UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY FIZYKY A INFORMATIKY

999d51cb-c7ab-4d0f-a2f5-8667e886140a

BALLBOT

2011

Šimon Miklušičák

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Šimon Miklušičák
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.9. aplikovaná informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský

Názov: Ballbot

Cieľ: Analyzovať spôsoby pohybu robotov s guľovou morfológiou. Navrhnúť a v simulácii overiť riadiaci a navigačný systém pre prototyp takéhoto robota. V prípade možnosti overiť na reálnom prototype.

Vedúci: Mgr. Pavel Petrovič, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 11.11.2009

Dátum schválenia: 12.10.2010

doc. RNDr. Mária Markošová, PhD.
garant študijného programu

Šimon Miklušičák
študent

Pavel Petrovič
vedúci

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY FIZYKY A INFORMATIKY

BALLBOT

Bakalárka práca

Študijný program : Aplikovaná informatika
Študijný odbor: 9.2.9 Aplikovaná informatika
Školiace pracovisko: Katedra aplikovanej informatiky
Školiteľ: Mgr. Pavel Petrovič, PhD.

Bratislava 2011

Šimon Miklušičák

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím citovaných zdrojov.

. . . . . .

Ďakujem Mgr. Pavlovi Petrovičovi, PhD. za podnetné návrhy a odborné vedenie pri písaní tejto bakalárskej práce.

Ďakujem svojej rodine a priateľom za pomoc a morálnu podporu

Abstrakt

Zostrojiť moderného, v dostatočnej miere samostatného mobilného robota s nadpriemernými manévrovacími schopnosťami ostáva pre návrhárov, konštruktérov a technologických zanietencov po celom svete i dnes veľká výzva.

Problémy s návrhom a realizáciou konštrukčných riešení a neraz i chýbajúce prostriedky, môžu často znamenať priveľké sústo i pre nadšených milovníkov robotiky. V tejto práci sa pokúsime navrhnúť netradičný typ guľového robota s plne funkčným systémom riadenia, ktorý bude možné v budúcnosti implementovať do reálneho protoypu, vo virtuálnom svete fyzikálnej simulácie a využiť tak všetky výhody plynúce z takéhoto riešenia. Počnúc vysokou mierou stability cez nadpriemernú rýchlosť výpočtov, bez technických porúch a problémov s komunikáciou a prakticky zadarmo. Otázkou zostáva, či budeme schopní dostatočne komplexne abstraktne premýšľať a analyzovať možné problémy.

Výsledok tejto práce predpokladá fungujúci model riadenia guľového robota a zároveň odpoveď na otázku, či možno bežne dostupné nástroje, aproximačne simulujúce dynamiku tuhých telies použiť ako relevantný nástroj pri návrhu riadiacih systémov robota.

Obsah

Kapitola 1

Úvod

1.1 Úvod

Napriek technologickému pokroku a výraznému rastu výpočtovej sily integrovaných obvodov a teda i schopnosti robotov analyzovať a riešiť stále zložitejšie úlohy, je i dnes pomerne náročné vyrobiť iného, než kolesového robota s nadpriemernými manévrovacími schopnosťami.
Inak povedané, robota schopného pohybovať sa vzhľadom na požiadavku okamžite a ľubovoľným smerom, aspoň približne konštantnou rýchlosťou a to bez hrozby zo straty stability a následného poškodenia.

Vďaka pevnej sférickej schránke, nízkej hmotnosti a výborným manévrovacím schopnostiam sa guľové roboty radia práve do tejto úzkej skupiny robotov. Hlavným problémom pri ich konštrukcii však ostáva návrh a implementácia vhodného, aspoň približne presného modelu riadenia, ktorý sa vždy líši v závislosti od konkrétnej implementácie.

V tejto práci sa pokúsim navrhnúť všeobecný model riadenia pre vybraný druh guľového robota, nezávislý na jeho veľkosti, či hmotnosti. Overiť funkčnosť navrhnutých algoritmov s využitím softvéru simulujúceho dynamiku tuhých telies a vytvoriť tak podklady na vytvorenie riadiaceho systému pre skutočný prototyp.
Vzhľadom na neštandardný typ robota, bude možné za úspech pokladať návrh a implementovanie aspoň základných algoritmov riadenia.

1.2 Cieľ práce

Cieľom práce je navrhnúť a v počítačovej simulácii implementovať model riadenia pre zvolený typ guľového robota. Dosiahnutie tohto cieľa je podmienené naplnením troch podcieľov:

1. Analýza spôsobov pohonu guľových robotov a ich porovnanie
2. Návrh konkrétneho prístupu a modelu riadenia
3. Implementácia navrhnutého riešenia v počítačovej simulácii

Analýza pozostáva predovšetkým z porovnania dôležitých vlastností rôznych typov guľových robotov. Jej cieľom je nájsť vhodné, čiže v univerzitných podmienkach implementovateľné riešenie.
Analýza si nekladie za cieľ podrobnú technickú špecifikáciu konštrukcií, alebo systémov riadenia v konkrétnych prípadoch. Je zameraná predovšetkým na porovnanie kľúčových vlastností, ktoré mali vplyv na výber a návrh konečného riešenia.
Súčasťou analýzy je načrtnutie základných princípov dynamiky tuhých telies a jej simulovania pomocou počítača.

Druhá kapitola sa postupne venuje voľbe vhodného typu robota na základe vybraných kritérií - predovšetkým na základe typu pohonu, ktorý vo veľkej miere ovplyvňuje kľúčové vlastnosti guľových robotov.
Definovaniu základných konštrukčných požiadaviek a návrhu teoretického modelu vybraného typu robota. Návrhu dôležitých súčasti počítačovej aplikácie, predovšetkým riadiaceho systému. Jednotlivé časti vhodne dopĺňa výber a opis nástrojov použitých pri realizácii projektu.

Tretia kapitola sa zaoberá predovšetkým implementáciou navrhnutého riešenia vo fyzikálnej simulácii. Podrobne opisuje dôležité algoritmy, prepojenie jednotlivých častí aplikácie a postupy použité v konkrétnych prípadoch. Ponúka pohľad na problémy spojené s vývojom aplikácie a implementovaním navrhnutých algoritmov.

Kapitola 2

Analýza

2.1 Guľový robot

Guľový robot je typ mobilného robota, charakteristický zväčša pevným vonkajším obalom tvaru gule, ktorý mu umožňuje v plnej miere využívať výhody valivého pohybu[1]. Tento obal, nazývaný tiež sférický exoskelet ho predurčuje k nadštandardným manévrovacím schopnostiam, vďaka ktorým preniká do rôznych sfér vedy a výskumu.

> *Ponúka značné výhody oproti viackolesovým - staticky stabilným robotom, vďaka výbornej manévrovateľnosti, dobrej dynamickej stabilite a nízkej miere odporu.* [Oum09]

Mnohé typy sú schopné pohybu v ľubovoľnom smere a to bez ohľadu na predchádzajúcu polohu a otočenie, čo je z hľadiska manévrovacích schopností obrovská výhoda v porovnaní s inými typmi mobilných robotov. Bývajú pomerne ľahké, často prenosné a vonkajšia, dobre uzatvoriteľná schránka poskytuje dostatočnú ochranu pre aktívne mechanizmy a senzory vo vnútri robota.
Z týchto dôvodov sú vhodné aj na použitie v teréne a v nepriaznivých podmienkach, akými sú prašné, vlhké, či dokonca teplotne nepriaznivé prostredie. Existujú funkčné prototypy, schopné pohybu po vodnej hladine. Vďaka ich všestrannosti a dobrým pohybovým schopnostiam sa o guľových robotoch často hovorí aj v súvislosti s vesmírnym, predovšetkým medziplanetárnym výskumom.

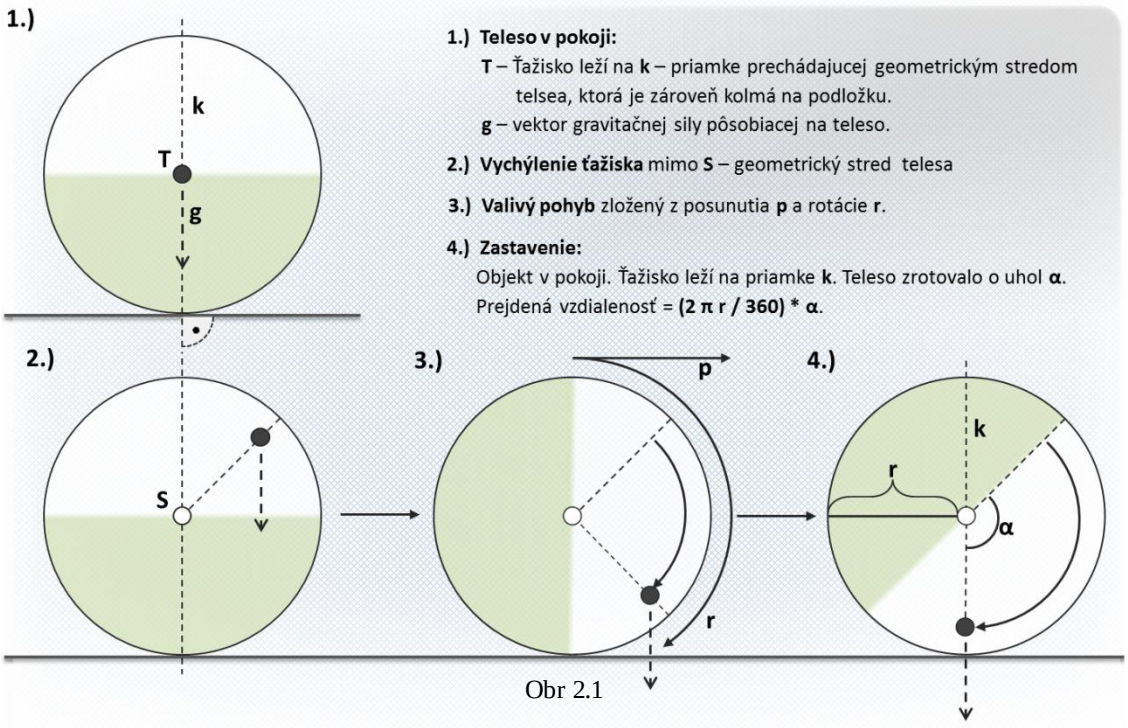
---

1 - Valivý pohyb je pohyb, zložený z rotácie a posunu

2.2 Základné princípy pohonu a riadenia

Kľúčovým pri vývoji guľového robota je samozrejme model riadenia. Ten sa vždy líši v závislosti od konkrétnej implementácie. Vzhľadom na konštrukčné odlišnosti jednotlivých typov a fakt, že sú tieto systémy navrhované v prevažnej miere pre jeden konkrétny prototyp, neexistuje žiadne univerzálne riešenie.
Základný princíp pohonu a riadenia však ostáva aj napriek iným odlišnostiam spoločný pre veľkú väčšinu týchto robotov. Jedná sa *riadené presúvanie hmoty vo vnútri guľového tela*.

Predstavte si guľové teleso položené na vodorovnej podložke. Pokiaľ sa ťažisko telesa nachádza na myslenej kolmici s podložkou, ktorá prechádza geometrickým stredom telesa a dotykovým bodom s podložkou, teleso ostáva v pokoji.
Ak premiestnime časť hmotnosti telesa mimo túto kolmicu, guľa sa dostane do nerovnovážneho stavu a pôsobením gravitačnej sily dochádza k valivému pohybu v smere vychýlenia ťažiska. Keď sa teleso vráti do rovnovážnej polohy, pohyb ustane.



Obr 2.1

Tento jednoduchý princíp poskytuje v praxi dostatočne veľký priestor pre množstvo rozličných riešení, no stanovuje im zároveň niektoré podstatné limity. Maximálny uhol stúpania pre guľové roboty sa zvykne pohybovať niekde na hranici

dvadsiatich stupňov. Rovnako maximálna rýchlosť pohybu robota pre konkrétnu implementáciu vždy závisí od jeho veľkosti a hmotnosti.

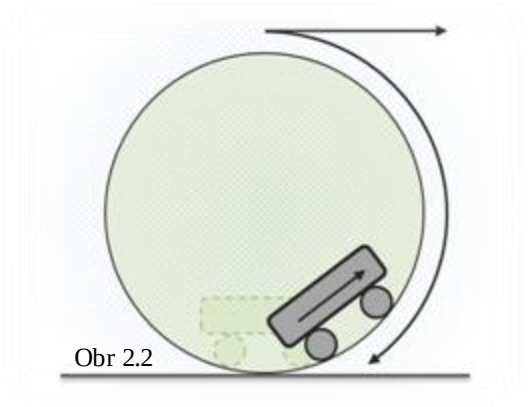
2.3 Typy guľových robotov

Hoci existuje mnoho rozdielnych implementácií, navrhnutých prevažne podľa individuálnych požiadaviek konkrétnych projektov, môžme guľové roboty rozdeliť do niekoľkých základných skupín, na základe použitých technológií a konštrukčných prístupov.

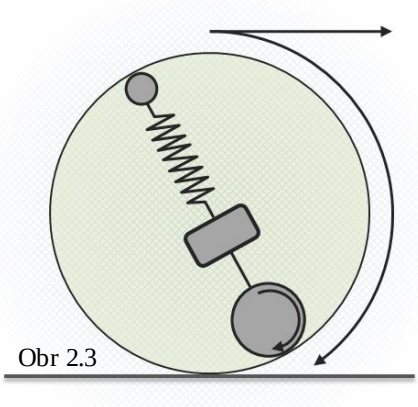
> *Nie je ľahké porovnávať vlastnosti jednotlivých typov robotov, keďže žiaden z nich nebol optimalizovaný na používanie. To znamená, že výkonnostné faktory ako rýchlosť, prekonávanie prekážok, jazda po naklonenej plošine, či citlivý systém ovládania, ktoré sú kľúčové pre nezávislý pohyb, nie sú doladené na maximum.* [Oum09]

2.3.1 Vnútorná riadiaca jednotka

Jedná sa o skupinu robotov, o ktorých pohyb sa stará samostatná mobilná jednotka, nazývaná tiež IDU[2]. Zväčša ide o troj, alebo štvorkolesový podvozok voľne umiestnený vo vnútri robota. Ovládaním IDU, ktorá sa pohybuje po vnútornej stene guľovej schránky sa dostáva robot do pohybu. Toto riešenie sa z hľadiska konštrukcie radí k tým najjednoduchším. Pozri Obr. 2.2



Obr 2.2



Obr 2.3

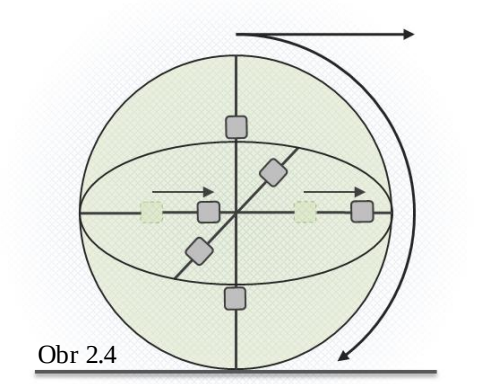
Špeciálnym prípadom IDU je tzv. *Centrálny pružinový článok*[2]. Jeho základ tvorí jedno aktívne - motormi ovládané a jedno pasívne koleso. Sú prepojené pružinou, ktorá ich tlačí vždy na protiľahlé bodoch plášťa. Pozri Obr. 2.3

---

2 - Z anglického Inside Driving Unit

2.3.2 Presúvanie závaží po osiach;

Systém pohonu a riadenia, spočívajúci v precíznom presúvaní závaží po sústave telesových uhlopriečkach. Spoločná výslednica gravitačných síl pôsobiacich na závažia pri dobre vyváženom prototype jednoznačne určuje polohu vychýleného ťažiska a teda momentálny, či budúci smer pohybu robota.



Obr 2.4

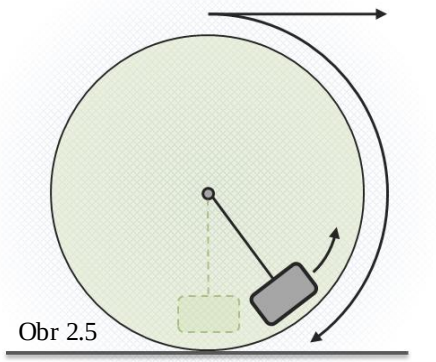
Na príklade z obrázku 2.4 vidieť model s troma navzájom kolmými osami. Tento zaujímavý koncept vytvára vo vnútri robota trojrozmernú súradnicovú sústavu, ktorá uľahčuje potrebné výpočty. Pri práci s takýmto modelom riadenia, je dôležité uvedomiť si, že sa celá súradnicová sústava otáča spoločne s robotom a je nevyhnutné zohľadňovať túto rotáciu pri riadení robota. Aby to vôbec bolo možné, musí robot navyše disponovať vhodnými senzormi, schopnými v reálnom čase poskytovať informácie o otočení v priestore. Tento precízny prístup ponúka pri správnom prevedení výborné manévrovacie schopnosti a citlivé ovládanie, avšak za cenu konštrukčne i programátorsky pomerne náročného riešenia.

2.3.3 Kyvadlový systém (pendulum)

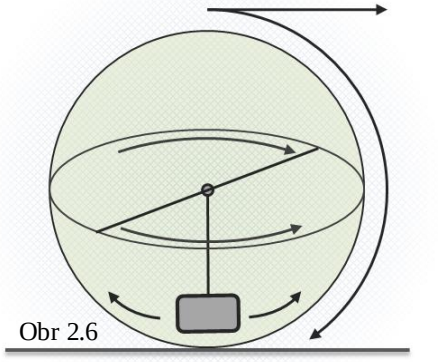
Ide pravdepodobne o najrozšírenejší a najobľúbenejší typ guľového robota vôbec. Existuje dokonca komerčne vyrábaný model, využívaný ako súčasť bezpečnostných systémov. Za svoju obľubu vďačí pomerne jednoduchému a ľahko pochopiteľnému modelu riadenia. Na centrálnej osi zavesené kyvadlo sa pomocou motorov vychyľuje zvoleným smerom, čím mení polohu ťažiska.

Bežne rozlišujeme dva rozdielne prístupy k systému kyvadla. Prvý vidieť na obrázku 2.5. Ide o kyvadlo vychyľované v dvoch na seba kolmých smeroch. Jeden motor sa stará o pohyb dopredu a dozadu. Druhý vychyľuje kyvadlo do strán, čím riadi

smer pohybu. Jedná sa o veľmi efektívny model z hľadiska zrýchlenia, maximálnej rýchlosti a plynulosti pohybu. Robot je schopný bez problémov sa pohybovať, zastaviť a opätovne sa pohnúť po naklonenej ploche.Tento kolesu podobný pohyb však prináša i svoje nedostatky. Menovite neschopnosť zahájiť pohyb ľubovoľným smerom.



Obr 2.5

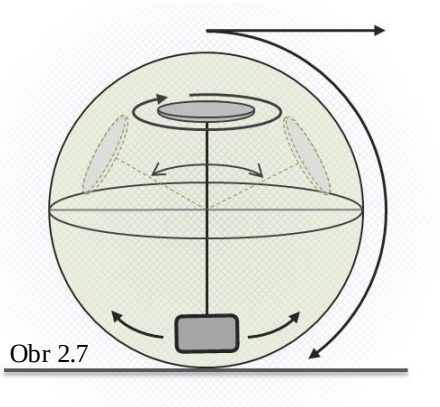


Obr 2.6

Druhý prístup, obrázok 2.6, rieši práve spomínaný problém. Rovnako ako v prvom prípade sa kyvadlo vychyľuje dopredu a dozadu, no o zmenu smeru sa stará špeciálny mechanizmus, ktorý otáča centrálnu os a teda celý mechanizmus kyvadla po vnútornom obvode exoskeletu.Vzhľadom na zložitosť mechanizmu je však robot konštrukčne náročnejší ako robot s jednoduchým kyvadlom.

2.3.4 Gyroskopická stabilizácia

Špeciálny druh robota, ktorého exoskelet nemusí byť guľa. Pretože sa o jeho stabilizáciu stará zabudovaný gyroskop, vonkajší obal môže mať tvar kolesa, či disku. Na pohyb dopredu a dozadu môže používať napríklad systém kyvadla, no o zmenu smeru sa starajú servomotory, ktoré podľa potreby vychyľujú gyroskop, čím dochádza k zmene náklonu robota a nálslednej zmene smeru pohybu. Obr. 2.7.



Obr 2.7

Zoznam vybraných implementácii a reálnych prototypov je obsiahnutý v dodatku **A**

2.4 Simulácia dynamiky tuhých telies

Nasleduje stručný pohľad na dynamiku tuhých telies, problematiku implementácie tejto dynamiky v počítačovej simulácii a relevantnosť takejto simulácie vo vzťahu k realite.

2.4.1 Základy dynamiky tuhých telies

Cieľom tejto časti nie je podrobne sa zaoberať teóriou mechaniky a dynamiky tuhých telies. Pre jasné a zrozumiteľné pochopenie zvyšnej časti práce je však dôležité aspoň čiastočne vysvetliť základné pojmy.

Dynamika: je časť klasickej mechaniky, ktorá sa zaoberá vplyvom pôsobenia síl na pohyb telies. Opisuje zmeny pohybového stavu telies. Jej základy položil Isaac Newton v 18. storočí, keď sformuloval tzv. Newtonove pohybové zákony

Tuhé teleso: je ideálne teleso z pevnej látky, ktoré pri pôsobením síl nemení svoj tvar - nedeformuje sa. Tieto sily naň majú výlučne pohybový účinok. Neberie sa do úvahy časticovú štruktúru látok. Pri uvažovaní o takomto telese a jeho vlastnostiach si treba uvedomiť, že dokonale tuhé telesá v reálnom svete neexistujú. Uvažujeme o nich výlučne vrámci fyzikálnych výpočtov. Tento prístup nám dovoľuje pomocou fundamentálnych princípov dynamiky počítať a simulovať správanie tuhých telies pri pôsobení síl. Tuhé telesá sa používa k teoretickému skúmaniu čisto pohybových účinkov síl na teleso v prípadoch, keď ho nemožno nahradiť hmotným bodom, pretože jeho rozmery a tvar nie sú zanedbateľné.

Sila a skladanie síl: sila je vektorová veličina, ktorá vyjadruje mieru vzájomného pôsobenia telies alebo polí. Skladanie síl je postup, ktorým sa z jednotlivých síl pôsobiacich na teleso určí výsledná sila (*výslednica*). Účinok všetkých síl je potom rovnaký ako účinok výslednice.

Newtonove pohybové zákony: sú tri základné zákony mechaniky, ktoré publikoval sir Isaac Newton v diele Principia Mathematica v r. 1687. Tieto zákony a predovšetkým druhý z nich tvoria absolútny základ pre pochopenie a simulovanie

dynamiky tuhých telies. Umožňujú určiť pohyb telesa v inerciálnej vzťažnej sústave, ak sú známe sily naň pôsobiace.

Inerciálna vzťažná sústava: je vzťažná sústava, v ktorej platí Prvý Newtonov pohybový zákon.

Prvý Newtonov pohybový zákon - Zákon zotrvačnosti: Každý hmotný bod v inerciálnej sústave zotrváva v pokoji alebo v rovnomernom priamočiarom pohybe, kým nie je nútený vonkajšími silami tento svoj stav zmeniť.

Druhý Newtonov pohybový zákon - Zákon sily: V inerciálnej vzťažnej sústave sa výsledná sila pôsobiaca na hmotný bod rovná prvej derivácii hybnosti hmotného bodu podľa času. Ak sa pri pohybe hmotnosť telesa nemení, platí : Ak na teleso pôsobí sila, teleso sa pohybuje zrýchlením, ktoré je priamo úmerné pôsobiacej sile a nepriamo úmerné hmotnosti telesa, čiže:

$$a = F / m$$

kde $\boldsymbol{a}$ *je vektor zrýchlenia,* $\boldsymbol{F}$ *je vektor sily,* $\boldsymbol{m}$ *je hmotnosť telesa.*

Tretí Newtonov pohybový zákon - Zákon akcie a reakcie: Dva hmotné body na seba pôsobia rovnako veľkými silami opačného smeru,ktoré súčasne vznikajú a súčasne zanikajú. Jedna zo síl sa nazýva sila akcie a druhá sa nazýva sila reakcie. Ak pôsobí prvé teleso na druhé teleso, pôsobí aj druhé teleso na prvé teleso.

2.4.2 Počítačové simulácie

Počítačovú simuláciu si možno predstaviť ako matematický model aplikovaný na zvolený dynamický systém[3] pomocou počítača. Počítačové simulácie obyčajne používajú zjednodušený model dynamického systému, pretože reálny systém, býva nezriedka zložitý a výpočty by boli časovo príliš náročné.
Vhodným príkladom sú moderné počítačové hry, využívajúce fyzikálny systém. Pri hraní hier nezáleží na korektnosti výpočtov do takej miery, ako povedzme pri vedeckých projektoch. Preto používajú veľmi zjednodušený model dynamiky. To zaručuje rýchle vyhodnocovanie a vysokú mieru interaktivity. Povedané zjednodušenie: rýchlosť na úkor presnosti.

---

3 - Rozumieme systém, ktorého okamžitý stav sa mení v čase v závislosti od predchádzajúceho stavu a vonkajších podnetov.

Pokiaľ chceme hovoriť o fyzikálnej simulácii o niečo konkrétnejšie, je potrebné si uvedomiť, o aký typ simulácie máme záujem. Na trhu existuje pomerne bohatá škála komerčne i voľne dostupných riešení, ktorých funkcie a vlastnosti sa líšia v závislosti od spôsobu využitia. Preto je vhodné stotožniť nároky na presnosť, rýchlosť, či možnosť interaktívne do procesu simulácie vstupovať s potrebami konkrétneho projektu. Ak uvažujeme nad vzťahom dynamického modelu a simulácie z pohľadu časovej porovnateľnosti, možno rozlišovať štyri typy simulácii.

Online simulácia: výpočty prebiehajú porovnateľne, alebo rovnako rýchlo, ako v skutočnosti. Väčšinou ide o simuláciu jednoduchých modelov.

Offline simulácia: v značnej miere pomalšia ako reálny model. Simulácie zložitých systémov. Napríklad simulácie počasia, alebo podobne náročných modelov.

Interaktívna simulácia: dostatočne rýchla na to, aby bolo možné zasahovať do jej priebehu. Napríklad už spomínané hry, alebo simulácie v robotike či strojárenstve.

Realtime simulácia: simulácia zaručuje update systému fixný počet krát za sekundu. Jedná sa o veľmi presné väčšinou vedecké simulácie.

Keďže sa jedná o simuláciu riadiaceho systém mobilného robota, nepotrebujeme ani extrémnu výpočtovú silu, ani nanometrickú presnosť. Zato je nevyhnutná možnosť priamej interakcie s riadiacim systémom za okamžitej odozvy robota. Navyše by bolo vhodné použiť voľne dostupné riešenie. Existuje pomerne široké spektrum nástrojov dostupných vo forme SDK[4] . Takýto systém dynamických knižníc je pre projekt prinajmenšom vhodný, keďže je potrebné algoritmy riadenia navrhnúť na úrovni programovacieho jazyka.

2.4.3 Simulácie pomocou SDK

Ako taká simulácia pomocou SDK v skutočnosti funguje. Formou rovníc a vzorcov navrhnutý dynamický model konkrétneho fyzikálneho systému (mechanika, počasie), spracúvajú algoritmy schopný riešiť tieto rovnice a vypočítať, ako sa systém mení v čase. Ak tieto algoritmy implementujeme v počítačovom program, jeho spustenie možno pokladať za aktívnu simuláciu daného fyzikálneho systému.

---

4 - Z anglického Software Development Kit

Kapitola 3

Návrh

3.1 Filozofia návrhu

Než začneme s návrhom konkrétnych algoritmov, musíme si ujasniť vzťahy medzi kľúčovými prvkami projektu a načrtnúť nami preferovaný proces riadenia robota. V zjednodušenej forme možno tento proces vidieť na diagrame Chart 3.1.

Základnou myšlienkou celého projektu, je vytvoriť počítačový program, chopný transformovať požiadavky užívateľa na postupnosť príkazov, ktoré za účelom pohybu robota vhodne ovládajú jeho elektromotory. Tento program má byť spustený na počítači a komunikovať s robotom skrz bezdrôtové pripojenie. Užívateľ interaguje so systémom pomocou jednoducého rozhrania. Napríklad terminálu.

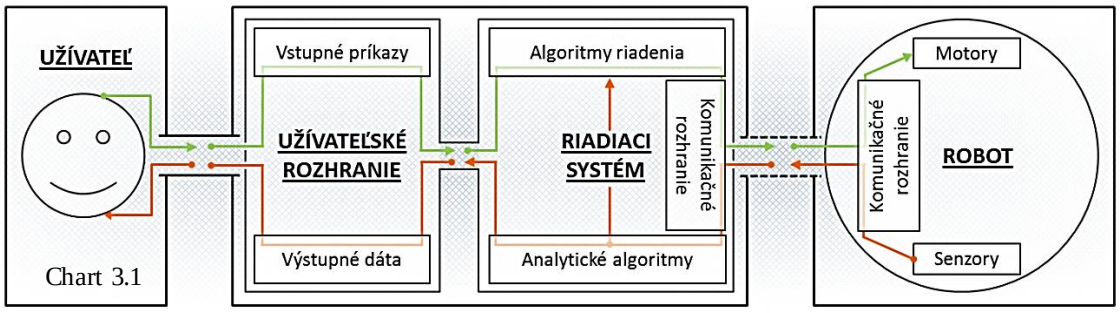


Chart 3.1

Odhliadnuc od technickej stránky projektu, mechanických vlastností robota a komplikácií spojených s jeho konštrukciou, i tak dostávame návrh, závislý na bezproblémovej komunikácii riadiaceho sytému s robotom a precíznych reakciách elektromotorov. Keďže cieľom projektu je práve systém, ktorý je univerzálny a nezávislý na konkrétom modeli robota zvoleného typu, pokladali sme sa vhodné použiť ako testovacie prostredie práve počítačovú simuláciu.

Jednoznačnou výhodou navrhovaného prístupu je absencia konštrukčných a komunikačných problémov a mechanických častí, akými sú elektromotory a závažia. Aj keď budeme v procese simulácie budeme brať do úvahy teoretickú polohu závaží, simulovať budeme iba vychyľovanie ťažiska. Samozrejme na základe teoretického modelu robota.Môžeme teda definitívne povedať, že návrh a následná implementácia, zahŕňa iba tri nezávislé subsystémy:

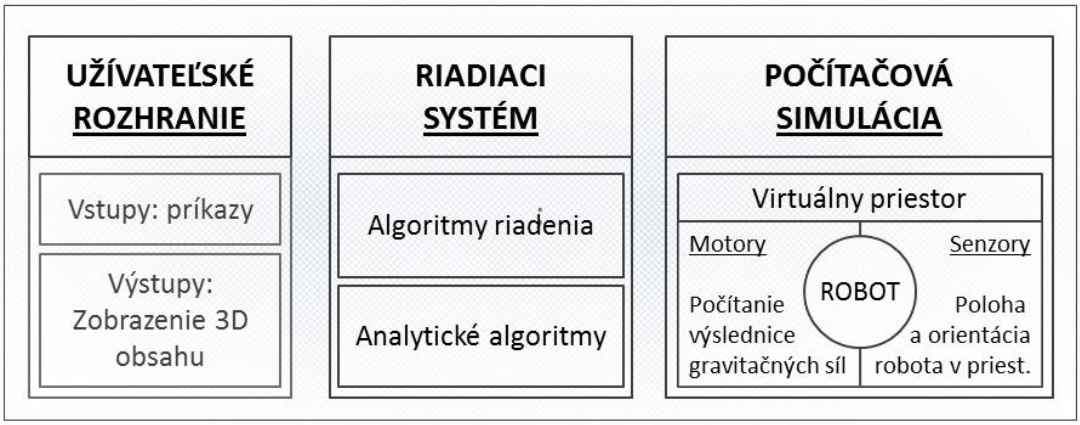


Chart 3.2

3.2 Teoretický model robota

Skutočnosť, že cieľom práce nie je výroba skutočného robota, umožňuje, či dokonca núti pristupovať k technickým otázkam výlučne na teoretickej úrovni. Použitie počítačovej simulácie si navyše žiada zjednodušený prístup ku konštrukcii.

Preto nemá zmysel, venovať sa podrobne konštrukčným a technickým otázkam a ich riešeniam. Zameriame sa predovšetkým na aktívne mechanizmy, ktoré priamo súvisia s náhonom a riadením robota.

V prvom rade je potrebné navrhnúť základné konštrukčné charakteristiky, ktorými sa budeme riadiť pri tvorbe algoritmov riadenia. Správne navrhnutie konštrukcie pomôže zadefinovať vlastnosti riadiaceho systému a zároveň mu stanoví hranice funkčných požiadaviek.

3.2.1 Zvolený prístup

Pri výbere konkrétneho prístupu sme za rozhodujúcu považovali čo najvyššiu mieru manévrovateľnosti. Vzhľadom na to, že projekt vzniká na akademickej pôde a v blízkej budúcnosti môže poslúžiť ako podklad pre zostrojenie skutočného robota,

mal by byť použiteľný na edukačné a vedecké účely.
Dôraz sme preto kládli predovšetkým na presnosť a schopnosť okamžite reagovať na požiadavky zo strany užívateľa. Na základe týchto požiadaviek sme sa rozhodli navrhnúť systém riadenia pre robota s troma navzájom kolmými osami, po ktorých sa presúvajú závažia.

3.2.2 Konštrukčné požiadavky

Dôležitou vlastnosťou nášho modelu riadenia má byť aj určitá miera univerzálnosti. To znamená, že musí byť implementovateľný bez ohľadu na technické parametre, akými sú veľkosť či hmotnosť robota, hmotnosť závaží alebo rýchlosti elektromotorov. Aby bolo vôbec možné takýto model riadenia aplikovať v praxi, musíme zadefinovať niektoré technické a konštrukčné špecifiká, pre ktoré náš model riadenia navrhneme. Ako som načrtol na začiatku kapitoly, nejedná sa o exaktnú technickú špecifikáciu, ale skôr o súbor pravidiel, ktoré je nutné dodržať pre bezproblémovú implementáciu modelu riadenia. Začnime teda zoznamom najdôležitejších konštrukčných prvkov.

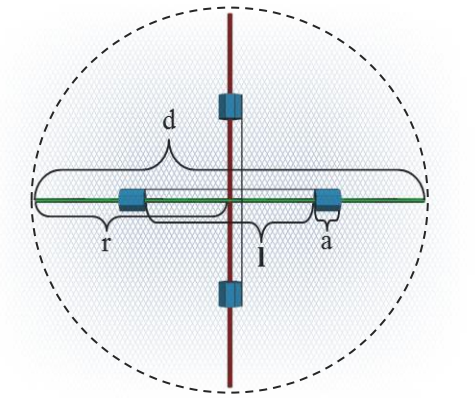
Vonkajšia guľová schránka: Základným stavebným prvkom robota, je vonkajšia schránka v tvare guľe: *sférický exoskelet*. Dovoľuje robotovi plne využívať potenciál valivého pohybu. Pre lepšiu zotrvačnosť a čo najnižší odpor sa uprednostňuje schránka z pevných a ľahkých materiálov, ako je napríklad PVC. Z praktických dôvodov je dobré, aby sa dala schránka jednoducho otvoriť.

Sústava troch osí: Ďalšou dôležitou súčasťou robota sú tri rovnaké, navzájom kolmé priečky - osi. Ich dĺžku je potrebné prispôsobiť vnútorným rozmerom guľového obalu. Pre efektívne presúvanie hmotnosti sa vyžaduje čo najväčšia možná dĺžka osí.

Elektromotory: Jedinou požiadavkou na použité motorčeky je schopnosť posúvať závažia o presne stanovenú vzdialenosť. Ďalšie parametre, umiestnenie, či prepojenia so závažiami nie je sú pre túto prácu dôležité.

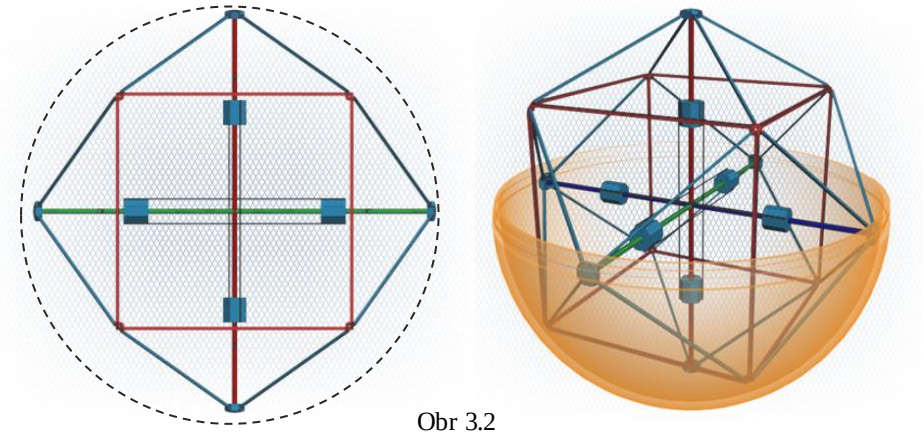
Senzory: Musia byť schopné poskytnúť riadiacemu algoritmu aktuálnu informáciu o otočení v priestore vo forme eulerových uhlov[5], alebo takej, z ktorej možno tieto uhly vhodným výpočtom odvodiť.

Posuvné závažia: Po každej osi sa posúvajú dve napevno spojené závažia rovnakej veľkosti a hmotnosti. Pozri obr. 3.1. Nech ***d*** je dĺžka osi, ***a*** nech je dĺžka jedného závažia a ***r*** nech udáva vzdialenosť medzi hraničnými pozíciami tohoto závažia. Potom optimálnu vzdialenosť medzi nimi, možno vyjadriť vzťahom: ***l = d - r - a***.



Obr 3.1

Vnútorná kostra: Aby bolo možné vo vnútri robota nainštalovať osi so závažiami, elektromotory, senzory, prípadne ďalšie súčasti, potrebujeme vyrobiť vhodnú nosnú konštrukciu, ktorá zaistí pevné ukotvenie všetkých potrebných komponentov. Jedným z riešení je napríklad kostra kocky, doplnená o ramená pre upevnenie osí a lepšie ukotvenie celého systému vnútri gule - Obr 3.2.
Hoci pre potreby simulácie by bolo zbytočne na obtiaž, brať takúto konštrukciu do úvahy, nech poslúži ako námet pre prípadný reálny prototyp.


Obr 3.2

Žiadne ďalšie súčiastky, ako integrované obvody, napájanie motorov a senzorov, alebo komunikačné zariadenia, nie sú predmetom tohto návrhu a pri tvorbe riadiaceho systému nebudú brané do úvahy. V prípade reálnej implementácie je potrebné dbať na čo najdokonalejšie vyváženie celého robota - ak sa všetky závažia nachádzajú vo východzej pozícii, ťažisko robota leží čo najbližšie k jeho geometrickému stredu. Samozrejme, že určitú mieru nerovnováhy možno ošetriť softwarovo, ale znižuje sa tým celkový manévrovací potenciál robota.

3.2.3 Stabilita a Orientácia

Pri skúmaní konceptu guľového robota, narazíme na dve dôležité vlastnosti, ktoré priamo vyplývajú z ich konštrukcie.

Stabilita: Jednou z najvýraznejších vlastností sférickej schránky, je absencia straty stability. Každý nerovnovážny stav je pre robota predpokladom pohybu, nie hrozbou poškodenia. Povedané veľmi zjednodušene - guľový robot nemôže za bežných podmienok spadnúť. Gúľa sa. Tento napohľad zrejmý fakt je veľkou devízou, špeciálne na poli mobilných robotov. Umožňuje nám navrhnúť model riadenia nezávislý na kontrole stability.

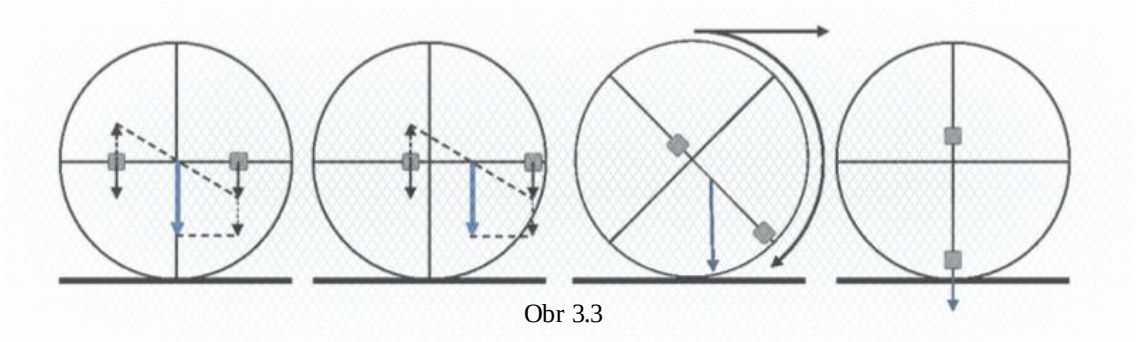
Orientácia: Pri pohľade na guľového robota vyvstáva otázka orientácie: Kde je pre takéhoto robota vpredu, a vzadu? Hore a dole? Existuje niekoľko prístupov k riešeniu tohto problému. Načrtnem základné dva.

Prvé, jednoduchšie riešenie nemožno aplikovať na všetky typy guľových robotov. Je vhodné predovšetkým pri systémoch založených na koncepte kolesa, ale aj vo všetkých implementáciách, kde sa riadiace systémy - motory, senzory a pod. neotáčajú spoločne s guľovým obalom. Dobrým príkladom je napríklad systém kyvadla. Pre pohyb dopredu sa kyvadlo vychyľuje vždy jedným, rovnakým smerom. Motory sa otáčajú vždy rovnako. To isté platí o jazde vzad, alebo zmene smeru.
Vďaka týmto vlastnostiam si riadici systém vystačí so sjednodušeným pohľadom na problém orientácie robota v priestore.

Druhým, o niečo všestrannejším, no zároveň komplikovanejším riešením je použitie gyroskopu, alebo iného senzoru, ktorý dokáže presne určiť svoju orientáciu v priestore. Výstupná informácia získaná z takéhoto senzoru obsahuje tri čísla. Tie reprezentujú veľkosti **Eulerovych uhlov**.
Leonard Euler (1707-1783) dokázal vetu o rotačnom pohybe, ktorá tvrdí, že každý rotačný pohyb v priestore sa dá vyjadriť iba tromi parametrami, uhlami otáčania okolo troch osí v priestore, nazývaných Eulerove uhly. Tieto uhly možno chápať aj ako postupnosť troch rotácií. Túto skutočnosť využijeme aj mi, pri návrhu modelu riadenia a tvorbe kľúčových algoritmov. Podobný prístup sa bežne využíva pri určovaní orientácie lietadiel, či vesmírnych plavidiel. Preto vytvoriť funkčný a efektívny algoritmus aplikovaný na robota možno pokladať za najväčšiue výzva projektu.

3.2.4 Základná mechanika

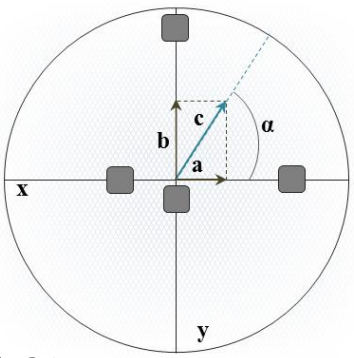
Ešte než začneme s návrhom softwaru, pozrime sa bližšie na princíp prenášania hmotnosti a pôsobenie gravitačnej sily vo vnútri robota. Pochopenie tohto princípu je nevyhnutné pre návrh vhodného modelu riadenia. Obrázok 3.3 zjednodušene popisuje princíp skladania gravitačných síl pôsobiacich na závažia. Ide o veľmi jednoduchý princíp skladania rovnobežných síl.



Obr 3.3

Ďalším dôležitým krokom je zmena smeru pohybu. Ak chceme, aby sa robot pohyboval presne určeným smerom, musíme zapojiť do procesu vychýlenia ťažiska, viac ako jednu os. Pozri obrázok 3.4 - pohľad zhora.

Berme do úvahy dve vzájomne kolmé osi ***x*** a ***y***. Obe nech sú rovnobežné s podložkou, na ktorej stojí robot. Chceme vykonať pohyb smerom vychýleným od osi ***x*** o uhol ***α***. Zvoľme maximálne posunutie po osi ***y*** o veľkosti ***b***. Potom posunutie po osi ***x*** musí mať veľkosť $a = b / tg(\alpha)$.



Obr 3.4

Treba si uvedomiť, že miera vychýlenia závaží ***c*** ovplyvňuje celkovú rýchlosť pohybu. Čím väčšie **c**, tým vyššia rýchlosť. Tento fakt nám dovolí meniť nie len smer, ale i ovplyvňovať rýchlosť pohybu robota. Najväčšia možná hodnota, akú ***c*** nadobúda pri tomto type robota, je polovica zo vzdialenosti medzi hraničnými pozíciami konkrétneho závažia.

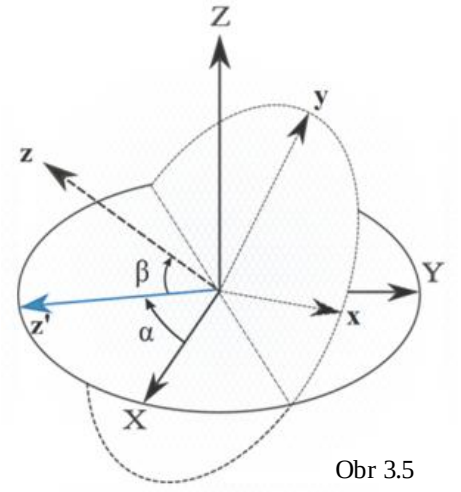
3.3 Model riadenia

Cieľom tejto práce je predovšetkým navrhnúť riadiaci systém pre vybraný typ guľového robota. Dôraz sa pritom kladie na univerzálnosť systému riadenia pri neskoršej implementácii.

Predpokladajme teda, že existuje robot, ktorý spĺňa nami definované konštrukčné požiadavky. Máme zabezpečený vzdialený prístup k jeho elektromotorom a senzorom. Sme schopní presúvať závažia a získavať dáta o otočení robota v priestore. Začnime teda s návrhom algoritmov riadenia. Základ celého systému tvoria dva nezávislé procesy. Prvý z nich analyzuje okamžitý stav robota. Druhý následne počíta vhodnú polohu závaží.

3.3.1 Analýza okamžitého stavu robota

Prvoradou úlohou analýzy je počítať vychýlenie osí vo vzťahu k ich nulovým pozíciám a ukladať tieto informácie v dočasnej pamäti. Sledovať, ako sa tieto dáta menia v čase a počítať niektoré doplňujúce informácie, akými sú momentálna rýchlosť pohybu a tiež prejdená vzdialenosť. Pozri obrázok 3.5.



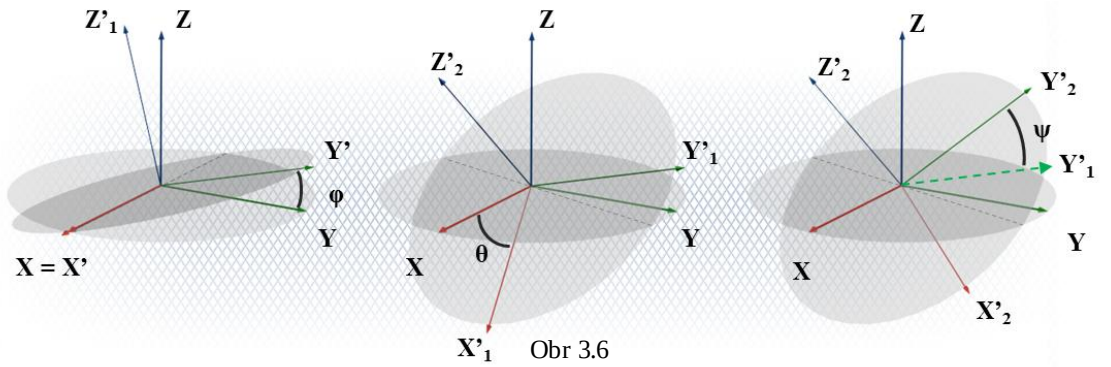
Obr 3.5

Na obrázku je zobrazená statická súradnicová sústava označená veľkými písmenami ***X,Y*** a ***Z***. Je to východzí stav pre robota. Senzor vracia uhly (0,0,0). Druhá súradnicová sústava označená malými písmenami ***x, y*** a ***z*** predstavuje otočenie robota v určitom okamihu. Pri dodržaní konštrukčný pokynov, možno túto sústavu považovať za totožnú so sústavou troch osí robota. Pre správne fungovanie rozhodovacích a riadiacich algoritmov musíme vypočítať dva špecifické uhly pre každú os robota. Na obrázku 3.5 vidieť požadované uhly pre os ***z***. Uhol ***α*** vychýlenia osi od osi ***X***, ktorú pokladá senzor za nulovú a uhol ***β,*** ktorý zviera s rovinou horizontu, resp. s vlastnou projekciou do tejto roviny ***z'***. Rovinou horizontu rozumieme rovinu na ktorej ležia statické osi ***X*** a ***Y*** a os ***Z*** je na ňu kolmá. Táto rovina by mala byť vždy vodorovná a prechádzať priesečníkom geometrickým stredom robota.

Najdôležitejším algoritmom v celom procese analýzy je práve počítanie spomínaných uhlov α a β. Existuje viacero spôsobov počítania, napríklad pomocou rotačných matíc, no vzhľadom na to, že očakávame dáta zo senzora vo forme eulerovych uhlov, navrhneme vhodnú formu výpočtu priamo z týchto hodnôt.
Keďže eulerove uhly možno pokladať za postupnosť otočení po jednotlivých osiach v poradí tej ktorej konvencie, budeme aj uhly jednotlivých osí odvodzovať postupne. Síce nemožno predpokladať rovnakú konvenciu pre všetky existujúce senzory, bolo by nad rámec tejto práce navrhovať všeobecný algoritmus, ktorý by bol schopný pracovať s každou z dvanástich konvencií. Pre potreby projektu budeme preto predpokladať konvenciu **X - Y'- Z'**.
Je pravda, že algoritmus stráca takýmto striktným výberom na svojej univerzálnosti, no je postačujúci pre potreby projektu a môže poslúžiť minimálne ako podklad, či inšpirácia v prípade použitia inej konvencie.

Pozrime sa bližšie na problematiku počítania uhlov. Pozri obrázok 3.6. Senzor nám na požiadanie dodá tri uhly (φ, θ, ψ). Použitie konvencie **X - Y'- Z'** znamená, že momentálnu orientáciu senzora a teda robota v priestore získame, ak ho budeme postupne otáčať v nasledujúcom poradí:

- o uhol φ po pôvodnej osi **X** robota.
- o uhol θ, po jedenkrát zrotovanej osi **Y** robota, čiže po $\mathbf{Y'_1}$.
- o uhol ψ, po dvakrát zrotovanej osi **Z** robota, čiže po $\mathbf{Z'_2}$.



Obr 3.6

Ako správne vyhodnotiť tieto informácie a vypočítať veľkosti uhlov α a β z obrázku 3.5 pre všetky osi robota? Eulerove uhly vyjadrujú postupné skladanie troch rotácií, čo zabraňuje vypočítať pomocou jedinej funkcie uhly pre všetky osi naraz.

Je však možné tieto uhly odvodzovať v nadväznosti krokov za pomoci takejto jednoduchej funkcie. Je dobré si uvedomiť, že pre každú os stačí brať do úvahy dve rotácie. Keď objekt rotuje po danej osi, jej poloha sa predsa nemení. Ak vychádzame z konvencie **X - Y'- Z'**, môžme povedať, že pri skladaní troch rotácií sa dostane do konečnej pozície ako prvá os **Z**. Po prvých dvoch rotáciách. Využijeme túto skutočnosť a pokúsime sa vypočítať uhly $\boldsymbol{\alpha}$ a $\boldsymbol{\beta}$ pre túto os. Za týmto účelom sme si odvodili nasledovnú funkciu **F1:**

$$\alpha Z = arctg\ \frac{\sqrt{(1-u)^2 * v^2}}{u}$$

$$u = cos(\theta - 90)$$

$$v = cos(\varphi - 90)$$

Táto funkcia je iba upravenou formu **stredovej rovnice elipsy** za vhodného využitia vlastností goniometrických funkcií a jednotkovej kružnice. Funkcia je do veľkej miery všeobecná, ale nemá zmysel pre $\boldsymbol{\theta = 0}$. Táto možnosť musí byť vo fáze implementácie vhodne ošetrená, napríklad pripočítaním jednej tisíciny k nulovému uhlu.
Pre vypočítanie uhla $\boldsymbol{\beta}$ osi **Z** upravíme hodnoty $\boldsymbol{u}$ a $\boldsymbol{v}$ nasledovne:

$$u = cos(\varphi - 90)$$

$$v = cos(\alpha Z - 90)$$

A použijeme funkciu **F1**. Čaká nás výpočet uhlov pre os **Y**. Výpočet je nutné rozšíriť o ďalší krok a opäť vhodne upraviť hodnoty $\boldsymbol{u}$ a $\boldsymbol{v}$:

$$w = \arcsin \frac{sin(\varphi - 90)}{\sin(\beta Z - 90)} * \frac{\varphi}{|\varphi|} * \frac{\theta}{|\theta|}$$

$$u = \cos(\psi - w)$$

$$v = cos(|\beta Z| - 90)$$

$$\alpha Y = arctg\ \frac{\sqrt{(1-u)^2 * v^2}}{u} - \alpha Z$$

Opäť stačí upraviť hodnoty $\boldsymbol{u}$ a $\boldsymbol{v}$ a použiť funkciu **F1** na výpočet $\boldsymbol{\beta}$ pre os ***Y***:

$$u = cos(|\beta Y| - 90)$$

$$v = cos(\alpha Y - 90)$$

Výpočet uhlov pre os **X**, je totožný s počítaním pre os **Y** s jediným rozdielom: počítame pre $\psi = \psi - 90$.

Máme navrhnutý algoritmus, ktorý dokáže zo získaných dát okamžite vypočítať veľkosť odchýlky od východzieho, nultého smeru sezora pre každú s troch osí robota a veľkosť uhla, ktorý daná os zviera s rovinou horizontu. Tieto uhly budeme vo všeobecnosti označovať ako $\boldsymbol{\alpha}$ – pre uhol smeru osi a $\boldsymbol{\beta}$ – pre uhol zvieraný s rovinou horizontu. Viď obr. 3.5.

Ak chceme hovoriť o uhloch $\boldsymbol{\alpha}$ a $\boldsymbol{\beta}$ vo vzťahu ku konkrétnym osiam robota, zadefinujme si tieto osi a k nim relevantné uhly. Povedzme, že súčasťou guľového robota sú tri, navzájom kolmé osi. Ich spoločný priesečník leží v geometrickom strede tohto robota. Po každej z osí dokážeme v oboch smeroch presúvať rovnako ťažké závažia tak, ako je to zadefinované v časti 3.2 Teoretický model robota. Pokiaľ skúšame uvažovať o orientácii robota v priestore, musíme si určiť tzv. východzí – nulový stav, od ktorého budeme odvodzovať ďalšiu rotáciu. Súčasťou robota je senzor, ktorý vracia trojicu eulerovych uhlov. Ak senzor vracia v niektorom okamihu tri nulové uhly (0;0;0), hovoríme, že má nulovú orientáciu. V tomto momente existujú dva odlišné prístupy k otázke navigáce robota.

Definitívny princíp: ktorý vychádza priamo z geocentrickej sústavy planéty, v našom prípade Zeme. Uvažujme myslenú trojrozmernú súradnicovú sústavu, odvodenú od geocentrickej sústavy planéty, ktorej začiatok, bod [0,0,0] leží v strede senzora. Má tri nasledovne orientované osi: dve vodorovné, ktoré ležia v rovine horizontu senzora, kde X smeruje zo severného na južný magnetický pól planéty a os Y smeruje zo západu na východ. A tretiu os Z, kolmú na rovinu horizontu, ktorá smeruje kolmo do stredu planéty.

Povedzme teda, že má senzor nulovú orientáciu. Prispôsobme osi robota tak, aby sa zhodovali s geocentrickou sústavou senzora.

Potom ***Os_X*** je tá os robota, ktorá v momente nulovej orientácie senzora leží v rovine horizontu, resp. zviera s rovinou horizontu uhol o veľkosti 0° a zároveň smeruje priamo na sever. Tento smer, budeme tiež nazývať nulový, alebo východzí smer pohybu robota. ***Os_Y*** tiež leží v rovine horizontu a je vychýlená od nulového smeru o +90°.

Os_Z je taká, že s rovinou horizontu zviera uhol 90° a od nulového smeru je vychýlená o -45°. Smer pohybu robota v tomto prípade odvodzujeme od magnetického severu

planéty, resp. od inak definovaného nulového stredu v závislosti od modelu, či typu použitého senzora.

Relatívny princíp: Tento princíp jednoducho definuje jednu z osí robota, ako nulový smer a polohu zvyšných dvoch osí podobne ako v predošlom prípade. Zvolený smer pohybu odvodzuje od orientácie tejto vybranej osi.

Pre potreby návrhu riadiaceho systému si teraz o ničo exaktnejšie zadefinujeme uhly $\boldsymbol{\alpha}$ a $\boldsymbol{\beta}$ z obrázku 3.5. vo vzťahu, ku konkrétnym osiam robota, keďže sa jedná o súčasť analýzy okamžitého stavu, ktorá je kľúčom k úspešnému modelu riadenia.

α_X – uhol α pre os, ktorej východzia pozícia, je totožná s nulovou pozíciou senzora. Pri použití definitývneho prístupu to znamená, ak senzor obsahuje napríklad kompas a jeho nultá pozícia je nastavená na niektorú zo svetových strán, povedzme na severný magnetický pól, potom os_X tiež smeruje priamo na severný pól a uhol α_X je v takom prípade rovný 0°. V prípade použitia relatívneho princípujurčovania orientácie, je uhol α_X, vždy rovný nule.

β_X – už spomínaný uhol, ktorý zviera os_X s rovinou horizontu. Tento uhol sa rovná nule, ak leží os_ X v rovine horizontu.

α_Y – Analogicky, nech os_Y tiež leží v rovine horizontu spolu s osou_X, potom zviera s touto osou pravý uhol a rovnako s vychodzím smerom senzora uhol o veľkosti 90°.

β_Y – pre tento uhol platia rovnaké pravidlá, vo vzťahu k osi_Y, ako pre uhol β_X vo vzťahu k osi_X.

α_Z – zviera os_Z s východzím smerom senzora. O tomto uhle môžeme konštatovať, že pokiaľ sa os_X nachádza vo svojej východzej pozícii, má uhol ***α_Z*** veľkosť -45°.

β_Z – ak os_ X a os_Y ležia v rovine horizontu, zviera s nimi tretia os_Z pravý uhol a teda má uhol ***β_Z*** veľkosť 90°.

Posledná Doležitá informácia na záver je, že nami navrhnutý algoritmus počíta uhol $\boldsymbol{\alpha}$ vždy z intervalov: $\langle 0;180) \vee \langle -180;0)$ a $\boldsymbol{\beta}$ vždy ako uhol z intetrvalu: $\langle -90;90)$.

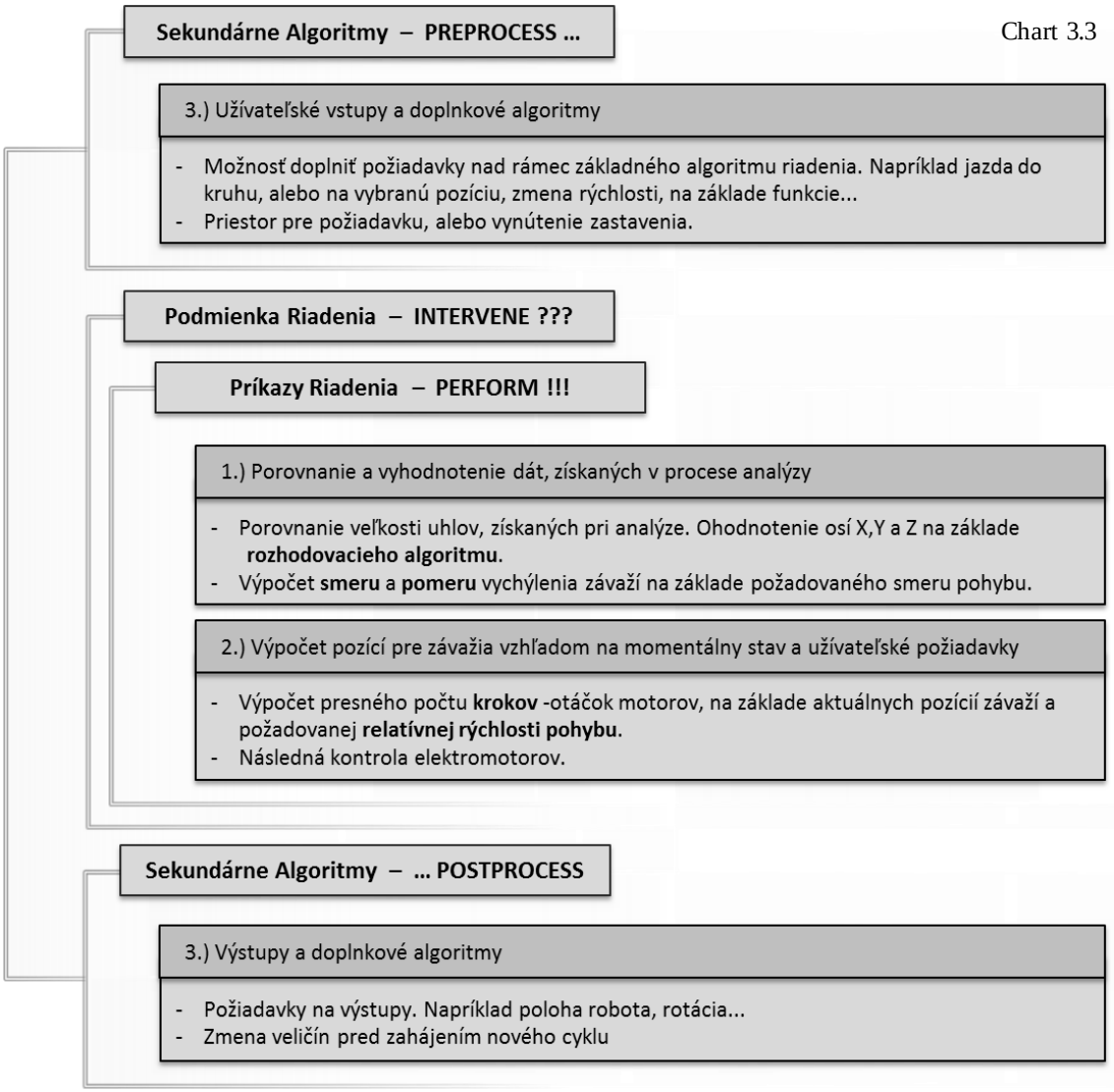
3.3.2 Algoritmy Riadenia

Máme zadefinovaných všetkých šesť potebných uhlov, poznáme ich vzťahy ku konkrétnym osiam a dokážeme ich za pomoci vhodných algoritmov vypočítať. Môžme sa teraz podrobnejšie zaoberať samotným riadením pohybu.

Keďže sa má jednať o dynamický model riadenia robota, uvažujme slučku – cyklicky sa opakujúci algoritmus, ktorý vyhodnocuje momentálny stav robota vzhľadom na zadefinované podmienky pohybu a následne, berúc do úvahy orientáciu robota, užívateľské vstupy a mechanické vlastnosti robota, efektívne vyhodnocuje nové polohy pre závažia a posiela elektromotorom robota správne požiadavky .

Našou úlohou je teda určiť vhodnú postupnosť jednotlivých krokov algoritmu a bližšie tieto kroky zadefinovať. Pozri Chart 3.3.

Chart 3.3



Na diagrame 3.3 možno vidieť približnú postupnosť krokov vykonaných počas jedného „úplného“ cyklu. Rozumie sa cyklus v ktorom je potrebné vykonať zmeny polôh závaží.

Pozrime sa podrobnejšie na konkrétne kroky a skúsme si uvedomiť, čo presne proces riadenia obnáša a vyžaduje. Už vieme, že prvým dôležitým prvkom, je výpočet uhlov, ktoré zvierajú osi robota s nulovými pozíciami senzora. Povedzme si teda, ako bude prebiehať samotný pohyb robota. Pozornosť venujme iba jednoduchému pohybu vybraným smerom, ktorý tvorí absolutný základ riadiacich algoritmov.

Príkazy Riadenia: Prvým krokom je, na základe vypočítaných uhlov, vybrať minimálne dve osi, z ktorých možno vysunutím závaží vychýliť ťažisko presne určeným smerom a upresniť smer, ktorým sa budú musieť otáčať elektromotory.
Návrh takéhoto algoritmus nie je úplne triviálny. Je potrebné zvažovať nie len jeho presnosť, ale aj rýchlosť vykonania a v neposlednom rade mieru efektivity vybraných osí. To znamená, vysunúť ťažisko nie len vybraným smerom, ale s čo najvyšším možným zotrvačným potenciálom. Povedané inak, treba zvoliť kompromis, medzi čo najväčšou pákou a rezervou zotrvačnosti.
Navyše je vhodné nájsť algoritmus dosť všeobecný, aby ho bolo možné aplikovať bez ohľadu na orientáciu robota v priestore. Algoritmus, ktorý sa nám podarilo navrhnúť možno veľmi zaobalene zhrnúť do dvoch krokov:

Vybrať os, s čo najväčšou odchýlkou od rovinou horizontu, resp. os s najvačším uhlom $\boldsymbol{\beta}$ *v smere pohybu. Túto os budeme nazývať* ***hlavná os.***

Vybrať druhú os s opačným znamienkom uhla $\boldsymbol{\alpha}$*, ako* ***hlavná os.*** *Prioritne však os s rovnako orientovaným uhlom* $\boldsymbol{\beta}$*. Budeme ju nazývať* ***doplnková os.***

Rozdiel medzi hlavnou a doplnkovou osou je taký, že po hlavnej osi budeme vysúvať závažie vždy na maximálnu dovolenú pozíciu a polohu závaží na doplnkovej osi dopočítame, pomocou pytagorovej vety.
Takto zadefinované osi, v sebe nesú najväčší potenciál pre pohyb vybraným smerom.. Tento spôsob výberu nám zaručí, že výslednica, ktorú osi vytvoria, bude v prvom okamihu zvierať s horizontom uhol $\boldsymbol{\beta}$ väčší, ako 0° a menší ako 45°, čo dáva robotovy najvyšší možný rotačný potenciál a v spolupráci s vhodnou podmienkou riadenia zaručí čo najnižšie požiadavky na presúvanie závaží.

Druhou dôležitou časťou, je plynulé rozloženie pohybu elektromotorov. Venovať sa tejto časti však z pohľadu simulácie nemá veľký význam, keďže nám budeme umožnené pracovať priamo s ťažiskom.
Istú mieru identity, však musíme zachovať a preto zadefinujme aspoň nasledovné. Na základe technickej špecifikácie konrétneho prototypu je vhodné, prepočítavať polohu závaží na tzv. „kroky“ pevne stanovenej dĺžky. Požadovaná poloha závažia sa potom rovná dĺžke jedného kroku násobenej počtom krokov potrebných k dosiahnutiu požadovanej pozície. Tiež si treba uvedomiť, že miera vychýlenia závaží po osi vybranej, ako hlavná os riadenia robota udáva parcentuálnu mieru maximálnej dosahovanej rýchlosti otáčania a teda i posunu robota. Inak povedané, čím vyššiu dokážeme vysunúť maximálnu hodnotu na hlavnej osi, tým rýchlejšie sa bude robot pohybovať.

Podmienka Riadenia: ak chceme aby sa robot pohyboval, musí nutne dôjsť k vychýleniu jeho ťažiska. Reakciou na tento nerovnovážny stav je rotácia a s ňou spojený priamočiary pohyb. Aby sme docielili efektívne opakovanie tohto procesu, musíme si zadefinovať vhodné pravidlo, ktorého naplnenie resp. nenaplnenie definitívne určuje, či je potrebné zapojiť ďalšie algoritmy riadenia.
Nájsť takúto univerzálnu podmienku nebolo triviálne a predchádzalo jej mnoho úvah, pozorovaní a analýz rotácií objektov v 3D modelovacích programoch. Nakoniec však možno túto podmienku definovať pomerne jednoducho:

Príkazy riadenie je potrebné použiť a zadefinovať novú výslednicu v momente, keď uhol β *osi, ktorú pokladáme za hlavnú vrámci prebiehajúceho pohybu prekročí veľkosť + 45° v smere pohybu. (kladná hodnota v tomto prípade znamená, že sa závažia na osi sa približijú k zemi).*

Táto jednoduchá podmienka zabezpečí minimálny pohyb závaží, počas pohybu. Konkrétne dvakrát za jedno otočenie robota v smere pohybu a využíva tak celý potenciál výslednice, bez straty rýchlosti.

Sekundárne Algoritmy: priestor na zadávanie užívateľských vstupov a generovanie požadovaných výstupov a na úpravu vybraných premenných pred vykonaním, alebo po vykonaní príkazov riadenia.

Kapitola 4

Implementácia

4.1 Použité nástroje

Výber vhodných nástrojov je samozrejme jedným z dôležitých aspektov implementácie. Najdôležitejšími prvkomi projektu sú model riadenia a fyzikálna simulácia. Keďže sme sa rozhodli vytvoriť simuláciu za pomoci SDK, a potrebujeme všestranný programovací jazyk, ako je ***C++***, zvolili sme ako najvhodnejšiu kombináciu nástrojov spoluprácu všestraného programocieho nástroja, akým je ***Visual Express Studio*** a súbor knižníc ***Open Dynamics Engine***. Jedná sa o profesionálne a predsa voľne dostupné nástroje.

4.1.1 Open Dynamics Engine

ODE je súbor profesionálnych voľne dostupných open source dynamických knižníc, použiteľných pri simulácií dynamiky pevných telies, navrhnutý pre spoluprácu s C/C++ API. Zahŕňa pokročilé techniky fyzikálnej simulácie, ako počítanie trecích a zotrvačných síl, či pokročilú detekciu kolízií.
Je vhodný a teda aj používaný pri simulovaní rôznych prvkov virtuálnej reality, akými sú prostredia, stroje, či dokonca živé bytosti v mnohých profesionálnych projektoch, alebo známych počitačových hrách, ako napríklad *S.T.A.L.K.E.R, World of Goo,* alebo *X-Moto*. Súčasťou ODE je i základná vykresľovacia knižnica *drawstuff.dll,* vychádzajúca z dobre známeho OpenGL, ktorá slúži k nenáročnému zobrazeniu výstupných dát v 3D a samozrejme všetky zdrojové kódy, dostupné v prípade potreby k modifikácii.

4.2 Aplikácia ako celok

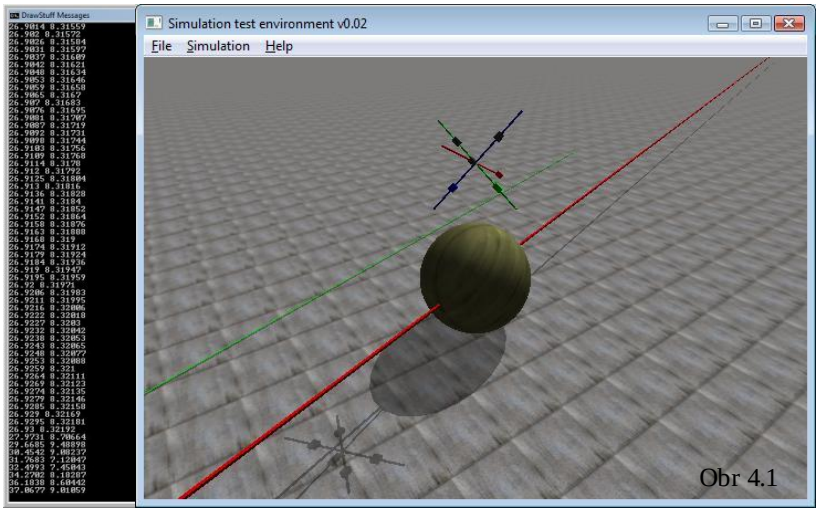
Budeme vychádzať z návrhu aplikácie, zobrazeného na diagrame char 3.2. v kapitole 3. Návrh. Aplikácia pozostáva z troch logicky samostatných - oddeliteľných častí: užívateľské rozhranie, simulácia virtuálneho sveta a guľového robota v ňom a riadiaci systém pre tohto robota.

4.2.1 Užívateľské rozhranie (GUI)

Pre potreby nášho projektu postačilo veľmi jednoduché užívateľské rozhranie. Kedže aplikácie nie je určaná bežnému užívateľovi a je iba prostriedkom k navrhnutiu a ďalšiemu vylaďovaniu algoritmov riadenia, nedisponuje žiadnymi funkčnými komponentami, či pomocnými nástrojmi, alebo nápoveďou. Skladá sa iba z dvoch samostatných okien.

Prvým je OpenGL okno vytvorené a spravované pomocou knižnice *drawstuff.dll*, ktoré slúži na "online" zobrazovanie 3D obsahu a dát generovaných simuláciou – obr 4.1.

Druhým je jednoduché terminálové okno, nazývané tiež *command line* – obr 4.1. Obe okná slúžia predovšetkým na kontrolu algoritmov riadenia podľa požiadaviek užívateľa, či skôr programátora.



Obr 4.1

Treba podotknúť, že grafické okno slúži zároveň, ako zachytávač správ (event catcher) a je možné jeho prostredníctvom interagovať so simuláciou i samotným robotom. Nasleduje zoznam funkčných kláves a priradených funkcií v tabuľke Tab 4.1.

KLÁVESA	FUNKCIA	POPIS
t / T	*terminate*	preskakuje riadiace algoritmy
f / F	*freeze*	pause / unpause simuláciu
c / C	*camera*	statická / dynamická kamera
r / R	*skelet*	zobraz / skry endoSkelet (osi a závažia)
w / W	*X + 20*	Závažia na osi X posuň o + 20
s / S	*X - 20*	Závažia na osi X posuň o - 20
a / A	*Y + 20*	Závažia na osi Y posuň o + 20
d / D	*Y - 20*	Závažia na osi Y posuň o - 20
q / Q	*Z + 20*	Závažia na osi Z posuň o + 20
e / E	*Z - 20*	Závažia na osi Z posuň o - 20
< / >	*+/- angle*	zmeň smer pohybu o ANGLE_STEP_SIZE (čaká na potvrdenie)
b / B	*new angle*	potvrdí zmenu smeru pohybu nastavenú za pomoci < / >
p / P	*ToPoint*	zapni / vypni automatizovaný pohyb do bodu [TO_POINT_X, TO_POINT_Y] (definované užívateľom)
v / V	*circle*	zapni / vypni automatizovaný pohyb po kružnici
n / N	*stop*	zabrzdi robota
m / M	*move*	zaháji pohyb s ohľadom na nastavené parametre
z / Z	*unblock*	pri zablokovaní robota, spustí znovu analytický algoritmus, odblokuje závažia
x / X	*reset*	nastaví všetky závažia do vychodzej / nultej pozície

4.2.2 Simulácia

Aktívnou simuláciou rozumieme vhodne parametrizovaný trojrozmerný virtuálny priestor, v ktorom platia základné zákony mechaniky pevných telies a v ňom umiestnený dynamický, interaktívny model robota, navrhnutý podľa teoretického modelu z tretej kapitoly.O správne fungovanie takejto simulácie v našom programe sa stará jediná inštancia nami definovanej triedy *Simulation* nazvaná *mainSimulation.* Trieda i jej inštancia sú definované v súbore *Simulation_Class.h*.

4.2.3 Riadiaci systém

Vzhľadom na to, že počítačová simulácia je v istom slova zmysle samo o sebe cyklom, využili sme túto skutočnosť pri navrhovaní riadiacich algoritmov a obalili sme všetky dôležité algoritmy do jedinej triedy: *DrivingControl_Class*. Funkcionalita triedy je rozdelená to dvoch hlavných metód, *RotationAnalyze()* a *RidersControler()*. Ich interakcia je zahrnutá v hlavnej slučke programu a vhodne doplnená ďalšími algoritmami, ako bolo zadefinované na diagrame Char 3.3 v predchádzajúcej kapitole.

4.3 Kľúčové algoritmy

4.3.1 Simulácia virtuálneho sveta

Simulácia je vytvorená, následne inicializovaná a po ukončení práce "deštruovaná" v *main funkcii* aplikácie *Ballbot*. Aby mohla inštancia *mainSimulation* správne fungovať, musí byť v rámci programu ešte pred jej aktiváciou inicializované samotné ODE: *dInitODE()*.

Až následne možno úspešne inicializovať inštanciu *mainSimulation*. Treba si uvedomiť, že ODE je súbor DLL a teda nepodporuje priamo objektový prístup.

Preto bolo potrebné, vhodným spôsobom obaliť základnú funkcionalitu do metód nami zadefinovanej triedy. Pozrime sa teraz aspoň na zjednodušené "zapuzdrenie" triedy *Simulation*, obsahujúci niektoré základné metódy: obr. 4.2.

```
static class Simulation {
        private:
                dWorldID  simWorldID;                          //  parametrizovateľná množina hmotných entít
                dSpaceID  simSpaceID;                          // množina geometrických vlastností objektov
                dJointGroupID simJointID;                      //  množina kolízií objektov

                void simDrawing();                             // vykresľovanie dát pomocou drawstuff.dll

        public:
                void simNearCallback(void *, dGeomID, dGeomID); // riešenie kolízií
                void simCmndLine(int);                         // užívateľské vstupy
                void simMainLoop(int);                         // hlavná slučka simulácie
                void simStartLoop();                           // spúšťa hlavnú slučku
}
```

Obr 4.2

O správne fungovanie simulácie sa vo veľkej miere postará samotné ODE a teda by bolo zbytočné, podrobne sa venovať všetkým jeho funkciám a metódam triedy *Simulation*. Preto aspoň stručný pohľad na niektoré najdôležitejšie parametre pri vytváraní vyrtuálneho sveta a kolízneho priestoru: obr 4.3.

```
dJointGroupIDsimJointID = dJointGroupCreate(0);   ////////////////////////////////////////////////////////////////
dSpaceID simSpaceID = dHashSpaceCreate(0);        // Volania metód Create v ODE spravidla vracajú jedinečné int ID //
dWorldID simWorldID = dWorldCreate();             ////////////////////////////////////////////////////////////////

dWorldSetCFM(simWorldID, 0.1e-10);                // Constraint Force Mixing
dWorldSetERP(simWorldID, 1.0);                    // Počet "Error Correction" počas jedného kroku

dWorldSetGravity(simWorldID, 0.0, 0.0, 0.0);      // Sila gravitácie - globálny parameter pre konkrétny dWorld
dWorldSetDamping(simWorldID, 0.02, 0.02);         // Miera útlmu zotrvačnosti. Predchádzanie nekonečne gravitačným silám
```

Obr 4.3

Dalo by sa povedať, že základ pre simuláciu virtuálneho sveta pozostáva z troch rozlišných entít, konkrétne z:

- *dWorld* – vhodne naparametrizovaný dynamický svet podriadený zákonom mechaniky pevných telies. Simuluje sa v ňom pôsobenie síl(priamych, zotrvačných, trecích,..) na v ňom definované telesá na základe ich hmotnosti.
- *dSpace* – Kolízny priestor, zastrešuje kolízie objektov definovaných v prislúchajúcom *dWorld*, na základe ich geometrických vlastností.
- *dJointGroupIDsimJointID* – množina spojení, ktoré vznikajú pri kolízii.

4.3.2 Simulácia guľového robota

Robota možno vo vzťahu k virtuálnemu svetu a kolíznemu priestoru, opísať ako hmotnú entitu existujúce v patričnom *dWorld* svete, s dokonale guľovým tvarom definovaným vo vhodnom *dSpace*.

Navyše však vieme definovať ťažisko tohto telesa mimo jeho geometrický stred. Táto kľúčová vlastnosť nám dovoľuje simulovať pohyb závaží v jeho vnútry a robota ovládať. Takéhoto robota, zastrešuje nami definovaná trieda SphereBot. Pozrime sa na zapuzdrenie a niektoré z zaujímavé metódy: obr. 4.5.

```
class SphereBot {
  private:
    dBodyID sphereBodyID, skeletBodyID;                  // dBodyID je ID entity existujúcej vo vhodnom dWorld
    dGeomID sphereGeomID, skeletGeomID;                  // dGeomID je ID entity existujúcej vo vhodnom dSpace
    dSpaceID skeletSpaceID;                              // vlastný kolízny priestor pre kostru robota (aby nedošlo k interakcii)
    dWorldID skeletWorldID;                              // vlastný svet pre kostru robota (aby nedošlo k interakcii)
    dMass sBotMass;                                      // ID entity robota v dWorld
    dReal sBotRadius;                                    // polomer robota
    dReal sBotWeight;                                    // vlastné hmotnosť robota

    dReal riderXPosition, riderYPosition, riderZPosition;   // momentálne pozície závaží vo vzťahu k jednotlivým osiam

    int riderXDirection;                                 // smer pohybu závažia (rotácia motora) = +/- 1
    int riderXStepsCount;                                // momentálne pozície závaží vo vzťahu k jednotlivým osiam

    void riderXMakeStep();                               // posúva závažia po osipodľa stanovených pravidiel
  public:
    void resetRiderX(bool);                              // nastavý závažia na vychodziu pozíciu
    void sBotDraw(bool);                                 // súčast drawstaff vykresľovacieho procesu
    void sBotStep();                                     // súčast mainLoop aplikácie

    void getEulerAnglesXYZ(dReal &, dReal &, dReal &);          // z rotačnej matice robota počíta aktuálne eulerove uhly
    void moveRiderX(int steps) { riderXStepsCount = steps; };   // inc/dec počet "krokov" závažia

    dReal getGravityCenterX()  { return riderXPosition - RIDER_ZERO_POSITION;  };     //vráti pôsobisko gravitačnej sily danú os
    const dReal *getRobotPosition()  { return dBodyGetPosition(sphereBodyID);  };     //vracia súradnice v 3D priestore
    const dReal *getRobotRotation()  { return dBodyGetRotation(sphereBodyID); };     //vracia rotačnú maticu robota
};
```

Obr 4.4

Z mnoštva metód možno aspoň približne vidieť, že simulácia pohybu závaží bola navrhnutá veľmi obozretne a s dôrazom na identitu s teoretickým modelom robota. Z pohľadu priebehu simulácie a ovládania robota je zaujímavá ešte metóda *sBotStep()*, ktorá je súčasťou hlavného cyklu: Obr. 4.5.

```
void SphereBot::sBotStep() {

   riderXMakeStep();                    //ak došlo k posunu závažia, počíta jeho novú pozíciu
   riderYMakeStep();
   riderZMakeStep();
                                        //simulácia pôsobenie gravitačnej sily na základe polohy závažia na jednotlivých osiach
   dBodyAddForceAtRelPos(sphereBodyID, 0.0, 0.0, GRAVITY_FORCE * 0.333, riderXPosition - RIDER_ZERO_POSITION, 0.00, 0.00);
   dBodyAddForceAtRelPos(sphereBodyID, 0.0, 0.0, GRAVITY_FORCE * 0.333, 0.00, riderYPosition - RIDER_ZERO_POSITION, 0.00);
   dBodyAddForceAtRelPos(sphereBodyID, 0.0, 0.0, GRAVITY_FORCE * 0.333, 0.00, 0.00, riderZPosition - RIDER_ZERO_POSITION);

};
```

Obr 4.5

Tu vidieť netradičnú silu ODE. Funkcia: *dBodyAddForceAtRelPos()* umožňuje nahradiť gravitáciu virtuálneho sveta, pôsobiacu vždy v geometrickom strede telesa, za silu pôsobiacu medzi dvoma závažiami každej z osí a tým simulovať vysunutie ťažiska. Inou zaujímavou funkciou triedy *Spherebot* je metódu *getEulerAnglesXYZ(dReal &angleX, dReal &angleY, dReal &angleZ)*. Obr.4.6, ktorá prepočíta rotačnú maticu robota na eulerove uhly. Ako sa spomína v kapitole 3 Návrh, existuje viacero spôsobov výpočtu týchto uhlov, rovnako ako možno zvoliť rôzne konvencie: Obr 4.6.

```
void SphereBot::getEulerAnglesXYZ(dReal &angleX, dReal &angleY, dReal &angleZ) {

        const dReal *rot = getRobotRotation();          // Rotačná matica robota
        angleY           = asin( rot[2]);               // Počíta veľkosť otočenia podľa osi Y
        dReal D          = angleY;
        dReal C = cos( angleY );
        angleY *= RADIANS;

        dReal tr_x, tr_y;

        if ( fabs( C ) > 0.005 ) {                      // Gimball lock Nie

                tr_x  =  rot[10] / C;                   // Veľkosť otočenia podľa osi Y
                tr_y  = -rot[6]  / C;
                angleX = atan2( tr_y, tr_x ) * RADIANS;

                tr_x  =  rot[0] / C;                    // Veľkosť otočenia podľa osi Z
                tr_y  = -rot[1] / C;
                angleZ = atan2( tr_y, tr_x ) * RADIANS;

          } else {                                      // Gimall lock Áno

                angleX = 0;                             // Nastav otočenie po osi X na nulu */
                tr_x   = rot[5];                        // Vypočítaj otočenie po osi Z*/
                tr_y   = rot[4];
                angleZ = atan2( tr_y, tr_x ) * RADIANS;

        }

          angleX = angleX == 0.00 ? 0.001 : angleX;     // Zabráni deleniu nulov
          angleY = angleY == 0.00 ? 0.001 : angleY;     // Zabráni deleniu nulov
          angleZ = angleZ == 0.00 ? 0.001 : angleZ;     // Zabráni deleniu nulov

};
```

Obr 4.6

4.3.3 Riadiaci systém

Ako je spomenuté vyššie, všetky dôležité riadiace algoritmy sú obsiahnuté v jedinej triede: *DrivingControl_Class*. Funkcionalita triedy je rozdelená to dvoch hlavných metód, *RotationAnalyze()* a *RidersControler()*. Ich interakcia je zahrnutá v hlavnej slučke programu a vhodne doplnená ďalšími algoritmami, ako bolo zadefinované na diagrame Char 3.3 v predchádzajúcej kapitole.

```
class DrivingControl {
  private:
    dReal angleX, angleY, angleZ;          // veľkosti eulerovych uhlov X, Y, X
    dReal toBeDirection;                   // nový smer pohybu, nastavený užívateľom
    dReal moveDirection;                   // aktuálny smer pohybu: <-180;+180);
    dReal moveRelSpeed;                    // percentuálne - desatinným číslom vyjadrená rýchlosť rotácie;
    dReal smallAlfaX,                      // odchýlka osi X od nulového smeru(severu)
    dReal smallBetaX;                      // odchýlka osi X od roviny horizontu
    dReal fullAlfaX,                       // úplná veľkosť uhla medzi osou X a nulovým smerom
    dReal fullBetaX;                       // úplná veľkosť uhla medzi osou X a rovinou horizontu

    int selectedAxis;                      // hlavná os (X = 1, X = 2, Z = 2)

    dReal getBasicAngle(dReal a, dReal b); // pomocný algoritmus počíta smallAlfa a smallBeta uhly
    dReal getComplementToResultant(dReal speed ... // pomocný algoritmus počíta doplnok k hlavnej osi
    void getZZAngles(dReal &smallAlfa, dReal ... // výpočet fullAlfaZ a fullBetaZ uhlov
    void getXYAngles(dReal &smallAlfa, dReal ... // výpočet fullAlfaX/Y a fullBetaX/Y uhlov
    void RotationAnalyze();                // hlavný analitický algoritmus, počíta uhly pre všetky tri osi
    bool RidersControler();                // hlavný riaidiaci algoritmus, rozhoduje a posúva závažia
  public:
    void dCtrlStep();                      // súčasť hlavného cyklu simulácie
}
```

Obr 4.7

Týmto sa dostávame k samotnému jadru projektu. Analytické algoritmy, spoločne s algoritmami riadenia a ďalšími, sekundárnymi algoritmami sú v presne danej postupnosti volané jeden za druhým v metóde *dCtrlStep()*, ktorá je súčasťou hlavného cyklu simulácie. Keďže využívame jediný cyklus, teda i jediné vlákno, vyhli sme sa problémom so synchronizáciou procesov. Môžeme teda perdpokladať, že spustenie a úspešné ukončenie metódy *dCtrlStep()* v každom cykle, možno pokladať za univerzálne riešenie momentálneho stavu robota vo vzťahu k požiadavke

Poďme sa teda pozrieť krok za krokom na metódu *dCtrlStep()*. Z hľadiska samotného riadenia možno funkciu rozdeliť na tri časti: *Preprocess, Inner Process* a *Post Process.* Každá z týchto častí má vlastnú úlohu, no môžeme povedať, že jadrom systému a jeho najdôležitejšou súčasťou je práve *Inner Process*, ktorý je zložený z analýzi okamžitého stavu a samotného riadenia. *Preprocess*, možme označiť predovšetkým ako priestor na vstupy. Naopak, *Postprocess* zase za priestor navýstupy a doplnkovú funkcionalitu.

Aby sme správne pochopili proces riadenia ako celok, pozrime sa krok za krokom na medódu *dCtrlStep()*.

```
void DrivingControl::dCtrlStep() {

   switch (newCommand) {
      case 'M,  :  { Stop = false;            Stoped = false; }     break;     // Príkaz Move – naparametrizované automatické riadenie
      case 'S'  :  { Stop = true;             Stoped = false; }     break;     // Príkaz Stop  – spustí sa brzdenie a zastaví sa ďalšie riadenie
      case 'P'  :  { ToPoint = ! ToPoint;     Circle = false;  }    break;     // Príkaz To Point – zaháji pohyb k vybranému bodu v priestore
      case 'C'  :  { Circle = ! Circle;       ToPoint = false; }    break;     // Príkaz Circle – zaháji pohyb do kruhu, podľa zadného uhla
      case 'D'  :  { moveDirection = toBeDirection;          }      break;     // Príkaz Direction – aktualizácia smeru pohybu robota
      case 'T'  :  { Terminated = !Terminated;               }      break;     // Príkaz Terminate – pozastaví analýzu a riadenie robota
   };
   newCommand = NULL;
```

Obr 4.8

Preprocess: Obrázok 4.8 zobrazuje prvú funkčnú časť každého simulačného cyklu. Preprocess je kľúčový prvok pokiaľ ide o užívateľské vstupy. Umožňuje vstupovať do prcesu riadenia, a parametrizovať vybrané premenné, prípadne vyžiadať inú špeciálnu funkcionalitu, ako napríklad príkaz *"Circle"*. Pokiaľ ide o samotnú implementáciu do programovacieho jazyka a začlenenie do procesu riadenia, jedná sa o veľmi jednoduchý spôsob interakcie s robotom.

```
void DrivingControl::dCtrlStep() {

   if( mainBallBot.getStepNumbersXYZ(true) == 0 )          {          // Doležitá podmienka – riadenie prebieha iba ak sú závažia v pokoji

      if( ! Terminated )          {                                   // Užívateľom vyžiadané prerušenie riadenia

         RotationAnalyze();                                           // Doležité analytické algoritmy

         Controls = RidersControler();                                // Doležité riadiace algoritmy – vracia úspešnosť podmienky riadenia
                                                                      // Či vôbec bolo nutné zahájiť v danom kroku riadenie.
                                                                      // Používa sa pri špeciálnych algoritmoch v Postprocesse
      };

   };
```

Obr 4.9

Innerprocess: znázornený na Obrázku 4.8 je najdôležitejším krokom v procese riadenia. Je zložený z dvoch za sebou volaných metód: *RotationAnalyze()* zasteršuje analytické algoritmy a stará sa o aktualizáciu dát, konkrétne uhlov, ktoré zvierajú jednotlivé osi s rovinou horizontu a "smerom nula", ktoré pre svoje výpočty vyžaduje metóda *RidersControler()*. Táto sa následne venuje rozhodovaniu a riadeniu, na základe presne stanovených pravidiel.

Nasleduje pohľad na algoritmy metódy *RotationAnalyze()*. Návrhu týchto algoritmov sa venovala kapitola 3, v časti s názvom Analýza okamžitého stavu robota. Pri implementácii týchto algoritmov, dochádzalo k nezrovnalostiam a nesprávnym výsledkom, preto bolo potrebné algoritmus, ošetriť niekoľkými podmienkami. Obr 4.10.

```
int integer1 = (abs(angleY) >= 90.00 ?   1 : -1) * (angleX / abs(angleX));      // použité pri úprave odchýlky na výsledný uhol
int integer2 =  angleY < 00.00 ? 180 :  0;                                      // použité pri úprave odchýlky na výsledný uhol

smallAlfa = getBasicAngle((angleX - 90) * DEGREES, (angleY - 90) * DEGREES);   // výpočet odchýlky Alfa          < - 90 ; + 90>
fullAlfa  = integer1 * (integer2 + smallAlfa);                                  // výpočet výsledného uhla Alfa  < -180; +180)


integer1 = angleX >= +90.00 ?  1 : -1;                                          // použité pri úprave odchýlky na výsledný uhol
integer2 = angleY >= +90.00 ? -1 :  1;                                          // použité pri úprave odchýlky na výsledný uhol

integer1 = angleX < -90.00 ? -integer1 : integer1;                              // použité pri úprave odchýlky na výsledný uhol
integer2 = angleY < -90.00 ? -integer2 : integer2;                              // použité pri úprave odchýlky na výsledný uhol

smallBeta = getBasicAngle((smallAlfa  - 90) * DEGREES, (angleX - 90) * DEGREES);// výpočet odchýlky Beta          < - 90 ; + 90>
fullBeta  = abs(smallBeta) * integer1 * integer2 ;                              // výpočet výsledného uhla Alfa  < - 90 ; + 90>

// metóda počíta na základe stredovej rovnice elipsy a goniometrických funkcií, odchýlko osi od jej nultej pozíciie //
dReal getBasicAngle(dReal a, dReal b) {     return atan( sqrt((1 - pow(cos(b),2)) * pow(cos(a),2)) / cos(b) ) * RADIANS;    };
```

Obr 4.10

Funkcia zobrazená na obrázku 4.10. sa zaoberá konkrétne výpočtom uhlov pre os *Z*. Pre výpočet uhlov pre osi *X* a*Y* je použitá podobná funkcia, založená na totožnom princípe počítania odchylky pomocou stedovej rovnice elipsy. Líšia s v spôsobe výpočtu výsledného uhla, čo bohužiaľ vyplýva z vlastností *eulerových uhlov*.
Ako sa neskôr pri testovaní riadiaceho systému ukázalo, práve táto druhá funkcia spôsobovala problémy pri pčítaní uhlov α a β pre osi *X* a *Y*.
Dôsledkom týchto chýb, dochádzalo miestami k nesprávnym výsledkom a následne i chybám v riadení a odchylkám v trajektórii pohybu. *V čase dokončenia tejto práce, ešte nebola závada presne definovaná a odstránená.*

Pokiaľ ide o rozhodovacie a riadiace algoritmy metódy *RidersControler()*, nevyskytli sa v procese implementácie a testovania žiadne zásadné problémy. V súlade z návrhom z kapitoly 3, časť ***Algoritmy Riadenia***, bolo možné jednoducho implementovať základné podmienky a následé výpočty. Zdrojové kódy sú príliž rozzsiahle a k nahliadnutiu v prílohe práce. Preto nasleduje iba stručná postupnosť krokov, rozhodovacích a riadiacich algoritmov, tak ako boli navrhnuté a následne implementované v metóde *RidersControler()*.

RidersControler( ) {

1.) výpočet odchýlky uhla α pre os XYZ

difrXYZ *= (α - moveAngle)* *// moveAngle = uhol pohybu*

// vzdy pre vsetky osi (difrX, difrY, difrZ)
2.) výpočet orientácie osi XYZ k semru pohybu *// ktorým smerom sa posúvajú závažia*

directXYZ *= +1 ak |difrXYZ| <= 90;* *// analogicky vzdy pre vsetky osi*
directXYZ *= -1 ak |difrXYZ| > 90;*

```
3.) podmienka pohybu / zastavenia                    // chcem zahájiť pohyb, alebo brzdenie?

        ak Stop == true
        4.)     vyber os s najvacsim | β |           // predpoklad najrýchlejšieho zastavenia
                pošli závažia tejto osi na maximálnu pozíciu v smere (- directXYZ)

        ak Stop == false
        4.) podmienka riadenia              // je potrebna zmena polohy zavazi?

                ak je (β * directXYZ) osi označenej ako hlavná > 45° ?

                5.) označ os s najmenším (β * directXYZ) ako hlavnú

                6.) posuň závažia na hlavnej osi maximum v smere directXYZ

                7.) označ os s opačným difrXYZ, ako má hlavná os za vedľajš
                //prioritne os s (β * directXYZ) < 0

                8.) počítaj a vhodne vychýľ závažia na vedľajšej osi.

                9.) nastav závažia tretej osi na vychodziu - nulovú pozíciu.

};
```

Postprocess: priestor určený pre zobrazovanie dodatočných výstupov, ale predovšetkým pre aplikáciu doplnkových algoritmov. Na obrázku 4.11 vidieť príklad dvoch takýchto jednoduchých algoritmov. Pohyb po kružnici, alebo presun na zvolenú pozíciu.

```
const dReal *pos = mainBallBot.getRobotPosition();                  // poloha robota v priestore [X,Y,Z]

if ( Circle )   {                                                   // pohyb po kruznici

   MainOutput  << pos[0] << "|" << pos[1] << "\n";                  // výstup do súboru
   cout        << pos[0] << " " << pos[1] << "\n";                  // výstup na terminál

   moveDirection = moveDirection + ANGLE_STEP_SIZE;                 // mení veľkosť uhla podľa zadaných parametrov
   moveDirection = moveDirection > 180 ? moveDirection - 360 : moveDirection;      //úprava na formát <-180;180)

} else if ( ToPoint )   {                                           // pohyb k zvolenému bodu

   MainOutput  << pos[0] << "|" << pos[1] << "\n";                  // výstup do súboru
   cout        << pos[0] << " " << pos[1] << "\n";                  // výstup na terminál

   dReal a = sqrtf( (pos[0] - TO_POINT_X) * (pos[0] - TO_POINT_X) );    // vzdialenosť robota od nového bodu po osi X
   dReal b = sqrtf( (pos[1] - TO_POINT_Y) * (pos[1] - TO_POINT_Y) );    // vzdialenosť robota od nového bodu po osi Y

   if( (a <= 1) && (b <= 1) ) {  newCommand = 'S';       ToPoint = false;  }     // úspech a zastavenie
   else                          moveDirection = atan( (b / a) * 1 ) * RADIANS;  // prepočítavanie smeru pohybu

} else   {

   MainOutput  << pos[0] << "|" << pos[1] << "\n";                  // výstup do súboru
   cout                         << pos[0] << " " << pos[1] << "\n";  // výstup na terminál

};
```

Kapitola 5

Záver

V tejto kapitole nasleduje zhrnutie výsledkov nášho snaženia na základe skúseností a dát získaných počas testovania. Zhodnotenie doterajšieho prístupu vzhľadom na tieto výsledky. Prehľad výhod a nevýhod nami implementovaného riešenia a plány budúcich riešení.

5.1 Testovanie

Pri testovaní sme sa sústredili na kľúčové vlastnosi požadované od mobilného robota. Predovšetkým presnosť a plynulosť pohybu. Rýchlosť pohybu a schopnosť zastaviť. Následne na obmedzenia v teréne, ako napríklad pohyb po naklonenej rovine a niektoré špeciálne spôsoby pohybu, ako napríklad pohyb po kružnici.

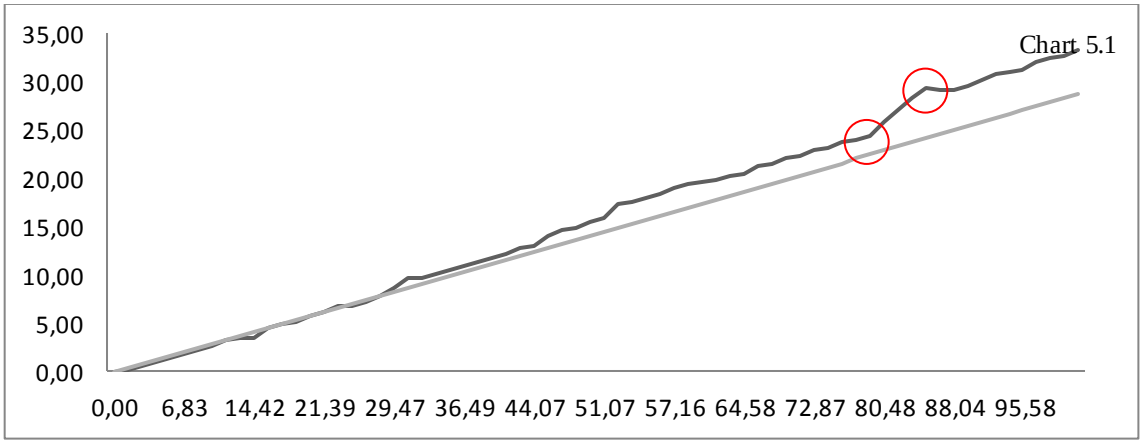
5.1.1 Plynulosť pohybu

Na základe pozorovaní možno konštatovať, že pohyb robota je vo väčšine prípadov plynulý a udržiava si konštantnú rýchlosť. Miestami však dochádza k istej forme útlmu zo strany robota. Robot náhle spomalí, stráca zotrvačnosť a automatizovaný pohyb ustane. Dochádza k úplnej pasivite robota a je potrebné zo strany užívateľa vynútiť reštart analityckého algoritmu.
Predpokladáme, že tento stav vzniká v dôsledku nedostačne optimalizovanej ***Podmienky Riadenia***, definovanej v kapitole Návrh. Táto domienka sa potvrdila predovšetkým pri pokusoch o pohybe po naklonenej rovine.

Zistili sme, že ***Podmienky Riadenia*** nie je ani zďaleka optimalizovaná pre takýto druh pohybu. Vzhľadom na tento fakt sme upustili od akých koľvek ďalších testov na šikmej plošine. V konečnom dôsledku však možno konštatovať, že robot si napriek častému prenášaniu ťažiska udžiava plynulý pohyb.

5.1.2 Presnosť pohybu

Už od návrhu teoretického modelu sme spomedzi všetkých vlastností robota, kládli najväčší dôraz predovšetkým na presnosť pohybu a možno konštatovať, že robot nedopadol najhoršie. Naneštasie sme zistili, že v procese analýzy dochádza k chybám, ktoré vyzerajú byť pomerne náhodné. Je pravdepodobné, že **analytický algoritmus** sa v niektorých, zatiaľ v bližšie nešpecifikovaných prípadoch dopúšťa chyby vo výpočte a následne ponúka riadiacemu algoritmu nekorektné dáta, čoho výsledkom sú neočakávané odchylky zo smeru pohybu. Pozri Chart 5.1. a Chart 5.2



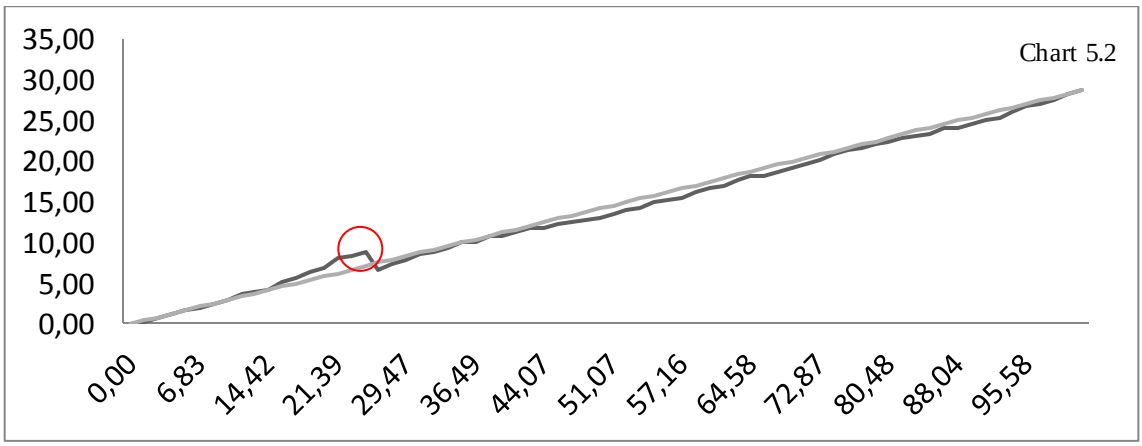


Chart 5.1 reprezentuje pokus o priamočiary pohyb, bez kontroly pozície. (move)

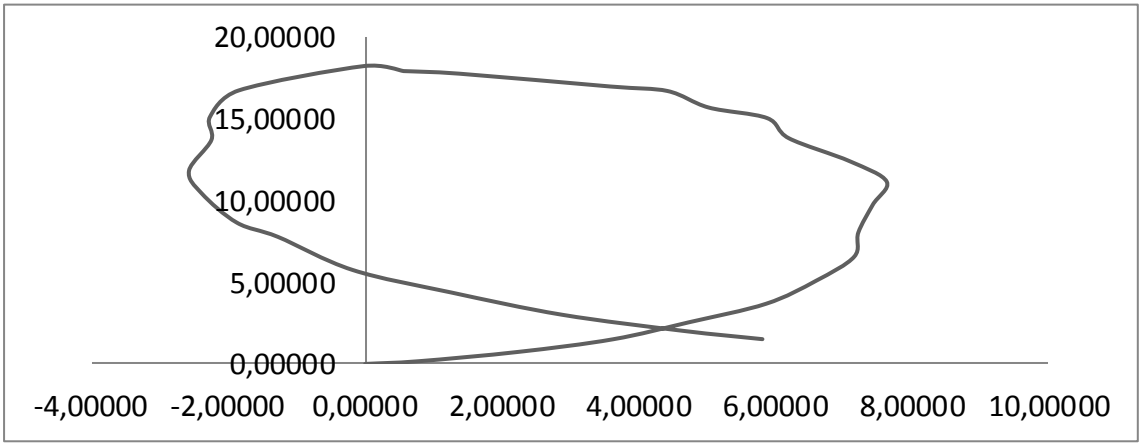
Chart 5.2 reprezentuje pokus o priamočiary pohyb, so sekundárnou kontrolou pozície. (move to point)

5.1.3 Brzdenie

Keďže sme upustili od meraní naklonenej plošine, samotné brzdenie prešlo iba jednoduchým testovaním, ktoré ukázalo, že najdlhšia možná brzdná dráha, respektíve odchylka, pokiaľ hovoríme o rýchlosti, ktorú dokáže robot vyvinúť bez pomoci akýchkoľvek vonkajších síl sa rovná 1/8 obvodu robota, čo pokladáme za dobrý výsledok.

5.1.4 Pohyb po kružnici

Vzhľadom na to, že sa nejedná priamo o funkčnosť riadiaceho systému robota, ale skôr o čo najľahšie implementovateľný experiment, nepokladáme slabý výsledok za osobné sklamanie, alebo nenaplnenie cieľou práce, ale skúr za fakt. Chart 5.3



5.2 Záver

Z osobného hľadiska považujem hlavné cieľe tejto práce za splnené. Po podrobnej analýze sa podarilo navrhnúť a v počítačovej simulácii úspešne implementovať funkčný model riadenia guľového robota.

Že sa v rámci testovania ukázali slabé miesta použitých algoritmov, nepokladám za neúspech, ale skôr za výzvu do budúcna.

Za čiastočný neúspech však pokladám absenciu podrobnejšej technickej, ale aj funkčnej špecifikácie, ktoré by dali priestor rýchlejšiemu vývoju a znásobili použiteľnosť tejto práce.

Dodatok A

Príklady vybraných typov guľových robotov:

- 1. IDU = inside driving unit. Príklady konkrétnej implementácie:
  - SAR[1]
  - SAR 2[2]
  - ME 189 – Robotic Drive[3]

- 2. Systém kyvadla, oba používané typy:
  - The Orb[4]
  - Rotundus The Ground Bot[5]
  - IEEE VR 2009 - Spherical remote-controlled robot[6]
  - Picorover[7]
  - Robot vyrobený pomocou Lego Mindstorms NXT[8]
  - Rollo[9]

- Gyroskop:
  - A Single-Wheel, Gyroscopically Stabilized Robot – GYROVER[10]

1 YOUTUBE. *SAR.* [online] YouTube.com. [Cit. 19.11.2010] Dostupné na internete:<http://www.youtube.com/watch?v=c9Uph7oyV8s&feature=related >

2 YOUTUBE. *SAR2.* [online] YouTube.com. [Cit. 9.11.2010] Dostupné na internete:<http://www.youtube.com/watch?v=6pjgq2wq0Do&NR=1 >

3 YOUTUBE. *ME189 – Robotic Drive.* [online] YouTube.com. [Cit. 16.11.2010] Dostupné na internete:<http://www.youtube.com/watch?v=8yPc524_ASI&NR=1>

4 PISCSART.NET. *The Orb.* [online] Picsart.net. [Cit. 8.12.2010] Dostupné na internete:<http://www.picsart.net/orb/index.html>

5 ROTUNDUS.SE [online] Rotundus.se. [Cit. 8.10.2010] Dostupné na internete:<http://www.rotundus.se/>

6 YOUTUBE. *IEEE VR 2009 - Spherical remote-controlled robot.* [online] YouTube.com. [Cit. 29.11.2010] Dostupné na internete:<http://www.youtube.com/watch?v=lGvYJzfpfG0&feature=related>

7 WIKI.XPRIZE. *Picorover.* [online] [Cit. 16.12.2010] Dostupné na internete:<http://wiki.xprize.frednet.org/index.php/Portal:Picorover>

8 VÖLKER, N. [online] nilsvolelker.com [Cit. 30. 9. 2010] Dostupné na internete: <http://www.nilsvoelker.com/nxt/spherical/index.html>

9 AALTO UNIVERSITY SCHOOL OF ELECTRICAL ENGINEERING. *Rollo.* [online] autsys.tkk.fi. [Cit. 18.10.2010] Dostupné na internete:http://autsys.tkk.fi/en/Rollo

10 A Single-Wheel, Gyroscopically Stabilized Robot - GYROVER. [online] [Cit. 18.10.2010] Dostupné na internete:< http://www.cs.cmu.edu/~cyberscout/gyrover.html>

Použité zdroje

Knižné zdroje

ENGLER, O., RANDLE, V. 2010. *Introduction to Texture Analysis:* Macrotexture, Microtexture and Orientatinon Mapping. 2nd ed. Boca Raton: Taylor and Francis Group, LLC, 2010. ISBN 978-1-4200-6365-3.

Elektronické zdroje

ALVES, J., DIAS, J.: *Design and control of a spherical mobile robot*. 2003. [online]. 11 p. [Cit. 16. 11. 2010] Dostupné na internete: <http://mail.isr.uc.pt/~mrl/admin/upload/18.pdf>

HALME, A. - SUOMELA, J. - SCHÖNBERG, T. - WANG, Y.: *A Spherical Mobile Micro-Robot for Scientific Applications*. [online]. [Cit. 12. 5. 2011] Dostupné na internete: <http://www.isr.uc.pt/~jalves/astrpap2.pdf>

MING, Y. - ZONGQUAN, D. - XINYI, Y. - WEIZHEN, Y.: Introducing HIT Spherical Robot: Dynamic Modeling and Analysis Based on Decoupled Subsystem. 2006. [online]. [Cit. 7. 11. 2010]Dostupné na internete: <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4141861>

MUKHERJEE, R. 2001. *Spherical Mobile Robot*. US Patent 6,289,363, September 11, 2001. [online]. [Cit. 14. 10. 2010] Dostupné na internete: <http://www.freepatentsonline.com/6289263.pdf>

SMITH, R. 2004. *Dynamics Simulation:* A whirlwind tour. [online]. [Cit. 30. 1. 2011] Dostupné na internete: <http://www.ode.org/slides/parc/dynamics.pdf>

VÖLKER, N. [online] nilsvolelker.com [Cit. 30. 9. 2010] Dostupné na internete: <http://www.nilsvoelker.com/nxt/spherical/index.html>

VÖLKER, N. [online] nilsvolelker.com [Cit. 30. 9. 2010] Dostupné na internete: <http://www.nilsvoelker.com/nxt/spherical/programming.html>

VYBÍRAL, B.: *Kinematika a dynamika tuhého tělesa:* Studijní text pro řešitele FO a ostatní zájemce o fyziku. [online]. 67 s. [Cit. 10. 4. 2011] Dostupné na internete: <http://fo.cuni.cz/texty/dynamika.pdf>

AALTO UNIVERSITY SCHOOL OF ELECTRICAL ENGINEERING. *Rollo*. [online] autsys.tkk.fi. [Cit. 18.10.2010] Dostupné na internete:<http://autsys.tkk.fi/en/Rollo>

MECHANICALMUTANTS.COM. *Schollarballs*. [online] mechanicalmutants.com. [Cit. 13.12.2010] Dostupné na internete:<http://mechanicalmutants.com/index.php?p=sb>

PISCSART.NET. *The Orb*. [online] Picsart.net. [Cit. 8.12.2010] Dostupné na internete:<http://www.picsart.net/orb/index.html>

POPSCI THE FUTURE NOW. *Ground Bot*. [online] Popsci.com. [Cit. 18.10.2010] Dostupné na internete:<http://www.popsci.com/bown/2008/product/ground-bot>

ROBOSEL. *Spherical micro-robots could explore Mars*.[online] Robosel.blogspot.com. [Cit. 18.12.2010] Dostupné na internete:<http://robosel.blogspot.com/2009/03/spherical-micro-robots-could-explore.html>

ROTUNDUS.SE [online] Rotundus.se. [Cit. 8.10.2010] Dostupné na internete:<http://www.rotundus.se/>

TECHBRIEFS.COM. *Alternative Way of Shifting Mass To Move a Spherical Robot.* [online] Techbriefs.com. [Cit. 3.11.2010] Dostupné na internete:<http://www.techbriefs.com/component/content/article/854>

TECHNOVELGY.COM WHERE SCIENCE MEETS FICTION. *Rotundus And Rover: Robotic And Fictional Guardians (Updated)*. [online] Technovelgy.com. [Cit. 6.11.2010] Dostupné na internete:<http://www.technovelgy.com/ct/Science-Fiction-News.asp?NewsNum=332>

WIKI.XPRIZE. *Picorover*. [online] [Cit. 16.12.2010] Dostupné na internete:<http://wiki.xprize.frednet.org/index.php/Portal:Picorover>

Záverečné práce

NAGAI, M. 2008. Control System for a Spherical Robot. Luleå University of Technology, 2008. ISSN: 1653-0187 [online]. 108 p. [Cit. 8. 1. 2011] Dostupné na internete: <http://epubl.ltu.se/1653-0187/2008/116/LTU-PB-EX-08116-SE.pdf>

[Oum09] OUMER, N. W. 2009. *Development of Wireless Control System for a Spherical Robot*. Luleå University of Technology, 2009. ISSN: 1653-0187 [online]. 102 p. [Cit. 15. 1. 2011]Dostupné na internete: <https://pure.ltu.se/ws/files/31139928/LTU-PB-EX-09079-SE.pdf>

Audiovizuálny materiál

YOUTUBE. *SAR*. [online] YouTube.com. [Cit. 19.11.2010] Dostupné na internete:<http://www.youtube.com/watch?v=c9Uph7oyV8s&feature=related >

YOUTUBE. *SAR2*. [online] YouTube.com. [Cit. 9.11.2010] Dostupné na internete:<http://www.youtube.com/watch?v=6pjgq2wq0Do&NR=1 >

YOUTUBE. *ME189 – Robotic Drive*. [online] YouTube.com. [Cit. 16.11.2010] Dostupné na internete:<http://www.youtube.com/watch?v=8yPc524_ASI&NR=1>

YOUTUBE. *Documentary about Rotundus GroundBot. Interview with Viktor Kaznov, CTO at Rotundus*. [online] YouTube.com. [Cit. 1.12.2010] Dostupné na internete:<http://www.youtube.com/watch?v=oMy-H74g-yw&NR=1 >

YOUTUBE. [online] YouTube.com. [Cit. 19.12.2010] Dostupné na internete:<http://www.youtube.com/watch?v=ZcSb9F2Xn10&feature=related >

YOUTUBE. *Rotundus spherical robot*. [online] YouTube.com. [Cit. 1.11.2010] Dostupné na internete:<http://www.youtube.com/watch?v=4zXXyqqLy0o&feature=related>

YOUTUBE. *Rotundus GroundBot runs at a parking lot in NY*. [online] YouTube.com. [Cit. 4.11.2010] Dostupné na internete:<http://www.youtube.com/watch?v=0LPzSgxVkdY&NR=1 >

YOUTUBE. *Rotundus GroundBot water tests*. [online] YouTube.com. [Cit. 7.12.2010] Dostupné na internete:<http://www.youtube.com/watch?v=t_UIJ3d8XNI >

YOUTUBE. *Robot for navigating dangerous terrain*. [online] YouTube.com. [Cit. 12.1.2011] Dostupné na internete:<http://www.youtube.com/watch?v=ML2UjOeiZDw&feature=player_embedded# >

YOUTUBE. *Spherical robot climbing out of a hole*. [online] YouTube.com. [Cit. 20.11.2010] Dostupné na internete:<http://www.youtube.com/watch?v=OWkK-o4Vq-A&feature=player_embedded >

YOUTUBE. *Spherical robot climbing up a step*. [online] YouTube.com. [Cit. 29.11.2010] Dostupné na internete:<http://www.youtube.com/watch?v=mecsepWXuuc&feature=related>

YOUTUBE. *IEEE VR 2009 - Spherical remote-controlled robot*. [online] YouTube.com. [Cit. 29.11.2010] Dostupné na internete:http://www.youtube.com/watch?v=lGvYJzfpfG0&feature=related