

3.3 Softvér a komunikačný protokol

Na základnej doske robota bežia dva subsystémy naprogramované v jazyku C (subsystém na riadenie ramena a subsystém na riadenie motorov kolies). Obrázok č. 14 znázorňuje dekompozíciu systému robota a sú na ňom vyobrazené spomínané subsystémy, ktoré bežia na mikročipoch ATtiny2313 a ATmega88.

Po naštartovaní mobilného robota prebehne inicializácia systémov a robot začne počúvať na štandardnom sériovom porte, na ktorý sa možno pripojiť a zadávať mu inštrukcie. Pripojenie je možné s nastavením rýchlosti na 115200 bit/s na príslušnom porte. Komunikácia spočíva v zadávaní príkazov začínajúcich symbolom '.' a spĺňajúcich nasledovný protokol:

1. *Servo control board* (ATmega88 @ 11.0592MHz)
   - `.sXN;` move servo to specified position, X - servo number 1-5, N - position 100-200 for S1-S3, 27-55 for S4-S5
     s1 - efektor, ktorý má za úlohu uchopenie predmetu
     s2 - odpovedá natočeniu kĺbu zápästia, môže dosahovať hodnoty <0, 125>
     s3 - odpovedá natočeniu kĺbu lakťa, môže dosahovať hodnoty <0, 147>
     s4 - odpovedá natočeniu kĺbu ramena, môže dosahovať hodnoty <0, 90>
     s5 - odpovedá natočeniu kĺbu základne, môže dosahovať hodnoty < -180, 0>
   - `.sv` print servo states
   - `.sl` flash servo LED
   - `.sk` invert servo LED
   - `.sx` move arm to neutral position
   - `.sz` move arm to position 1
   - `.b` print voltage of the batteries
2. *Motor control board* (Attiny2313 @ 11.0592MHz)
   - `.mXYN;` set speed of a motor, X - motor number 1-4, Y - direction (1=fwd,2=bwd,3=brake,4=float), N - speed 0-255

`.mv`	print motor states
`.mh`	brake all motors
`.mj`	float all motors
`.mf`	all motors fwd
`.mg`	all motors bwd
`.ml`	flash motor LED
`.mk`	invert motor LED

3.4 Platforma Gumstix

Gumstix je americká spoločnosť so sídlom v kalifornskom Redwood City . Vyvíja a vyrába malé SBC počítače (single-board computer). Medzi výhody takéhoto počítača patrí jeho nízka spotreba a malé rozmery, preto sa ideálne hodí do prostredia robotiky. Jedným z produktov tejto firmy je aj model Gumstix Overo Air, ktorý v danom prípade bude slúžiť na spracovanie obrazu zo stereokamier a následné odosielanie inštrukcií na platformu mobilného robota pomocou sériového prepojenia.

Platforma Gumstix sa skladá z dvoch častí. Prvou je COM doska Gumstix Overo Air obsahujúca procesor, pamäť RAM, slot na SD kartu a tiež WiFi a Bluetooth modul. Vzhľadom na to, že je potrebné prepojenie medzi Gumstixom a mobilnou platformou robota cez sériový port, a kvôli pohodlnejšiemu ladeniu (pripojenie na externú konzolu, monitor), je potrebná druhá časť – rozširujúca doska (expansion board). Rozširujúca doska obsahuje 40 pinov, z ktorých 4 konkrétne poslúžia ako low-voltage (1,8V) sériový port (Rx, Tx, VCC, GND). Procesorom Gumstixu je štandardný procesor s architektúrou ARM Cortex-A8 o frekvencii 600Mhz. Pamäť RAM má kapacitu 512MB.

Ako operačný systém pre Gumstix Overo Air bola zvolená distribúcia Linuxu špeciálne upravená pre platformu Gumstix, vytvorená pomocou systému Yocto Project. Táto distribúcia je okresanou verziou Linuxu, čím robí tento operačný systém ideálnym pre naše potreby, nakoľko je veľmi rýchly a hardwarové zdroje zaťažuje iba minimálne. V Linuxe je predinštalovaný balíčkovací systém Smart, ktorý nám poslúži na doinštalovanie potrebných knižníc. Keďže na platforme je dostupná celá rada GNU kompilátorov, chýbajúce balíčky a

knižnice si možno skompilovať zo zdrojových kódov, či už priamo na doske alebo pomocou cross-compilera na PC.

3.5 Surveyor Stereo Vision Sytem

Surveyor Stereo Vision System je zariadenie umožňujúce stereovidenie. Obsahuje dve farebné kamery OmniVision OV7725 (s rozlíšením max 640x480 px) umiestnené na prednom paneli a prepojené s riadiacimi doskami SRV-1 s procesormi Blackfin. Hoci dosky SRV-1 sú tiež programovateľné a môže na nich bežať aj OS Linux, v systéme sú zapojené tak, aby obraz okamžite poskytovali cez vysokorýchlostný sériový port prostredníctvom embedded web servera Lantronix Matchport. Pomocou tohto hardvéru je tiež možné ovládanie ôsmich servomotorov a dvoch DC motorov, ktorým dokáže poskytnúť prúd o veľkosti až 1A. Spojenie so systémom je možné nadviazať cez Ad-hoc WiFi prepojenie. Detaily nastavenia a prístupu k obrazu kamier sú popísané v podkapitole 5.1.

4. NÁVRH A ŠPECIFIKÁCIA

Kapitola sa venuje opisu funkcionality systému bežiaceho na platforme Gumstix s operačným systémom Linux. Hlavnou úlohou systému je prijímať podnety z prostredia pomocou stereokamier, vyhodnocovať ich, a následne demonštrovať použiteľnosť na konkrétnom správaní robota. Celková architektúra je v štýle Behavior-Based. Jednou z demonštrácii funkčnosti bude naprogramované špecifické správanie robota spočívajúce v hľadaní červenej kocky okolo seba (otáčanie sa dookola, kým ju neuvidí v zábere) a pohyb k nej. Akonáhle sa kocka dostane do dosahu robotického ramena, vypočíta sa jej pozícia pred robotom a pomocou ramena je uchopená a presunutá na nákladný priestor robota. Robot bude automaticky hľadať ďalšie kocky. Bude sa tiež vedieť vysporiadať so situáciou, v ktorej sa v jeho zornom uhle nachádza väčší počet kociek. Cieľom je, aby vytvorený systém bol modulárny – jednotlivé moduly bude možné v prípade potreby preprogramovať prípadne úplne vymeniť.

Systém bude naprogramovaný v jazyku C++, ktorý bude využívať externé knižnice OpenCV. Hotový systém nájde uplatnenie pri vykonávaní experimentov v umelej inteligencii a kognitívnej vede.

4.1 Architektúra systému

Vzhľadom na ciel práce – modulárnosť systému – bude systém rozdelený do viacerých nezávislých častí (modulov):

- robot_arm_module
- robot_base_module
- robot_arm_motion_module
- robot_base_motion_module
- camera_module
- stereo_depthmap_module
- object_recognition_module

V princípe by mal každý modul pracovať nezávisle na ostatných, avšak posedné tri moduly (camera_module, stereo_depthmap_module, object_recognition_module) musia úzko

spolupracovať a vymieňať si informácie cez zdielanú pamäť. Každý modul bude reprezentovaný jednou hlavnou triedou, ktorá bude poskytovať public metódy pre prácu s ním. Programovanie systému bude orientované na to, aby bola výsledkom práce **knižnica** použiteľná pre ďalšie projekty. Dôraz bude kladený najmä na jednoduchosť a zrozumiteľnosť použitia metód tak, aby knižnicu bolo možné používať bez potreby študovania jej vnútornej štruktúry.

4.1.1 Modul robot_arm_module

Modul robot_arm_module bude z veľkej časti tvorený kódom z bakalárskej práce študenta P. Pukančíka. Bude to svojim spôsobom akýsi wrapper nad robotickou rukou a jej servomotormi. Bude obsahovať implementáciu výpočtu inverznej kinematiky, pri ktorom zadá sa bod v priestore a uhol, pod ktorým sa má do neho dostať posledným článkom ramena. Následne algoritmus vráti na výstupe true/false podľa realizovateľnosti. Bakalárska práca študenta P. Pukančíka však nerieši rýchlosť a plynulosť pohybu robotického ramena, ani plynulý presun efektora ramena z jedného bodu do druhého.

4.1.2 Modul robot_base_module

Modul robot_base_module bude tiež z veľkej časti tvorený kódom z bakalárskej práce študenta P. Pukančíka. Bude to wrapper nad robotickou základňou a jej DC motormi kolies.

4.1.3 Modul robot_arm_motion_module

Modul robot_arm_motion_module bude zabezpečovať pohyb ramena v priestore. Bude zohľadňovať pracovný priestor robota a riadiť pohyb ramena takým spôsobom, že nebude dochádzať ku kolíziám s telom robota alebo samotnou robotickou rukou.

4.1.4 Modul robot_base_motion_module

Modul robot_base_motion_module bude zabezpečovať pohyb robota v priestore (otáčanie, pohyb dopredu a dozadu).

4.1.5 Modul camera_module

Modul camera_module bude zabezpečovať komunikáciu so stereokamerovým systémom Surveyor SVS, súčasťou ktorého je aj získavanie obrázkov z kamier. Najskôr prebehne inicializácia kamier (nastavenie kvality obrazu a rozlíšenia).

4.1.6 Modul stereo_depthmap_module

Modul stereo_depthmap_module bude mať za úlohu produkovať hĺbkovú mapu snímaného priestoru. Na spracovanie obrazu z kamier bude využívať niektorú z implementácii stereo matching knižníc OpenCV. Je potrebné spraviť výskum použiteľnosti v reálnom nasadení a porovnať kvalitu výstupov.

4.1.7 Modul object_recognition_module

Modul object_recognition_module bude zabezpečovať rozpoznávanie objektov v obraze poskytovanom kamerami. Rozpoznávanie objektov bude prebiehať na základe farby, pričom ako vstup bude definované rozmedzie detekovaného spektra farieb. Po rozpoznaní objektu sa vypočíta jeho vzdialenosť od kamier a pripočíta sa posun základne od kamier. Výstupom bude bod v priestore s tromi súradnicami reprezentujúci stred objektu. Modul bude úzko spolupracovať s modulom stereo_depthmap_module.

4.2 Triedy a metódy

Kapitola tvorí dokumentáciu k navrhovanému kódu, ktorý je rozdelený na sedem modulov (tried) podľa funkcionality popísanej v predošlom návrhu v podkapitole 4.1.

4.2.1 Trieda `RoboticArm`

Trieda `RoboticArm` reprezentuje robotické rameno robota. Po zavolaní konštruktora triedy `RoboticArm` sa vytvorí inštancia robotickej ruky obsahujúca pole objektov triedy `Servo` (servomotorov ramena) a pole piatich prepojení medzi nimi typu integer. Trieda `RoboticArm` má nasledovné metódy:

- `initConnection()` - otvorí sériové pripojenie na zadaný port.
- `getServoPositions()` - vráti pointer na pole hodnôt aktuálne nastavených na servomotoroch robotického ramena.
- `setServo_i_degrees()` - nastaví zvolené servo do zadanej pozície.
- `find_solution_and_do_InverseKinematics()` - nájde riešenie inverznej kinematiky do zadaného bodu v priestore a vráti true, inak vráti false. V prípade, že existuje riešenie, automaticky nastaví premenné servomotorov na výsledné hodnoty. Následne je potrebné volať metódu `sendServoPacket()` pre reálne nastavenie robotického ramena do zadaného bodu v priestore.
- `sendServoPacket()` - odošle príkaz na nastavenie zvoleného servomotora.

Trieda `Servo` reprezentuje servomotor. Obsahuje niekoľko atribútov: `min_degrees`, `max_degrees`, `signal_value_for_min_deg`, `signal_value_for_max_deg`, `rozpatie`, `akt_signal_value`. Konštruktoru triedy sa posiela teda minimálna a maximálna poloha servomotora v stupňoch (z nich sa vypočíta rozsah) a príslušná minimálna a maximálna hodnota signálu. Metódy triedy sú:

- `conversion_signal_value_to_degrees()` / `conversion_degrees_to_signal_value()` - metódy slúžiace na konverziu medzi hodnotou signálu a hodnotou uhla v stupňoch.
- `isInRange_signal_value()` / `isInRange_degrees()` - metódy vracajú true/false podľa toho, či parameter je v rozsahu min-max.

- `set_servo_by_signal_value()` / `set_servo_by_degrees()` - nastavenie servomotora na hodnotu zadanú ako parameter.
- `get_akt_signal_value()` - vráti aktuálnu hodnotu servomotora.
- `getAngle_degrees()` / `getAngle_radians()` - vráti aktuálnu hodnotu servomotora v stupňoch alebo v radiánoch.

Okrem vyššie uvedených tried sú zadeklarované dve štruktúry:

- `MyPoint` - reprezentuje bod v priestore (obsahuje 3 premenné *x, y, z*) a preťažuje operátor `!=` (pre porovnanie či sú dva body zhodné).
- `RoboArm_IK_Result` - používa sa ako návratová štruktúra metódy `find_solution_and_do_InverseKinematics()` a obsahuje 5 premenných `q1-q5` typu double, ktoré zodpovedajú hodnotám uhlov robotického ramena a jednu premennú typu bool, ktorá hovorí o dosiahnuteľnosti zadaného bodu v priestore robotickým ramenom.

4.2.2 Trieda `RoboticBase`

Trieda `RoboticBase` reprezentuje telo robota, resp. jeho štyri kolesá, slúžiace na pohyb celého robota v priestore. Po zavolaní konštruktora triedy `RoboticBase` sa vytvorí inštancia poľa motorov. Každý motor je objekt triedy `Motor`, obsahujúci dva atribúty - `direction` a `speed`. Hodnota premennej `direction` zodpovedá štyrom možným stavom motora kolesa - forward, backward, brake a float. Premenná `speed` nadobúda hodnoty 0-255, čo zodpovedá rýchlosti otáčania kolesa 0-150 ot./min. Oba atribúty majú v tejto triede prislúchajúce metódy typu get a set. Trieda `RoboticBase` poskytuje nasledovné metódy:

- `initConnection()` - otvorí sériové pripojenie na zadaný port.
- `getMotor()` - vráti pointer na príslušný objekt triedy `Motor` (podľa zadaného parametra).
- `sendMotorPacket()` - metóda na odosielanie príkazov motorom.

4.2.3 Trieda `RobotArmMotion`

Trieda `RobotArmMotion` obsahuje implementáciu plynulého pohybu robotickej ruky - presun robotickej ruky z jedného bodu do druhého. Trieda poskytuje metódy:

- `initArmPosition()` - po zavolaní nastaví robotickú ruku do pohotovostnej pozície (do konkrétneho bodu).
- `grip()` - efektor robotickej ruky za zatvorí (uchopenie predmetu).
- `ungrip()` - otvorenie efektoru robotickej ruky na maximum.
- `moveArm()` - plynulý pohyb robotickej ruky z jedného zadaného bodu do druhého. Body sú zadávané ako štruktúra `cv::Point3D`. Metóda si vypočíta vhodnú trajektóriu pomocou kontrolných bodov. Trajektória je vypočítaná tak, aby nedošlo ku kolízii s telom robota alebo základňou robotickej ruky.

4.2.4 Trieda `RobotBaseMotion`

Trieda `RobotBaseMotion` obsahuje implementáciu pohybu celého robota - otáčanie, pohyb vpred a vzad. Trieda poskytuje metódy:

- `moveFWD()` - pohnutie robota dopredu zadanou rýchlosťou počas trvania zadaného časového intervalu.
- `moveBWD()` - pohnutie robota dozadu zadanou rýchlosťou počas trvania zadaného časového intervalu.
- `turnLeft()` - otočenie robota doľava zadanou rýchlosťou počas trvania zadaného časového intervalu.
- `turnRight()` - otočenie robota doprava zadanou rýchlosťou počas trvania zadaného časového intervalu.

4.2.5 Trieda `StereoCam`

Objekt triedy `StereoCam` reprezentuje systém kamier robota. Dokáže sa pripojiť na kamerový modul SVS a získavať obraz z kamier. Trieda poskytuje metódy:

- `initConnection()` - inicializuje pripojenie na modul SVS a nastaví správne rozlíšenie (640x480 px) a kvalitu získavaného obrazu vo formáte JPEG.
- `percept()` - volaním metódy získame zábery z oboch kamier.

4.2.6 Trieda `DepthMap`

Objekt triedy `DepthMap` zabezpečuje transformáciu obrazu z kamier do 3D priestoru, výsledkom čoho je hĺbková mapa. Trieda má v sebe zadefinované všetky parametre použitého algoritmu stereovidenia nastavené pre použitie s konkrétnym modulom kamier. Nachádza sa tu aj konštanta vzdialenosti medzi kamerami a vzdialenosti kamier od stredu základne robotického ramena pre korektný výpočet pozície objektu pred robotom. Trieda poskytuje nasledovnú metódu:

- `calculateDepthMap()` – vracia objekt triedy `cv::Mat` reprezentujúci hĺbkovú mapu.

4.2.7 Trieda `ObjectRecognition`

Objekt triedy `ObjectRecognition` sa využíva na rozpoznávanie objektov v obraze poskytovanom triedou `StereoCam` a následný výpočet pozície objektu v priestore za pomoci hĺbkovej mapy poskytovanej triedou `DepthMap`. Trieda poskytuje metódy:

- `isObjectInView()` - vracia true/false podľa toho, či sa hľadaný objekt (triedy `MyObject`) nachádza v zábere alebo nie (či bol rozpoznaný alebo nie).
- `getObjectPositionXYZ()` - vráti stred najbližšieho rozpoznaného objektu (triedy `MyObject`) ako štruktúru `cv::Point3i`.
- `getObjectRect()` - vráti štruktúru `cv::Rect`, ktorá označuje obdĺžnik ohraničujúci rozpoznaný objekt (Bounding Box).
- `getObjectDistance()` - vráti vzdialenosť rozpoznaného objektu v milimetroch.

- `getObjectX()` - vráti posun stredu rozpoznaného objektu od osi prechádzajúcej stredom tela robota a stredom medzi kamerami v milimetroch (záporná hodnota ak je objekt v ľavej časti; kladná ak je objekt v pravej časti).

Aby bolo možné lepšie popísať hľadaný objekt v priestore, posiela sa metódam, ktoré volajú hľadanie objektu, ako parameter objekt triedy `MyObject`. Trieda obsahuje nasledovné public metódy:

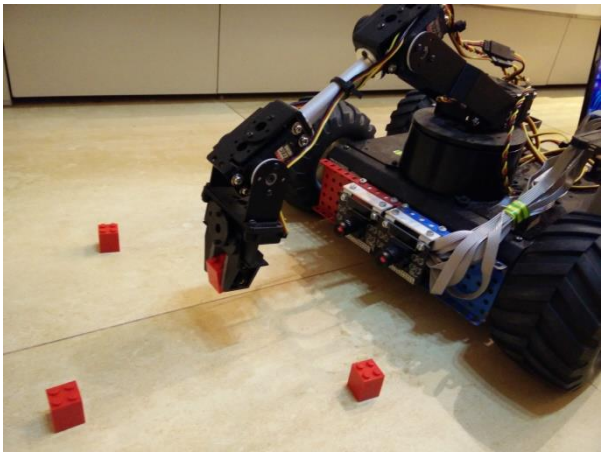
- `setLowHSV()` - nastaví dolnú hranicu farebného spektra pre detekciu podľa farby, pričom argumentom sú tri hodnoty (hue, saturation, value).
- `setHighHSV()` - nastaví hornú hranicu farebného spektra pre detekciu podľa farby, pričom argumentom sú tri hodnoty (hue, saturation, value).
- `setDimensions()` - dvomi hodnotami parametrov nastaví šírku a výšku hľadaného objektu.
- `setColorRed()` - nastaví detekované spektrum HSV na červenú farbu.
- `setColorBlue()` - nastaví detekované spektrum HSV na modrú farbu.
- `setColorGreen()` - nastaví detekované spektrum HSV na zelenú farbu.
- `setColorYellow()` - nastaví detekované spektrum HSV na žltú farbu.

4.3 Jednoduchý agent-robot s umelou inteligenciou

Demonštrácia funkcionality by mala byť predvedená na konkrétnom príklade použitia v praxi. Navrhnutými prípadmi použitia sú tri rôzne situácie, s ktorými sa má robot vysporiadať.

4.3.1 Demo 1 - zbieranie červených kociek

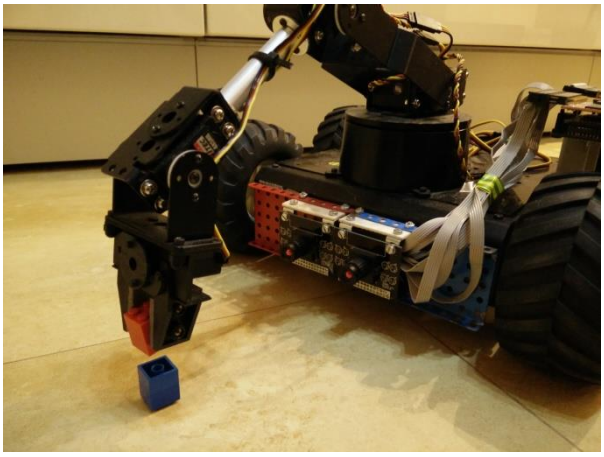
Rozpoznávaný objekt bude kocka o dĺžke hrany 2 cm červenej farby. Rozpoznávanie takéhoto objektu by malo prebiehať s minimálnou chybovosťou za rôznej intenzity osvetlenia. Robot sa umiestni do neznámeho priestoru a jednoduchá umelá inteligencia bude založená na princípe otáčania sa do momentu, kým robot v zornom poli nerozpozná hľadaný predmet. Následne sa začne pohybovať smerom k predmetu, pričom kontroluje, či sa stále objekt nachádza pred ním. V prípade, že dráha pohybu k rozpoznanému predmetu nie je priamočiara, robot upraví trajektóriu podľa potreby. Akonáhle sa predmet dostane do vopred definovaného pracovného priestoru robotickej ruky (priestor, v ktorom je robotická ruka schopná predmet uchopiť), dopočítajú sa za pomoci informácie o vzdialenosti (súradnica z) aj zvyšné dve súradnice x a y v priestore. Následne sa iniciuje proces presunu robotickej ruky smerom k predmetu a pomocou efektora dôjde k uchopeniu. Ďalším krokom je presun a umiestnenie predmetu na nákladný priestor robota, čiže na vopred definované miesto. Pohyb robotickej ruky musí byť plynulý a dostatočne jemný, aby dokázala jemne uchopiť aj ľahšie predmety a nedošlo k nechcenej kolízii s uchopovanými predmetmi. Robot opakuje toto správanie a neustále hľadá ďalšie kocky vo svojom okolí.



Obr. č. 15: Zbieranie červených kociek

4.3.2 Demo 2 - ukladanie predmetov na seba

Rozpoznávanými objektmi budú kocka o dĺžke hrany 2 cm červenej farby a rovnaká kocka modrej farby. Rovnako ako v predchádzajúcom príklade musí rozpoznávanie takéhoto objektu prebiehať s minimálnou chybovosťou za rôznej intenzity osvetlenia. Robot sa umiestni do neznámeho priestoru a jednoduchá umelá inteligencia bude založená na princípe otáčania sa do momentu, kým v zornom poli nerozpozná prvý hľadaný predmet. Následne sa začne pohybovať smerom k predmetu a v momente, keď sa predmet dostane do vopred definovaného pracovného priestoru robotickej ruky, dôjde k uchopeniu predmetu. Robot po sebe vykonanú akciu vizuálne skontroluje – ak sa predmet pred ním už nenachádza, znamená to, že je naozaj v uchopení efektora ramena. Následne prebieha lokalizácia druhého predmetu a pohyb k nemu. Robot zahájí presun uchopeného predmetu smerom k lokalizovanému predmetu a pokúsi sa položiť jeden na druhý. Určovanie polohy predmetov pred robotom je v tejto úlohe naozaj kritické, vzhľadom na to, že oba predmety majú rovnakú styčnú plochu, a teda ich umiestnenie na seba vyžaduje veľkú presnosť.

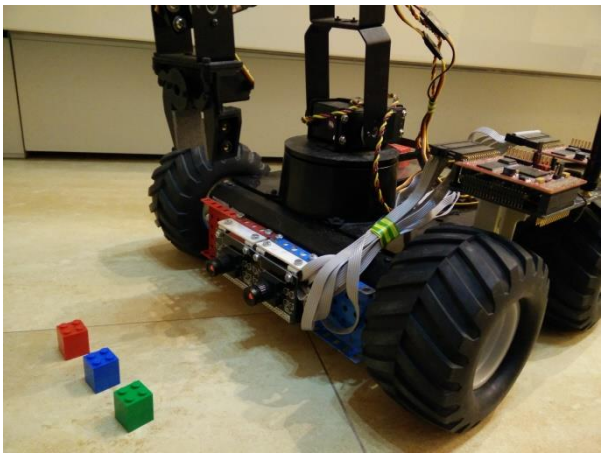


Obr. č. 16: Pokladanie červenej kocky na modrú

4.3.3 Demo 3 - preusporiadanie predmetov

Uvažujeme situáciu, v ktorej sa pred robotom nachádzajú tri rovnaké predmety rôznych farieb. Prvou časťou úlohy je rozpoznať všetky tri predmety na základe farby a určiť ich polohu v priestore. Predmety budú vopred umiestnené tak, aby boli v dosahu robotického ramena. Druhou časťou úlohy je využitie plánovania STRIPS popísaného v podkapitole 1.6. Stavom sveta je konkrétna konfigurácia (obsadenie pozícii predmetmi) a možnou akciou je

presun predmetu na voľnú pozíciu. Rozpoznaná poloha predmetov je počiatočným stavom sveta a koncový stav sveta je daný na vstupe. Pomocou prehľadávania stavového priestoru do šírky robot najskôr nájde riešenie (postupnosť akcií presunu kociek) a následne sa tieto akcie pokúsi vykonať. Po každej akcii sa svoje konanie snaží aj vizuálne skontrolovať, aby mal istotu, že predmet naozaj uchopil a presunul. Tri predmety pred ním určujú tri možné pozície a štvrtou (na začiatku voľnou) pozíciou je jeho nákladný priestor.



Obr. č. 17: Riešenie problému preusporiadania kociek

5. REALIZÁCIA A IMPLEMENTÁCIA

Kapitola popisuje implementačnú časť diplomovej práce, v ktorej sú obsiahnuté všetky riešenia problémov spojených s vývojom robotického systému slúžiaceho na autonómny pohyb a konanie robota. Aplikácia je naprogramovaná v jazyku C++ a je kompilovateľná takmer na akejkoľvek platforme, na ktorej sú OpenCV knižnice a knižnica Curl. Súčasťou zdrojového kódu sú aj XML súbory, v ktorých sú definované matice koeficientov skreslenia, matice kamier a tiež matica translácie/rotácie medzi kamerami (popísané v podkapitole 4.1). Tieto matice sme získali kalibráciou kamier, ktorú sme vykonali za pomoci návodu uverejneného na blogu M. Perisa [1]. Zdrojové kódy práce podliehajú open-source licencii a sú k dispozícii na url: http://dai.fmph.uniba.sk/projects/manipulators/.

5.1 Inicializácia objektov a komunikácie

Pre spustenie všetkých modulov je potrebné vytvoriť objekty vyššie spomínaných tried, inicializovať **pripojenie na sériový port** a tiež nastaviť rozlíšenie a kvalitu obrazu kamier. Predpokladom úspešného naštartovania je funkčné prepojenie počítača a základnej dosky cez sériový port a vytvorené WiFi Ad-hoc prepojenie medzi počítačom a doskou SVS. Inicializácia teda prebehne nasledovne:

```
StereoCam stereoCam = StereoCam();
stereoCam.initConnection();

Robot robot = Robot();
int fd = robot.initConnection(port);

RobotMotion robotMotion = RobotMotion();
robotMotion.initArmPosition(robot, fd);
```

Získavaniu obrazu z kamier musí predchádzať **WiFi Ad-hoc pripojenie** na dosku Surveyor SVS. Vzhľadom na to, že nastavenie a vytvorenie tohto spojenia nie je úplne triviálne, uvediem postup. V prvom rade prvé je potrebná modifikácia súboru `/etc/network/interfaces` a v ňom záznam týkajúci sa `wlan0` nasledovne:

```
iface wlan0 inet static
address 169.254.99.190
netmask 255.255.0.0
wireless-mode ad-hoc
wireless-essid SRV1
```

Táto modifikácia je potrebná, aby sa po vytvorení spojenia priradila správna IP adresa. Po uložení zmien odporúčam reštart systému. Pripojenie cez WiFi Ad-hoc vytvoríme postupnosťou nasledovných príkazov:

```
ifup wlan0
iw wlan0 set type ibss
ip link set wlan0 up
iw dev wlan0 ibss join SRV1 2412
```

V tomto štádiu by už malo byť spojenie aktívne. Adresa servera Surveyor SVS je 169.254.0.10, pričom ľavá kamera je dostupná na porte 10001 a pravá kamera na porte 10002. V metóde `StereoCam::initConnection()` sa využíva **knižnica Curl** na **HTTP requesty**. Príkladom použitia je nastavenie rozlíšenia kvality ľavej kamery na „c“ (640x480 px):

```
CURL *curl;
CURLcode res;

curl = curl_easy_init();
curl_easy_setopt(curl, CURLOPT_URL, "http://169.254.0.10:10001/robot.cgi?c");
curl_easy_setopt(curl, CURLOPT_FOLLOWLOCATION, NULL);
curl_easy_setopt(curl, CURLOPT_TIMEOUT, 1);
res = curl_easy_perform(curl);
curl_easy_cleanup(curl);
```

V nasledovnej tabuľke sú všetky možné príkazy pre kamerový modul:

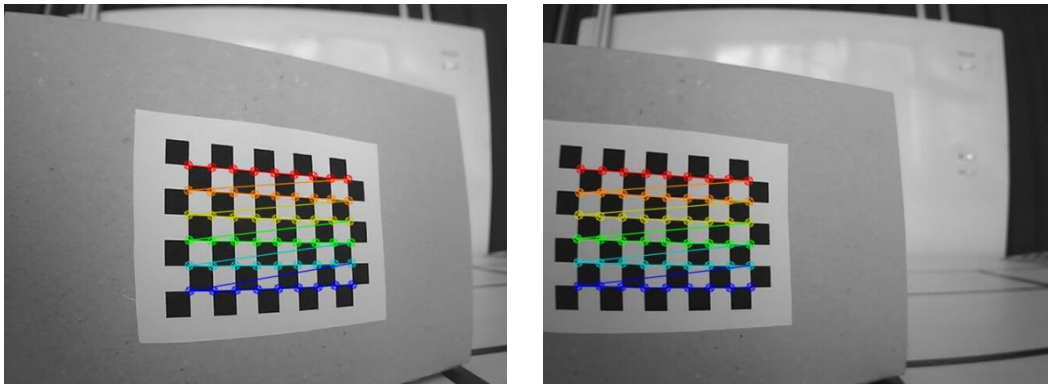
Command	Description
'a'	set capture resolution to 160x120
'b'	set capture resolution to 320x240
'c'	set capture resolution to 640x480

'd' or 'A'	set capture resolution to 1280x1024
'qx'	sets JPEG quality between 1-8 ('x' is an ASCII decimal character). 1 is highest, 8 is lowest

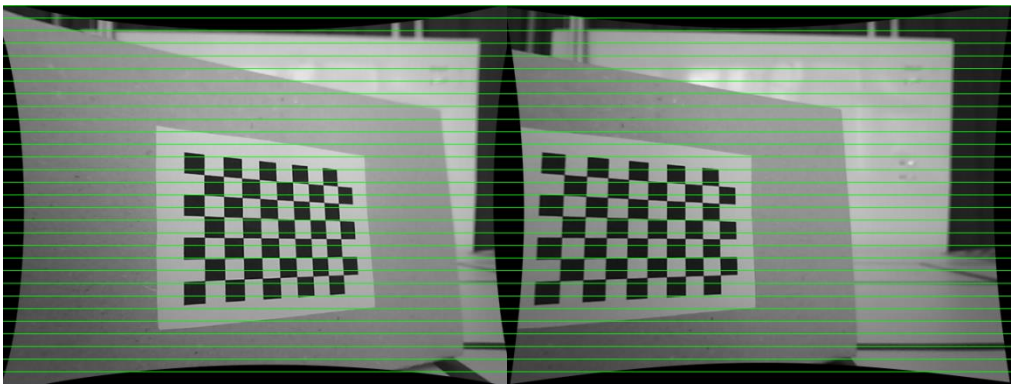
5.2 Spracovanie obrazu

5.2.1 Kalibrácia kamier

Skôr ako je možné použiť metódy stereovidenia poskytované OpenCV, je potrebné vykonať kalibráciu kamier. Kalibráciu je potrebné spraviť iba raz pre jednu konkrétnu sadu kamier (za predpokladu, že ich vzájomná poloha sa už nebude meniť). V danom prípade bolo zhotovených 14 rôznych párov záberov, pričom šachovnica bola rôzne natáčaná voči kamerám tak, aby jej vnútorné vrcholy bolo vidieť vždy v oboch záberoch. Veľkosť hrany štvorčeka bola 0,815 cm a šachovnica mala rozmery 10x7 štvorčekov. Na obrázku č. 18 je možné vidieť rozpoznanie vnútorných vrcholov šachovnice a aplikáciu výslednej rektifikácie na vybranej dvojici obrázkov (obr. č. 19).



Obr. č. 18: Vyznačenie rozpoznaných vrcholov šachovnice na ľavom a pravom zábere



Obr. č. 19: Aplikácia rektifikácie na dvojicu záberov

Na kalibráciu kamier pomocou metód OpenCV je vhodné použiť program Martina Perisa [1]. Výstupom programu sú kalibračné matice uložené v súboroch formátu XML, ktoré je potrebné skopírovať do priečinka /xml v koreňovom adresári projektu.

5.2.2 Načítanie obrázkov a matíc

Získané JPEG **obrázky z kamier načítame** metódou cv::imread() ako objekty typu cv::Mat, čo sú vlastne objekty reprezentujúce matice pixlov. Do rovnakých typov objektov načítame aj matice kamier (mx1.xml, my1.xml, mx2.xml, my2.xml, Q.xml), ktoré sme predtým získali kalibráciou kamier [1]. Obrázky načítavame ako cv::IMREAD_GRAYSCALE (pre stereovidenie) aj cv::IMREAD_COLOR (pre identifikáciu objektov):

```
cv::Mat img_l = cv::imread("left.jpg", cv::IMREAD_GRAYSCALE);
cv::Mat img_r = cv::imread("right.jpg", cv::IMREAD_GRAYSCALE);
```

Následne je potrebná aplikácia **rektifikácie** pomocou rektifikačných matíc:

```
cv::Mat img_l_rectified = cv::Mat::zeros(imageHeight, imageWidth, CV_8U);
cv::Mat img_r_rectified = cv::Mat::zeros(imageHeight, imageWidth, CV_8U);
cv::remap(img_l, img_l_rectified, mx1, my1, cv::INTER_LINEAR);
cv::remap(img_r, img_r_rectified, mx2, my2, cv::INTER_LINEAR);
```

5.2.3 Rozpoznávanie objektu

Rozpoznávanie objektu je vykonávané na základe jeho farby. Rozmedzie hľadaných farieb je definované vo farebnom modely HSV (hue-saturation-value). Hľadanie objektov spočíva v dvoch krokoch - **vytvorenie thresholded obrázku**, čo je obraz tvorený čiernymi a bielymi pixlami, pričom biele znamenajú rozpoznanú farbu a následné ohraničenie celých rozpoznaných objektov metódou Bounding Box. Vytvorenie thresholded obrázku:

```
cv::Mat imgHSV;
cv::cvtColor(sourceImg, imgHSV, cv::COLOR_BGR2HSV); //Convert the captured frame
from BGR to HSV

cv::Mat imgThresh;
```

```
cv::inRange(imgHSV, cv::Scalar(iLowH, iLowS, iLowV), cv::Scalar(iHighH, iHighS, iHighV), imgThresh); //Threshold the image

//morphological opening (removes small objects from the foreground)
cv::erode(imgThresh, imgThresh, cv::getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(5, 5)));
cv::dilate(imgThresh, imgThresh, cv::getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(5, 5)));

//morphological closing (removes small holes from the foreground)
cv::dilate(imgThresh, imgThresh, cv::getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(5, 5)));
cv::erode(imgThresh, imgThresh, cv::getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(5, 5)));
```

Nasleduje **hľadanie objektov** v thresholded obrázku a ich ohraničenie obdĺžnikmi typu cv::Rect metódou Bounding Box:

```
vector<vector<cv::Point> > contours;
vector<cv::Vec4i> hierarchy;
cv::findContours(imgThresh, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, cv::Point(0, 0));
vector<vector<cv::Point> > contours_poly(contours.size());

for (int i = 0; i < contours.size(); i++) {
    cv::approxPolyDP(cv::Mat(contours[i]), contours_poly[i], 3, true);
    cv::Rect rect = cv::boundingRect(cv::Mat(contours_poly[i]));
    resultBoundRectArray.push_back(rect); //add rectangle to result array
}
```

5.2.4 Kalkulácia 3D pozície

Na **vytvorenie hĺbkovej mapy** slúži objekt typu `cv::StereoBM`. Ako najvhodnejšie parametre som testovaním zistil nasledovné hodnoty:

```
cv::StereoBM bm = cv::StereoBM();
bm.state->preFilterCap = 28;
bm.state->preFilterSize = 13;
bm.state->SADWindowSize = 9;
```

```
bm.state->minDisparity = 25;
bm.state->numberOfDisparities = 256;
bm.state->textureThreshold = 314;
bm.state->uniquenessRatio = 2;
bm.state->speckleWindowSize = 100;
bm.state->speckleRange = 257;
```

Hĺbkovú mapu `vdisp` získame volaním:

```
cv::Mat disp = cv::Mat::zeros(imageHeight, imageWidth, CV_16S);
vdisp = cv::Mat::zeros(imageHeight, imageWidth, CV_8U);
bm(img_l_rectified, img_r_rectified, disp);
cv::normalize(disp, vdisp, 0, 255, CV_MINMAX, CV_8U);
```

Hĺbkovú mapu je možné zobraziť v novom okne (za predpokladu, že máme grafické rozhranie OS) volaním:

```
cv::imshow("disparity view", vdisp);
```

Aby bolo možné určiť presnú vzdialenosť každého pixlu hĺbkovej mapy, je potrebné previesť **rekonštrukciu bodov 2D hĺbkovej mapy do 3D priestoru**. Na tento úkon použijeme metódu `cv::reprojectImageTo3D()` nasledovne:

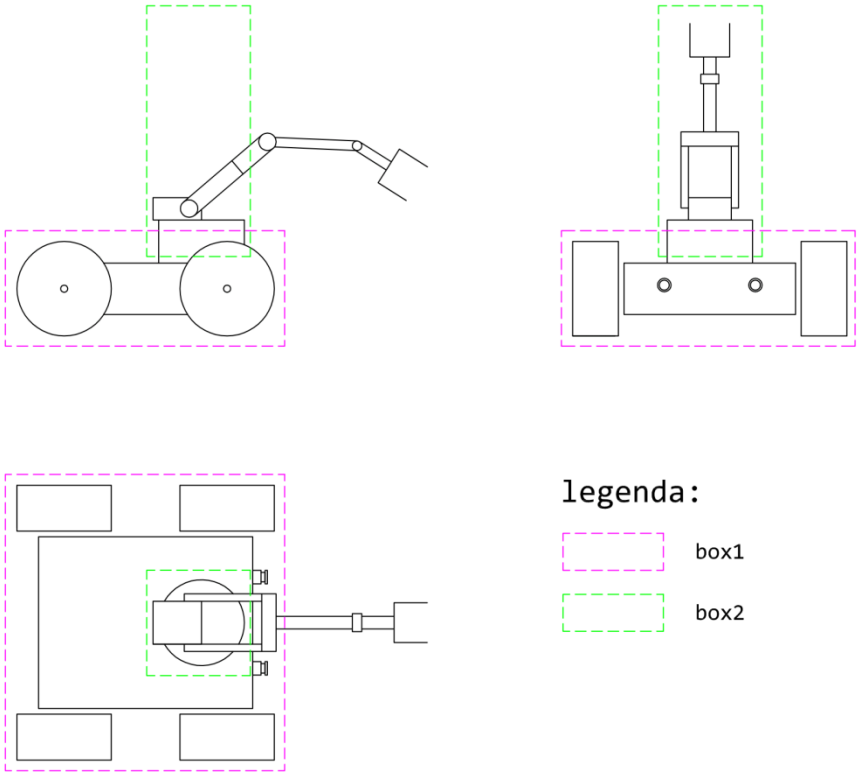
```
cv::Mat recons3D(vdisp.size(), CV_32FC3);
cv::reprojectImageTo3D(vdisp, recons3D, Q, false, CV_32F);
```

Pre každý bod matice `recons3D` následne vypočítame hodnoty *x*, *y*, *z* v priestore a uložíme ich do objektu typu `cv::Point`:

```
for (int y = yStart; y <= yEnd; y++) { //rows
    float* recons_ptr = recons3D.ptr<float>(y);
    for (int x = xStart; x <= xEnd; x++) { //cols
        cv::Point3d point;
        point.x = recons_ptr[3 * x];
        point.y = recons_ptr[3 * x + 1];
        point.z = recons_ptr[3 * x + 2];
        // do something with point
        ...
    }
}
```

5.3 Riadenie pohybu robotického ramena

Riadenie pohybu robotického ramena zabezpečuje objekt triedy `RobotArmMotion`. Pri programovaní pohybu robotického ramena vznikla otázka, ako sa pri jeho pohybe vyhnúť kolíziám (napríklad s telom robota). Prvé riešenie spočívalo v určení správneho poradia nastavovania servomotorov z jednej pozície do druhej. Toto riešenie sa však nedalo jednoducho unifikovať, nakoľko pohybu k objektu zodpovedalo iné poradie ako pohybu od objektu (zdvíhanie a presun objektu). Lepším riešením je definovanie pracovného priestoru robota pomocou vymedzenia priestoru, do ktorého sa nikdy nesmie efektor ramena dostať. Telo robota a základňa robotickej ruky sú reprezentované zjednodušeným spôsobom - pomocou dvoch kvádrov v priestore (`box1, box2`).



Obr. č. 20: Zjednodušená reprezentácia tela robota (box1) a robotickej ruky (box2)

Problém je teda zadefinovaný ako nájdenie vhodnej trajektórie z bodu *A* do bodu *B* tak, aby nepretínala `box1` ani `box2` pričom obmedzením je ešte polomer otáčania robotického ramena (cca 200°) a jeho dosah. Algoritmus bol navrhnutý a implementovaný nasledovne:

1. Uvažuj úsečku z bodu *A* do bodu *B*.
2. Zisti, či trajektória (úsečka) nepretína `box1` alebo `box2`.
3. Ak pretína, rozdeľ úsečku na dve polovice pomocou kontrolného bodu. Uprav pozíciu kontrolného bodu tak, aby neležal v `box1` ani v `box2`. Na novovzniknuté úsečky rekurzívne opakuj postup podľa bodu 1.
4. Ak nepretína, koniec.
5. Výsledkom je cesta z bodu *A* do bodu *B* reprezentovaná ako usporiadané pole bodov.

5.4 Použitie knižnice v praxi

Ako príklady reálneho použitia sú naprogramované tri rôzne správania robota podľa návrhu popísaného v podkapitole 4.3, ktoré sú zároveň akousi ukážkou použitia knižnice.

5.4.1 Implementácia dema 1 – zbieranie červených kociek

Kvôli demonštrácii funkčnosti systému bolo naprogramované jednoduché správanie robota. Jeho úlohou je pozbierať všetky červené kocky v jeho okolí (viď návrh v oddiely 4.2.1). Nasledovný pseudokód popisuje logiku správania robota:

```
while (true) {
	percept();
	if (je objekt v zabere) {
		if (objekt je prilis daleko) {
			moveFWD();
		} else if (objekt je prilis blizko) {
			moveBWD();
		} else {
			//objekt je v dobrej vzdialenosti
			if (objekt je prilis napravo) {
				turnRight();
			} else if (objekt je prilis nalavo) {
				turnLeft();
			} else {
				//objekt je v dosahu robotickeho ramena
```

```
                        pickUpObject();
                    }
                }
        } else {
                //objekt nie je v zabere
                turnRight();
        }
}
```

Celé správanie robota je naprogramované ako nekonečný cyklus, pričom v každej iterácii robot najskôr prijme vnemy z prostredia, spracuje ich (volanie metódy `percept()`) a následne sa rozhoduje akú akciu vykoná. Ak rozpozná objekt, tak sa k nemu snaží dostať bližšie a zodvihnúť ho, inak sa točí do strany.

5.4.2 Implementácia dema 2 – ukladanie predmetov na seba

Podobne ako v predchádzajúcom príklade, správanie robota je naprogramované v nekonečnom cykle no s tým rozdielom, že máme zadefinované dva rôzne objekty – červenú a modrú kocku:

```
//red lego cube
MyObject redObject = MyObject();
redObject.setColorRed();
//blue lego cube
MyObject blueObject = MyObject();
blueObject.setColorBlue();
```

Objekt triedy `MyObject` je posielaný ako parameter do funkcie `StereoCam.getObjectPositionXYZ()` vracajúcej polohu predmetu (`cv:Point3i`):

```
cv::Point3i objMidPoint = stereoCam.getObjectPositionXYZ(redObject);
cout << "redObject position is: " << objMidPoint << endl;
```

5.4.3 Implementácia dema 3 – preusporiadanie predmetov

Prvou časťou úlohy je identifikácia predmetov (troch rôznych kociek) podobným spôsobom ako v predchádzajúcom príklade (viď oddiel 5.4.2). Určenie poradia je zároveň určením počiatočného stavu sveta. Druhou časťou je nájdenie riešenia s využitím plánovania, čím dostávame postupnosť akcií, ktoré je potrebné vykonať, aby sme dostali želaný koncový stav sveta. Poslednou časťou je vykonanie akcií. Pseudokód dema 3 je nasledovný:

```
WorldState startWS = detectObjects();
Actions[] acts = solveProblem();
for(action in acts){
	execute(action);
}
```

Reálny kód je samozrejme o niečo zložitejší. Napríklad v časti `execute(action)` musí robot rozpoznať objekt, zdvihnúť objekt, overiť úspešnosť zdvihnutia, presunúť objekt a nakoniec ho položiť na novú pozíciu. Riešenie problému plánovania je naprogramované univerzálnym spôsobom, takže problém môže byť aj oveľa väčších rozmerov a použitý môže byť aj na riešenie iných problémov v iných projektoch.

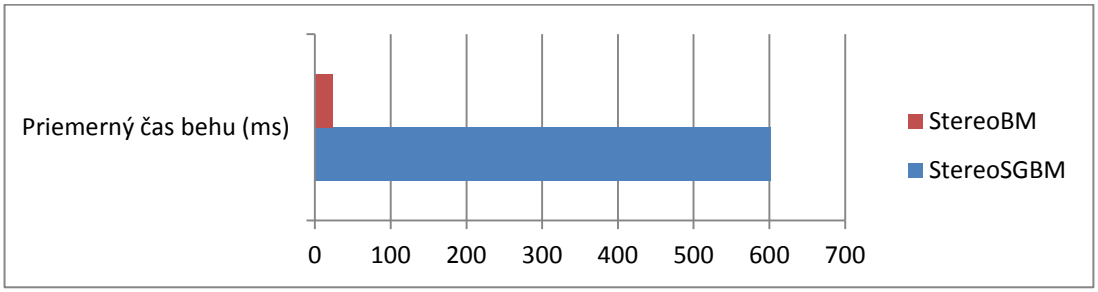
6. EXPERIMENTÁLNE NASADENIE SYSTÉMU

6.1 Porovnanie algoritmov StereoBM a StereoSGBM

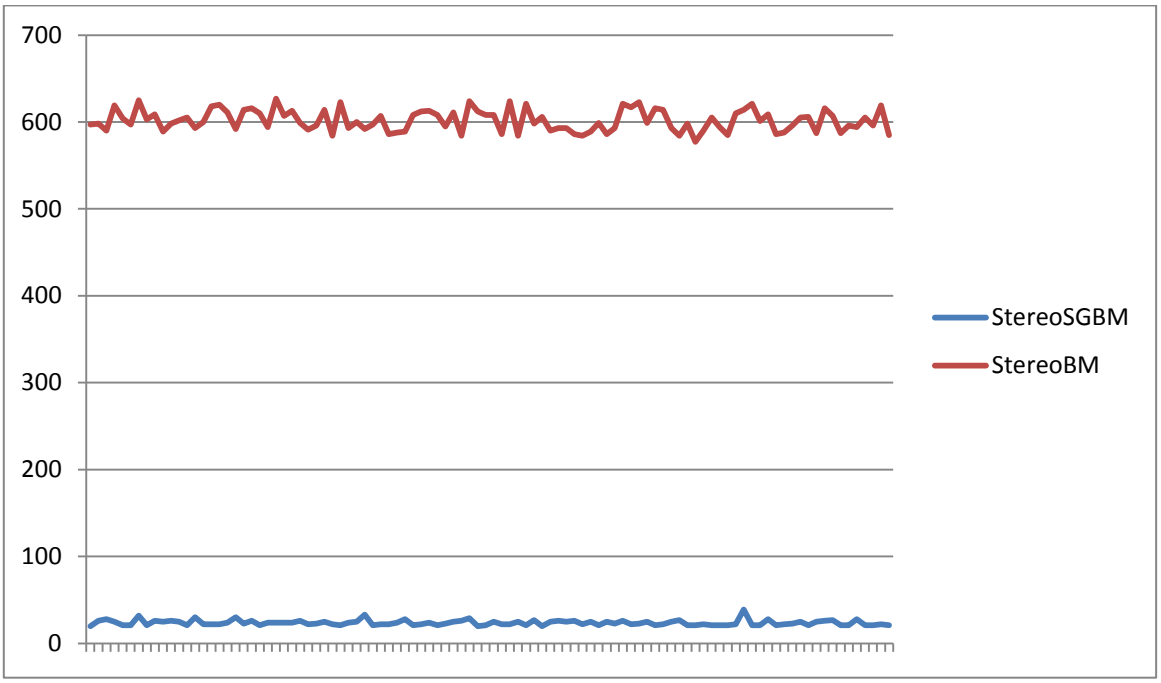
V tejto práci boli porovnávané dva algoritmy stereovidenia implementované v OpenCV. Algoritmy dostávali na vstupe rovnaké parametre. Podľa dokumentácie OpenCV, StereoSGBM algoritmus je akousi nadstavbou nad StereoBM, ktorá má zlepšiť kvalitu hĺbkovej mapy za cenu dlhšieho času potrebného na výpočet. Hľadanie najvhodnejších parametrov algoritmov bola časovo náročná operácia, keďže žiadne priame riešenie prakticky neexistuje. Dokumentácia OpenCV a tiež návod Martina Perisa na jeho blogu [1] však poskytli cenné rady, vďaka ktorým sa podarilo nájsť vhodné parametre:

```
int preFilterSize = 13;
int preFilterCap = 28;
int SADWindowSize = 9;
int minDisparity = 25;
int numberOfDisparities = 256;
int textureThreshold = 314;
int uniquenessRatio = 2;
int speckleWindowSize = 100;
int speckleRange = 257;
int P1 = 8 * 1 * SADWindowSize * SADWindowSize;
int P2 = 32 * 1 * SADWindowSize * SADWindowSize;
int disp12MaxDiff = 1;
bool fullDP = false;
```

Meraním času výpočtu po 100 pokusoch sa zistilo, že čas behu algoritmu StereoBM je omnoho menší ako StereoSGBM:



Obr. č. 21: Porovnanie priemerného času behu algoritmov StereoBM a StereoSGBM. Priemerný čas behu algoritmu StereoBM bol 23,77 ms a StereoSGBM bol 601,69 ms.



Obr. č. 22: Porovnanie času behu algoritmov StereoBM a StereoSGBM (100 meraní)

Avšak po kvalitatívnej stránke, čo je možné vidieť z obrázku č. 23, je hĺbková mapa vytvorená algoritmom StereoSGBM podstatne lepšia ako StereoBM:

Obr. č. 23: Zhora nadol: ľavý záber, výstup StereoBM, výstup StereoSGBM

Merania časov výpočtu jednotlivých častí kódu ukázali, že paradoxne najdlhšie netrvá výpočet hĺbkovej mapy, ale prenos záberov z kamier cez WiFi Ad-hoc, a to v priemere 4,35 sekundy. Z toho dôvodu, a tiež z dôvodu značne vyššej kvality hĺbkovej mapy, som sa rozhodol využiť algoritmus StereoSGBM, keďže sa ukázal ako vhodnejší pre použitie na robotovi Janko Hraško (pri popísanej hardvérovej konfigurácii).

6.2 Testovanie v reálnom prostredí

Nasadenie v reálnom rôznorodom prostredí možno označiť za pomerne úspešné, no pre použitie mimo prostredia laboratória by bolo vhodnejšie použitie inej metódy detekcie objektov než na základe farby. Pri testovaní sa tiež ukázalo, že rozpoznávanie objektu na základe farby fungovalo podstatne spoľahlivejšie v prípade, že objekt bol matnej farby (bez odleskov). Ak by hľadaný objekt mal jemnú textúru (napríklad mriežku podobnej farby), výsledky stereovidenia by to ešte mierne zlepšilo.

Prezentácia robota Janko Hraško bola súčasťou sprievodného programu Univerzitnej Regaty 2015. Systém implementovaný na robotovi sa ukázal ako stabilný a schopný riešenia a vykonávania rôznorodých úloh.

Obr. č. 24: Prezentácia robota počas sprievodného programu na Univerzitnej Regate 2015

ZÁVER

Výsledkom mojej práce je plne funkčný robot, ktorý je schopný vykonávať úlohy potrebné pre výskum v oblasti umelej inteligencie a kognitívnych vied. Robot dokáže rozpoznávať predmety pomocou stereovidenia, je schopný pohybovať sa v priestore a pomocou robotického ramena vykonávať rôzne akcie.

Súčasťou práce bolo podrobné zoznámenie sa s existujúcou hardvérovou platformou a jej významné zdokonalenie, implementácia stereovidenia a tiež vyriešenie problému plánovania trajektórie robotického ramena. V ukážkovej aplikácii je implementovaný všeobecný plánovač, ktorý umožňuje priame nasadenie robota na skúmanie a porovnávanie plánovacích algoritmov umelej inteligencie v nadväzujúcich prácach.

Výsledky tejto práce sú ďalším stupňom vývoja, ktorý môže smerovať k zlepšeniu kvality rozpoznávania priestoru pomocou stereokamier a tiež k zdokonaleniu výpočtov inverznej kinematiky. V prípade výmeny robotického ramena za zložitejšie (viac stupňov voľnosti) by nebolo možné využiť súčasnú implementáciu inverznej kinematiky. V takom prípade by bolo potrebné použiť niektorý z numerických alebo heuristických prístupov [7]. Zdokonalenie stereovidenia by mohlo spočívať v zmenšení vzájomnej vzdialenosti kamier a v ich dokonalejšej kalibrácii. Spresnenie navigácie robota v priestore by bolo možné dosiahnuť pomocou kombinácie obrazu a údajov zo senzorov na meranie vzdialenosti.

ZOZNAM POUŽITEJ LITERATÚRY

[1] Peris M. (2001). OpenCV: Stereo camera calibration.
http://blog.martinperis.com/2011/01/opencv-stereo-camera-calibration.html

[2] Peris M. (2001). OpenCV: Stereo matching.
http://blog.martinperis.com/2011/08/opencv-stereo-matching.html

[3] Peris M. (2012). 3D reconstruction with OpenCV and Point Cloud library
http://blog.martinperis.com/2012/01/3d-reconstruction-with-opencv-and-point.html

[4] Dröppelmann S., Hueting M., Latour S., van der Veen M. (2010). Stereo Vision using the OpenCV library.
http://tjpstereovision.googlecode.com/hg-history/551f9b6e2e9549337e7c26b4bac6a9a69a6c509c/doc/verslag.pdf

[5] Eifler M. (2003). Stereo videnie (vnímanie hĺbky). Projekt na predmet Kognitívna Veda, FMFI UK, Bratislava.
http://people.ksp.sk/~misof/skola/!to%20process/Kognitivna%20veda/MEifler.pdf

[6] Haladová Z., Kučerová J. Stereovidenie. Cvičenia z počítačového videnia. FMFI UK, Bratislava.
https://dai.fmph.uniba.sk/upload/a/ad/MATLAB_stereovidenie.pdf

[7] Pukančík P. (2012). Riadiaci systém s inverznou kinematikou pre mobilné robotické rameno. Bakalárska práca, FMFI UK, Bratislava.

[8] Monasse P., Morel J.-M., Tang Z. (2010). Three-step Image Rectification. British Machine Vision Conference, Aberystwyth, UK.
http://imagine.enpc.fr/publications/papers/BMVC10.pdf

[9] C. Arkin R. (2000). Behavior Based Robotics. The MIT Press.

[10] Brooks R. (1999). Cambrian Intelligence: The early history of the new AI. The MIT Press.

[11] Busquets D. (2003). A multiagent approach to qualitative navigation in robotics. Doctoral thesis, UNIVERSITAT POLITECNICA DE CATALUNYA.
http://eia.udg.es/~busquets/thesis/thesis.pdf

[12] Kolmogorov V., Monasse P., Tan P. (2014). Kolmogorov and Zabih's Graph Cuts Stereo Matching Algorithm. Image Processing Online 4 (2014), pp. 220-251.
http://dx.doi.org/10.5201/ipol.2014.97

[13] Olson E., Hao M. (2001). An Introduction to Stereo Vision and Disparity Computation. Final project for 6.344 Digital Image Processing Course, MIT. http://www.ravenousbirds.com/eolson/papers/stereovision/stereovisionpaper.pdf

[14] Gerig G. (2012). Image Rectification (Stereo). Lecture of Image Processing. University of Utah, Salt Lake City. http://www.sci.utah.edu/~gerig/CS6320-S2013/Materials/CS6320-CV-F2012-Rectification.pdf

[15] Fikes R., Nillson N. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Standfort Research Institute, California. http://ai.stanford.edu/~nilsson/OnlinePubs-Nils/PublishedPapers/strips.pdf