SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA CHEMICKEJ A POTRAVINÁRSKEJ TECHNOLÓGIE

EVIDENČNÉ ČÍSLO: FCHPT-113709-82056

PLÁNOVANIE A SLEDOVANIE TRAJEKTÓRIE PRE ROBOTICKÉ SYSTÉMY

BAKALÁRSKA PRÁCA

2018 **Roman Kohút**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA CHEMICKEJ A POTRAVINÁRSKEJ TECHNOLÓGIE

Evidenčné číslo: FCHPT-113709-82056

PLÁNOVANIE A SLEDOVANIE TRAJEKTÓRIE PRE ROBOTICKÉ SYSTÉMY

BAKALÁRSKA PRÁCA

Študijný program: automatizácia informatizácia a manažment v chémii a potravinárstve
Študijný odbor: 5.2.14. automatizácia, 5.2.52. priemyselné inžinierstvo
Školiace pracovisko: Ústav informatizácie, automatizácie a matematiky (FCHPT)
Vedúci záverečnej práce/školiteľ: doc. Ing. Michal Kvasnica, PhD.
Konzultant: doc. Ing. Michal Kvasnica, PhD.

Bratislava 2018 **Roman Kohút**

Slovenská technická univerzita v Bratislave
Ústav informatizácie, automatizácie a matematiky

Fakulta chemickej a potravinárskej technológie
Akademický rok: 2017/2018
Evidenčné číslo: FCHPT-113709-82056

STU FCHPT

ZADANIE BAKALÁRSKEJ PRÁCE

Študent:	**Roman Kohút**
ID študenta:	82056
Študijný program:	automatizácia, informatizácia a manažment v chémii a potravinárstve
Kombinácia študijných odborov:	5.2.14. automatizácia, 5.2.52. priemyselné inžinierstvo
Vedúci práce:	doc. Ing. Michal Kvasnica, PhD.

Názov práce: **Plánovanie a sledovanie trajektórie pre robotické systémy**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Cieľom práce je navrhnúť a implementovať systém na monitorovanie a riadenie pohybu robotických vozidiel v rovine. Študent využije systém vizuálneho rozpoznávania polohy bodov, pričom vytvorí rozhranie na napojenie tohto systému do Matlabu. Následne navrhne a implementuje regulátory na sledovanie trajektórie pohybu tak, aby sa robot presunul do cieľovej lokácie. Tento systém ďalej rozšíri na plánovanie a sledovanie bezkolíznej trajektórie v prítomnosti prekážok a viacerých robotických vozidiel.

Rozsah práce: 30

Riešenie zadania práce od: 12. 02. 2018

Dátum odovzdania práce: 06. 05. 2018

Roman Kohút
študent

prof. Ing. Miroslav Fikar, DrSc.
vedúci pracoviska

prof. Ing. Miroslav Fikar, DrSc.
garant študijného programu

Poďakovanie

Na tomto mieste by som sa chcel poďakovať ľuďom, ktorí mi výraznou mierou pomohli pri vytváraní mojej bakalárskej práce. Bez ich pomoci by som nezískal potrebné vedomosti pre jej dokončenie. Za materiálovo technologickú podporu ďakujem nášmu ústavu informatizácie, automatizácie a matematiky. Vďaka patrí aj môjmu vedúcemu práce, doc. Ing. Michalovi Kvasnicovi, PhD. za odborné vedenie a cenné rady. Za pripomienky vopred ďakujem aj svojmu oponentovi.

Abstrakt

Cieľom bakalárskej práce je navrhnúť a implementovať systém na monitorovanie pohybu robota v rovine. Ovládané budú jednoduché robotické vozidlá napájané Arduinom Yún. Na monitorovanie objektov v priestore sa použil systém senzora OptiTrack. V rámci tohoto systému a počítača vytvoríme server NatNetSDK, ktorý vysiela a následne spracováva údaje o polohe a natočení robotov. Server spracujeme v Matlabe, ďalej bude vysielať požadované údaje do Arduina cez wifi sieť. Celé toto rozhranie bolo popísané v prvých dvoch kapitolách práce. Vyhradenú kapitolu v práci má implementácia PID regulátora do algoritmu a popis riešenia, ako zabezpečiť presun robota do cieľovej lokácie. Posledná kapitola bola venovaná nájdeniu vhodných metód na vyriešenie bezkolíznej trajektórie v dynamickom prostredí s prekážkami a ich implementácii v Matlabe. V práci boli použité dve metódy na vyriešenie daného problému: Metóda pravdepodobnostného plánovania a heuristická metóda pod názvom A* („A star“). Algoritmus pre plánovanie cesty bol rozšírený o prostredie s viacerými robotmi, ktorý sa pohybujú bezkolízne v jednej miestnosti. Všetky algoritmy použité pre plánovanie boli naprogramované v Matlabe. Riešenie navrhnuté v tejto práci je jednoducho ovládateľný systém jedného alebo viacerých robotov, ktorý sa bezkolízne pohybujú v známom prostredí.

Kľúčové slová: Matlab; implementácia; regulátor; algoritmus; bezkolízna trajektória

Abstract

The aim of the bachelor project is to design and implement a system for tracking robot movement in a plane. We will control simple robotic vehicle powered by Arduino Yun. The OptiTrack system is a sensor which was used for space monitoring. A NatNetSDK server was created between the camera and the computer, this server broadcast and then process the position and rotation of robots. The server we process in Matlab further it will transmit the requested data to Arduin over the wifi network. This entire interface was described in the first two chapters. One of these chapter was focused on implementation of the PID controller into the algorithm and description of the solution how to ensure transmition of the robot to the target location. In the last chapter we looked for suitable methods for solving the noncolliding trajectory in dynamic environment whit obstacles. We have also looked for implementation these methods in Matlab. There were used two methods for solving this problem: Probabilistic Roadmap Method and Heuristic Method A* called „A star “. The algorithm for path planning was extended to a multi-robot environment that move in a single-room without collision. All algorithm used for path planning were implemented in Matlab. The solution proposed in this project is a simple-to-use system of one or more robots that move without collision in a known environment.

Key words: Matlab; implementation; controller; algorithm; noncolliding trajectory

Obsah

Úvod

Súčasný stav automatizácie priemyslu potvrdzuje, že robotizácia sa stáva súčasťou automatizácie výrobných aj nevýrobných procesov. Pod pojmom automatizácia rozumieme modernizovanie prevádzok, nahrádzanie starých zaužívaných postupov novými, modernejšími a hlavne efektívnejšími. V mnohých prípadoch automatizácia zahŕňa využitie robotov tam, kde človek nemôže pracovať , alebo pracuje neefektívne. Napriek tomu, že robotické pracoviská sú známe presnosťou, rýchlosťou a konštantnou kvalitou výstupov, nahradenie ľudského faktora nie je vždy bezpečná voľba. Dôvodom tohto stavu je, že robotika sa stále ešte nachádza na začiatku svojho rozvoja. Nemožno zaradiť roboty do bežného života tam, kde sa vyskytuje veľa nepredvídateľných faktorov a kde jediná chyba môže ohroziť okolie. Napriek tomu môžeme konštatovať, že prostredie, ktoré je určené pre roboty, sa dá v praxi veľmi dobre pripraviť, a tým vytvoriť vhodné podmienky pre precízne pracujúci systém. Pod takýmto prostredím rozumieme napríklad veľké sklady, otvorené plochy bez voľného pohybu osôb, ale aj prostredie s podmienkami nevhodnými pre ľudí ako sú radiačné, horiace a kontaminované priestory a iné im podobné.

V nami predloženej práci sa budeme zaoberať pohybom jedného či viacerých robotov v priestore vhodnom na ich zaradenie. Naše okolie bude predstavovať miestnosť s prekážkami, ktoré budeme poznať. Na základe informácií o prostredí a polohe robota budeme hľadať vhodnú trajektóriu na presun robota z počiatočného miesta do cieľa. Cesta musí byť navrhnutá tak, aby počas nej robot nenarazil na statické ani dynamické prekážky.

V poslednej časti práce sa budeme venovať reálnejšej situácii, v rámci ktorej sa v prostredí nepohybuje iba jeden, ale aj viacero robotov. Tieto roboty budú pracovať nezávisle od seba v rovnakom priestore a pohyby nimi vykonávané budú bezkolízne.

Jednou z najdôležitejších otázok, ktoré práca rieši je, akým spôsobom budú lokalizované prekážky a ako bude monitorovaný pohyb robota v priestore. Jedným z vhodných riešení môže byť senzor OptiTrack. Tento senzor bol vyvinutý na sledovanie retroreflexívnych materiálov, ktoré zachytáva ako body s presnou polohou a vďaka tomu vie poskytovať informácie o natočení v zadefinovanej súradnicovej sústave a mnohé iné potrebné údaje.

Prvá časť práce je venovaná predstaveniu zvoleného senzora a jeho spôsobov použitia. Ďalej bude nutné vytvoriť neustálu komunikáciu medzi robotickými vozidlami a senzorom OptiTrack. Túto komunikáciu nie je možné vytvoriť priamym spôsobom. Dáta z kamery sa dajú spracovať len pomocou zložitého programu, ktorý je možné si nainštalovať do počítača. Nutné bude teda

použiť počítač ako medzičlánok komunikácie. Dáta prijaté do počítača sa spracujú a následne upravia.

Druhá kapitola je venovaná práve tejto komunikácií. Zaoberáme sa v nej opisom vytvorenia servera a prekladom údajov z počítača (jazyk Matlab) do robotického vozidla (jazyk C). Robotické vozidlo predstavuje robota, ktorý má dve zadné klesá. Každé z nich je poháňané vlastným motorom. Tento robot je napájaný mikroovládačom od Arduina, v ktorom beží algoritmus pre pohyb. Pri pohybe robota je potrebné naň aplikovať určitú formu riadenia, ktorá zabezpečí, že robotické vozidlo príde čo najpresnejšie a bez vybočenia z dráhy k súradniciam, ktoré mu boli zaslané. V našom prípade toto riadenie predstavujú P,PI a PID regulátory. Riadenou veličinou bude uhol natočenia robota a riadiacou veličinou je zmena uhlovej rýchlosti kolies.

V tretej kapitole je opísaný princíp pohybu robotického vozidla a spôsob implementácie PID regulátora do algoritmu.

V poslednej časti práce je navrhnutý algoritmus, ktorý vytvorí bezkolíznu trajektóriu, vhodnú na pohyb robota v priestore s prekážkami. Rozhodli sme sa použiť dve metódy, porovnať ich a následne vyhodnotiť, ktorá je vhodnejšia pre náš systém. Prvá metóda je založená na náhodnom generovaní bodov v priestore a druhá je založená na heuristickom hľadaní optimálnej trajektórie v rámci stavového priestoru robota. Pomocou metódy, ktorú sme vybrali ako vhodnejšiu, navrhneme a implementujeme algoritmus na plánovanie trajektórie viacerých robotov. Táto trajektória bude tiež bezkolízna, založená na prioritách pridelených robotom. V rámci tohto zadania si roboty nebudú navzájom rovnocenné. Ak to budú okolnosti vyžadovať, podľa zvoleného kritéria, sa bude rozhodovať medzi obchádzaním robota s vyššou prioritou, alebo zastavením robota s menšou prioritou, kým sa trajektória neuvoľní. Všetky plánovacie algoritmy budú programované v Matlabe.

1 Monitorovanie pohybu

V tejto práci budeme na sledovanie pohybu robota a detekciu prekážok v priestore využívať systém OptiTrack. Je to jediný senzor, s ktorým budeme pracovať. Bude nám poskytovať všetky potrebné údaje o polohe a natočení v nami zadefinovanej súradnicovej sústave. Zároveň nám ich bude poskytovať v reálnom čase, aby sme s nimi mohli ďalej narábať.

1.1 Zoznámenie so senzorom OptiTrack

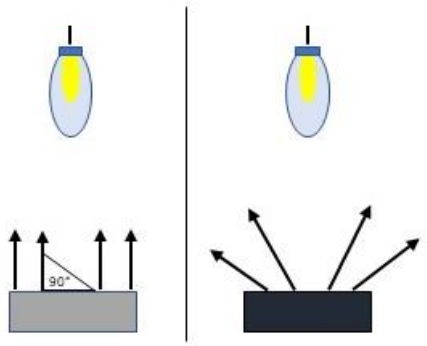
OptiTrack je systém jednej alebo viacerých kamier, ktoré sú prepojené a navzájom zosynchronizované tak, aby vytvorili čo najpresnejšie zachytený obraz pohybu v priestore. Kamery sa môžu vyskytovať vo viacerých variantoch. Najpresnejšie je rozdelenie po jednom oku kamery okolo celej miestnosti, ktoré zabráni vzniku slepého bodu či ovplyvneniu rušivými vplyvmi okolia.

Produkty od OptiTracku majú širokú škálu využitia, používajú sa napríklad pri vytvorení virtuálnej reality, v ktorej predstavuje senzor motoriky a prenáša ho do virtuálneho sveta. Ďalej sa využíva na skúmanie pohybu človeka ako aj zvierat. Napriek tomu, že má jednoduché užívateľské rozhranie, výsledky sú veľmi presné. Precíznosť systému OptiTrack má vysoké využitie aj v robotike. S vyspelejšími technológiami rástli aj požiadavky na ich chod. Použitie OptiTracku môže významne uľahčiť prácu s robotmi. Poskytuje všetky potrebné údaje v 6DoF (skratka z angličtiny „six degrees of freedom“), napríklad o polohe, natočení v reálnom čase. Tieto údaje sú veľmi spoľahlivé, najlepšia kamera dokáže presne detegovať 16 mm značku až na vzdialenosť 30,5 m. Na základe týchto vlastností budeme môcť vytvoriť systém na monitorovanie robota [1].

OptiTrack taktiež ponúka kamery, ktoré dokážu snímať 6DoF bez nutnosti rozmiestnenia kamier okolo celej miestnosti. V nami riešenom prípade budeme využívať práve túto kameru, ktorá bude schopná v rámci priestoru vytvoriť dostatočne presný obraz o našom systéme a dokáže poskytovať vhodné údaje, ktoré budeme ďalej využívať. Komunikácia kamery OptiTrack s počítačom prebieha pomocou vysokorýchlostného USB kábla. Informácie sa zobrazujú v originálnom softvéri, odtiaľ sa ďalej môžu „streamovať“ do rôznych programovacích jazykov po serveri.

1.1.1 Retroreflexívna značka

Retroreflexívna značka je vyrobená z takého materiálu, ktorý dokáže odrážať svetlo s minimálnou odchýlkou odrazu. Pre OptiTrack je tento materiál veľmi dôležitý, keďže je potrebný na lokalizovanie ich značiek pomocou jednoduchého svetla. Zachytený odraz sa používa pre výpočet 2D pozície značky, a táto pozícia sa pomocou Motívu prepočíta na 3D pozíciu cez rekonštrukciu [2].

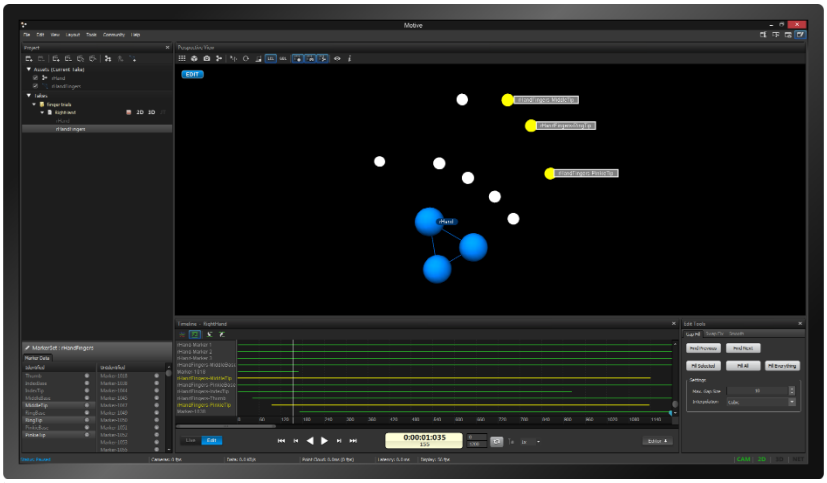


Obrázok 1: Princíp retroreflexívneho materiálu

Na obrázku č.1 môžeme vidieť ako sa líši odrážanie svetla od retroreflexívneho (vľavo) a bežného (vpravo) materiálu.

1.1.2 Motive: Tracker

Motiv:Tracker je softvér na zachytávanie optického pohybu vyvinutý pre systém OptiTrack. Navrhnutý bol tak, aby vyhovoval najnáročnejším technickým a vedeckým aplikáciám na sledovanie objektov v 6DoF, s čo najväčšiu presnosťou. Podporuje prácu v reálnom čase alebo off-line režime. Motive:Tracker umožňuje pridaním vlastných značiek do programu manuálne označovanie trajektórie malých alebo neexistujúcich pevných bodov či značiek. Každý bod sa v rámci „rigid body“ dá označiť a sledovať zvlášť. Nasledovný obrázok je z prostredia programu Motiv kde si môžeme všimnúť v rámci ruky zadefinované existujúce a umelé body.



Obrázok 2: Motiv vlastné značky

Prameň: [3]

Takisto je možné veľmi jednoducho synchronizovať video s 3D prekrytím. Pridaním ľubovoľnej sledovacej kamery do referenčnej skupiny videí. V správcovi kamier je možné nastaviť ju na plnoformátovú, kalibrovanú, synchronizovanú kameru s podporou 3D prekrytia [3].

1.1.3 Nastavenie hardvéru

Pre najlepšie „trackingové“ výsledky je nutné vyčistiť kamerou zachytávaný priestor. Pred nastavením systému je potrebné odstrániť prekážky, ktoré by mohli zabraňovať výhľadu kamery. Pri používaní kamery v budove musia byť zakryté otvorené okná a minimalizované prichádzajúce slnečné svetlo. Potrebné je vyhnúť sa inštalácii systému na lesklom a reflexnom podklade, pretože sa bude odrážať LED svetlo z kamier. Predmety s reflexnými povrchmi alebo svietiacimi prvkami by mali byť odstránené alebo zakryté nereflexnými materiálmi, aby sa zabránilo vonkajším odrazom. Kamera V120: Duo nepotrebuje zvláštne nastavenia a kalibráciu, všetko je vopred nastavené, stačí ju umiestniť do ľubovoľnej výšky tak, aby zachytávala potrebný priestor pred ňou [4].

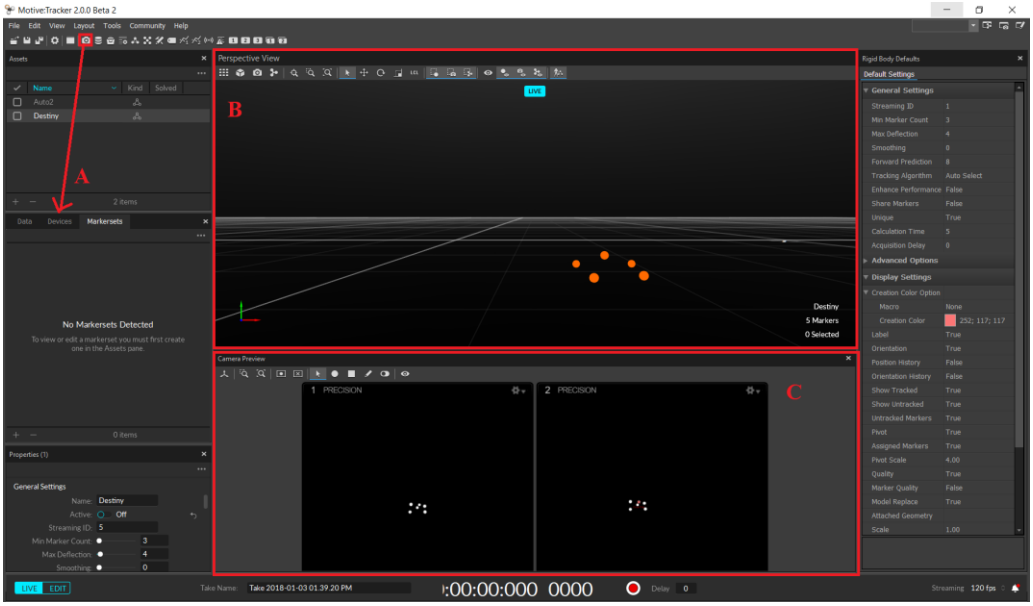
1.1.4 Orientácia v programe Motiv: Tracker

Na riešenie nami stanovenej úlohy bude nutné stiahnuť si softvér od OptiTracku pre komplexný pohyb objektov v priestore a tým je Motiv: Tracker. Najnovšia verzia Motive 2.0 sa nachádza na originálnej stránke OptiTracku, odkaz http://www.optitrack.com/products/motive/. Stiahnuť sa

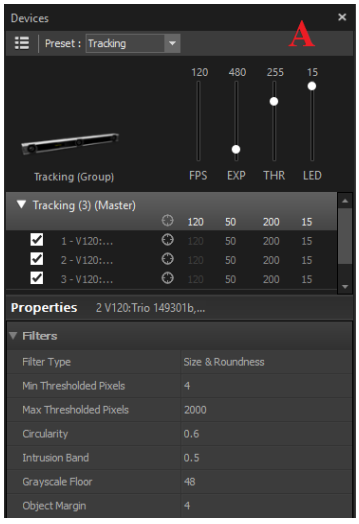
dá zadarmo, ale je potrebná spoplatnená licencia na spustenie programu. V našom prípade sa licencia automaticky potvrdí po pripojení kamery V120: Duo k počítaču pomocou USB kábla. Ďalej je potrebné stiahnuť si NatNet SDK, čo je bezplatné, vopred naprogramované rozhranie server/klient pre rôzne programovacie jazyky ako je napríklad aj Matlab. K dispozícii sa nachádza na tej istej stránke ako Motiv.

Po prvom spustni Motivu sa zobrazí okno „Quick start", pomocou tohto okna môžeme rýchlo začať so zadávaním konkrétnych úloh. Podľa nastavenia sa začne kalibrácia kamery, v našom prípade už je kamera vopred nakalibrovaná. Podľa potrieb sa dá táto kalibrácia zmeniť, popisu tohto postupu sa budeme venovať v ďalšej časti práce. Po zrušení okna „Quick start", je možné vidieť hlavné okno programu, viď. obrázok č.3. Zobrazíme si panel s nástrojmi obrázok č.3-A a obrázok č.4, v ktorom sa nachádzajú všetky pripojené kamery. Práve tu sa dajú konfigurovať nastavenia kamier (FPS, expozícia, LED a mnohé ďalšie). Pre každú kameru je nutné vybrať, na čo sa bude používať. Môže byť nastavená na 3D „tracking"  alebo referenčné video  . Iba kamery, ktoré sú nastavené na 3D „tracking" budú schopné rekonštruovať 3D súradnicový systém. Kamery, ktoré sú v režime referenčného videa, zachytávajú obraz v odtieňoch šedej pre referenčné účely. Rekonštruovaný 3D obraz obrázok č.3-B, zachytený „trackingovou" kamerou, môžeme nájsť v paneli perspektívny pohľad. Tu sa nachádza na náhľad, analýzu a označovanie 3D súradnicového systému v rámci zachyteného obrazu [5].

Panel je možné využívať počas živých záberov alebo aj pri prehrávaní nahratých údajov. Taktiež je možné vybrať tri a viac značiek na vytvorenie „rigid body", „skeletonu". Panel s ukážkou kamery obrázok č.3-C ukazuje 2D pohľad kamier v systéme. Tu sa môže monitorovať pohlaď každej kamery zvlášť a aplikovať „mask filter", ktorý buď automaticky zakryje všetky odrazové materiály, alebo sa dajú manuálne zakryť takéto miesta pomocou manuálneho filtra. Nevýhodou je, že po zakrytí filtrom, sa znemožní ďalšie snímanie na danom mieste. Tento panel sa tiež používa na skúmanie 2D objektov (kruhových odrazov), ktoré sú zachytávané alebo filtrované na preskúmanie toho, aké odrazy sa zachytávajú a rekonštruujú do 3D súradníc. Ovládací panel, umiestnený v spodnej časti aplikácie Motive, umožňuje ovládať nahrávanie (režim „live") alebo prehrávanie (režim úprav) údajov o snímaní. V režime živé je možné použiť ovládací panel na spustenie nahrávania a priraďovanie názvov pre súbory snímania. V režime úprav môžeme pomocou tohto panelu ovládať prehrávanie zaznamenaných záberov [5].



Obrázok 3: Program Motiv



Obrázok 4: Nastavenie kamery

Obrázok 5: Kalibrácia roviny

Prameň: [6]

1.1.5 Nastavenie obrazu kamery

Kamery V120: Duo, obrázok č.6, nie je potrebné kalibrovať, sú už pred kalibrované pomocou pevných umiestnení kamier. Ak je potrebné nastaviť rovinu zeme, tak sa použije koordinátor systému Motive a „Set Ground Plane". Rovina sa nastavuje tak, že sa na príslušnú podložku uložia tri značky, ktoré budú reprezentovať rovinu, obrázok č.5. Tiež je nutné, aby sa v zachytávanom obraze nachádzali len práve tieto tri značky. Následne sa v kalibračnom paneli stlačí „Set Ground Plane" [6].

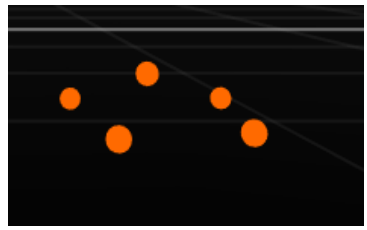
Obrázok 6: Kamera V120: Duo

Prameň: http://optitrack.com/products/v120-duo/

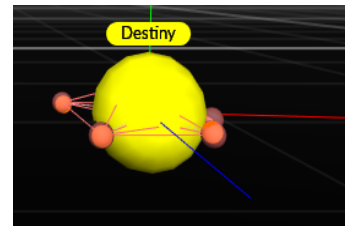
1.1.6 Značkovanie

Retro-reflexívne značky, obrázok č.7, ukladáme na objekt („rigid body" alebo „skeleton"), ktorý chceme sledovať. Treba sa uistiť, že značky sú dobre upevnené. Nastavovanie pre „skeleton" je odlišné, ako pre „rigid body", obrázok č.9, ale to je pre nás irelevantné. Pre vytvorenie „rigid body" sú potrebné minimálne tri značky, obrázok č.8. V okne pre perspektívny pohľad si označíme body, klikneme pravým tlačidlom a vyberieme „create rigid body" [7].

Obrázok 7: Značka



Obrázok 8: Označené body



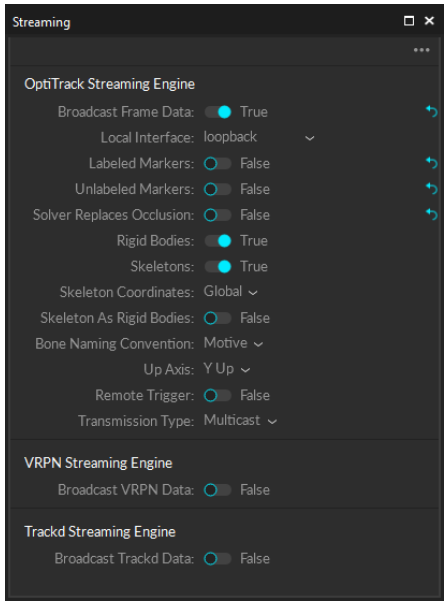
Obrázok 9: Rigid body

V Motive sa značky/body rekonštruujú do 3D súradníc. Pre Motiv musia byť všetky body označené, aby sa dali odlíšiť v rámci zachyteného obrazu a rekonštrukcie, „rigid body" a „skeleton" majú hneď po vytvorení automaticky označené všetky body.

Po tom, ako sa „rigid body“ zadefinuje, je všetko pripravené na „tracking“. Dáta sa dajú používať buď pri živom vysielaní alebo sa dajú nahrávať a spracovávať po „trackingu“. Pre náš projekt je relevantné iba spracovávanie dát naživo – „live streaming“.

1.1.7 Vysielanie dát

V Motive stačí zakliknúť v paneli „streaming“, obrázok č.10 „Broadcast Frame Data“. Okrem tejto možnosti ponúka tiež niekoľko možností „streamingu“ dát na externé aplikácie v reálnom čase. „Trackingové“ dáta môžu byť vysielané v živom alebo editovaciom móde. „Streamingové“ zásuvné moduly sú k dispozícii pre aplikácie Autodesk Motion Builder, Visual3D, Unreal Engine 4, 3ds Max, Maya (VCS), VRPN a trackd. Všetky sú k dispozícii na stránke OptiTracku. Pre všetky ostatné možnosti „streamingu“ je možné použiť NatNet SDK, ktorý ponúka vytvorenie vlastných klient/server aplikácií [8].



Obrázok 10: Panel pre „streaming“ dát

My sa budeme venovať práve variantu NatNet SDK.

1.2 Nastavenie OptiTracku

Po inštalácii Motive:Tracker s pripojením OptiTrack kamery V120: Duo k počítaču pomocou USB kábla, je nutné nastaviť rovinu podlahy, zorné pole a referenčný bod, teda súradnice

[x_0,y_0,z_0]. Zorné pole sa nastavuje podľa potreby pohybom stojana kamery a jeho náklonom. Keďže sa nachádzame v budove, je nutné upraviť miestnosť podľa kapitoly 1.1.3 (Nastavenie hardvéru) .

Rovina sa nastavuje podľa pokynov v kapitole 1.1.5 (Nastavenie obrazu kamery), referenčný bod sa určí ako stredný bod pri kalibrácii roviny. Je potrebné, aby bol vždy vhodne zvolený. V našom prípade sme ho zvolili v ľavom rohu zorného poľa kamery tak, že všetko, čo je pred kamerou, sa nachádza v kladných číslach a nie je nutné ďalšie prepočítavanie.

Ďalej je potrebné pripevniť retroreflexívne značky k Arduino autíčku. Pri viacerých robotoch alebo objektoch je vždy potrebné umiestňovať body v jedinečnom a špecifickom usporiadaní. V špecifickom preto, aby sa značky neprekrývali a boli na dobre viditeľných miestach, jedinečné usporiadanie je nutné, aby bolo možné od seba odlíšiť rozdielne objekty. Vždy po pridaní nového objektu je nutné zvoliť iný vzorec usporiadania značiek!

Akonáhle sú značky rozmiestnené, objekt sa položí do zorného poľa kamery. Ak je všetko, vrátane kamery zapnuté, mali by sa tieto body objaviť v okne pre perspektívny pohľad. V pravom dolnom rohu tohto okna sa nachádza aj počet nájdených značiek a počet zakliknutých značiek. Postupne body zaklikáme pomocou Ctrl + ľavé tlačidlo myši, potom klikneme pravým tlačidlom a vyberieme „rigid body“ a „create rigid“ body. V paneli „assets“, ktorý sa nachádza v ľavom hornom rohu, sa objaví nová položka „rigid body“. V nej je možné premenovať objekt alebo ho uložiť do počítača.

Zakliknutím na „rigid body“ sa v paneli „properties“ objavia vlastnosti tohto objektu. Jednou z nich je aj jeho ID číslo, ktoré budeme potrebovať neskôr. Po zapnutí „live stremingu“ je automaticky nastavená adresa počítača, na ktorom je spustený Motive, aby prebiehal „streming“ z toho istého PC na ten istý PC. Akonáhle sa zapne stream, presunieme sa z Motivu do Matlabu.

1.3 Princíp rotácie

Keďže kamera od OptiTracku spracováva údaje z reálneho sveta a vytvára z nich 3D projekciu, musí pracovať s istou formou rotácie, ktorá vie vhodne popísať pohyb v takomto prostredí. Rotáciu v 3D priestore najlepšie vyjadruje quaternion alebo teória Euilerových uhlov. OptiTrack narába vo svojom programe Motiv práve s quaternionom, keďže prináša určité benefity oproti Euilerovým uhlom. Neskoršie pri komunikácii si toto matematické prevedenie rotácie premeníme do Euilerových uhlov.

1.3.1 Eulerove uhly

Podľa Euilerovej rotačnej teorémy, každý druh rotácie môže byť vyjadrený pomocou troch uhlov. Najčastejšie sa tieto uhly vyjadrujú v karteziánskom súradnicovom systéme ako rotácia v osiach x, y a z ,teda podľa označenia φ, ψ, θ , kde φ predstavuje priečny náklon, po anglicky „roll“, ψ je vybočenie anglicky „yaw“ a θ označuje uhol pozdĺžneho sklonu, anglicky „pitch“.

Euilerove uhly boli pôvodne používané na riešenie diferenciálnych rovníc. Postupne sa z nich vyvinula najčastejšie používaná metóda na parametrizáciu rotácie v priestore.

Ak je rotácia popísaná pomocou troch uhlov φ, ψ, θ, alebo tromi rotačnými maticami, súhrne sú nazývané Euilerove uhly. Celkovú rotačnú maticu A môžeme popísať ako súčin troch rotačných matíc B,C a D, $A = B * C * D$.

Existuje niekoľko konvencií ako zapísať Eulerove uhly, v závislosti od osí, v ktorých sa rotácie vykonávajú. Maticu A si rozpíšeme

$$A \equiv \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}. \quad (1)$$

1. Prvá rotácia je podľa z-ovej osi θ predstavujúca maticu D.
2. Druhá rotácia podľa y-ovej osi $\psi \in [0, \pi]$ popísaná maticou C.
3. Posledná rotačná matica B popisuje rotáciu podľa x-ovej osi a je označená φ.

Teraz môžeme rozpísať rotačné matice pre každú os

$$D \equiv \begin{bmatrix} cos\,\theta & sin\,\theta & 0 \\ -sin\,\theta & cos\,\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad (2)$$

$$\mathrm{C} \equiv \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & \sin\psi \\ 0 & -\sin\psi & \cos\psi \end{bmatrix}, \quad (3)$$

$$B \equiv \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (4)$$

Cez násobenie matíc B, C a D rovnice (2), (3) a (4) môžeme vyjadriť celkovú rotačnú maticu A rovnica.

$$a_{11} = \cos\varphi * \cos\theta - \cos\psi * \sin\theta * \sin\varphi \quad (5)$$

$$a_{12} = \cos\varphi * \sin\theta + \cos\psi * \cos\theta * \sin\varphi \quad (6)$$

$$a_{13} = \sin\varphi * \sin\psi \quad (7)$$

$$a_{21} = -\sin\varphi * \cos\theta - \cos\psi * \sin\theta * \cos\varphi \quad (8)$$

$$a_{22} = -\sin\varphi * \sin\theta + \cos\psi * \cos\theta * \cos\varphi \quad (9)$$

$$a_{23} = \cos\varphi * \sin\psi \quad (10)$$

$$a_{31} = \sin\psi * \sin\theta \quad (11)$$

$$a_{32} = -\sin\psi * \cos\theta \quad (12)$$

$$a_{33} = \cos\psi \quad (13)$$

Celková rotačná matica A (14)

$$A \equiv \begin{bmatrix} \cos\varphi * \cos\theta - \cos\psi * \sin\theta * \sin\varphi & \cos\varphi * \sin\theta + \cos\psi * \cos\theta * \sin\varphi & \sin\varphi * \sin\psi \\ -\sin\varphi * \cos\theta - \cos\psi * \sin\theta * \cos\varphi & -\sin\varphi * \sin\theta + \cos\psi * \cos\theta * \cos\varphi & \cos\varphi * \sin\psi \\ \sin\psi * \sin\theta & -\sin\psi * \cos\theta & \cos\psi \end{bmatrix}$$

Pomocou matice A môžeme vypočítať súradnice bodu po rotácii

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = A * \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}. \quad (15)$$

Kde [x, y, z] sú súradnice bodu po rotácii , [x_0, y_0, z_0] sú súradnice bodu pred rotáciou a rovnice a_{nm} sú zložky rotačnej matice A literatúra [9], [10].

Transformácia rotačnej matice späť na uhly je nasledovná

$$\text{Pitch - } \theta = \tan^{-1}\left(\frac{a_{21}}{a_{11}}\right), \quad (16)$$

$$\text{Yaw - } \psi = -\sin^{-1}(a_{31}), \quad (17)$$

$$\text{Roll - } \varphi = \tan^{-1}\left(\frac{a_{32}}{a_{33}}\right). \quad (18)$$

Uhly okolo osi x a z, teda „pitch“ a „roll“, budú vždy v rozsahu od –π po +π (-180° po +180°). A y-ové, čiže natočenie „yaw“, bude v rozmedzí od – π/2 po +π/2 (-90° po +90°) [9], [10]. Viac rozvinuté vysvetlenie princípu Euilerových uhlov nájdete v použitej literatúre [11].

1.3.2 Quaternion

Quaternion je súčasť nekomutatívnej časti algebry, ktorú popísal William Rowan Hamilton. Označuje sa v matematike ako H, alebo Q_8 a je jedným z príkladov pre všeobecnú triedu hyperkomplexných čísel objavených Hamiltnom. V porovnaní s Euilerovými uhlami je quaternion jednoduchší na porozumenie a predchádza sa ním takzvanému „gimbal lock“ problému.

Situácia „gimbal lock“ nastáva pri vyjadrení rotácie pomocou Euilerových uhlov. Môžeme ju pozorovať aj na gyroskope, keď sa dve osi otáčania vyrovnajú, teda splynú. Následne v danom momente predstavujú namiesto dvoch natočení len jedno v rámci dvoch osi. Tomuto javu sa dá predísť použitím quaternionu. Quaternion má širokú škálu uplatnenia a využitia v počítačovej grafike, robotike, navigácii, leteckej dynamike, satelitoch a mnohých ďalších. Analogicky komplexnými číslami môžeme quaternion vyjadriť cez sumu reálnych a imaginárnych zložiek, [12]

$$H = a * 1 + b * i + c * j + d * k. \quad (19)$$

Teóriu popisujúcu quaternion a jej dôkaz nebudeme rozoberať v našej práci, keďže o celkový chod rotácie sa stará užívateľsky prívetivý program Motiv. Stačí nám len ovládať, že dané matematické vyjadrenie H nám predstavuje rotáciu vo všetkých smeroch x(rbqx),y(rbqy) a z(rbqz) ako aj konštantu w(rbqw). Viac rozvinuté vysvetlenie quaternionu nájdete v použitej literatúre [11].

2 Komunikácia

Keďže nie je možné priame prepojenie Arduina s OptiTrackom, budeme musieť pridať medzičlánok, ktorý bude túto komunikáciu sprostredkovávať. Takže v tejto kapitole sa budeme venovať prepojeniu senzoru OptiTrack, počítača a Arduina navzájom. Celé prepojenie budeme realizovať v Matlabe.

2.1 NatNet SDK

Možnosti komunikácie medzi OptiTrackom a počítacom sú popísané v kapitole 1.1.7 (Vysielanie dát). Pre nás najvhodnejší spôsob komunikácie je pomocou servera NatNet SDK, ktorý je multifunkčný pre viacero jazykov, vrátane nami používaného Matlabu. Všetky uvedené kódy naprogramované v matlabovskom jazyku budú uvedené rovnakým písmom a špeciálne odsadené.

Ak nemáte stiahnutý NatNet SDK, je možné nájsť ho na hlavnej stránke OptiTracku zadarmo. Odkaz pre stiahnutie servera NatNet SDK http://www.optitrack.com/downloads/developer-tools.html#natnet-sdk. Je potrebné otvoriť si NatNetSDK/Samples/MatlabWrapper/natnet, kde sa nachádzajú vopred naprogramované skripty a môžete si ich prezerať, upravovať a ďalej používať. My sme si ich postupne pripravili a zjednodušili pre uľahčenie chodu programu.

2.1.1 Vytvorenie servera

Ako prvé si inicializujeme natnet server.

```
opti = natnet;
```

Dostupné vlastnosti pre natnet nájdeme pomocou príkazu `properties` v okne pre príkazy môžeme vidieť nasledovné vlastnosti servra.

```
>> properties natnet
```

- `ClientIP`
- `ConnectionType`
- `HostIP`
- `IsReporting`
- `IsConnected`
- `Mode`
- `Version`

Pre nás budú podstatnejšie dostupné metódy pre tento server. Zobraziť si ich vieme pomocou príkazu `methods`, následne sa nám v príkazovom okne zobrazia metódy pre daný server. My si ukážeme len niektoré, ktoré budeme reálne využívať.

```
>> methods natnet
```

- `addlistener`
- `connect`
- `enable`
- `disable`
- `disconnect`

Potrebné bude pridať zložku, v ktorej sa nachádza quaternion.m, NatNetEventHandlerSample.m a ďalšie. Všetky potrebné skripty nájdeme v jednej zložke, ktorú si vieme jednoducho pridať do cesty pre náš program. Z toho dôvodu ich budeme poznať aj mimo tejto zložky.

```
addpath('NatNetSDK\Samples\Matlab');
```

Tieto skripty zabezpečujú spojenie so serverom a quaternion zabezpečuje implementáciu quaterinionskej matematiky a 3D rotácie. Pomocou príkazu metódy `connect` sa napojíme na server natnet a pomocou príkazu `addlistener` sprístupníme neustálu komunikáciu OptiTracku s Matlabom. Neskôr si vytvoríme funkciu pohyb_bez_prekazok, ktorá bude komunikovať a vysielať dáta.

```
opti.connect
opti.addlistener(1,'pohyb_bez_prekazok);
opti.enable(0)
```

Po nadviazaní komunikácie Matlabu s OptiTrackom nasleduje prepojenie Arduino autíčka s Matlabom. Arduino Yún poskytuje dve možnosti, ako vytvoriť stabilné prepojenie medzi počítačom a samotným robotickým autíčkom. Vlastnosťou tohto mikroovládača je vytvorenie wifi hotspotu, na ktorý sa vieme počítačom priamo napojiť. Tento spôsob je veľmi jednoduchý a rýchly, jeho nevýhodou je, že v tom istom čase by mohlo byť na počítač napojené práve jedno autíčko. Ďalšou možnosťou je použiť wifi router ako spojovací bod, ktorý bude sprostredkovávať prepojenie medzi počítačom a robotom. Vieme naň pripojiť počítač a ďalší n-počet robotov. Router vie priradiť ku každému pripojenému zariadeniu vlastnú IP adresu, na základe ktorej s ním môžeme komunikovať v rámci tejto siete. Aby nebolo nutné zakaždým kontrolovať, akú IP adresu router priradil robotovi, je potrebné ju v nastaveniach routra rezervovať pre každého robota zvlášť.

2.1.2 Nastavenie Arduina na sieť

Pri nastavení zapneme Arduino ShieldBot, pripojíme sa na hotspot, ktorý vytvorí a v internetovom prehliadači na základnej adrese Arduina (192.168.240.1) nájdeme nastavenia mikroovládača. Klikneme na možnosť konfigurácia a vyberieme v políčku pre automatické pripájanie k wifi sieti nami požadovanú sieť, na ktorú chceme pripájať robota a reštartujeme ho. V Matlabe sa pripojíme na autíčko pomocou IP adresy, ktorú si zadeklarujeme, a príkazom `connect`. Adresu, pomocou ktorej sa pripojíme na autíčko, nájdete podľa návodu v predošlej kapitole 2.1.1 (Vytvorenie servera).

```
mauto = Tracker('192.168.1.211');
mauto.connect
```

2.1.3 Spracovanie dát zo servera

Po nastavení Arduina na sieť ďalším krokom je vytvorenie funkcie, ktorá bude podľa nastaveného času komunikovať s OptiTrackom. Funkciu nazveme semestralny_optitreck. Z tejto funkcie nepotrebujeme vyberať žiadne informácie, lebo všetky sa budú spracovávať a odosielať v rámci nej, takže nebude mať žiadne výstupy. Budeme ju volať pomocou funkcie `addlistener`.

Premenná mauto musí byť globálna, aby sme s ňou mohli pracovať v rámci oboch funkcií. Potrebné je vedieť ID čísla z Motivu pre každý objekt, ktorý chceme zaznamenávať. Tieto identifikačné čísla sa priraďujú pre rigid body pri ich vytvorení, alebo sa dajú zmeniť manuálne. V našom prípade potrebujeme dve ID čísla: robotické autíčko a cieľ/destiny, čiže miesto, kam sa má autíčko dostať. Tieto čísla si inicializujeme.

```
rbnum = 1; % ID číslo autíčka
destnum = 4; % ID číslo cieľa
```

Nasledujúcim príkazom vieme zo servera dostať informácie o polohe a rotácii jednotlivých rigid body.

```
rb = evnt.data.RigidBodies( rbnum ); % načítanie zo servera
```

Informácie o polohe prichádzajú vo forme bodu so súradnicami [x, y, z] a nastavené sú na milimetre * 10^{-1}. Dané súradnice stačí premeniť na centimetre. Ako je zvyčajné, umiestňujeme x-ovú aj y-ovú súradnicovú os na rovinu, napriek tomu, že v OptiTracku je to inak. Nesúlad sa dá ľahko odstrániť načítaním osi z ako os y. Z načítaných súradníc zo servera si vytiahneme jednotlivé hodnoty do príslušných osí nasledovnými príkazmi.

```
x= rb.x * 100; %pozícia osi x v cm
y= rb.z * 100; %pozícia osi y v cm
```

```
z= rb.y * 100; %pozícia osi z v cm
```

Takým istým postupom pripravíme aj lokalizáciu cieľového bodu.

```
da = evnt.data.RigidBodies( destnum );
dx= da.x * 100;
dy= da.z * 100;
dz= da.y * 100;
```

Pri rotácii úprava bude o niečo zložitejšia, keďže OptiTrack pracuje s komplexným pohybom v priestore. Ten vhodne popisuje quternion. Máme k dispozícií predprogramovanú funkciu quaternion(), ktorá zadefinuje rotáciu podľa quaternionskej matematiky. Prevádzanie do formy Eulerových uhlov zabezpečí funkcia `EulerAngle()`. V tejto podobe je jednoduchšie narábať s rotáciou, keďže ju máme prevedenú do radiánov. O celé prevedenie sa postará séria uvedených príkazov.

```
q = quaternion( rbqx, rbqy, rbqz, rbqw );
qRot = quaternion( 0, 0, 0, 1);
q = mtimes( q, qRot);
a = EulerAngles( q , 'zyx' );
```

Rovnako ako pri polohe aj tu bude nutné postupne načítať všetky prijaté dáta zo servera do jednotlivých premenných.

```
ax = a( 1 ) * 180.0 / pi; %roll
ay = a( 2 ) * 180.0 / pi; %yaw
az = a( 3 ) * 180.0 / pi; %pitch
```

V tomto bode je dôležité určiť, aké údaje sa budú do autíčka posielať. Potrebná je jeho poloha a miesto, kam sa má dostať. Princíp pohybu robota tiež vyžaduje vybočenie, teda natočenie doľava a doprava, ktoré predstavuje rotácia okolo osi z v karteziánskej súradnicovej sústave (x, y, z). V OptiTracku je táto os označovaná ako os y, tiež známa pod anglickým názvom „yaw“. Okrem toho sa vyžaduje ešte údaj o ľubovoľnom natočení v hociktorej inej osi, tento údaj je potrebný kvôli riešeniu Eulerových uhlov. Bližšie vysvetlenie nájdete v kapitole 1.3.1 (Euilerove uhly).Všetky tieto údaje vieme posielať autíčku naraz, jediným príkazom `setData()`.

```
mauto.setData([x,y,ax,ay,dx,dy]) % odosielanie údajov
```

Komunikácia Matlabu s autíčkom prebieha s periódou 0.4 sekundy. Časový interval zabezpečujeme pomocou jednoduchého vopred naprogramovaného časovača `tic-toc`.

Celí kód je súčasťou prílohy B nachádza sa v nej jednoduchý pohyb robota v priestore bez prekážok.

2.2 Odosielanie dát

Odosielanie dát po serveri si vyžaduje presnú podobu, keďže ide o posielanie údajov z matlabovského jazyka do jazyka C v Arduine. Takémuto typu odosielania dát je potrebné vytvoriť formu, ktorá sa zakóduje a následne sa odošle po serveri. V robotovi sa odoslané opätovne dekóduje. V našom prípade sa dáta odosielajú do funkcie Tracker, ktorá dáta podľa poradia šifruje v nasledovanej podobe

```
msg = sprintf('{"x":%.2f,"y":%.2f,"ax":%.2f,"ay":%.2f,
"dx":%.2f,"dy":%.2f}',data(1),data(2),data(3),data(4),
data(5),data(6));
```

Prenos cez server zabezpečuje Matlab websocket client, ktorý je k dispozícii na stiahnutie na nasledovnom odkaze https://github.com/kvasnica/wsclient. V prílohe A nájdete šifrovač údajov Tracker.

3 Arduino

Arduino má k dispozícii aj softvér arduino IDE (integrované vývojové prostredie), ktorý uľahčuje prácu s Arduino produktmi a veľmi ľahko nahráva kódy. Stiahnuť sa dá na hlavnej stránke zadarmo alebo za dobrovoľný príspevok. Tento program budeme potrebovať na programovanie robotického autíčka Arduino ShieldBot a Arduino Yún. Programovanie v Arduino je pomerne jednoduché, v jazyku C má dve základné funkcie: *void setup()* a *void loop()*. Vo *void setup()* sa nachádza kód, ktorý je spustený len raz -pri zapnutí. *Void loop()* obsahuje kód, ktorý sa vykonáva dookola, ako vyplýva už z názvu: je to určitý druh cyklu. Knižnice a premenné sa definujú pred *void setup()*. Všetky uvedené kódy, ktoré boli naprogramované v jazyku C a implementované do Arduina, budú odlíšené šikmým písmom.

Program v Arduine si rozdelíme na dve časti. V prvej spracujeme prijaté dáta pomocou jazyka aJson a premeníme ich do jazyka C. Druhá časť bude obsahovať už konkrétnu prácu s prijatými údajmi.

3.1 Preklad dát

Knižnicu aJson je možné nájsť na internete. Je potrebné mať na zreteli, aby ste mali stiahnutú najnovšiu verziu. Skratka aJson je Arduino JavaScript Object Notation. Pri výmene údajov medzi prehliadačom a serverom môžu byť údaje iba v podobe textových reťazcov. Ľubovoľný objekt JavaScriptu môžeme premeniť na textový reťazec JSON a naopak, môžeme previesť akýkoľvek JSON, prijatý zo servera, do objektov JavaScriptu. Týmto spôsobom môžeme pracovať s údajmi ako objektmi JavaScriptu, bez komplikovaných analýz a prekladov.

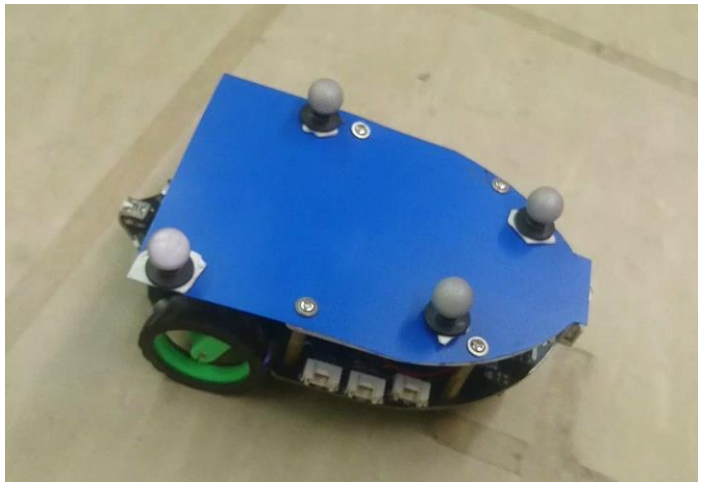
Predtým, ako sa začnú prekladať všetky premenné, musíme zadefinovať, do akých premenných sa budú ukladať:

```
int index = 0;
float x = 0.0;
float y = 0.0;
float ax = 0.0;
float ay = 0.0;
float dx = 0.0;
float dy = 0.0;
```

Následne pomocou jazyka aJSON preložíme prijaté údaje v takom istom tvare, v akom sme ich odoslali pomocou Trackera z Matlabu. Postupne ich načítame do zadefinovaných premenných a pripravíme ich na ďalšie použitie.

3.2 Ovládanie robotického autíčka

Pre aplikáciu našich algoritmov a programov sme si vybrali robotické autíčko Arduino ShieldBot obrázok č.11, na ktoré sa dá veľmi jednoducho pripojiť Arduino Yún. Dá sa s ním pomerne rýchlo komunikovať. Autíčko sa skladá z dvoch zadných kolies, každé je poháňané vlastným motorčekom a vpredu sa nachádza guľôčka, ktorá slúži ako opora. Robotické vozidlo má teda celkovo dve riaditeľné veličiny, uhlovú rýchlosť ľavého a pravého kolesa. Táto rýchlosť sa dá ovládať pomocou príkazu *shieldbot.drive(prave_koleso,lave_koleso).*



Obrázok 11: Robotické vozidlo

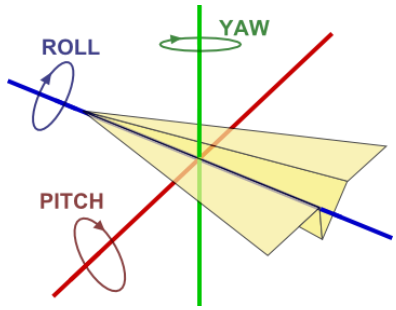
Cieľom je dostať robota z miesta A do miesta B v čo najkratšom čase, s minimálnymi odchýlkami od optimálnej dráhy. Toto riadenie môžeme zabezpečiť pomocou P,PI alebo PID regulátorov, ktoré veľmi rýchlo a presne dokážu regulovať pochod autíčka tak, aby sa dostalo do zvoleného miesta. Na robotovi sa budeme snažiť riadiť uhlovú rýchlosť kolies tak, aby bol otočený presne na cieľové miesto, ktoré má dosiahnuť. Podrobnejší popis je v kapitole 3.4 (Pohyb robota).

3.3 Prijaté údaje

V nasledovnej kapitole si uvedieme, aké údaje sú vysielané do robota a ako ich spracujeme. Zadefinujeme si ďalšie potrebné podmienky pre chod robota a vysvetlíme si mechanizmus PID regulátora, aj to, ako nám pomôže dosiahnuť náš cieľ.

3.3.1 Uhol natočenia

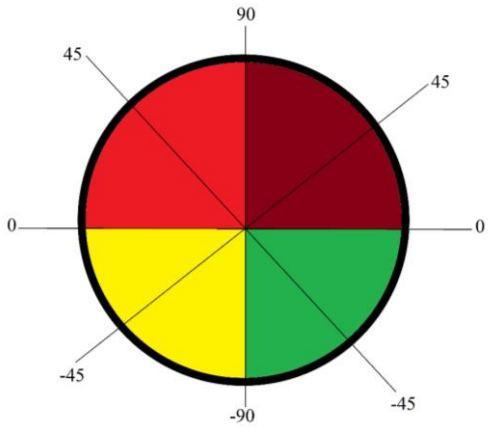
Ako prvé je potrebné vysvetliť, v akej podobe nám prichádza informácia o natočení. O natočení prichádzajú dve informácie, a to natočenie v y-ovej a x-ovej osi. Pre pohyb autíčka na rovine je pre nás dôležitá iba y-ová os, anglicky yaw. V tomto prípade počítame, že y-ová os je vedená kolmo na rovinu obrázok č.12 (zelená farba), čiže prestavuje pohyb doľava a doprava.



Obrázok 12: Rotácia okolo osí

Prameň: http://www.xojo3d.com/tut012.php

Tento údaj je definovaný v Eulerových uhloch obrázok, č.13. Podrobnejšie sme sa zmienili o tom, čo je Eulerov uhol alebo quaterinion, v samostatnej kapitole 1.3 (Princíp rotácie).



Obrázok 13: Natočenie v Euilerových uhloch

Motiv má schopnosť zapamätať si, ako sa prvýkrát zadefinuje „rigid body“. Každý pohyb, ktorý vyvolá natočenie o x° sa ráta od referenčného (nulového) natočenia. Vďaka tejto schopnosti si vieme (pre lepšiu manipuláciu) previesť tieto uhly do 360° systému, kde nula stupňov je pôvodné natočenie autíčka.

Predstavme si, že máme kruh na rovine otáčania okolo autíčka. Toto natočenie predstavuje natočenie „yaw“. V Eulerových uhloch horná výseč predstavuje kladné čísla od 0° po 90°, spodná výseč predstavuje záporné čísla od 0° po -90°, pravá výseč kruhu má vlastnosť, že x-ové a z-ové natočenie je práve rovné 0°, pri ľavej výseči sú tieto natočenia práve rovné -180°. Na základe týchto poznatkov vieme vytvoriť jednoduché rovnice na prepočet a vytvoriť tak údaje v intervale od 0 po 360°.

Príklad: Autíčko je natočené o 260° od svojho pôvodného natočenia. V Eulerových uhloch sa nachádzame v žltej výseči na -80°. Inak povedané, y-ové natočenie je rovné -80°, x-ové a z-ové natočenie je rovné -180°.

Vytvorili sme si funkciu nezávislú od hlavných funkcií *setap()* a *loop()*. Funkcia má názov prepocet_uhlaAY, vždy sa zavolá na začiatku *loopu()* a prepočíta uhly. Vstupné premenné sú nespracované údaje o natočení v Eulerových uhloch.

```
int prepocet_uhlaAY (float vstupAY,float vstupAX)
    {
      float vystupAY2;
      if ((vstupAY > 0) && (abs(vstupAX) < 50 )) { vystupAY2 = vstupAY ;}
      else if (abs(vstupAX) >= 50 ) { vystupAY2 = 180 - vstupAY ;}
      else if ((vstupAY < 0) && (abs(vstupAX) < 50   )) { vystupAY2 = 360 + vstupAY ;}
      return vystupAY2;
    }
```

Podmienky sa nepatrne líšia od našej teórie, lebo autíčko sa nenachádza zarovno s rovinou a z toho dôvodu môžu nastať odchýlky. Na základe zvolených podmienok vieme presne určiť výseče. Výstupom z funkcie je teda prepočítaný uhol v 360° systéme, vzhľadom na pôvodné natočenie. Tieto uhly sa používajú pri ďalších operáciách.

3.3.2 Poloha

Údaje o polohe prichádzajú už v upravenej forme tak, ako sme ich v Matlabe zadefinovali. Chodia ako súradnice v x-ovom a y-ovom smere od nami zadefinovaného referenčného bodu $[x_0,y_0]$, jednotkovo v cm.

3.4 Pohyb robota

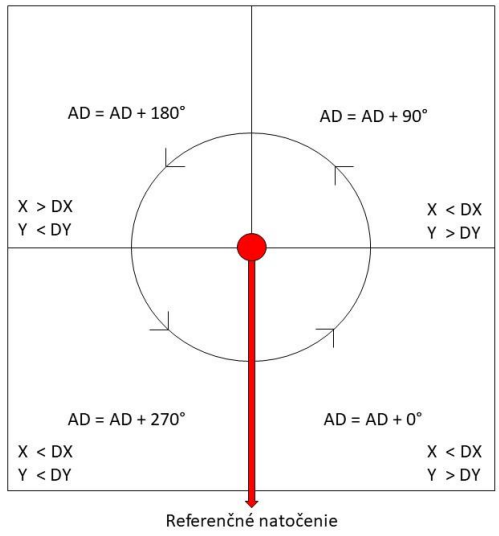
V projekte bol zvolený jednoduchý pohyb autíčka založený na fakte, že poznáme pôvodné natočenie autíčka. Pomocou tangensovej vety si vieme vypočítať, aký uhol má mať autíčko, aby bolo nasmerované na cieľ. Tangens α je pomer dĺžok odvesny protiľahlej k tomuto uhlu a dĺžky odvesny k nemu priľahlej.

$$\tan\alpha = \frac{a}{b} = \frac{\Delta Y}{\Delta X} \text{ po prepísaní } \alpha = \tan^{-1}\left(\frac{|vzdialenosť\ ciela\ Y - poloha\ autíčka\ Y|}{|vzdialenosť\ ciela\ X - poloha\ autíčka\ X|}\right) \quad (20)$$

Vytvorili sme si funkciu na daný prepočet a nazvali sme ju prepocet_uhlaAD. Zavolá sa vždy na začiatku *void loopu()* a vypočíta, aký uhol má mať autíčko vzhľadom na jeho referenčný stav. Vstupné premenné sú informácie o polohe autíčka a cieľa.

```
int prepocet_uhlaAD(float DX,float DY,float X,float Y) {
    float vystupAD;
    if ((DX > X) && (DY >= Y)) { vystupAD = atan(abs(DY-Y)/abs(DX-X))*180/M_PI + 270; }
    else if ((DX <= X) && (DY > Y)) { vystupAD = atan(abs(DY-Y)/abs(DX-X))*180/M_PI + 180;}
    else if   ((DX < X) && (DY <= Y)) { vystupAD = atan(abs(DY-Y)/abs(DX-X))*180/M_PI + 90;}
    else if ((DX >= X) && (DY < Y)) { vystupAD = atan(abs(DY-Y)/abs(DX-X))*180/M_PI ; }
    return vystupAD; }
```

Keďže podľa tangensovej vety by sme vedeli iba spočítať uhol medzi dĺžkou v x-ovom a y-ovom smere medzi cieľom a začiatkom, musíme ho ešte prerátať na náš systém, vzhľadom k referenčným podmienkam obrázok č.14.



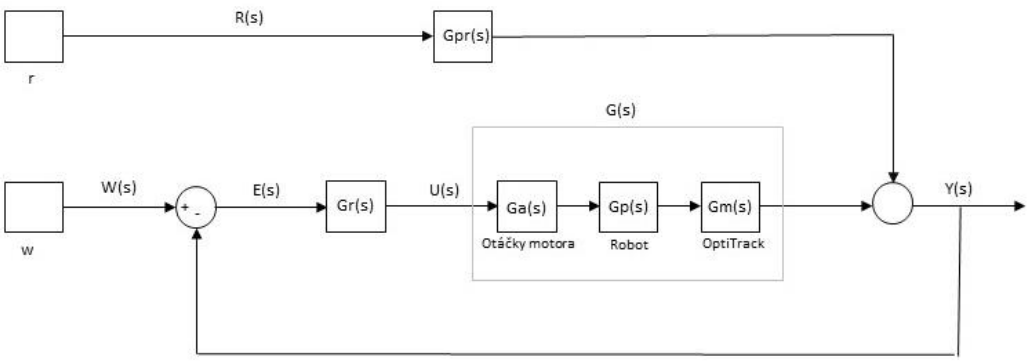
Obrázok 14: Uhol natočenia robota

Na obrázku č.14 sme si jednoducho vysvetlili logiku výpočtu požadovaného uhla natočenia autíčka v nami zadefinovanom pevno súradnicovom systéme. AD predstavuje požadované natočenie, X a Y sú súradnice robota v príslušných osiach a DX a DY sú súradnice cieľa, kam má robot prísť v daných osiach. Červená šípka znázorňuje referenčné natočenie autíčka. Podľa smeru šípky sa toto natočenie zvyšuje od 0°. V prvej časti vypočítané natočenie predstavuje zároveň skutočné natočenie. V ďalšej časti vypočítané natočenie už nepredstavuje skutočné a preto musíme prirátavať v každom ďalšom sektore od prvého o 90° naviac. Prechod z poslednej časti do prvej je číselne prevedený z 359° do 0°.

3.4.1 Uzavretý regulačný obvod

Uzavretý regulačný obvod je termín pre systém, ktorý pracuje v automatickom režime v istom zapojení. Je tvorený procesom, ktorý chceme riadiť a prostriedkami, ktoré zabezpečujú toto riadenie. Uzavretý regulačný obvod, inak aj URO, môžeme vyjadriť aj jeho blokovou schémou zobrazenou na obrázku č.15. Je zostavená zo štyroch základných blokov, ide o merací člen, akčný člen, riadený systém a regulátor [13].

Pomocou uhla natočenia robotického autíčka a uhla vzhľadom na cieľ vieme skonštruovať jednoduchú schému URO. Požadovaná veličina W je uhol, ktorý je potrebný dosiahnuť autíčkom. Skutočne nameranú hodnotu Y predstavuje uhol autíčka vzhľadom na začiatočné natočenie, teda riadenú veličinu, ovplyvnenú poruchou Gpr. Pomocou týchto dvoch veličín si vieme vygenerovať regulačnú odchýlku E, ktorá bude vstupovať do nami naprogramovaného regulátora. Blok Gr predstavuje regulátor, ktorý generuje hodnotu akčnej veličiny (Ga), Gp predstavuje riadený prenos a Gm je merací člen [13].



Obrázok 15: Uzavretý regulačný obvod

Schéma URO zjednodušene predstavuje prácu nášho systému. Vplyv poruchy sa môže vyskytovať vo všetkých častiach bloku G(s) naraz alebo aj jednotlivo. V tomto projekte sme sa nezaoberali odstránením týchto porúch a šumov, čiže ich nekompenzujeme žiadnym spôsobom.

3.4.2 PID regulácia

PID regulátor sa skladá z troch zložiek, proporcionálnej (P), integračnej (I) a derivačnej (D) zložky. Prvá, proporcionálna časť, predstavuje prítomnosť, kde poslednú alebo momentálnu regulačnú odchýlku násobí zosilnením regulátora. Jej matematické vyjadrenie pomocou zákona riadenia je

$$P = Z_r * e(t) \tag{21}$$

Z_r predstavuje zosilnenie systému a e(t) je regulačná odchýlka v spojitom čase.

Ďalšia v poradí je integračná zložka, ktorá predstavuje minulosť. Je to nestabilná zložka regulátora, odstraňujúca trvalú regulačnú odchýlku. Bude teda generovať akčnú veličinu až do chvíle, kým regulačná odchýlka nebude rovná nule. V matematickom prevedení cez zákon riadenia môžeme tento člen zapísať integráciou nasledovne. Integračnú konštantu si budeme označovať ako T_i.

$$I = \frac{Z_r}{T_i} \int_0^t e(\tau) * d\tau \tag{22}$$

Posledná, derivačná zložka, je určená na zlepšovanie stability URO a zrýchľuje riadenie, inak povedané, predpovedá budúcnosť. Svojím zásahom ukazuje budúci vývoj regulačnej odchýlky. Opísaná je v zákone riadenia cez deriváciu regulačnej odchýlky podľa času.

$$D = Z_r * T_d * \frac{de(t)}{dt} \tag{23}$$

T_d predstavuje derivačnú konštantu. Výstup regulátora u(t) teda môžeme vyjadriť nasledovne.

$$u(t) = Z_r \left((t) + \frac{1}{T_i} \int_0^t e(\tau) * d\tau + T_d * \frac{de(t)}{dt} \right) \tag{24}$$

Rovnica č.24 vyjadruje PID regulátor podľa zákona riadenia v spojitom čase.

3.4.2.1 Implementácia regulátora

PID regulátor sme predstavili v časti 3.4.2, na základe opísaného matematického prevedenia P, I a D zložky nemôžme ani jednu z nich priamo implementovať ako výraz do algoritmu. Keďže sú všetky vyjadrené cez časovo spojitý zákon riadenia, musíme ich najskôr previesť do diskrétnej časovej oblasti a až potom ich môžeme priamo implementovať do algoritmov.

Ako prvé si v Arduine pred funkciou *void setup()* zadefinujeme konštanty: kp_konštanta, ki_konštanta a kd_konštanta s pridelenými experimentálnymi hodnotami, ktoré je možné nájsť v časti 3.4.2.2 (Porovnanie regulátorov).

Regulačná odchýlka môže byť prepísaná ako

$$e(t) = e(n). \tag{25}$$

Zadefinujeme si ju v Arduine ako premennú chyba a bude vypočítavaná cez sériu podmienok, ktoré zabezpečia, aby sa bralo do úvahy otáčanie v oboch smeroch, doľava a doprava. Proporcionálnu zložku P bude predstavovať jednoduché zosilnenie Z_r, ktoré bude násobené regulačnou odchýlkou. Toto zosilnenie si prevedieme ako k_p a v Arduine si ho inicializujeme ako kp_konštantu.

$$P = k_p * e(n). \tag{26}$$

Vyjadrenie v jazyku C :

```
P = kp_konstanta * chyba
```

Integračnú zložku I vyjadríme pomocou aproximácie integrálu ako sumu jednotlivých oblastí vypočítanú pomocou lichobežníkového pravidla [14]. Pomocou tohto pravidla môžeme približne nahradiť plochu funkcie pomocou lichobežníkov, ktoré je jednoduché vypočítať. Rozdiel časov, teda fixná perióda Δt , bude predstavovať šírku lichobežníka. V našom prípade je táto perióda rovná 0,4 sekundy. Dĺžku jednej strany bude predstavovať minulá regulačná odchýlka e(n-1) a druhú dĺžku momentálna regulačná odchýlka e(n). Teda celková aproximácia integračnej zložky do diskrétnej časovej oblasti vyzerá nasledovne

$$I \approx k_i * \sum_{j=1}^{n} \left(\frac{e(i-1)+e(i)}{2}\right). \tag{27}$$

Konštanta k_i predstavuje podiel šírky lichobežníka, teda periódy Δt a integračnej konštanty T_i.

Vyjadrenie v jazyku C :

```
if ((chyba < 5)){
    integral += ki_konstanta * (chyba+predosla_chyba)/2;
    I = integral;
   }
else{
    integral = 0;
    I = 0;
    }
predosla_chyba = chyba;
```

Integračná zložka sa zapína len vtedy, keď dosiahneme pomocou P zložky okolie požadovanej hodnoty. Mimo neho je nulová. Jej úlohou je udržať toto okolie a dosiahnuť požadovanú hodnotu. Ak by sme mali integračnú zložku k dispozícii neustále, kumulovala by sa v priebehu času

a neustrážili by sme požadovanú hodnotu. Tá by bola extrémne kmitavá. Deltu okolia sme zvolili na základe experimentu. Keďže autíčko sa ovláda len pomocou rýchlosti jeho kolies, je ťažké toto okolie zmenšovať.

Poslednú zložku regulátora D vieme jednoducho aproximovať, nahradením derivácie spätou diferenciou [15]. Takto vieme približne riešiť v diskrétnej časovej oblasti derivačnú zložku

$$D \approx k_d * \frac{e(n-1)-e(n)}{\Delta t}. \quad (28)$$

Vyjadrenie v jazyku C :

delta_chyby = chyba - predosla_chyba ;

derivacia = delta_chyby / 0,4;

*D = kd_konstanta * derivacia;*

Konštanta k_d predstavuje vynásobenie zosilnenia systému a derivačnej konštanty, ak by sa vyskytovala porucha, ktorá by ovplyvňovala našu periódu a menila ju v priebehu času. Vieme ľahko prerátavať danú hodnotu periódy a tým pádom sa poruchy vyvarujeme. Čiže Δt vieme nahradiť v integračnej aj derivačnej zložke nasledovne:

Ako prvé si zadefinujeme premennú predošlý čas a priradíme jej hodnotu 0, táto premenná bude globálna, čiže musí ísť mimo funkcie.

predosly_cas = 0;

Potom si inicializujeme začiatok počítania času vo *void setup()*.

cas = millis();

Deltu času teda vypočítame ako rozdiel času, ktorý skutočne ubehol od posledného prijatia údajov.

delta_casu = (cas - predosly_cas)/1000 ;

predosly_cas = cas;

Je nutné si prepočítať čas z príkazu milis() na sekundy.

3.4.2.2 Porovnanie regulátorov

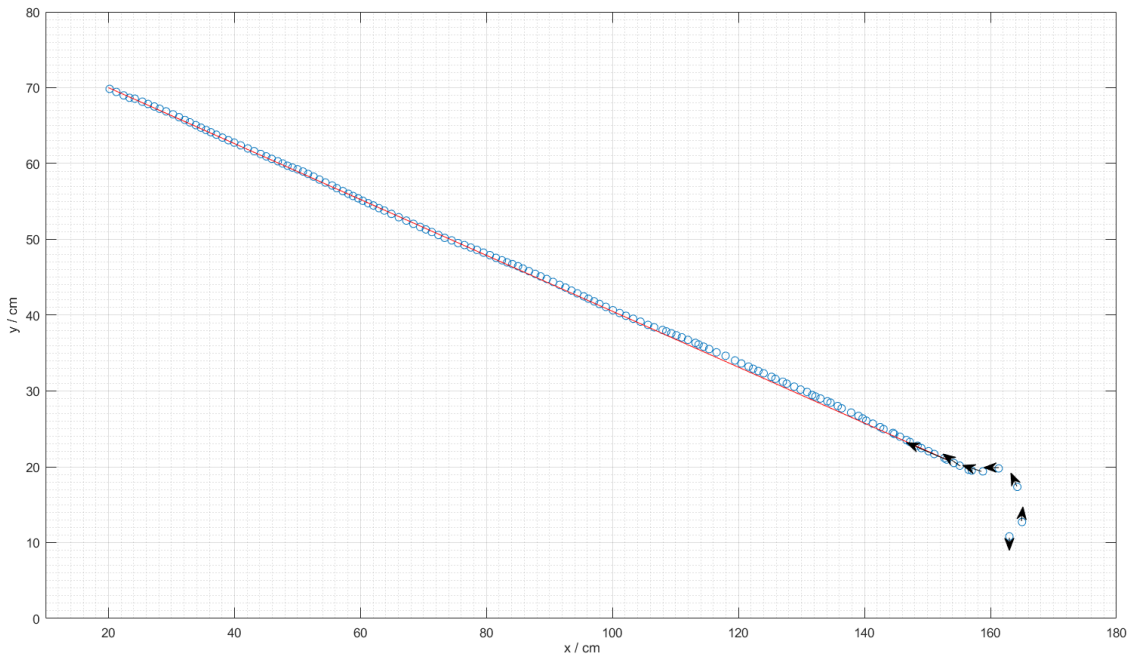
Pre reguláciu uhlovej rýchlosti koliesok sme si vybrali P, PI a PID regulátor. Rozdiel medzi P a PI regulátorom nie je veľmi veľký. I zložka je aktívna, iba ak robot dosiahne požadované okolie uhla smerujúceho k cieľu. Tým pádom sa skoro vôbec nebude líšiť priebeh otáčania robota. Cesta k danému cieľu je ale rôzna. Pri veľmi veľkej I zložke narážame na problém rýchlosti našej komunikácie a obmedzujeme rýchlosť akčných zásahov robota. Čiže nie je vhodné voliť veľkú integračnú konštantu, lebo systém sa stane kmitavým.

Nasledovný experiment ukáže, aký je celkový priebeh regulácie otáčania autíčka a cesty za cieľom.

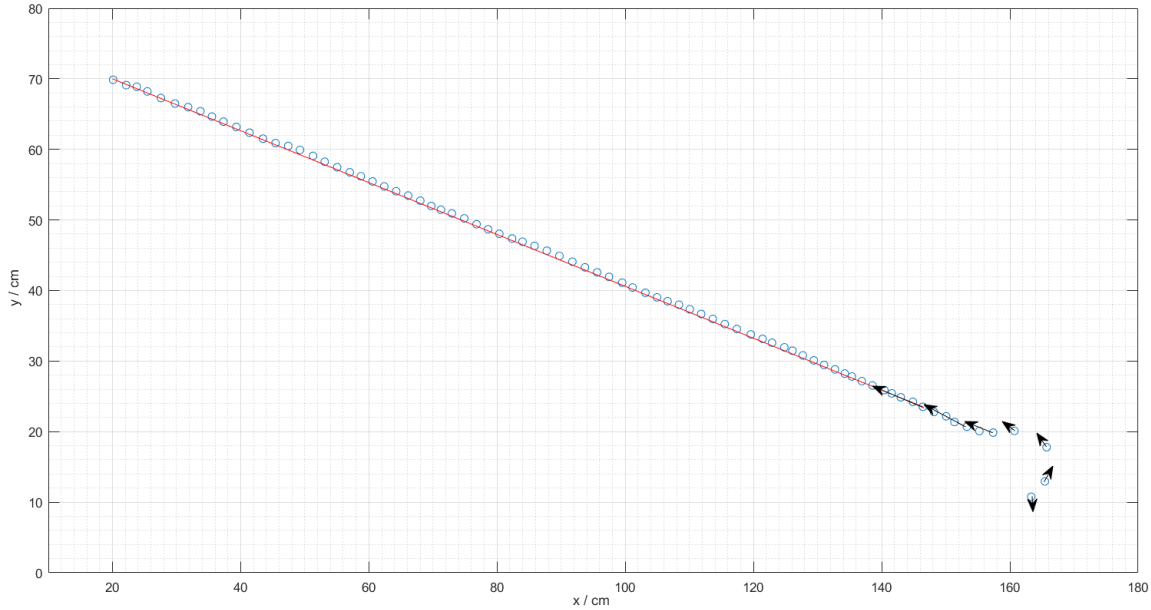
Tabuľka 1: Experimentálne hodnoty pre regulátory

k_r	k_i	k_d	obrázok
0,5	-	-	č.15
0,5	0,015	-	č.16
0,3	0,05	0,1	č.17

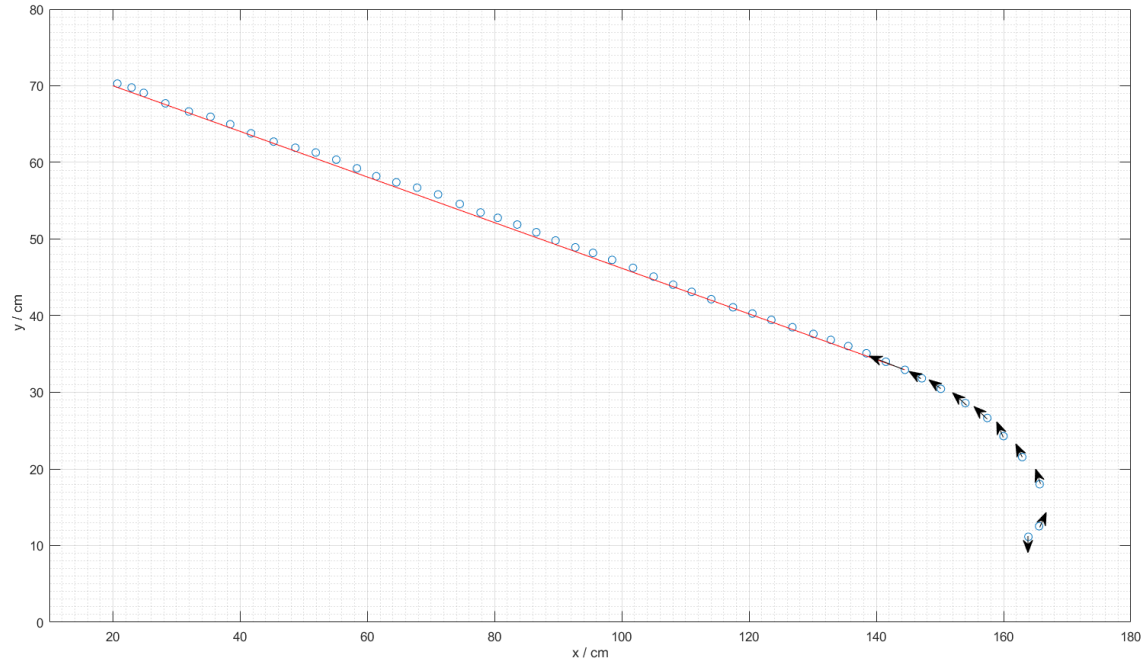
Údaje boli získane na základe experimentu, v ktorom autíčko bolo na začiatku otočené o viac ako 180° stupňov od cieľa. Jeho počiatočná pozícia bola [165, 10] cm a konečná [20, 70] cm. Cieľom experimentu bolo zistiť vhodné konštanty pre P, PI a PID reguláciu. Na nasledovných grafoch si môžeme pozrieť nájdené vhodné konštanty pre každý regulátor. Modré guličky reprezentujú pohyb robotického vozidla v čase z počiatočnej pozície do cieľového miesta. Prvú časť priebehu pohybu sme si označili čiernymi šípkami. Tieto šípky nám orientačne ukazujú, ako sa robot otáčal, až kým nenašiel požadovaný uhol natočenia. Červená čiara nám ukazuje ideálnu cestu po nájdení požadovaného uhla natočenia. Čiže uvádza, ako robot udržiaval túto požadovanú hodnotu. Pozícia každej vzorky bola nameraná s periódou 0,4 sekundy.



Obrázok 16: Priebeh riadenia pomocou P regulátora



Obrázok 17: Priebeh riadenia pomocou PI regulátora



Obrázok 18: Priebeh riadenia pomocou PID regulátora

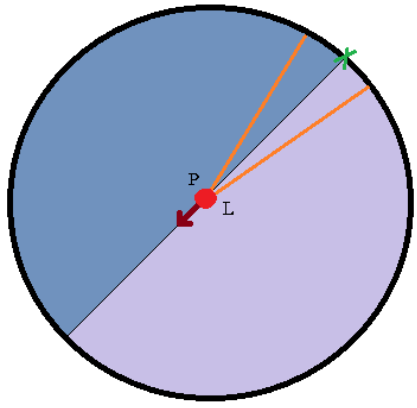
Na prvých dvoch priebehoch otáčania (body so šípkou) si môžeme všimnúť, že je veľmi rýchle. Robot sa otočí na malom priestore za minimálny čas. Prvý priebeh s P reguláciou, obrázok č.16 je agresívny, dobre reaguje na dynamický cieľ. Po nájdení vhodného uhla už P regulátor len udržuje správne natočenie a nijako neurýchľuje pohyb samotný.

PI regulátor, obrázok č.17, reaguje lepšie na dynamické prostredie, aj keď počiatočné otáčanie je rovnaké, ako pri predošlom priebehu, keďže I zložka je aktívna len v požadovanom okolí. PI regulátor tlmí agresívny zásah v okolí požadovanej hodnoty a pomáha ju rýchlejšie ustáliť. Celkový pohyb po nájdení vhodného uhla je urýchlený I zložkou, ktorá v čase naberá na veľkosti.

Posledný priebeh s PID reguláciou, obrázok č.18, je odlišný. Toto riešenie najlepšie našlo vhodný uhol, aj keď otáčanie zabralo najviac času. Vzhľadom na naše požiadavky bolo tiež dostatočne rýchle. PID regulátor reaguje na dynamické prostredie trošku horšie, ako predošlé regulátory, keďže D zložka približne predpovedá budúce regulačné odchýlky a pri neustálom menení požadovanej hodnoty tieto sa odchýlky môžu výrazne meniť. Ale na druhú stranu si môžeme všimnúť, že veľmi dobre urýchlilo celkový pohyb autíčka. Pre naše potreby sú teda všetky tri regulátory vhodné. V prípade P a PI regulátora budeme radšej voliť možnosť s I zložkou, keďže dosahovala lepšie výsledky.

3.4.3 Ovládanie kolies

V tele programu sa odohráva rozhodovanie, či sa má autíčko otáčať doprava alebo doľava. Toto rozhodovanie je zabezpečené cez sériu podmienok. Tieto podmienky môžeme vysvetliť na základe nasledovného zobrazenia, viď. obrázok č.19.



Obrázok 19: Podmienka otáčania

Jednotlivé písmená znamenajú: P-pravé koleso, L-ľavé koleso. Červená šípka ukazuje, kam je autíčko otočené. Zelený krížik označuje cieľové miesto a oranžové čiary vyraďujú okolie požadovanej hodnoty.

Príklad 1 : Ak by sa cieľ nachádzal v tmavej výseči, autíčko musí ísť rýchlejšie ľavým kolesom, čiže pripočítavame akčný zásah z regulátora k ľavému kolesu a prvé koleso je spomaľované akčným zásahom. Akčný zásah pre spomalenie kolesa sa berie s polovičnou váhou, aby nebola regulácia príliš agresívna, inak by sa nám pomocou jedného regulátora nepodarilo regulovať obe kolesá.

Príklad 2: Ak by sa cieľ nachádzal v bledej výseči, autíčko musí ísť rýchlejšie pravým kolesom a pomalšie ľavým.

Tieto podmienky, samozrejme, platia pre ľubovoľný prípad natočenia autíčka, rozdeľovacie rozhranie sa vždy predelí nanovo.

Celý kód pre ovládanie robotického vozidla nájdete implantovaný v jazyku C príloha F.

4 Hľadanie cesty

Pojem hľadanie cesty, alebo anglicky „path planning“, bol vyvinutý vo viacerých oblastiach, napríklad v umelej inteligencii, robotike alebo pri teórii riadenia. Preto sa môžeme stretávať s viacerými definíciami tohto pojmu.

V oblasti robotiky sa plánovanie cesty zaoberá problematikou presunutia robota z jedného miesta na druhé. S pokročilými technológiami a s novými matematickými riešeniami tento problém okrem trasy zahrnul mnohé ďalšie faktory, ako napríklad prekážky, dynamické prostredie, viacerých robotov alebo neistoty.

V umelej inteligencii predstavuje tento pojem hľadanie sekvencie logických úkonov, ktoré premieňajú začiatočný stav robota na cieľový stav. Takéto plánovanie zahŕňa veľa teoretických myšlienok, rozhodovacích procesov a mnohé iné faktory.

V teórii riadenia plánovanie cesty je otázkou stability, spätej väzby a optimálnosti. Ako môžeme vidieť, plánovanie cesty mobilných robotov je zložitý problém a existuje veľa prístupov a metód ako je ho možné riešiť.

Vo všeobecnosti sa v robotickom hľadaní cesty zameriavame na navrhovanie algoritmov, ktoré vytvárajú užitočné pohyby pomocou spracovania jednoduchých alebo komplexnejších geometrických modelov. V robotike môžeme rozdeliť plánovanie cesty na viacero skupín.

Jednou z najdôležitejších je plánovanie pohybu alebo plánovanie trajektórie. Táto časť práce bude zameraná na nájdenie algoritmu na plánovanie vhodnej trasy, umožňujúcej prejsť dráhu za čo najkratší čas. Plánovanie trasy je určené pre automatizáciu mechanických systémov, ktoré majú akčné členy, senzory a hlavne výpočtové schopnosti. Plánovanie pohybu a trajektórie sú definované ako základné potreby v robotike a sú opísané algoritmami.

Úlohou týchto algoritmov je premeniť špecifickú úlohu, na vysokej úrovni nezávislosti od človeka, do jednoduchého popisu, ako sa hýbať. Tieto potreby sú vysvetlené takzvaným pianovým problémom. Úloha je definovaná ako presný model domu s pianom ako so vstupom do algoritmu. Algoritmus musí vyhodnotiť, ako hýbať pianom z jednej izby do druhej bez toho, aby sa ním zasiahlo do iného predmetu alebo steny.

Plánovanie cesty robota je definované podobným spôsobom. Plánovanie pohybu zvyčajne ignoruje dynamiku a iné obmedzenia a zameriava sa predovšetkým na rotáciu riadeného objektu (robota). Až nedávne výskumy priniesli v tejto oblasti zohľadňovanie ďalších aspektov ako sú neistoty, diferenciálne obmedzenia, optimalita a podobne.

Plánovanie trajektórie sa zase vzťahuje na výsledok plánovania pohybu a vymedzuje riešenie pre pohyb vzhľadom na mechanické obmedzenia robota.

Zvyčajnou úlohou mobilného robota je navigácia v priestoroch budovy. Pre robota je potrebné vytvoriť obraz zmapovaného prostredia, v ktorom sa nachádza, určiť jeho presnú polohu na mape a nakoniec riešiť bezkolízny prechod do požadovaného miesta. Prvé dve úlohy sa dajú spojiť do jedného problému, známeho pod skratkou SLAM (simulačná lokalizácia a mapovanie). Navigácia robota je závislá od informácií získaných o prostredí pohybu. Ak robot nemá žiadny popis prostredia, je nútený používať iba svoje senzory, ktoré zachytávajú v určitom dosahu jeho okolie. Na základe týchto údajov, získaných robotom zo senzorov, môže tento vyhodnotiť okolie a pohybovať sa v ňom z jedného miesta na druhé. Takýto prístup sa nazýva reaktívna navigácia, pretože robot reaguje len na podnety senzorov.

Ak však má robot k dispozícií presnú mapu prostredia, potom môže využiť viacero spôsobov pre navigáciu a svoj pohyb optimálnou cestou. Toto sa nazýva globálna navigácia, alebo inak plánovanie trajektórie.

V tejto práci budeme rozoberať plánovanie trajektórie dvomi metódami, ktoré vedú k vytvoreniu bezkolíznej cesty pri použití viacerých robotov.

Každá metóda plánovania cesty je závislá od stavového priestoru. Tento priestor reprezentuje všetky možné pozície a orientácie robota. Existuje niekoľko spôsobov, ako opísať takýto priestor. Obyčajne je reprezentovaný implicitne plánovacím algoritmom, a preto ho môžeme rozdeliť na základe stavového priestoru.

Najrozšírenejšie používaný stavový priestor v robotike je sieťovo- metrická mapa. Hoci sa využíva aj geometrická, alebo topologická reprezentácia, metrická mapa je najvhodnejšia pre získavanie nových údajov o priestore. Jednotlivá bunka z mriežky reprezentuje iba malý kúsok priestoru, a preto metrická mapa môže narastať do obrovských rozmerov. Algoritmus, ktorý používame v našej práci, je založený práve na mriežkovej reprezentácii priestoru.

Väčšina plánovacích problémov zahŕňa postupnosť rozhodnutí, ktoré sa uplatňujú v priebehu času. Navyše pri formulovaní plánovania je potrebné zahrnúť, ako sa mení stavový priestor v priebehu vykonávania akcií. V mobilnej robotike je zvyčajne stavovo hodnotná funkcia vytvorená ako sekvencia stavov z konfiguračného priestoru, ktorú robot musí dosiahnuť. Pri každom plánovaní cesty sa zahŕňa počiatočný a koncový bod.

Vo všeobecnosti existujú dva druhy plánovacích neistôt, založených na type kritérií. Prvým je realizovateľnosť, nájdenie plánu, ktorý naisto privedie robota do cieľového stavu bez ohľadu na efektívnosť cesty. Druhá je optimalizácia, nájdenie vhodného plánu , ktorý optimalizuje výkon určitým spôsobom. V tejto práci budeme narábať s algoritmami, ktoré sú iba realizovateľné.

Vo všeobecnosti tento rozbor hľadania cesty veľmi dobre popisuje danú problematiku bol prevzatý a upravený z obsiahlej literatúry (Trebuňa F. Modelling of Mechanical and Mechatronic Systems. 2014. New York: Elsevier Procedia ISBN: 978-1-5108-0071-7 str. 59-69) informácie sme čerpali v rámci článku [16].

4.1 Algoritmy pre plánovanie trajektórie

V nasledujúcich kapitolách si predstavíme dve metódy, ktoré sme využívali pre nájdenie vhodných ciest. Každá metóda má vytvorenú kapitolu zvlášť pre vysvetlenie princípu vyhľadávania a podkapitolu pre implementáciu v Matlabe. Na konci tejto kapitoly budeme porovnávať obe metódy a vyberieme tú, ktorá je pre nás vhodnejšia.

4.1.1 Pravdepodobnostné plánovanie

Pravdepodobnostné plánovanie, anglicky „probabilistic roadmap“, skratka PRM, je jeden z najviac využívaných algoritmov na plánovanie. Za posledných desať rokov bola táto technika skúmaná v mnohých štúdiách rôznymi výskumníkmi. To viedlo k vytvoreniu rôznych prístupov. Každý riešiteľ obohatil vývoj algoritmu niečím novým a tým vznikali mnohé jeho varianty. V tejto práci nebudeme vzájomne porovnávať tieto odlišnosti, využijeme iba jeden variant, ktorý bol vyvinutý pre užívateľov Matlabu. Prakticky všetky prístupy k pravdepodobnostnému plánovaniu majú rovnaký základ, ktorý si vysvetlíme nasledovne.

Pravdepodobnostné plánovanie je algoritmus vyvinutý pre robotiku, ktorý rieši problém určenia cesty medzi počiatočnou a cieľovou lokáciou robota, pričom sa zabraňuje kolíziám s prekážkami. Myšlienkou PRM je náhodne rozmiestniť body v známom prostredí. Body sa môžu rozmiestniť hocikde vo voľnom priestore. Testovaním, či sa nachádzajú mimo prekážok, zabránime generovaniu nežiaducich bodov. Použitím plánovača sa vytvorí sieť pospájaných bodov. Plánovač má za úlohu vytvoriť priamku medzi susednými bodmi tak, aby sa nekrížila s prekážkou. Ak sa priamka kríži s prekážkou, plánovač ju nepridá do siete. Potom sa pridá počiatočná poloha robota a cieľové miesto, kam má robot dôjsť. Tým sa vytvorí cesta s okolitými vygenerovanými bodmi. Vyhľadávací algoritmus nájde bezkolíznu cestu medzi začiatočnou a konečnou pozíciou [17].

4.1.1.1 Aplikácia PRM

V rámci tejto práce budeme pracovať s už existujúcimi programami, ktoré sú súčasťou matlabovského robotického toolboxu. Tento toolbox poskytuje algoritmy pre vývoj autonómnych robotických aplikácií pre letecké a pozemné vozidlá, robotické paže a humanoidné roboty. Pre nás je najdôležitejšie, že tento toolbox zahŕňa aj algoritmus pre pravdepodobnostné plánovanie.

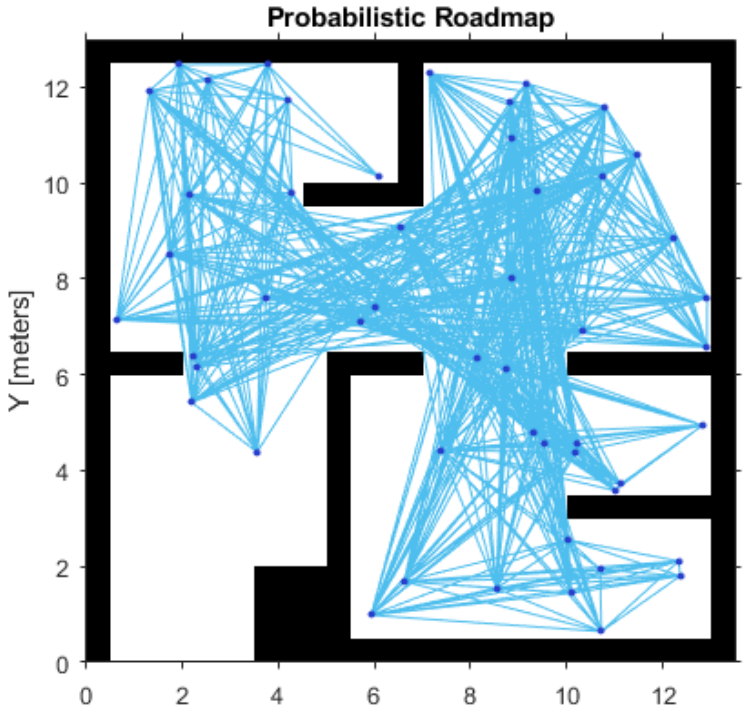
Podľa logiky, ktorú sme si popísali v kapitole 4.1.1 (Pravdepodobnostné plánovanie), je ako prvé potrebné poznať mapu prostredia, kde sa robot bude pohybovať. Mapu si vygenerujeme ako pole jednotiek a núl. Jednotky budú predstavovať prekážku a nuly prázdne miesto. Mapu si uložíme ako maticu a pomocou príkazu `load()` si ju vieme kedykoľvek načítať. Funkcia robotického toolboxu BinaryOccupancyGrid nám umožňuje vytvárať ľubovoľné 2D prekážky. Vieme jednoducho integrovať údaje zo senzorov o prekážke a umiestniť ich polohu do mapy. Táto funkcia používa hodnotu pravda, teda 1 pre reprezentáciu obsadeného miesta (prekážka) a nepravda, čiže 0 reprezentujúca voľné miesto.

```
load('mapa.mat')
mapa = robotics.BinaryOccupancyGrid(flipud(Mapa),10);
```

Po vytvorení mapy sa vygenerujú náhodne body v celom priestore pomocou príkazu robotics.PRM(). Vstupmi do tejto funkcie sú mapa, vytvorená predošlými príkazmi a počet bodov, ktoré sa majú vygenerovať.

```
prm = robotics.PRM(Map,50);
show(prm)
```

Následne sa vytvorí sieť, ktorá prepojí body medzi sebou tak, že cesta nikdy nebude križovať prekážku, viď. obrázok č.20.

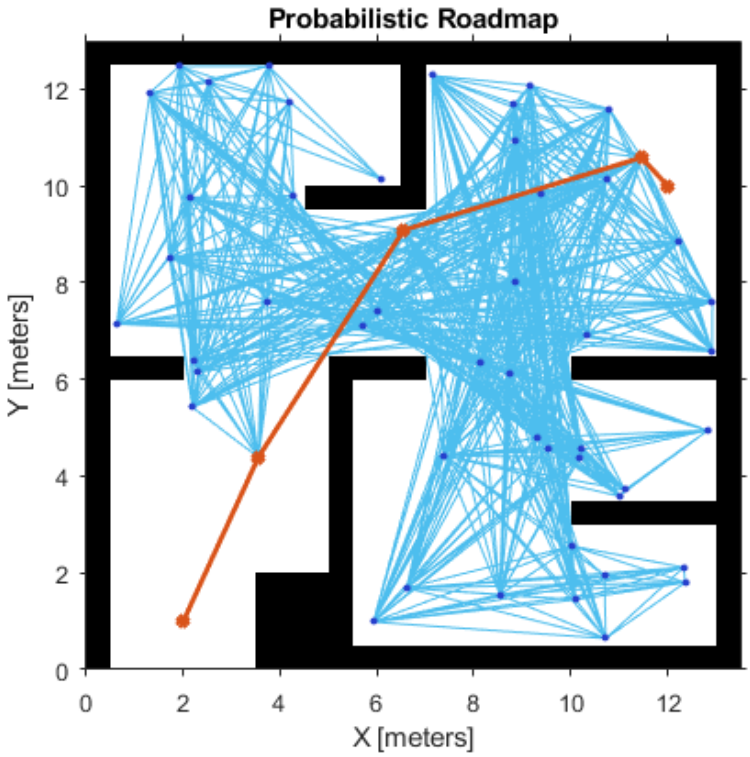


Obrázok 20: Mapa vygenerovaná pomocou robotického toolboxu

Ak sa pridá väčší počet bodov, vzniká oveľa komplexnejšia sieť. Táto poskytuje viacero možností na vylepšenie správnej cesty. Teraz už stačí len pridať polohu robota (začiatok) a cieľ (koniec), kam sa má dostať, v tvare vektora [x, y]. Cez príkaz `robotics.findpath()` nájdeme požadovanú cestu. Vstupom do funkcie je mapa s vygenerovanými bodmi, začiatočná a konečná poloha robota.

```
zaciatok = [2 1];
koniec = [12 10];
path = robotic.findpath(prm,zaciatok,koniec)
show(prm)
```

Vvýsledkom riešenia je bezkolízna trasa, ktorá vedie zo štartu do cieľa obrázok č.21.



Obrázok 21: Nájdená trasa pomocou PRM

Pre kompatibilitu s našimi programami sme si vytvorili zvlášť funkciu, ktorá obsahuje algoritmus pre nájdenie trasy. Vstupom do nej je poloha robota zaznamenaná z OptiTracku, premenená na metre a miesto, kam sa má robot dostať. Aby sme zabezpečili reakciu na dynamiku prostredia, musíme túto funkciu volať vždy nanovo, aby sa prepočítavala trasa a zabránilo sa tak nežiaducej zrážke s pohyblivými objektami. Práve tu je problematický bod nami zvoleného algoritmu. Je síce rýchly a spoľahlivý, no vždy, keď sa prepočítava trasa, tak sa nájde nová, bez ohľadu na to, aká bola predošlá. Tým pádom robot síce nájde cestu, ale za pomerne dlhú dobu príde do cieľa a celková trasa je nekoordinovaná a pohyb robota je trhaný. Informácie o aplikácií a robotickom balíčku pre Matlab sme čerpali z [18]. Algoritmus je súčasťou prílohy D.

4.1.2 Metóda plánovania A*

Jednou z priorít práce je aj vytvorenie vlastného algoritmu na nájdenie vhodnej trasy. Podstatné bolo vytvoriť ho tak, aby bol dostatočne rýchly, v určitých situáciách pracoval optimálne a aby bol prispôsobiteľný robotickým vozidlám. Rozhodli sme sa pre aplikáciu A* algoritmu, ktorý je

nazývaný aj prvý najlepší vyhľadávací algoritmus. Je vhodný pre náš systém, keďže môže byť aplikovaný pre metrický alebo topologický konfiguračný priestor. Tento algoritmus sa v súčasnosti používa a stále je zaradený medzi najlepšie algoritmy na plánovanie trajektórie. Dôvodom jeho úspechu je, že každá bunka v konfiguračnom priestore má priradenú hodnotu f(n).

$$f(n) = g(n) + h(n) \tag{29}$$

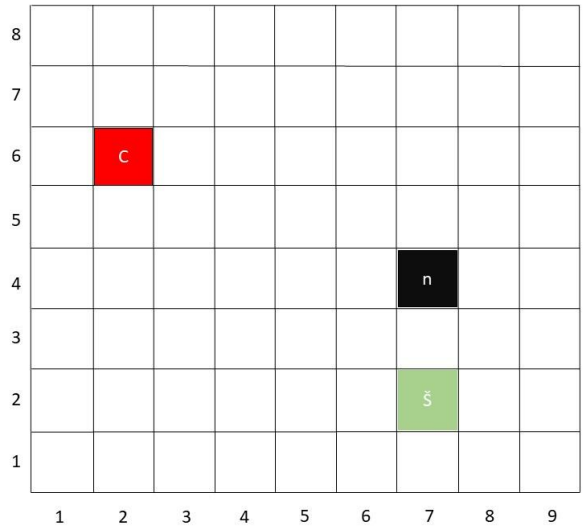
V tomto prípade g(n) predstavuje vzdialenosť od počiatočného bodu do bodu n a h(n) je heuristický odhad veľkosti najkratšej vzdialenosti z bodu n do cieľa. Každá susedná bunka momentálnej bunky sa hodnotí hodnotou f(n). Kritérium pre výber bunky je hodnota f(n), bunka s najnižšou hodnotou, sa vyberá ako ďalšia vhodná. Výhodou tohto algoritmu je, že kritérium pre výber buniek, môže byť prispôsobivé. Vďaka tomuto kritériu si vieme algoritmus prispôsobovať vlastným potrebám. Jednoducho môžeme pridávať k hodnotám jednotlivých buniek f(n) náklady na energiu, čas, bezpečnosť dokonca aj náročnosť, ak by sme chceli prekážku preletieť a mnohé ďalšie podmienky [13], [19].

Cieľom našej práce nebolo optimalizovať žiadnu z podmienok energie, časovej náročnosti cesty, alebo vzdialenosti. Zamerali sme sa čisto iba na vytvorenie algoritmu, ktorý nájde spoľahlivo cestu z polohy robota do cieľového miesta.

4.1.2.1 Výpočet kritéria f(n)

Predstavme si mriežkový súradnicový systém, teda graf, ktorého súradnice sú reprezentované jednou bunkou, viď. obrázok č.22. Na základe tejto predstavy si môžeme vysvetliť výpočet hodnoty f(n). Ako prvé je nevyhnutné zadefinovať hodnotu prejdenia z jednej bunky do druhej horizontálnym aj vertikálnym spôsobom. Tento pohyb z jednej bunky do druhej je jednoduchý, bude mu teda pridelená hodnota za každý prejdený cm. Táto hodnota bude rovná 10, keďže v našom prípade jedna bunka predstavuje 1dm^2. Diagonálny pohyb je už zložitejší, keďže sa jedná o pohyb v horizontálnej a vertikálnej polohe naraz. Hodnota mu bude pridelená na základe Pytagorovej teorémy, rovnica číslo (30). Vypočítaná hodnota bude približne rovná $14 \approx \sqrt{10^2 + 10^2}$.

$$C = \sqrt{A^2 + B^2} \tag{30}$$



Obrázok 22: Mriežkový súradnicový systém

Červený štvorec predstavuje cieľ, zelený štart a čierny predstavuje n-té políčko. Vektorovým zápisom súradnice štartu sú Š = [7, 2], súradnice cieľa sú C = [2, 6] a n-tého políčka sú N = [7, 4]. Dôležité je zistiť, koľko diagonálnych, vertikálnych a horizontálnych pohybov musíme vykonať, aby sme sa dostali z jedného políčka na iné. Počet horizontálnych a vertikálnych pohybov môžeme zlúčiť. Tento počet krokov od n-tého políčka ku štartu (Hš) a cieľu (Hc) vypočítame podľa rovníc (31/a) a (31/b). Počet diagonálnych pohybov k štartu (Vš) a cieľu (Vc) vypočítame z rovníc (32/a) a (32/b).

$$Hc = |min(|N - C|) - max(|N - C|)| \quad (31/a)$$

$$H\check{s} = \left|min\left(\left|N + \check{S}\right|\right) - max\left(\left|N - \check{S}\right|\right)\right| \quad (31/b)$$

$$Vc = min|abs(N - C)| \quad (32/a)$$

$$V\check{s} = min\left|abs\left(N - \check{S}\right)\right| \quad (32/b)$$

Na základe týchto rovníc vieme vypočítať hodnotu kritéria f(n), rovnica číslo (33).

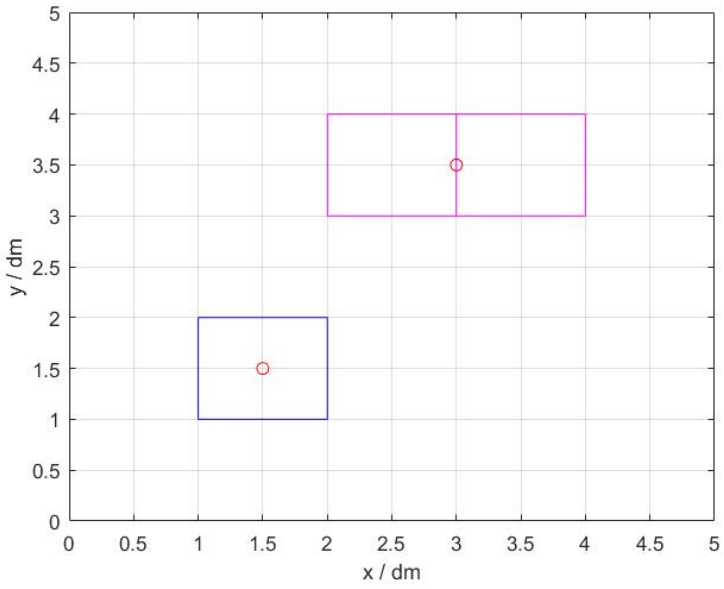
$$f(n) = g(n) + h(n) = (H\check{s} * 10 + V\check{s} * 14) + (Hc * 10 + Vc * 14) \quad (33)$$

$$g(n) = (H\check{s} * 10 + V\check{s} * 14) \quad (34)$$

$$h(n) = (Hc * 10 + Vc * 14) \quad (35)$$

4.1.2.2 Aplikácia A*

Celý plánovací algoritmus sme naprogramovali v Matlabe, aby bol kompatibilný s našimi už existujúcimi programami. Pred tým, ako sa začne celý proces plánovania, je potrebné vedieť, v akom prostredí sa robot má pohybovať (veľkosť miestnosti, pozícia prekážok, terén atď.). Dôležité je, aby bolo známe, kde sa v ňom robot nachádza a kde je jeho cieľ. Na vytvorenie digitálnej mapy sme využili v Matlabe maticu s rozmermi prostredia v decimetroch. Riadky predstavujú x-ovú a stĺpce zas y-ovú súradnicu bunky v matici sú poňaté ako štvorec o ploche 1 dm^2, kde riadok a stĺpec reprezentuje pravý horný roh štvorca. V bunkách matice budú jednotky reprezentovať prekážky a nuly zase prázdne miesto. Celá mapa má dookola prekážky, teda jednotky, aby sa zabránilo vychádzaniu robota z požadovaného územia. Keďže sa s prvkami matice dá pracovať len ako s celými číslami, bolo potrebné zaokrúhľovať všetky informácie o polohe. V našom prípade sme ich zaokrúhlili na decimetre. Toto zaokrúhlenie nespôsobuje problém. V reálnom svete má robot určité rozmery, v našej digitálnej mape bude predstavovať jednu bunku, čiže 1 dm^2. Budeme uvažovať, že všetky prekážky majú rovnaké rozmery, a to 5x5 cm , ale nie vždy sa musia nachádzať uprostred bunky, takže ich umelo nafukujeme podľa ich polohy. Toto nafúknutie zabraňuje robotovi pri reálnom pohybe naraziť do prekážky. Na nasledovnom obrázku č.23 môžeme vidieť príklad nafúknutia prekážky (štvorec), keď poznáme len jej stred (červený kruh).



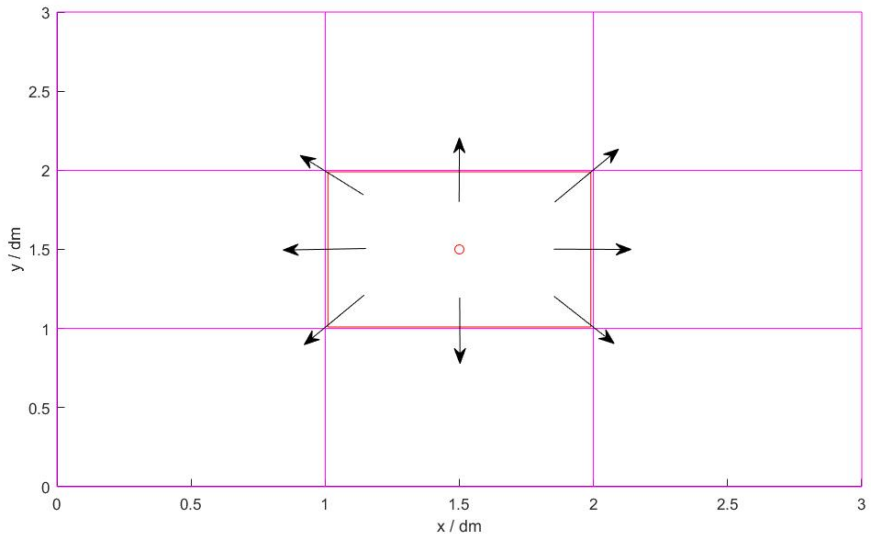
Obrázok 23: Nafukovanie prekážok

Všetky vybrané a otestované bunky sa ukladajú do matice s názvom (prehľadané), ktorá má počet riadkov podľa prehľadaného počtu buniek (m) a šesť stĺpcov. Matica má nasledovný tvar a obsah.

$$prehľadané = \begin{bmatrix} x & y & g(n) & h(n) & skontorolované & pôvod \\ : & : & : & : & : & : \\ x_m & y_m & g(n)_m & h(n)_m & skontrolované_m & pôvod_m \end{bmatrix} \quad (36)$$

Prvý a druhý stĺpec obsahuje x-ovú a y-ovú súradnicu, g(n) je hodnota vzdialenosti od pozície robota po n-tú pozíciu bunky, h(n) je hodnota vzdialenosti od n-tej bunky po cieľ. Skontrolované predstavuje, či dané súradnice už boli vybraté ako vhodné, a teda majú priradené hodnotu 1. Ak ešte neboli zvolené za vhodné, tak majú pridelenú hodnotu 0. V poslednej bunke (pôvod) sa nachádza číslo, z ktorého bola daná súradnica vygenerovaná.

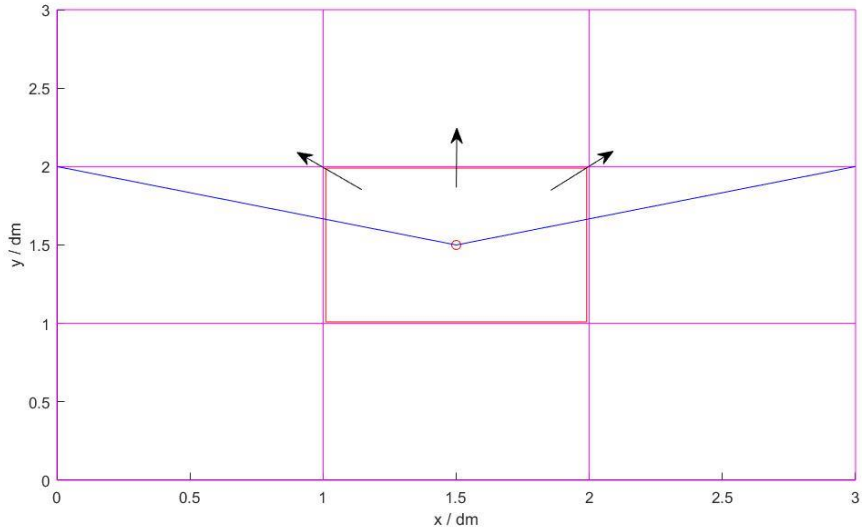
Po vytvorení digitálneho konfiguračného priestoru nasleduje algoritmus, ktorý vyhľadáva bezkolíznu cestu. Tento algoritmus bude prehľadávať bunku po bunke od začiatočnej pozície robota, až kým nenájde bunku, ktorá sa rovná súradniciam cieľa. Pomocou podmienky si zadefinujeme stavový priestor robota, teda jeho všetky možné pozície a orientácie. Tento priestor si nastavíme tak, že robotovi povolíme pohyby v určitých smeroch (uhloch), čiže môže prechádzať aj šikmo. Bunky sa teda budú vyhľadávať vo všetkých smeroch, viď. obrázok č.24.



Obrázok 24: Stavový priestor robota

Vždy, keď sa začne vyhľadávanie, prvé bunky sa volia podľa iného stavového priestoru, ktorý je vyhradený podľa toho, akým smerom je robot natočený, viď. obrázok č.25. Výber buniek teda prebieha podľa toho, či sa bunka nachádza +/- 60° od natočenia robota v našom súradnicovom

systéme. Natočenie robota z OptiTracku sa prerátava rovnako, ako v časti 3.3.1 (Uhol natočenia) a požadovaný uhol medzi robotom a zvolenou bunkou ako v časti 3.4 (Pohyb robota).



Obrázok 25: Počiatočný stavový priestor robota

Zakaždým je potrebné vybrať určitý počet buniek, na začiatku sú to tri, a potom osem, postupne až kým sa nenájde cieľová bunka. Postupné vyberanie buniek prebieha vo vnorených cykloch. Bunka sa zvolí z okolia robota a otestuje sa, či jej súradnice nie sú už v matici prehľadané. Ak sa jej súradnice nevyskytujú v zmienenej matici, tak prebehne druhá kontrola, či sa na týchto súradniciach v digitálnej mape nenachádza prekážka. Ak na nej prekážka neleží, vypočítajú sa hodnoty g(n) rovnica číslo (34) a h(n) rovnica číslo (35). Tento cyklus prejde cez určený počet buniek, všetky sa postupne priraďujú do matice prehľadané aj so všetkými potrebnými hodnotami. V ďalšom kroku sa vojde do matice prehľadané a hľadá sa riadok, ktorý ešte nebol vybraný a má najmenšiu hodnotou f(n). Ak sa nájdu dve rovnaké hodnoty f(n) a sú zároveň najmenšie, volí sa tá bunka, ktorá ma najmenšiu hodnotu h(n).

Celý tento postup sa opakuje, až kým sa zvolená bunka nerovná cieľu. Následne spätným vyhľadávaním cez posledné políčko, posledného riadku matice prehľadané (pôvod) dostaneme hľadanú cestu z počiatočného miesta robota do cieľového.

Keďže potrebujeme, aby tento algoritmus pracoval dynamicky, funkciu si voláme periodicky každých 0,4 sekundy a do autíčka posielame len prvú súradnicu trajektórie. Zvyšok nepoužitej trajektórie sa vymaže. Ak nenastane zmena polohy prekážok a robot neprešiel po nerovnosti, nová trajektória je skoro totožná s minulou a robot plynule pokračuje. Týmto spôsobom sme

vytvorili dynamické prostredie, ktoré vyhľadáva bezkolíznu cestu pri výskyte pohyblivých prekážok.

Pri danom algoritme môžeme pozorovať niekoľko nevýhod. Keďže pracuje heuristicky, nepočíta pri hľadaní trajektórie s prekážkami, preto môže vzniknúť nechcené predĺženie nájdenej cesty. Nájdená cesta je ideálna za predpokladu, že sa pri vyhľadávaní nenarazí na prekážku. V mnohých prípadoch, ak je prostredie príliš komplikované, má nájdená trajektória ďaleko od optimálnej. Tento problém je možné čiastočne vyriešiť dynamikou algoritmu. Pri každej perióde sa prepočítava trajektória nanovo, tým pádom sa berie iná počiatočná pozícia robota a v určitých situáciách môže dôjsť k zásadnému vylepšeniu trajektórie. Algoritmus pre vyhľadávanie trajektórie metódou A* nájdete v prílohe C.

4.1.3 Porovnanie metód

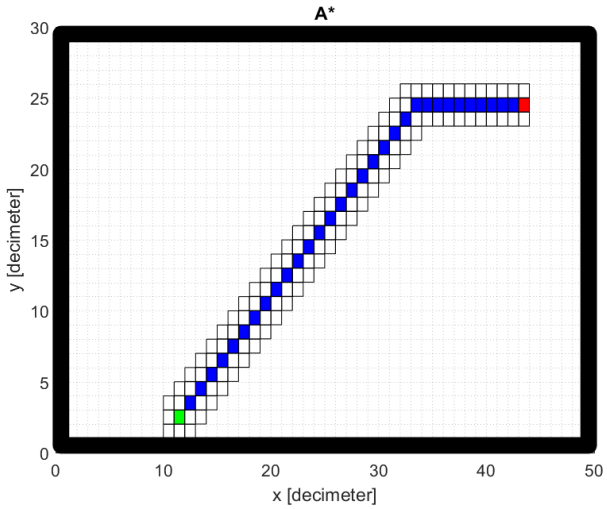
V tejto kapitole sa budeme venovať porovnaniu metód uvedených v kapitolách 4.1.2, 4.1.1. Obe metódy sme simulovali na PC a následne sme si ich vykreslili v Matlabe.

Tabuľka 2: Simulačné porovnanie vyhľadávacích metód

Č. simulácie	Prostredie č.	Štart	Cieľ
1.	1	[120, 30]	[440, 250]
2.	2	[120, 30]	[440, 250]
3.	3	[120, 30]	[440, 250]
4.	4	[50, 200]	[450, 200]
5.	5	[50, 150]	[380, 50]

Na nasledujúcich simuláciách si ukážeme výhody aj nevýhody oboch metód. Simulácie sme previedli v Matlabe za tých istých podmienok v oboch situáciách, v prostredí o veľkosti 5x3 m. Všetky simulácie sme si umelo pripravili tak, aby ukázali správanie sa metód v rôznych situáciách a to, ako sa s nimi vysporiadali. Metóda PRM v robotickom toolboxe nafukuje rozmery prekážok trošku inak. Prekážky sú o niečo väčšie, než nami požadované rozmery, ale ovplyvnenie simulácie je minimálne.

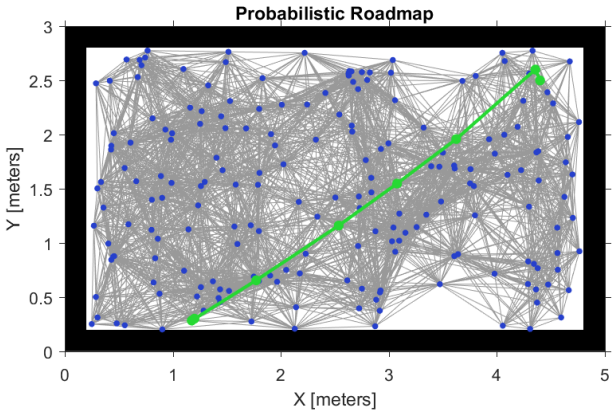
Mapa číslo 1. bez prekážok



Čas výpočtu: 0.006583s

Prehľadané bunky: 146

Dĺžka trasy : 396.9 cm

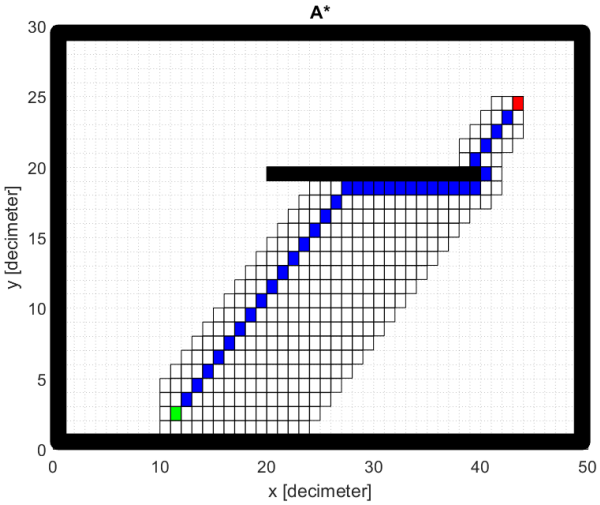


Čas výpočtu: 0.142943 s

Vygenerované body: 200

Dĺžka trasy : 408.3 cm

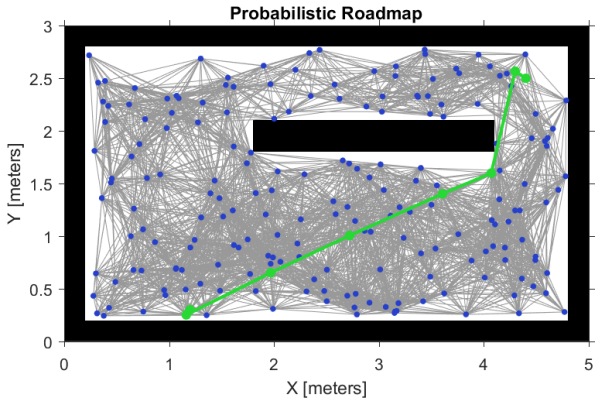
Mapa číslo 2. s prekážkou



Čas výpočtu: 0.046056 s

Prehľadané bunky: 351

Dĺžka trasy : 416.9 cm

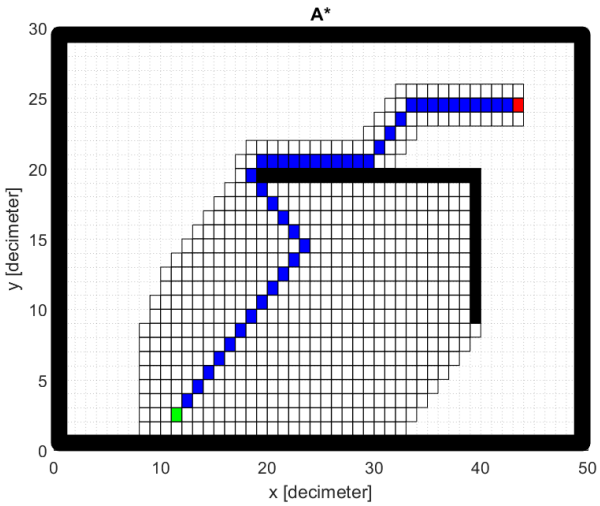


Čas výpočtu: 0.302854 s

Vygenerované body: 200

Dĺžka trasy : 438.4 cm

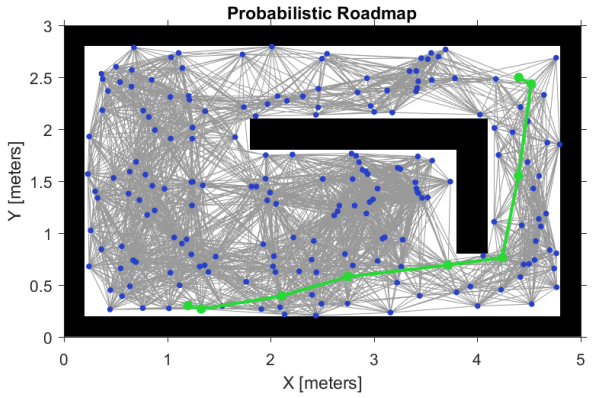
Mapa číslo 3. s prekážkou



Čas výpočtu: 0.070891 s

Prehľadané bunky: 628

Dĺžka trasy : 496.9 cm

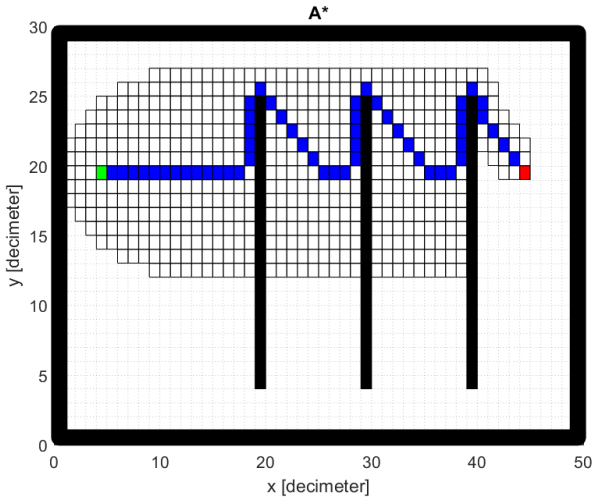


Čas výpočtu: 0.298287 s

Vygenerované body: 200

Dĺžka trasy : 492.7 cm

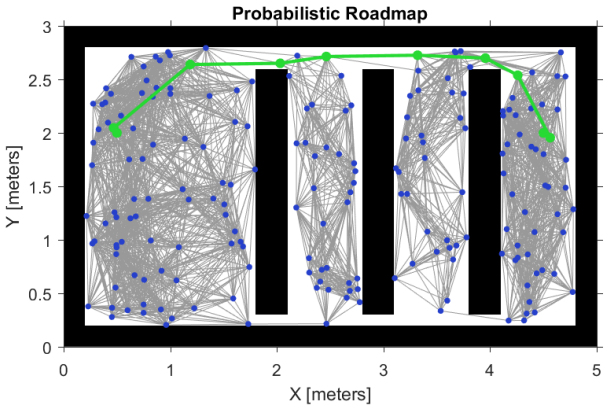
Mapa číslo 4. s prekážkami



Čas výpočtu: 0.067571 s

Prehľadané bunky: 571

Dĺžka trasy : 615.2 cm

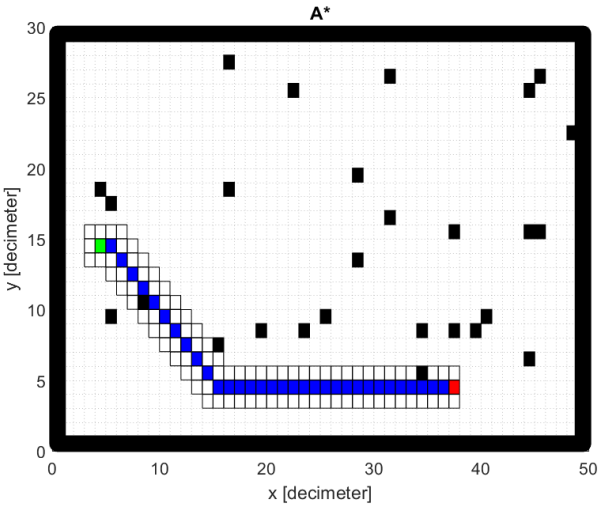


Čas výpočtu: 0.299848 s

Vygenerované body: 200

Dĺžka trasy : 494.1 cm

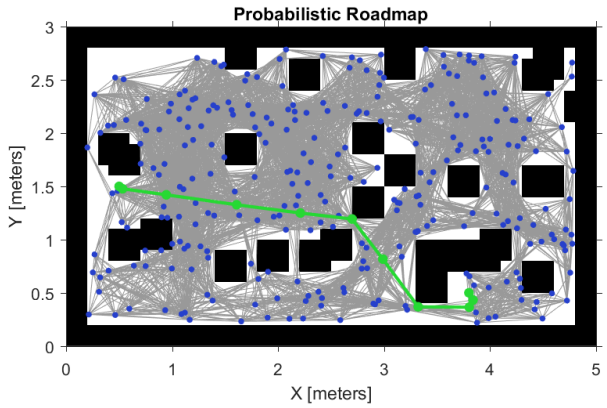
Mapa číslo 5. s prekážkami



Čas výpočtu: 0.031926 s

Prehľadané bunky: 125

Dĺžka trasy : 361.4 cm



Čas výpočtu: 0.477362 s

Vygenerované body: 300

Dĺžka trasy : 389.4 cm

Na mape č.1 môžeme vidieť, že metóda A* prehľadala minimum buniek a získali sme celkom dobrú cestu, ale stále dosť vzdialenú od ideálnej. Na rozdiel od toho metóda PRM, ktorá vygenerovala 200 buniek, aby mohla nájsť požadovanú cestu. Nájdená cesta je ešte nepresnejšia.

Na mapách číslo 2, 3 a 4 môžeme vidieť, ako sa správajú tieto metódy v prítomnosti prekážok. Zatiaľ čo heuristický algoritmus A* robil zbytočné obchádzky, ktoré mu predlžovali trajektóriu, PRM algoritmu sa podarilo veľmi dobre nájsť vhodnú cestu, ktorá bezproblémovo obišla prekážky. Rozdiel v nájdených cestách si môžeme najlepšie všimnúť v prípade mapy číslo 4.

V poslednom prípade, na mape č.5, máme náhodne vygenerované prekážky v rámci celej mapy. Vo väčšine prípadov v takomto prostredí, kde sú malé prekážky, dominuje metóda A*, ktorá nájde krátku cestu. Metóda PRM mala značný problém nájsť dostatočne krátku cestu. Museli sme vygenerovať o 100 buniek viac, ako pri predošlých simuláciách touto metódou a to zapríčinilo aj nárast času výpočtu trajektórie.

Vo všetkých prípadoch bola metóda A* omnoho rýchlejšia ako PRM. Rýchlosť A* metódy závisí od počtu prehľadaných buniek. Čím je počet prehľadaných buniek menší, tým bude metóda rýchlejšia. V tomto prípade sme mali dostatočne veľkú časovú rezervu pre našu periódu vzorkovania a vedeli by sme ju aplikovať aj na omnoho väčšie mapy. Naopak, metóda PRM narážala na naše limity komunikácie 0,4 sekundy. Dôvodom bol veľký počet generovaných buniek, ktoré zabezpečujú nájdenie dostatočne krátkej cesty. Pre náš cieľ sme zvolili za vhodnejší nami naprogramovaný algoritmus, s ktorým sa dá pracovať ďalej a môžeme si ho upravovať podľa našich potrieb.

4.2 Koordinácia s prioritou

Posledným cieľom našej práce bolo aplikovať vyhľadávanie cesty nielen v prostredí s dynamickými prekážkami, ale aj v prítomnosti viacerých robotov, ktoré plnia svoj cieľ. Takýmto spôsobom sa už približujeme k reálnym situáciám, nastávajúcim v praxi tam, kde sa využívajú automatizované vozidlá. Môžeme si ho všimnúť napríklad vo veľkých skladoch, na letiskách, parkoviskách či dokonca vo veľmi zložitej podobe, akú predstavuje cestná premávka.

V našom prípade sa bude jednať o vytvorenie systému, kde medzi sebou komunikujú viaceré autíčka. Tieto autíčka môžu teda slúžiť na presun nákladu z jedného miesta na druhé, alebo môžu pomocou merania určitej fyzikálnej veličiny (teploty, svietivosti, atď.) hľadať miesto zdroja. Zadanou úlohou je vytvoriť algoritmus, ktorý bude sledovať roboty a vytvorí také cesty, ktoré zabránia ich stretu.

Rozhodli sme sa vytvoriť súbor robotov, v ktorom má každý priradenú prioritu. Hlavnou myšlienkou je, že robot s najväčšou prioritou ignoruje všetkých ostatných a zvyšné sa mu podriaďujú.

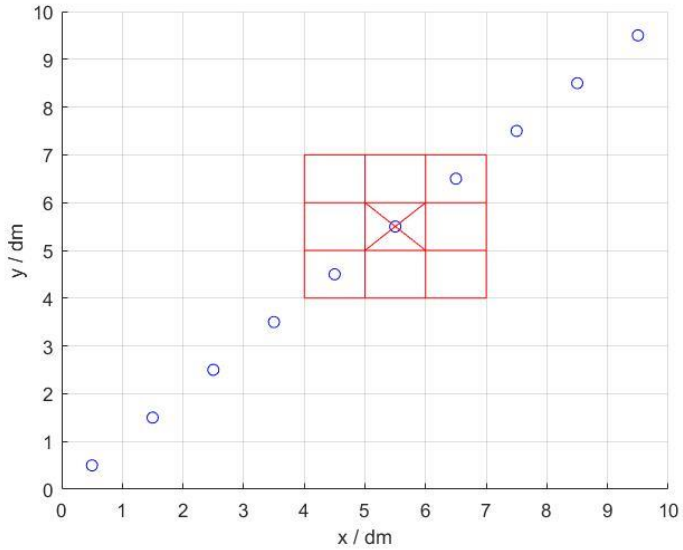
Príklad : V prítomnosti troch robotov jeden má najvyššiu prioritu a ostatné sa mu prispôsobujú, robot so strednou prioritou sa prispôsobuje iba hlavnému robotovi a posledný robot sa prispôsobuje obom.

Jedným z podstatných problémov bolo, akým spôsobom sa majú roboty prispôsobovať. Rozhodli sme sa pre dve možnosti. Po prvé, robot s menšou prioritou obíde robota s väčšou na základe novo vypočítanej trajektórie. Po druhé, ak je novo vypočítaná trajektória nevýhodná podľa nami vytvoreného kritéria, tak robot pokračuje podľa starej trajektórie. Tesne pred kolíziou na zvolenej bunke zastane a počká, kým sa mu cesta uvoľní. Kritérium sme si vytvorili na základe úspory času, kde zastavenie robota je rovné dvom bunkám určeným na pohyb. Za tento čas sa stihne robotovi uvoľniť cesta a môže pokračovať ďalej. Takže ak je novozvolená trajektória väčšia iba o dve bunky a menej, ako pôvodná, robot ide podľa nej. Akonáhle hocijaký robot zastane či príde na požadovanú pozíciu, stáva sa automaticky obyčajnou prekážkou pre ostatných.

4.2.1 Aplikácia koordinovaného plánovania

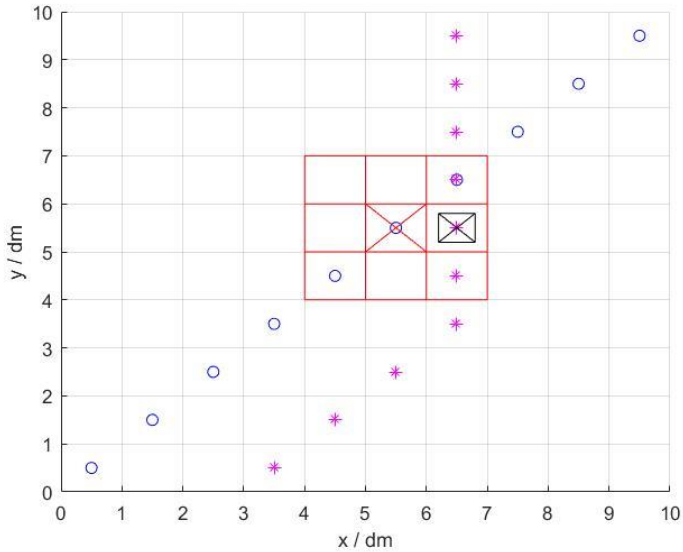
V tejto práci sme uvažovali o systéme troch robotických vozidiel, ale programy sa dajú ľahko rozšíriť pre ich väčší, alebo menší počet. Na základe logiky, ktorú sme si vysvetlili v predošlej kapitole, sme boli schopní vytvoriť funkciu pre takéto plánovanie. Funkcia sa periodicky zavolá každých 0,4 sekundy. Jej vstupmi sú natočenia a polohy všetkých robotických autíčok a všetky polohy prekážok.

Ako prvé sme vypočítali trajektórie pre každé autíčko zvlášť. Trajektórie sa hľadajú algoritmom, ktorý sme si predstavili v kapitole 4.1.2 (Metóda plánovania A*). Po nájdení všetkých trás skontrolujeme, či pri strete dvoch trajektórií nedochádza k vzniku kolízie, viď. obrázok č.26. Pod kolíziou si nepredstavujeme len stret dvoch trajektórií v tom istom čase na jednej bunke. Pri porovnávaní trajektórií pridávame tej s vyššou prioritou určité okolie, ktoré zabraňuje pri reálnom pohybe stretu autíčok. Toto okolie, obrázok č.26, sa pohybuje podľa porovnávaných buniek. Teda ak sa v rovnakom čase stretáva bunka trajektórie s nižšou prioritu, s okolím v tom čase prideleným bunke s výšou prioritou, nastáva kolízia.



Obrázok 26: Okolie bunky s vyššou prioritou

Prvý robot má najväčšiu prioritu, druhý strednú a tretí robot najmenšiu. Podľa tejto priority porovnávame druhého robota s prvým, tretieho s prvým a tretieho s druhým. Ak sa nájde zhoda a v jednom z týchto porovnávaní vzniká kolízia dvoch trajektórií, obrázok č.27, je nutné trajektóriu s nižšou prioritou prepočítať.



Obrázok 27: Kolízia dvoch trajektórií

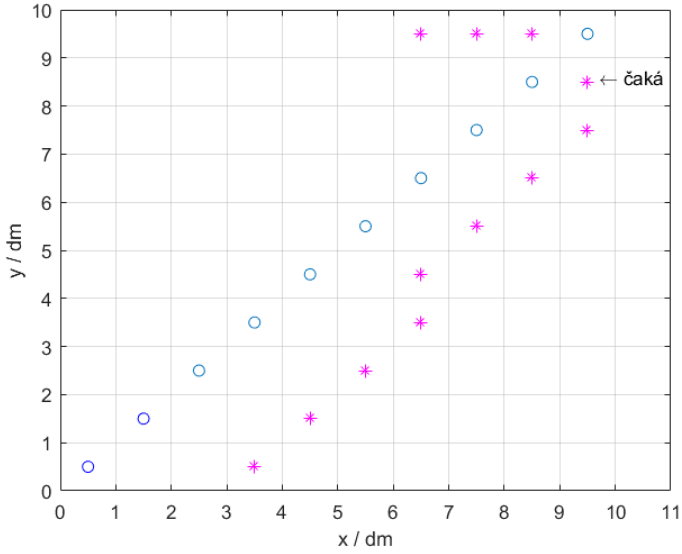
Modré značky znázorňujú robota s väčšinou prioritou, fialové robota s nižšou prioritou. Začiatočná pozícia hlavného robota bola na bunke [1, 1] a konečná na bunke [10, 10], podriadený

robot začínal z bunky [4, 1] a cieľ mal na bunke [7, 10]. Postupným prehľadávaním sa zistila kolízia v okolí hlavnej bunky číslo [6, 6] s podriadenou bunkou [7, 6] (zvýraznená v čiernom štvorci).

Prepočítavanie prebieha len od tej časti trajektórie, kde vznikla kolízia, zvyšná časť sa ponecháva nezmenená. Časť trajektórie, ktorú je nutné zmeniť, sa vymaže. Ponechajú sa iba dve bunky, prvá je bunka pred kolíziou a tá vstupuje do funkcie pre nájdenie cesty ako počiatočná pozícia robota. Druhá ponechaná bunka je tá, z ktorej bolo generované okolie kolízie. Táto vstupuje do rovnakej funkcie, ako prekážka s prideleným okolím. Cieľ robota vstupuje do funkcie nezmenený.

Funkcia na vyhľadávanie trajektórie ostáva nezmenená a je rovnaká, ako v kapitole 4.1.2 (Metóda plánovania A*). Po opravení trajektórie sa porovná nová trajektória so starou podľa kritéria, ktoré sme si vysvetlili v predošlej kapitole. Ak podľa kritéria nová trajektória vyhovuje, pokračuje sa v porovnávaní od miesta tejto kolízie. Ak sa vyskytla nová kolízia, tak sa opraví podľa rovnakého postupu. Ak však nová trajektória nevyhovuje kritériu, ponecháva sa pôvodná trajektória. Miesto kolízie podradeného robota sa uloží do matice na to určenej a pokračuje sa vo vyhľadávaní, ako keby kolízia nenastala.

Výstupom z funkcie na koordinované plánovanie sú len prvé súradnice nájdených trajektórií. Zvyšné časti sa vymažú. Vystupujúce súradnice sa ešte pred odoslaním kontrolujú, či sa na nich nenachádzala kolízia. Ak sa takáto súradnica nájde, namiesto nej sa odošle momentálna súradnica robota ako cieľ, kam mal prísť. Robot na to zareaguje tak, že zastane a bude stáť, až kým kolízia nezmizne z jeho trajektórie. Na nasledujúcom obrázku, viď. č.28, si môžeme pozrieť upravenú trajektóriu pre robota s nižšiu prioritou. Táto koordinovaná trajektória zabraňuje kolízií dvoch robotov a zabezpečuje tak plynulý chod. Na nasledujúcom grafe môžeme vidieť upravenú trajektóriou pre podradeného robota. Táto trajektória je zmiešaná, lebo robot obchádzal miesto kolízie niekoľko krát a pri jednej kolízií vyhodnotil, že je nutné počkať, kým sa cesta uvoľní (v obrázku je bunka označená ako „čaká”).



Obrázok 28: Prepočítaná trajektória

Výhodou takéhoto algoritmu je jeho rýchlosť a istota správneho chodu. Aj keby sa zmenil cieľový bod hocijakého robota, je zaistená podmienka nekolízneho pohybu. Jednu z nevýhod programu je, že dané trajektórie nepracujú optimálne, nepočítajú s budúcim vývojom trajektórie a tým sa môže predĺžiť trajektória obchádzajúceho robota. Tento problém by sa dal ľahko vyriešiť. Pri nájdení kolízie by robot iba zastal a počkal, kým by sa mu cesta uvoľnila. Toto, ale nebolo naším cieľom. My sme sa prioritne snažili dosiahnuť plynulý pohyb. Celý algoritmus koordinovaného plánovania pre viacerých robotov je zahrnutý v prílohe E.

Záver

Myšlienkou projektu bolo napodobniť vhodné prostredie pre prácu robotov a následne vytvoriť systém, ktorý bude bezproblémovo pracovať v takýchto podmienkach za pomoci pohybového senzora OptiTrack.

Priestory sme si predstavili ako jednoduchú miestnosť s prekážkami, v ktorej sa roboty budú pohybovať zo zadanej počiatočnej pozície do cieľa. Úlohou týchto robotov bolo teda hýbať sa v priestore tak, aby nenarazili do žiadneho iného objektu a prišli na zvolené miesto. Celá realizácia spracovávania údajov zo senzora a následné vytváranie vhodných trajektórií bolo realizované v Matlabe. Pohyb robotického vozidla bol riadený pomocou algoritmu implementovaného v jazyku C. Celú prácu sme rozdelili do štyroch hlavných kapitol.

Prvá kapitola bola venovaná senzoru OptiTrack a programu, ktorý s ním súvisí. Vysvetlili sme, ako pracuje a aké je jeho nastavenie.

V druhej kapitole sme spracovali princíp komunikácie medzi robotom a senzorom OptiTrack. Keďže túto komunikáciu nebolo možné previesť priamo, museli sme roboty pripojiť na wifi sieť a vytvoriť server v rámci jedného počítača. Tento server bol realizovaný pomocou pripraveného programu NadNet SDK s rozhraním pre Matlab. NatNet SDK je server, ktorý je výhradne určený pre užívateľov OptiTracku. Počítač sprostredkováva túto komunikáciu a zároveň prijíma údaje z kamery. Nebolo jednoduché odosielať údaje v rámci dvoch jazykov a zároveň ich odosielať po sieti. Ak sme chceli odosielať údaje po sieti, museli sme ich zakódovať v určitej forme. Vyriešili sme to tak, že sme zobrali dáta z Matlabu a vytvorili z nich textový reťazec, ktorý sa dal poslať po sieti. Následne sme v Arduine odkódovali prijaté údaje pomocou jazyka aJson.

V tomto bode je začiatok tretej kapitoly, opisujúcej pohyb robota. Ten bol realizovaný na základe prijatých údajov o polohe cieľa, robota a jeho natočení v našom súradnicovom systéme. Údaj o natočení sme museli spracovať a previesť z Euilerových uhlov do 360° systému.

O to, či sa má robot natáčať doľava alebo doprava, sa stará séria podmienok. Najzložitejšou časťou kapitoly o pohybe robota bola implementácia PID regulátora do jazyka C.

Všetky tri časti regulátora sú popísané časovo spojitým zákonom riadenia, takže je nutné ich pred implementáciou previesť do diskrétnej časovej oblasti. Túto premenu všetkých troch zložiek sme previedli cez aproximácie. Deriváciu sme nahradili spätnou diferenciou a integráciu pomocou lichobežníkového pravidla. Po implementácií sme sa museli rozhodnúť, ktorý regulátor nám bude vyhovovať najlepšie. K dispozícií sme mali P,PI a PID regulátory. Úlohou regulátora bolo dosiahnuť čo najrýchlejšie požadovaný uhol natočenia autíčka k cieľu. Pomocou niekoľkých

experimentov sme sa rozhodli pre PI regulátor, ktorý najlepšie zvládal dynamické prostredie, teda náhle zmeny požadovanej hodnoty. Informácie sme využili v poslednej časti práce.

Tá zahŕňala riešenie úlohy hľadania takej trajektórie, pomocou ktorej robot prejde v miestnosti z jedného bodu do druhého tak, že sa vyhne všetkým prekážkam. Na vyriešenie úlohy sme zvolili dve metódy a pomocou porovnania sme vybrali vhodnejšiu.

Prvá metóda bola založená na pravdepodobnostnom plánovaní. Metódu sme použili v rámci robotického balíčka Matlabu. Jej nevýhoda tkvie v závislosť na náhodne generovaných bodoch v priestore. Čím viac bodov sa vygeneruje, tým je nájdená trasa presnejšia. Na druhú stranu, so zvyšujúcim sa počtom bodov, sa zvyšoval čas nájdenia trajektórie. Pri použitý istého počtu generovaných bodov sa táto metóda stala nevyhovujúcou, lebo presiahla nami určenú maximálnu dobu hľadania.

Druhá metóda, „A star“, pracuje na základe heuristického hľadania optimálnej cesty. Metódu sme si upravili a implementovali v Matlabe. Táto metóda bola omnoho rýchlejšia a nebola závislá od počtu vygenerovaných bodov. Jej problémom bola heuristika, ktorá nepredpokladá výskyt prekážok v hľadaní ideálnej cesty. Kvôli tomuto problému v priestore s veľkými prekážkami môže robot robiť zbytočné obchádzky. Avšak táto metóda bola úspešnejšia aj v prostredí s veľkým počtom relatívne malých prekážok.

Na základe porovnania zo simulácií sme sa rozhodli pre nami naprogramovanú metódu „A star“. Následne sme túto metódu využili ako nadstavbu na riešenie systému viacerých robotov pohybujúcich sa v tom istom priestore. Metódu sme nazvali koordinácia s prioritou.

Princíp vytvárania trajektórie bol upravený o možnosť stretu viacerých robotov počas pohybu. Každému robotovi sme priradili prioritu v zostupnom poradí. Na základe tejto priority sa robot s nižšou prioritou prispôsobuje všetkým, ktoré majú vyššiu prioritu. Trajektória podradeného robota sa upraví podľa kritéria tak, že obíde miesto kolízie, alebo ak nevyhovuje kritériu, pokračuje vo svojej ceste a tesne pred kolíziou zastane a počká, kým sa mu uvoľní cesta.

Zoznam použitej literatúry

[1] http://www.optitrack.com/applications/

[2] https://v20.wiki.optitrack.com/index.php?title=Markers#Retroreflective_Markers

[3] http://optitrack.com/products/motive/tracker/indepth.html

[4] https://v20.wiki.optitrack.com/index.php?title=Prepare_Setup_Area

[5] https://v20.wiki.optitrack.com/index.php?title=Quick_Start_Guide:_Getting_Started

[6] https://v20.wiki.optitrack.com/index.php?title=Tracking_Bar_Coordinate_System

[7] https://v20.wiki.optitrack.com/index.php?title=Rigid_Body_Tracking

[8] https://v20.wiki.optitrack.com/index.php?title=Data_Streaming

[9] https://en.wikipedia.org/wiki/Euler_angles

[10] http://mathworld.wolfram.com/EulerAngles.html

[11] http://www.weizmann.ac.il/sci-tea/benari/sites/sci-tea.benari/files/uploads/softwareAndLearningMaterials/quaternion-tutorial-2-0.pdf

[12] https://en.wikipedia.org/wiki/Quaternion

[13] BAKOŠOVÁ, M. – FIKAR, M. 2012. Riadenie procesov. 2. vyd. Bratislava: Vydavateľstvo STU. ISBN: 978-80-227-3763-0

[14] https://en.wikipedia.org/wiki/Trapezoidal_rule

[15] https://en.wikipedia.org/wiki/Numerical_differentiation

[16] DUCHON, F. – BABINEC, A. – KAJAN, M. - BENO, P. FLOREK, M. – FICO, T. – JURISICA, L. 2014. Path planning with modified A star algorithm for a mobile robot. Článok z internetu <https://doi.org/10.1016/j.proeng.2014.12.098>

[17] https://en.wikipedia.org/wiki/Probabilistic_roadmap

[18] https://www.mathworks.com/help/robotics/examples/path-planning-in-environments-of-different-complexity.html

[19] https://en.wikipedia.org/wiki/A*_search_algorithm

Prílohy

Príloha A: https://github.com/RomanKohut/Tracker

Príloha B: https://github.com/RomanKohut/No-obstacles-planning

Príloha C: https://github.com/RomanKohut/A-star-pathfinding

Príloha D: https://github.com/RomanKohut/Probabilistic-roadmap

Príloha E: https://github.com/RomanKohut/Coordination-with-priorities

Príloha F: https://github.com/RomanKohut/Arduino-vehicle-algorithm