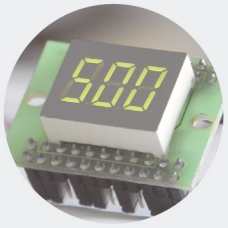
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Desenvolvimento e validação de um simulador 3D para prova de condução autónoma do FNR

DAVID LUÍS PEREIRA FERNANDES
novembro de 2019

POLITÉCNICO
DO PORTO

Desenvolvimento e validação de um simulador 3D para prova de condução autónoma do FNR

David Luís Pereira Fernandes
Nº 1081692

Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização de Sistemas Autónomos
Departamento de Engenharia Electrotécnica

Instituto Superior de Engenharia do Porto

2019

Dissertação, para satisfação parcial dos requisitos do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: David Luís Pereira Fernandes

Nº 1081692

Orientador: André Miguel Pinheiro Dias

Mestrado em Engenharia Eletrotécnica e de Computadores

Área de Especialização de Sistemas Autónomos

Departamento de Engenharia Electrotécnica

Instituto Superior de Engenharia do Porto

25 de Novembro de 2019

Esta página foi intencionalmente deixada em branco.

Abstract

One of the more relevant research challenge competitions in the Festival Nacional de Robótica (FNR) is the Autonomous Driving Competition, where fully autonomous robots have to navigate in a complex scenario replicating road environments. Here robots must overcome challenges ranging from perception to navigation and control.

In addition, there are challenges associated to the integration of sensors (Cameras, Inertial Measurement Unit (IMU), Laser and Encoders) able to support the navigation task or even at the mechanic development level, which sometimes, can be difficult to address by some research groups. Therefore, this paper presents a simulation environment developed for FNR Autonomous Driving Competition.

This paper also covers various topics of mobile robotics, the proposed simulator is used in the development and validation of a location system, trajectory tracking, and obstacle avoidance.

Keywords: Morse, ROS, Simulation, Localization, Obstacle avoidance, Path traking

Esta página foi intencionalmente deixada em branco.

Resumo

Inserida no Festival Nacional de Robótica (FNR), a prova Autonomous Driving Competition endereça desafios da condução autónoma, nesta competição os sistemas robóticos totalmente autónomos devem navegar num cenário de prova que retrata condições do ambiente rodoviário. Os robôs devem superar desafios relevantes da robótica móvel como a localização, percepção, navegação e controlo.

São diversas as dificuldades e condicionalismos por parte dos participantes a este modelo de competição. A participação carece de uma solução de robô nem sempre disponível ou de fácil desenvolvimento e tende a integrar um considerável número de sensores.

Esta dissertação apresenta um ambiente de simulação da prova de Condução Autónoma do FNR, sendo o simulador proposto utilizado no desenvolvimento e validação de um módulo de localização, *path traking*, *path following* e *obstacle avoidance*.

Palavras-Chave: Morse, ROS, Simulação, Localização, Obstacle Avoidance, Path traking, Path following

Esta página foi intencionalmente deixada em branco.

Conteúdo

Esta página foi intencionalmente deixada em branco.

Lista de Figuras

Lista de Tabelas

Esta página foi intencionalmente deixada em branco.

Lista de Siglas e Acrónimos

ADC Autonomous Driving Competition

ADAS Advanced driver-assistance systems

API Application programming interface

CCW Counterclockwise

EKF Extended Kalman Filter

FOV Field of View

FNR Festival Nacional de Robótica

GPL General Public License

GPS Global Positioning System

GE Game Engine

HIL Hardware-in-the-Loop

IMU Inertial Measurement Unit

INESC TEC Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência

IPM Inverse Perspective Mapping

KF Kalman Filter

LTI linear time invariant

LiDAR Light Detection and Ranging

LSA Laboratório de Sistemas Autónomos

LRF Laser rangefinder

MPC Model predictive control

MIMO Multi-Input-Multi-Output

MCL Monte Carlo localization

MIL Model-In-the-Loop

MPC Model predictive control

OSRF Open Source Robotics Foundation

PDF Probability density function

PID Proportional Integral Derivative controller

RSSI Received signal strength indication

ROS Robot Operating System

Rprop Resilient propagation

SIS Sequential Importance Sampling

SIR Sampling Importance Resampling

SIL Software-In-the-Loop

SLAM Simultaneous localization and mapping

SISO Single-input-single-output

WSN Wireless sensor network

Capítulo 1

Introdução

Os próximos anos da Indústria automóvel serão marcados pelo ponto de viragem na adoção de sistemas de condução totalmente autónomos. O desígnio teve início na década de 80 impulsionado pelo projeto Eureka PROMOTHEUS, que definiu o estado da arte dos veículos autónomos. Em 1995 foi alcançado o primeiro marco, protagonizado pelo Mercedes W140 S-Class num percurso de aproximadamente 1678 quilómetros (entre Munique e Copenhaga) realizado de forma praticamente autónoma.

A simulação como meio de emular o mundo real tem-se revelado um fator preponderante ao desenvolvimento dos sistemas de condução autónoma [1], principalmente em fases de desenvolvimento e teste que antecedem a introdução dos sistemas em ambiente real.

São numerosos os desafios colocados na avaliação de sistema em ambiente real, infraestruturas e logística associada ao teste de um simples carro robótico requer recursos monetários e humanos significativos [2]. Sistemas imaturos, requerem uma enorme quantidade de dados [3] e repetição de condições de teste para otimização, a serem adquiridas num ambiente real implicariam onerosos custos, situações de perigo e elevado tempo despendido. Segundo Susan M. Paddock [4] e Kalra *et al.* [5] é inclusivamente impossível em tempo ajustado, assegurar em ambiente real a segurança dos veículos autónomos.

A simulação tem evoluído consideravelmente, não sendo unicamente um meio de teste para hardware e software, é hoje utilizada como ferramenta de treino para os sistemas de condução autónoma. Abordagens recentes como *reinforcement learning*, *learning-by-demonstration* e *transfer-learning* são apontadas como técnicas necessárias à evolução e aprendizagem dos sistemas de condução autónoma, sendo numa primeira fase realizado em ambiente simulado [6].

Diversos simuladores comerciais como o Carcraft, Metadono, rFpro e opções *open-*

source como o CARLA e AIRSIM, replicam o mundo com detalhe gráfico com inclusão de objectos dinâmicos, incluem diferentes tipos de veículos, ciclistas e pedestres criando cenários e imprevisibilidades do mundo real. A simulação permite o teste intensivo a variações específicas, como por exemplo, alterações das condições meteorológicas ou presença de pedestres na estrada. Condições essas difíceis ou mesmo imprudentes de replicar na realidade.

A simulação democratizou o acesso aos meios de desenvolvimento, teste e treino dos sistemas associados à condução autónoma, concedendo o acesso a diversos modelos de plataformas robóticas, sensores e ambientes de simulação.

São numerosos os trabalhos que principiam em ambiente de simulação, no desenvolvimento de soluções previamente à implementação real, exemplos encontrados em Webots [7], VREP [8], Gazebo [9, 10] [11], MORSE [12, 13, 14] ou USARSim [15].

São diversas as competições no âmbito da condução autónoma com intuito de promover, desenvolver e difundir o conhecimento científico e tecnológico na área. O DARPA Robotics Challenge [16] foi uma dessas competições impulsionadora no desenvolvimento de robôs, software e adaptação de simuladores de robótica (Gazebo).

A um nível mais académico e em alguns casos com direta associação à indústria, a Audi Autonomous Driving Cup [17], Open Zeka MARC [18], CARLA Autonomous Driving Challenge [19] são provas com importante *playground* para ensinar, desenvolver e testar estratégias para robôs autónomos.

Inserida no Festival Nacional de Robótica (FNR) [20], a prova de Condução Autónoma (ADC) replica num cenário de pista as condições de uma estrada convencional. Escalada em níveis de dificuldade, os concorrentes são incitados a resolver diversos desafios da condução autónoma, desde a navegação entre faixas à necessidade de planear e executar trajetórias em percursos não estruturados.

Este trabalho pretende numa primeira fase, responder a estes constrangimentos disponibilizando a toda a comunidade um ambiente simulado da ADC-FNR. Recriado com elevado grau de realismo composto de todos os elementos da prova, permitindo o apoio nas tomadas de decisão aquando do desenvolvimento e da preparação prévia de todos os participantes.

Outro vetor deste trabalho é o desenvolvimento de soluções de perceção, navegação e controlo de um robô diferencial candidato à competição ADC, atestando o simulador proposto.

Este projeto desenvolve um módulo de Localização utilizando *landmarks* observadas pelo sistema de visão. A localização é uma peça basilar, determinante e essencial ao controlo de um robô [21] [22], mas num cenário *indoor* e parco em referências este é um

desafio de alguma complexidade e raramente encontrado nas soluções propostas pelos participantes da ADC-FNR.

Nos anos 80, a robótica foi definida como a ciência que estuda a ligação inteligente entre a perceção e ação [23], este trabalho não ficaria completo sem uma abordagem ao controlo e navegação sendo este um dos grandes tópicos da robótica móvel. Não pretendendo ser uma solução completa de resposta a todos os desafios da prova ADC-FNR, alguns *behaviors* são endereçados para a realização do percurso evitando obstáculos.

1.1 Motivação

São diversos os constrangimentos e desafios à participação no modelo de prova ADC-FNR. Para estudantes com foco no desenvolvimento das soluções de software de controlo, percepção, navegação, etc - as questões mecânicas inerentes ao desenvolvimento da plataforma e de todo o hardware associado, podem facilmente exacerbar de trabalho estas equipas frustrando objetivos [24]. Questões de acessibilidade, o custo do hardware (plataforma e diversos sensores) e a falta de experiências que sustentem as opções na fase de desenvolvimento, podem facilmente demover equipas à participação ou inibir a conclusão dos seus projetos.

Disponibilizar um simulador da prova ADC-FNR à comunidade é uma das motivações deste trabalho. Um simulador que permita a preparação prévia dos participantes tenderá a promover a adesão de um maior número de participantes, com soluções mais capazes de enfrentar os diferentes níveis de exigência da competição.

Numa outra vertente, o simulador é uma ferramenta passível de utilização na aprendizagem dos mais diversos tópicos da robótica móvel. O acesso a uma ferramenta apelativa e disponível no imediato a qualquer estudante, sem condicionalismo de acesso a *hardware* será uma mais valia a nível académico.

Por fim, a participação da competição na prova ADC-FNR. A participação na competição é um objetivo final e todos os desenvolvimentos realizados preparam uma solução para um sistema real a ser apresentado na ADC-FNR.

1.2 Objetivos

Os objetivos abaixo elencados resumem o alcance pretendido desta dissertação, não obstante outros desenvolvimentos, estes são os desafios aos quais se pretende encontrar solução:

- Desenvolver um simulador da prova ADC, com integração dos diferentes elementos da prova (sinais luminosos, túnel, zona de trabalho, etc);
- Interactividade do simulador para rápida definição de missões que retratam as diferentes mangas e dinâmica da competição (posicionamento de obstáculo, alteração dos sinais, inclusão de túnel, etc);
- Atingir elevado grau de realismo para os sistemas de visão, diminuindo impato na transição para a realidade;
- Dotar o cenário e todos os seus elementos de características físicas, que eleve o realismo do simulador (massas, gravidade, fricção, ruídos, entre outras);
- Integração do simulador num *framework* que permita desenvolvimento distribuído;
- Criar um módulo de localização com recurso a um filtro probabilístico;
- Desenvolver um algoritmo para *path following* que permita o robô percorrer a pista;
- Desenvolver abordagem genérica de *collision avoidance*, que permita utilização nos cenários de obstáculos, túnel e zona de trabalho delimitada por cones;
- Criar elementos visuais que retratem os desenvolvimentos realizados;
- Validação dos métodos desenvolvidos e avaliação de desempenho.

1.3 Estrutura

Esta dissertação encontra-se organizada em oito capítulos. No segundo capítulo encontra-se a Formulação do Problema, com descrição da prova de condução autónoma (ADC) do Festival Nacional de Robótica (FNR), tópicos da robótica presentes nesta competição e dificuldades com que se deparam os seus participantes.

O estado da Arte exposto no capítulo 3, endereça um estudo sobre as soluções existentes de simulação e dois tópicos da robótica móvel, sistemas de Localização e *path following/ path tracking*.

A fundamentação teórica, que dá suporte aos desenvolvimentos deste trabalho é descrita no capítulo 4. Neste é realizado uma introdução ao controlo e modelo cinemático da plataforma utilizada, a conceitos probabilísticos que endereçam os filtros igualmente descritos neste capítulo, o filtro de Kalman linear e estendido.

O Projeto é descrito no Capítulo 5, neste consta toda a estrutura e arquitetura do sistema desenvolvido, aplicações utilizadas e metodologia adotada.

O capítulo 6 apresenta os desenvolvimentos realizados, dividido entre o simulador Morse, o módulo de localização, *path following*, *path tracking* e manobra para evitar colisão com obstáculos.

Os resultados estão presentes no capítulo 7, o comportamento do sistema de localização, assim como dos controladores é aqui avaliado.

Por fim a conclusão no capítulo 8, tece comentários a todo o trabalho realizado e resume os desenvolvimentos futuros.

Esta página foi intencionalmente deixada em branco.

Capítulo 2

Formulação do problema

A prova de condução autónoma do FNR é um desafio de robótica móvel com características que tendem a replicar ambientes reais de condução numa pista em ambiente fechado. O ambiente estruturado da prova é composto por uma pista com formato do número oito, com faixas delimitadoras representativas de uma estrada real devidamente delimitada, assim como uma passadeira, sinais luminosos, obstáculos, zona de obras e um túnel. Para além destas características da pista e seus elementos, existe também uma zona delimitada para estacionamento com dois lugares, um dos quais estará ocupado.

A prova é composta por 3 mangas, sendo o grau de dificuldade incrementado a cada uma destas etapas. A primeira prova tem como objectivo a condução autónoma do robô em pista. O robô deve completar duas voltas ao circuito o mais rápido possível. Na segunda prova, o robô deve identificar os sinais dispostos no painel de informação e agir em conformidade. O painel luminoso apresenta 5 possíveis sinais, que dão ordem desde o sentido, parar, seguir em frente ou estacionar. O robô deve nesta fase ser capaz de realizar a manobra de estacionamento em área delimitada para o efeito, com dois lugares de estacionamento estando um deles ocupado, pelo que o robô deverá ser capaz de identificar o lugar livre e estacionar. Outro desafio desta fase está relacionado com a colocação de um obstáculo na via, simulando a presença de um veiculo. O robô deverá identificar o obstáculo e realizar manobra de ultrapassagem utilizando a faixa oposta à que transitava.

Na terceira e última fase, a dificuldade é incrementada pela simulação de zona de obras e um túnel que influência a percepção dos sensores devido às condições de baixa luminosidade. A zona de obras simulada por cones interligados por fitas, pressupõe um desafio à lógica que precede as restantes fases, o cenário deixa de ser estruturado. A zona de obras é definida sem conhecimento prévio definindo um novo percurso.

Figura 2.1: Cenário da prova de Condução Autónoma do FNR

A Prova de Condução Autónoma (ADC) endereça desafios relevantes da robótica móvel, localização, controlo do movimento, planeamento de trajectórias, entre outros que fazem desta competição um desafio relevante.

O prova ADC apresenta uma pista difícil de replicar pelos seus participantes, que raramente dispõem de condições para testar as suas soluções com todos os elementos presentes prova (túnel, painéis semafórico, zona de obras, etc). A ausência de acesso a um meio de teste que retrate a diferentes componentes da prova, condiciona o refinamento das soluções e restringe os participantes aos desafios mais simples da competição, afastando-os dos problemas mais complexos da robótica móvel. O acesso a plataformas robóticas e diversos sensores é outra condicionante à participação.

A simulação é um meio de preparação para esta competição e ferramenta de estudo para os diversos tópicos da robótica. A disponibilização de uma ferramenta de simulação à comunidade, que retrate com detalhe e realismo o cenário da prova, vai permitir um maior envolvimento na competição.

Capítulo 3

Estado da Arte

3.1 Simuladores

As ferramentas de simulação para análise, desenho, aquisição de dados e treino estão já bem enraizadas e tidas como essenciais na fase de desenvolvimento, testes e maturação dos sistemas.

A simulação permite replicar com grande realismo diversos cenários e condições do mundo, difíceis de serem avaliadas na realidade e permite fazê-lo de forma eficiente, a baixo custo e de forma segura. O recurso a simulação possibilita uma fácil e célere avaliação da confiabilidade das soluções de código e o teste intensivo de abordagens nas mais diversas situações.

Os simuladores permitem o acesso a diferentes modelos de plataformas, sensores e atuadores emulando diferentes níveis de realismo com inclusão de física e incerteza nas medidas.

Mais recentemente, a simulação é também utilizada como meio para aprendizagem de sistemas utilizando técnicas como *Machine learning*, *deep learning* ou *imitation learning*.

A simulação é também um recurso essencial para metodologias no desenvolvimento de sistemas. Modelos de desenvolvimento incluem o recurso a simulação para avaliação e teste em diferentes fases do processo, permitindo abstracções a diferentes níveis sobre o hardware. Fases de teste denominadas por *Model-In-the-Loop* (MIL), *Software-In-the-Loop* (SIL) e *Hardware-in-the-Loop* (HIL).

Existem diferentes tipos de simuladores, com enfoque e objectivos diferentes que podem ser combinados ou utilizados em diferentes fases do desenvolvimento de um sistema

autónomo, a selecção pelo mais ajustado pode efectivamente tornar-se complexa. Nas secções abaixo são descritos alguns destes simuladores, divididos entre Simuladores com inclusão de sistemas/veículo real e simuladores robóticos.

3.1.1 Simuladores com inclusão de sistemas/veículo real

O projecto para uma solução final de um sistema autónomo é moroso, complexo e dispendioso. O modelo V [25] está genericamente aceite na indústria e define diferentes fases de desenvolvimento e avaliação. O processo inicia por *Model-In-the-Loop*(MIL), desenvolvendo soluções com abstração sobre o *hardware* ou *frameworks*, no qual o controlador ou algoritmo é desenvolvido em *close-loop* com modelos da plataforma, sensores, actuadores e cenários externos.

Segue-se a verificação via *Software-In-the-Loop* (SIL), para validação das estratégias e algoritmos, o código desenvolvido é testado sobre o hardware a ser utilizado com simulação de sensores, actuadores e restantes componentes. O passo seguinte, a simulação por *Hardware-In-The-Loop* (HIL), consiste na possibilidade de combinar simulação com hardware real, sendo passível que sensores ou actuadores sejam alternativamente reais, emulados ou simulados, o tipo de sensor não deverá comprometer o *software* desenvolvido [26] que se comporta de igual forma.

O HIL permite num ambiente seguro, o teste intensivo e sistemático em diferentes condições de teste, com modularidade sobre a inclusão do tipo de sensores (reais, simulados ou emulados), perturbações e ruídos gerados por outros sensores, ou externos.

A simulação no modelo V foi amplamente utilizada no desenvolvimento dos sistemas ADAS [27] [26], [28]. Encontram-se para este modelo de desenvolvimento diversas referências na literatura com utilização de simuladores como o MATLAB/Simulink, PreScan, LabVIEW e Carsim.

3.1.1.1 CarSim

O Mechanical Simulation [29] é uma plataforma com simuladores para diferentes tipos de veículos (CarSim, TruckSim e BikeSim) com enfoque na simulação do comportamento dinâmico do veículo e partes constituintes nas mais diversas condições (inclinação, condições do piso), avaliando reacção do veículo em condições específicas de derrapagem, inclinação, travagem, assim como o comportamento à presença de objetos no cenário. O software permite a representação visual do movimento do veículo e a criação de cenários, embora o movimento da indústria tenha convergido na adopção de simuladores de percepção, nomeadamente no Unreal Engine [30]. Para resposta a esta demanda, a

Mechanical Simulation desenvolveu o VehicleSim Dynamics *plugin* [31], permitindo fácil integração dos modelos matemáticos do CarSim e TruckSim num ambiente Unreal.

O CarSim apresenta suporte sobre os processos *Software-in-the-loop*, *Model-in-the-loop*, *Hardware-in-the-loop* e *Driver-in-the-loop* com interface MATLAB/Simulink. São diversas as referências na adopção destas ferramentas combinadas, nomeadamente no desenvolvimento e avaliação de sistemas ADAS, [32, 33, 34, 35].

3.1.1.2 MATLAB e Simulink

O MATLAB [36] assim como o Simulink [37] são ferramentas *standards* da indústria e enraizadas a nível académico. O MATLAB permite a análise e desenvolvimento de algoritmos, modelos matemáticos e disponibiliza *apps* em diversas áreas de investigação, como *Deep Learning*, *Computer Vision*, *Robotics*, entre outras. Disponibiliza opção de conversão automática de código para as linguagens C/C++, HDL e CUDA, facilitando a portabilidade de soluções desenvolvidas para sistemas embebidos. O Simulink é uma ferramenta de simulação e programação gráfica *model-based design*. É uma constante a utilização simultânea de ambas ferramentas, combinando o editor de texto presente no MATLAB ao gráfico do Simulink, permitindo a inclusão de algoritmos e *datasets* através do MATLAB serem intensivamente simulados com recurso ao Simulink, e os resultados obtidos analisados no MATLAB.

Com enfoque nos sistemas de condução autónoma, o MATLAB em conjunto com o Simulink disponibiliza um conjunto de ferramentas de apoio ao desenho, desenvolvimento e teste como é exemplo o Automated Driving toolbox. Esta *toolbox* fornece um conjunto de funcionalidades concedendo acesso a algoritmos pré-concebidos que suportam o desenvolvimento de sistemas de percepção com base em visão, lidar, modelo de sensores, planeamento de trajectórias, fusão sensorial, processamento de sensores, entre outros. Inclui conversão de código para linguagem C, abrindo espaço à célere integração no *hardware* e realização de testes HIL. Diversas outras ferramentas completam o ecossistema de aplicações disponíveis pelo MATLAB e Simulink no domínio *Automotive*.

3.1.2 Simuladores de Robótica

Os simuladores de Robótica são genericamente utilizados na robótica móvel e foram igualmente adoptados para o desenvolvimento de soluções de sistemas para a condução autónoma, maioritariamente pela comunidade Académica. Existem diversos exemplos da adopção deste tipo de simuladores na preparação para competições de condução autónoma, inclusivamente presentes na mais paradigmática prova DARPA Grand Chal-

lenge [38] [39].

Os simuladores de robótica não são ferramentas exclusivas ao domínio da condução autónoma ou robótica móvel terrestre, integram contudo ferramentas ajustadas à sua utilização. Numa única ferramenta está habitualmente presente o acesso a um motor Gráfico, simulador de física (com diferentes graus de precisão), sensores, atuadores, plataformas, *frameworks* de comunicação, entre outras ferramentas de apoio ao desenvolvimento.

Simuladores de robótica como Gazebo [40], UsarSim [41], MORSE [42] ou V-Rep [43] permitem o desenvolvimento de cenários com detalhe gráfico, desenho e definição de estruturas robóticas, ou a utilização de modelos concebidos permitindo de forma célere o acesso a um meio de desenvolvimento e teste. Estes simuladores disponibilizam um vasto conjunto de sensores, com possibilidade de inclusão de incerteza nas medidas e perturbações elevando o grau de realismo. Os *middlewares* de comunicação são outro fator relevante, *frameworks* como ROS [44] ou YARP [45], fazem parte do ecossistema da robótica móvel e disponibilizam um conjunto de ferramentas, *plugins* e metodologia de desenvolvimento escalável, distribuída, flexível e facilmente transferida para uma solução de robô real.

A escolha de um simulador ajustada à necessidade, deve considerar os critérios:

- Realismo e robustez da visualização/renderização 3D
- Licenciamento, licença GPL (General Public License) ou comercial
- Plataformas, sensores e atuadores disponíveis
- Maturidade, comunidade e tutoriais disponíveis
- Ferramenta de desenvolvimento, nomeadamente para criação de cenários e plataforma
- Middlewares de comunicação, com avaliação por sensor
- Possibilidade de inclusão de ruído no output dos sensores, elevando o grau de realismo
- Performance do simulador

Com este enquadramento, serão avaliados os Simuladores referidos na tabela 3.1.

Tabela 3.1: Características dos simuladores

Simuladores	motor de simulação	motor gráfico	middlewares	licença	comunidade/tutoriais
Gazebo	ODE, Bullet, Simbody e DART	OGRE	ROS	open-source apache 2.0	grande comunidade/ bem documentado
MORSE	Bullet	Blender Game Engine	ROS, YARP, Pocolibs, MOOS, HLA e Mavlink	open-source BSD	pequena comunidade/ bem documentado
Webbots	ODE	OpenGL	ROS, TCP/IP	open-source apache 2.0 (Dezembro 2018)	pequena comunidade/ bem documentado

3.1.2.1 Gazebo

O Gazebo [46] é um simulador 3D inteiramente *open-source*, fez parte do projeto Player Project [47] entre 2004 a 2011 e evolui como projeto independente com apoio da Willow Garage [48]. Em 2012, a Open Source Robotics Foundation (OSRF) saiu da Willow Garage e tornou-se a responsável pelo projeto Gazebo.

O simulador Gazebo foi projetado para elevada precisão na simulação de ambientes dinâmicos, com intuito de servir a demanda de aplicações para uso de veículos robóticos em cenário *outdoor*. Contrariamente a este principio, a sua utilização acabou por revelar-se maioritariamente *indoor*.

O Gazebo apresenta uma arquitectura distribuída prevenindo dependência sobre ferramentas, tendo de forma separada bibliotecas para simulação de física, renderização, comunicação, API intuitiva para criação e inclusão de novos sensores, atuadores, robôs, denominados por modelos, e ainda um interface de utilizador. O simulador permite o acesso a diversos motores de física (ODE, Bullet, Simbody e DART), embora por defeito compilado com suporte ODE sendo necessário a instalação e compilação do Gazebo através do código fonte para utilização dos restantes. Assenta em renderização Ogre3D dispondo cenários com elevado detalhe gráfico e realismo, incluindo efeitos de iluminação, sombras e texturas. É um simulador com valências para múltiplos robôs, embora com comparativa baixa performance sendo o processamento distribuído uma das maiores limitações deste simulador [49]. Dispõem de um editor para criação de cenários e também permite a importação através do desenvolvimento em SDF [50], mas apresenta limitações como impossibilidade de edição de *meshes*. A biblioteca de sensores e atuadores é rica mas relativamente parca na diversidade de modelos de robôs [51]. Segundo [51] o interface de utilização é complexo com baixa usabilidade. Outro ponto negativo, é a complexa estrutura de dependências do simulador e modelos disponíveis.

Como *middleware*, o Gazebo disponibiliza *plugin* para comunicação com o ROS. Existe estreita ligação entre o Gazebo e o ROS, o Gazebo está inclusivamente presente no *package*

de instalação do ROS o que populariza ainda mais a sua utilização. O Gazebo é dos simuladores com maior adoção pela comunidade, apresenta grande dinamismo e está bem documentado sendo o efeito de comunidade um fator importante, existindo fórum activamente participativo.

3.1.2.2 MORSE

O simulador Morse [42], igualmente *open-source* oferece grande realismo de simulação 3D, com capacidade de simular ambientes internos ou externos de pequena e grande dimensão, gerindo de forma exímia a simulação simultânea de múltiplos robôs. O simulador é suportado pelo motor de jogo Blender Game Engine [52], que apresenta elevado realismo gráfico com renderização em tempo-real, modelando *meshes* e diversos efeitos como texturas, iluminação e sombras [53]. Como motor de física, o MORSE recorre à biblioteca Bullet Real-Time Physics Simulation [54], replicando efeitos do Mundo real, como a gravidade, forças, efeitos de colisão e propriedades estáticas como massa, definição de corpos rígidos, entre outros.

O MORSE recorre igualmente ao Blender como ferramenta para modelação de sensores, cenários, robôs, etc. O Blender é uma ferramenta *open-source* com uma enorme comunidade na criação de modelos 3D, animação, simulação, renderização e criação de jogos. Qualquer *mesh* representativa de um sensor, robô, cenário ou outro, uma vez criada em Blender pode imediatamente ser incluída para utilização no MORSE.

O MORSE é essencialmente uma aplicação desenvolvida em Python, controlada inteiramente via linha de comandos com arquitectura de componentes. Os sensores, atuadores e outros componentes modelados no Blender são codificados em *scripts* Python, a cada *script* corresponde um componente que a ter representação visual, tem correspondência a uma *mesh* desenhada no Blender.

O simulador inclui diversos sensores *standart*, atuadores e um conjunto considerável de robôs terrestres, incluindo também veículos aéreos e aquáticos, sendo possível a criação e inclusão de novos robôs. O MORSE suporta atualmente 6 open-source middlewares (ROS, YARP, Pocolibs, MOOS, HLA e Mavlink), que embora não compatíveis com todos os sensores, demonstra modularidade e independência possível na integração com outras arquitecturas.

O MORSE nos seus diversos componentes (sensores, atuadores e robôs) permite um ajustável grau de realismo e abstração. Diferentes objectivos de simulação incorrem em diferentes níveis de realismo pretendidos, a título de exemplo, simulações com objectivo primordial no processamento de imagem podem considerar o controlo do movimento secundário. Enquadrado neste contexto, o MORSE disponibiliza uma câmara que ren-

deriza o cenário e consequentemente gera grandes quantidade de informação, limitando o número de câmaras e performance da simulação, mas necessária a algoritmos de processamento de imagem. Disponibiliza contudo uma *Semantic camera*, que serve para identificação de objetos e posição dos mesmos no FOV da câmara, fá-lo com conhecimento do Mundo e dos objetos sem recorrer ao processamento de imagem, o que permite rapidamente simular este *output*. A informação de sensores e atuadores pode ainda ser manipulada por ação de *Modifiers*, elevando realismo com inclusão de ruído, como é disso exemplo a perturbação passível de ser gerada com ruído gaussiano no output do IMU. Esta interferência é adicionada à informação disponibilizada por estes componentes, não fazem parte dos mesmos, sensores e atuadores são agnósticos a esta perturbação ou ao *middleware* com que comunicam.

O MORSE é realmente diligente no processo de instalação e tempo para uma primeira simulação, sendo realmente simples alcançar ambos. O MORSE está disponível via package Debian/Ubunto sem dependências adicionais.

Embora referenciado como um projeto maduro e apresente tutoriais abrangentes e devidamente explanados, o MORSE não partilha do entusiasmo e comunidade de outros e em particular do Simulador Gazebo, o que efetivamente se apresenta como uma menos valia. A comunidade está basicamente restrita a uma lista de email, o que se revela insuficiente.

3.1.2.3 Webots

O simulador 3D Webots [55] é um projecto recentemente disponibilizado em *open-source* (Dezembro 2018) com vertente comercial apenas para o suporte, o simulador é desenvolvido pela Cyberbotics em estrita ligação com o instituto de tecnologia de Lausanne. O simulador é um projeto maduro com utilização por parte de mais de 800 universidades [56], muito bem documentado, estável e abrangente nas diversas áreas da robótica móvel.

O simulador assenta numa utilização aprimorada do motor de física ODE. Disponibiliza calibração de parâmetros como massa, coeficiente de fricção, amortecimento entre outras propriedades estáticas e dinâmicas para um comportamento ajustável à realidade dos modelos de sensores, atuadores e robots.

O Webots disponibiliza um ambiente completo de programação e criação de novos modelos, contudo e segundo Echeverria [53], o interface para desenvolvimento de novos robots e sensores é complexo e muito pouco intuitivo. O simulador permite incluir importação de modelos num formato VRML97, que abre espaço ao desenho em ferramentas de modelação como o Blender, Autocad e SolidWorks.

O IDE de programação do Webots integra directamente diversas API's, fornecendo acesso a diferentes meios de programação e interfaces, o ROS, MATLAB, TCP/IP para comunicação local ou remota com robôs são alguns exemplos. De forma análogo aos simuladores apresentados, o Webots inclui um conjunto de robots, sensores e atuadores considerável, mas vai mais longe no âmbito da condução autónoma, disponibilizando ferramentas e modelos particularmente endereçados a este tópico.

O simulador inclui um variado conjunto de cenários que retratam diversos ambientes do mundo real em diversas condições do mesmo, cidades retratadas de dia e noite, aldeias, autoestradas, entre outros. Estão também disponíveis modelos de veículos reais como é exemplo o Toyota Prius, Range Rover Sport SVR e um conjunto alargado de sensores que inclui diversos modelos de Câmaras, LiDARs e Radares normalmente utilizados nesta indústria.

O desenvolvimento é suportado com bibliotecas que disponibilizam um vasto conjunto de funções intuitivas com funcionalidades de mais alto nível. Toda a informação está documentada de forma exímia para as linguagens de programação C, C++, Java e Python.

3.2 Localização em cenários *indoor*

A estimação da localização de um robô é uma componente essencial da robótica móvel, sendo crucial para tarefas como mapeamento, navegação e controlo. Para sistemas robustos é normalmente assumida a multiplicidade de sensores para redundância e aumento de exatidão, sendo que nos cenários *outdoor* os sistemas são normalmente suportados com recurso a sensores GPS, algo impraticável nos cenários *indoor* como o presente neste trabalho em estudo.

A localização é dividida em duas abordagens, acompanhamento da posição (*pose tracking*) e localização global. Na abordagem *pose tracking* a posição inicial do robô é conhecida, o seu estado pode ser estimado acumulando informação do movimento desenvolvido pelo robô no tempo, obtendo a posição relativa a uma posição inicial. Esta técnica denominada por *dead reckoning* é relativamente simples e recorre habitualmente a odometria, sensores inerciais e magnetómetros. A ausência de exatidão na medição do deslocamento linear e angular de um robô, gerado por diferentes fontes de erro e acumulada no tempo, cria diferenças significativas na estimação da *pose*, inviabilizando esta técnica de forma isolada em sistemas com movimento de vários metros.

É frequente a utilização de abordagens *dead reckoning* aliadas a alguma observação, que localize momentaneamente o sistema no referencial global.

A Localização global é um problema mais complexo. Inserido em determinado mapa e desconhecendo a sua posição inicial, um robô tenta localizar-se com base na informação de sensores e do seu movimento. Abordagens probabilísticas são aceites como o melhor método à localização e podem diferenciar-se pela forma como representam o espaço de estados [57], divididas entre filtros de Kalman e localização topológica ou *Gridbase* de Markov.

As abordagens entre a

3.2.1 Localização com recurso a filtros de Kalman

O algoritmo do filtro de Kalman (KF) e derivações, em particular o algoritmo do filtro de Kalman estendido (EKF) foram e permanecem sendo alvo de uma ampla utilização na robótica móvel para determinação da localização, e mais recentemente na localização e mapeamento simultâneo (SLAM). São inúmeras as referências à utilização deste algoritmo [58, 59, 60, 61, 62, 63].

Os algoritmos dos filtros de Kalman carecem do modelo do sistema e observação e desenvolvem-se essencialmente em duas fases, *Predict* e *Update* numa abordagem Bayesiana. No KF, o algoritmo assume unicamente sistemas lineares, o que restringe utilização para a maioria dos sistemas presentes na realidades, o EKF pode ter-se como uma evolução e com recurso à Linearização dos modelos, emprega a mesma abordagem do KF para modelos não lineares de sistemas.

Os filtros de Kalman são baseados na representação da incerteza do estado por uma distribuição Gaussiana unimodal, este modelo é limitativo para a localização e impõem conhecimento sobre o estado inicial do robô para convergência do filtro. A incerteza das medidas são igualmente assumidas como distribuições gaussianas, o que não corresponde por diversas vezes à realidade [57]. Contudo, assumindo as condições referidas, o filtro de Kalman é uma ótima técnica para localização, robusta, eficiente e precisa.

Em [63], a implementação do algoritmo EKF é empregue na localização de um robô diferencial, utilizando odometria e um *laser-range-finder* (LRF).

A primeira fase do algoritmo, *preditc*, é suportada pelo modelo discreto cinemático do robô que inclui características do mesmo (raio das rodas, distância entre rodas, etc) e traduz o comportamento do movimento face aos inputs de velocidade e direcção. O estado, definido por $x_p(k) = [x_r(k), y_r(k), \varphi_r(k)]^{\mathrm{T}}$, representa a *pose* do robô em relação ao sistema de coordenadas global.

O robô desenvolve movimento a duas dimensões, num deslocamento linear e angular, apresentando erro no posicionamento obtido pelo modelo cinemático devido à incerteza

da odometria por efeitos diversos como deslizamento, derrapagem, incerteza na medição. A matriz de covariância (Q) é formulada com base no erro de rotação de cada roda, proporcional à velocidade de rotação e de uma constante que serve de *tuning*, obtida experimentalmente.

Na fase seguinte do algoritmo - o *Update* - a *pose* do robô é corrigida com recurso a um modelo de observação de segmentos de linhas. Com recurso ao sensor LRF, o modelo reconhece parâmetros de linhas utilizando o método mínimo dos quadrados, identificando correspondência dos segmentos de linha observada com as linhas do mapa global previamente conhecido. A observação de linha e correspondente posição face ao robô no mapa local, com a informação da correspondente linha e posição real no mapa global permite atualizar a posição do robô. Abordagem semelhante apresentada em [58, 63].

Em [60] numa abordagem restritiva a sensores, o algoritmo de localização EKF é implementado recorrendo apenas a informação de *encoders* e giroscópio. Os resultados são confiáveis para a experiência realizada, embora limitada no espaço a pouco mais de um dezena de metros.

Em [64] a localização é suportada por *beacons*, em detrimento de algum modelo de observação de *landmarks*. Uma rede de cinco sensores Wireless (WSN) em conjunto com dois transmissores ultrassónicos estão acoplados num robô, que comunicam com três recetores ultrassónicos fixos acoplados num suporte colocado em posição elevada. Os transmissores ultrassónicos têm posição fixa e conhecida, transmitindo periodicamente, informando simultaneamente via wireless a sua identidade. A *pose* do robô móvel é estimada com recurso à medicação dos seis tempos de voo (ToF).

No algoritmo EKF, é necessário ter conhecimento prévio sobre o ruído associado ao modelo do sistema e modelo dos sensores (matrizes Q e R). Conhecimento estatístico impreciso deste erro pode comprometer a performance do filtro. Em [61], os autores apresentam uma abordagem do filtro aliada à utilização de um sistema *neuro-fuzzy*, responsável pela estimação dos elementos da matriz R a cada iteração, aquando do *update* do filtro com informação das observações. Esta abordagem possibilita a utilização do filtro de Kalman para SLAM com modelos de sistemas estatisticamente pouco definidos, que de outra forma divergiriam muito rapidamente.

3.2.2 Localização topológica ou *Grid-based* de Markov

O algoritmo probabilístico de Markov considera todas as hipóteses possíveis de localização no espaço, não sendo restritivo a uma representação gaussiana e unimodal na incerteza da posição, conforme o filtro de Kalman. A possível posição do robô (estado) é representada

por uma função densidade de probabilidade multimodal.

O espaço é discretizado por um número finito de estados, com representação topológica ou baseada em *Grids*. Abordagens topológicas permitem uma localização aproximada da posição do robô recorrendo a identificação de *landmarks*. Na abordagem *Grid Base*, o espaço é dividido em células, a cada célula é atribuída uma probabilidade correspondente a função de *likelihood* do robô ocupar essa posição. A desvantagem deste modelo está sobretudo associada aos requisitos computacionais necessários, com elevado *overhead* e *tradeoff* na resolução e dimensão do espaço para aplicabilidade em *real-time*. O algoritmo divide-se em duas fases, *update* e *prediction* semelhante ao filtro de Kalman.

Num trabalho notável, em [57] uma abordagem de localização de Markov é exposta, denominada por localização de Monte Carlo (MCL) baseada num filtro de partículas com técnica de amostragem, ponderação e reamostragem. A ideia subjacente ao algoritmo MCL está na representação da probabilidade que descreve a possível posição do estado inicial, denominada por *belief* (bel), numa abordagem de partículas ponderadas, aleatoriamente distribuídas em detrimento da probabilidade em cada *Grid*. A abordagem foi testada em diversos cenários reais com recurso a diferentes robôs, nomeadamente o Pioneer, equipados com matrizes de sonares, *laser range finders* e uma ou duas câmaras. Os resultados revelaram ser significativamente mais precisos do que os prévios algoritmos de localização de Markov, com recurso a um menor custo computacional e memória, com inclusive sucesso na localização em condições em que outras abordagens falharam.

Em [65], o algoritmo de localização de Monte Carlo é utilizado com recurso a um sistema de *Bluetooth*. Combinando informação de sensores presentes num telemóvel, acelerômetro e bússola, com a variação de nível de RSSI de um sistema de beacons Bluetooth de baixo consumo. O algoritmo é aplicado segundo as fases do MCL, da seguinte forma:

1. Atualização da estimação da posição - baseada no movimento do robô.

   O estado inicial do robô é desconhecido, sendo a actualização da estimação da posição suportada no movimento, recorrendo à informação do acelerômetro e bússola. Aquando do movimento do robô, são geradas N amostras que traduzem possíveis posições e as amostras são pesadas com a função de *likelihood*.

2. Atualização das observações - baseado na leitura dos sensores Bluetooth.

   Nesta fase, a observação é utilizada para o processo da nova ponderação das amostras, baseado na localização de Markov. Os sensores dos beacons Bluetooth, através do nível de potência do sinal RSSI, são convertidos em distancias e estas medidas

são utilizadas de forma a atribuir maior peso às partículas que melhor expressam a posição do robô.

Com objetivo de resolução milimétrica na localização de um robô para aplicação industrial, em [66] é combinado a localização com MCL, técnica de amostragem KLD [67] e procedimentos de reposicionamento preciso usando alinhamento do *scan* proposto por Celsi em [68]. O algoritmo MCL é ajustado de forma a atingir critério de eficiência, resolução e confiabilidade. A performance do MCL é otimizada na distribuição de partículas com aplicação do algoritmo de amostragem KDL, que adapta o número de partículas a distribuir limitando o erro gerado pela divergência de Kullback-Leibler.

Adicionalmente, para melhoria de resolução pelo ainda limitado número de partículas e resolução da *grid* para os resultados pretendidos nas aplicações industriais, uma correspondência de *scans* é levada a cabo com observações de referência previamente registadas. A posição do robô é então calculada em relação aos *scans* de referência, que não estão sujeitos à resolução da *grid* do MCL. Na experiência conduzida, o erro máximo no posicionamento estimado face ao real foi de 17mm e 0.53 graus, o que prova o nível de resolução atingido e aplicabilidade de modelos probabilísticos, embora com abordagens refinadas em cenários com elevado critério de resolução no posicionamento.

São diversas as abordagens, em particular na ponderação e amostragem de partículas assim como nos sensores utilizados. Em [69] a localização é suportada de um sensor inercial e um *laser-range-finder*, o algoritmo MCL é empregue por meio de um filtro de partículas com amostragem SIS, não só para localização mas também para o mapeamento utilizando um mapa de *grids*. Em [70], a localização MCL recorre a sensor WIFI combinado com odometria, o mapa é definido pela sua geometria e informação sobre pontos de acesso *wifi* e nível de sinal. Numa fase de aprendizagem todos os dados do mapa são obtidos servindo de suporte ao processo de localização. Em [71] uma abordagem de localização para ambientes urbanos e aplicabilidade na indústria automóvel é apresentada, recorrendo a localização *Semi-Markov* com recurso a quatro radares.

3.2.3 Controlo do Movimento robô tração diferencial

Existe na Literatura o vasto número de referências com diferentes estratégias para o controlo de robôs móveis, a problemática do controlo é normalmente dividida em:

- Planeamento de trajetórias
- *Trajectory tracking* e *path following*
- Estabilização num ponto

A prova ADC restringe as necessidades do planeamento de trajetória, o caminho geométrico a ser percorrido encontra-se bem definido, não obstante possíveis situações de obstáculos, esses serão considerados com perturbações à trajetória a percorrer. O problema de estabilização num ponto, endereça cenários de estabilização em determinada *pose* pretendida, como é a condição de estacionamento. Nenhum destes dois pontos são foco deste trabalho.

- *Path following*

  Num *path following* o robô deve percorrer um caminho geométrico minimizando a sua distância a determinada caminho, d ilustrado na figura 3.1. A velocidade do robô é assumida como constante $v \neq 0$ não obedecendo a nenhuma lei de controlo temporal. Uma abordagem prática deste tipo de controlo, seria assumir a velocidade linear do robô constante mantendo-o a determinada distancia das faixas laterais da via de condução.

- *Path tracking*

  Num *path tracking* o robô deve atingir e seguir uma trajetória definida no plano cartesiano com determinada lei temporal associada, a velocidade linear passa a ser variável. O *tracking* da trajetória é tido como um problema de estabilização do erro em relação a determinada referência, habitualmente tido como um robô virtual que se desloca pela trajetória.

Em [72] e [73], são implementadas abordagens com utilização do algoritmo *pure-persuit*, o *pure-persuit* é um algoritmo com abordagem geométrica na definição da referência a seguir. Uma circunferência com centro geométrico no referencial do robô é definida, o *waypoint* de referência é determinado pela intercessão ou proximidade com esta circunferência de raio denominado por *lookahead*. Em [73], a implementação do algoritmo é realizada com *lookahead* variável para melhor adaptação da trajetória dependente da velocidade.

O *pure-persuit* é um controlador de implementação simples, é contudo aproximado no seguimento da trajetória e não permite estabilização num determinado ponto.

A linearização dinâmica por *feedback* é outra abordagem utilizada no controlo, tem por base o aumento das variáveis de estado para alcance da linearização exata do modelo do robô, a linearização alcançada permite projetar uma lei de controlo linear. Em [74] o controlo via linearização dinâmica por *feedback* é empregue no controlo de um modelo *piecewise bilinear* de aproximação parcial da dinâmica do robô, semelhante a carros.

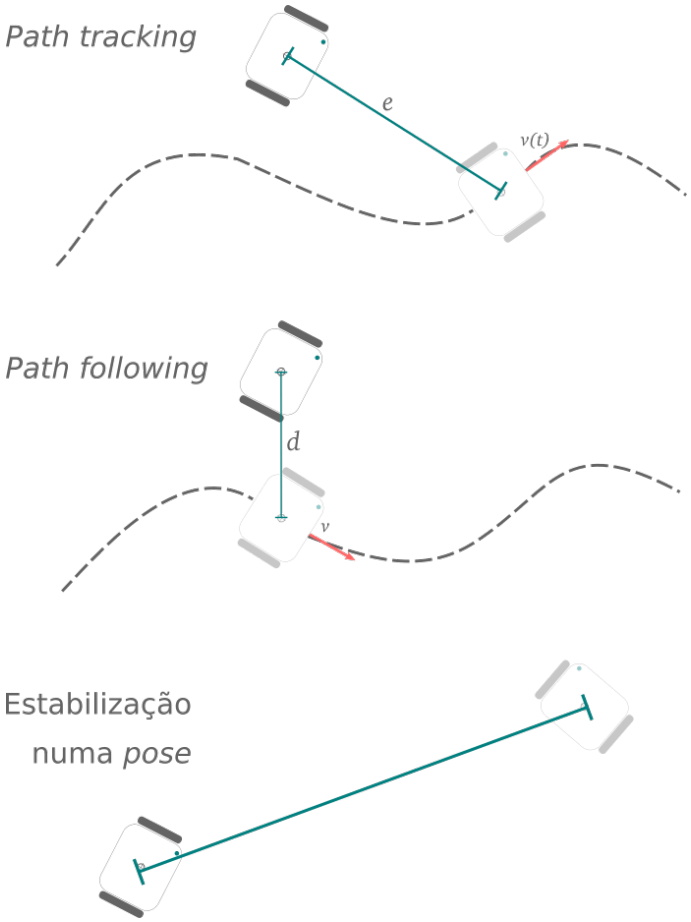


Figura 3.1: Problemas clássicos de controlo

Outra abordagem é o controlo via *backstepping*, as referências [75] e [76] apresentam implementações desta metodologia de controlo em robôs diferencias. Nguyen Gia Minh Thao *et al.* propõem em [76] o controlo com recurso a um PID *backstepping* para equilíbrio de um robô diferencial, a correção da orientação do robô com apenas duas rodas é alcançada por uma lei de controlo que deriva desta abordagem com recurso às funções de Lyapunov.

Mais recentemente, outras abordagens ao controlo têm sido empregues pela via de novos paradigmas como o *reinforcement learning*. Exemplos podem ser encontrados em [77, 78, 79]. Em [79] o controlo da direção é realizado recorrendo a *reinforcement learning* pelo método *Neural Fitted Q Iteration*. Consiste essencialmente na definição de uma função (Q) que cumpre a tarefa de controlo pretendida, obtida iterativamente via aprendizagem da rede neuronal. Este modelo de aprendizagem prescinde do conhecimento do modelo cinemático ou dinâmico do sistema, algo presente em todas as restantes abordagens.

O controlo por modelo preditivo (MPC) é outra abordagem muito utilizada no *path traking*, o método é apresentado nas referências [80, 81, 82] utilizando robôs com movimento diferencial.

A ideia básica do algoritmo é a elaboração de previsão de diferentes *outputs* utilizando o modelo do sistema, selecionando a previsão que minimiza o erro face à condição de controlo desejada. É um controlador multi-variável, que permite definir limitações do sistema e opera via realimentação.

Em [82] uma alteração ao método MPC é apresentada limitando o modelo a duas fases, uma fase de otimização segundo algoritmo RPROP e previsão do estado recorrendo ao modelo cinemático do robô e *cost function* alterada. O mapa é definido por *waypoints*, o *tracking* da trajetória é realizado com pontos de referência - (x, y) - em detrimento da abordagem de minimização da distância de erro relativa à trajetória. GUOXING BAI *et al.* [83] realiza abordagem semelhante não descartando contudo efeitos do deslizamento, apresentando para tal um modelo dinâmico.

3.3 Discussão do Estado da Arte

Na análise efetuada do estado da arte, foram identificados os principais Simuladores robóticos apresentando as suas vantagens e focos de aplicação, assim como características diversas que apoiam à decisão na escolha para o trabalho em estudo. Não obstante a maior adesão da comunicada ao simulador Gazebo, ou ao endereçamento concreto de ferramentas para a condução autónoma do Webots, o simulador preferencial e selecionado

para desenvolvimento nesta dissertação foi o MORSE. O MORSE apresenta ser promissor no realismo gráfico, simples de abordar, modular e dispõem dos sensores e modelo de plataforma pretendido. A metodologia para inclusão de incertezas, ajuste no grau de realismo e algumas abstracções foram melhor percecionadas na forma de as abordar.

A combinação de sensores a utilizar é algo condicionada pelos requisitos da prova de Condução Autónoma, o robô não pode depender de elementos externos, pelo que ficam excluídas metodologias com recurso a *beacons* ou marcadores. O robô deve estimar o seu posicionamento através do seu movimento e perceção da sua localização da pista. Fica claro que o sistema deverá incluir elementos de perceção, utilizando para isso um sensor ótico aliado a uma abordagem *dead reckoning* recorrendo a encoders e/ou sensor IMU.

Sendo bem definido o posicionamento inicial do robô para cada inicio de manga das provas de condução autónoma, a incerteza da *pose* do robô no estado inicial é residual, podendo o filtro ser alimentado desta informação. Com este enquadramento, aliado à maior eficiência computacional na fusão sensorial, o filtro de Kalman Estendido foi escolhido como solução para o módulo de localização.

Capítulo 4

Fundamentos Teóricos

4.1 Teoria do Controlo

O controlo de sistemas apresenta fundamentalmente duas abordagens, controlo sem realimentação do estado(*Open-loop*) e por malha de realimentação do estado (*closed-loop*), na robótica móvel o controlo de sistemas é maioritariamente realizado por realimentação. Nesta abordagem a saída do sistema é capturada por um ou diversos sensores e comparada com a referência pretendida. O sistema é atuado com base no erro observado pela ação do controlador.

O modelo *closed-loop* é apresentado no diagrama da figura 4.2.

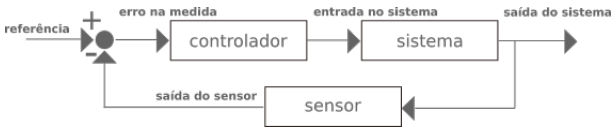


Figura 4.1: Diagrama de blocos do controlo em $closed - loop$

4.1.1 Função de transferência (*close-loop*)

A função de transferência de um sistema controlado por realimentação de estado descreve o comportamento do sistema. Assume-se que o sistema, o controlador e sensor são lineares e invariantes no tempo.

Os sistemas com captura de uma única variável de saída são denominados por SISO (Single-input-single-output) ao passo que os sistemas MIMO (Multi-Input-Multi-Output) aplicam a mesma metodologia de realimentação observando e atuando com múltiplas entradas e saídas, representadas por vetores de variáveis.

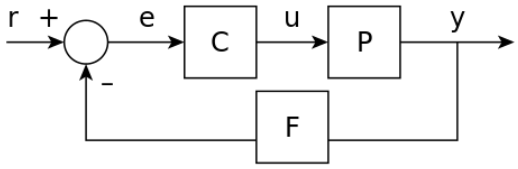


Figura 4.2: Diagrama de blocos modelo controlo $closed - loop$

A análise do função de transferência é obtida recorrendo à transformada de Laplace.

$$\begin{aligned} Y(s) &= P(s)U(s) \\ U(s) &= C(s)E(s) \\ E(s) &= R(s) - F(s)Y(s) \end{aligned} \tag{4.1}$$

Resolvendo pela relação de saída em função da entrada do sistema:

$$Y(s) = \frac{P(s)C(s)}{1 + P(s)C(s)F(s)} R(s)Y(s) = H(s)R(s) \tag{4.2}$$

Resultando na denominada função de transferência $H(s)$:

$$H(s) = \frac{P(s)C(s)}{1 + P(s)C(s)F(s)} \tag{4.3}$$

4.1.2 Modelo Espaço de Estados

A representação via modelo espaço estados é um método sistemático e genérico de representação de um sistema. As entradas e saídas de um sistemas são relacionadas com as variáveis de estado pela equação diferencial de primeira ordem.

A representação via modelo espaço de estados, no caso mais genérico de um sistema linear é definido pela equação 4.4.

$$\begin{aligned} \dot{x}(t) &= A(t)x(t) + B(t)u(t) \\ \dot{y}(t) &= C(t)x(t) + D(t)u(t) \end{aligned} \tag{4.4}$$

Equação 4.4 com representação via diagrama de blocos exposta na figura 4.3

$x(t)$: representa o vetor de estados

$y(t)$: representa o vetor de saídas

$u(t)$: representa o vetor de entradas ou de controlo

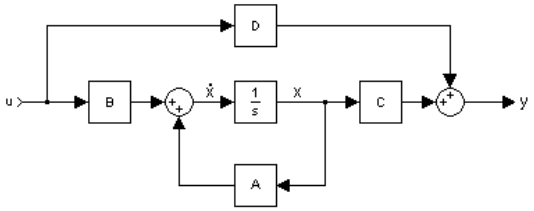


Figura 4.3: Diagrama Modelo espaço de estados

$A(t)$: matriz do modelo ou estado do sistema

$B(t)$: matriz de entradas do controlo

$C(t)$: matriz de saídas

$D(t)$: matriz de feedforward (assume valor 0 na ausência desta alimentação direta)

4.1.3 Sistema lineares e não lineares

Os controladores partem do conhecimento dos modelos matemáticos representativos do comportamento dos sistemas a entradas do controlo. Os sistemas são divididos entre lineares e não lineares.

Os sistemas lineares são bem comportados e bem definidos o que não ocorre na realidade, o mundo tende a ser descrito por equações não lineares conforme é exemplo o modelo cinemático do robô diferencial exposto neste trabalho na subsecção 4.1.5.

Os sistemas lineares, denominados por sistemas LTI (*Linear time-invariant*) são sistemas com saídas proporcionais às entradas e não dependem do tempo, são descritos por equações diferencias lineares.

Os sistemas não lineares, não obedecem ao princípio da sobreposição pelo que as saídas não são proporcionais às entradas, são geralmente descritos através de equações diferenciais não lineares. São matematicamente difíceis de descrever, complexos e tendem habitualmente na robótica a serem aproximados por equações lineares através do processo de Linearização. Neste trabalho o modelo de sistema do robô diferencial é linearizado conforme descrito na subsecção 4.4.1.

4.1.4 Análise da propriedade dos sistemas

Para o controlo, os sistemas têm que ser analisados e definidos na sua estabilidade, controlabilidade e observabilidade.

A estabilidade analisa o comportamento do sistema, definindo se o mesmo responde de forma esperada a entradas do controlo. Por definição no seu sentido mais amplo, um

sistema é estável se responder com saída limitada a qualquer entrada limitada.

Matematicamente, um sistema é estável se todos os pólos da sua função de transferência forem valores reais negativos. Caso sejam raízes simples que estejam sobre o eixo imaginário, o sistema é definido como marginalmente estável, na presença de pelo menos uma raiz nos quadrantes positivos do eixo real ou múltiplas no eixo imaginário, o sistema é instável.

A estabilidade do sistema pode igualmente ser aferida pelo critério de Lyapunov, que expondo de forma simplista refere, se uma solução inicia próxima de um ponto de equilíbrio, fica para sempre próxima desse ponto de equilíbrio.

A controlabilidade define se o sistema pode ser controlado levando-o a atingir qualquer estado pretendido, partindo de qualquer estado inicial num intervalo de tempo finito, esta é a definição de um sistema totalmente controlável. A controlabilidade é aferida pela análise da matriz de controlabilidade (R), esta tem subjacente um conceito, se o sistema é controlável significa que existe uma sequência de entradas que transfere o estado inicial do sistema x_0 em t_0 para o estado final pretendido x_n em t_n. Num sistema LTI:

$$R = [B\ AB\ A^2B\ \dots\ A^{n-1}B] \tag{4.5}$$

O número linhas ou colunas linearmente independentes da matriz R é obtido pelo *rank*, se o *rank* da matriz R for igual à dimensão do sistema (n), o sistema é totalmente controlável.

$$rank(R) = n \tag{4.6}$$

A Observabilidade de um sistema afere se é possível definir o comportamento do sistema observando as saídas do sistema. Num sistema não observável o estado ou algumas variáveis do estado não podem ser determinadas pela observação das saídas através de um sensor. Na definição formal um sistema é observável se em qualquer sequência de estado ou entrada do controlo, seja possível determinar o estado atual recorrendo unicamente à informação das saídas.

Analogamente à análise da controlabilidade, a observabilidade do sistema é aferida pelo número de linhas ou colunas linearmente independentes da matriz de observabilidade Ω, sendo esta definida por:

$$\Omega = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{4.7}$$

o sistema é completamente observável se e só se:

$$rank(\Omega) = n \tag{4.8}$$

4.1.5 Modelo do robô diferencial

O robô considerado para este trabalho assenta num modelo diferencial. O robô diferencial é constituído por duas rodas com controlo independente de velocidade e uma ou mais rodas adicionais para equilíbrio da plataforma. O robô diferencial apresenta limitações de movimento, a configuração e tipo de rodas aliado à condição de não deslizamento inibem o robô de gerar movimento sobre o seu eixo transversal. Esta configuração impõem restrições não-holonômicas, definem-se como não-holonômicos sistemas com dimensão finita onde algum tipo de restrição é imposta a um ou mais estados do sistema.

O controlo do robô diferencial é realizado pelas entradas $u = [v\ \omega]^T$, em que v representa a velocidade linear e ω a velocidade angular. O estado cinemático do robô, também denominado como *pose* é definido por $[x\ y\ \theta]^T$, em que (x, y) representa a posição em coordenadas Cartesianas do robô face ao referencial fixo e θ a orientação do robô face ao eixo x do referencial fixo.

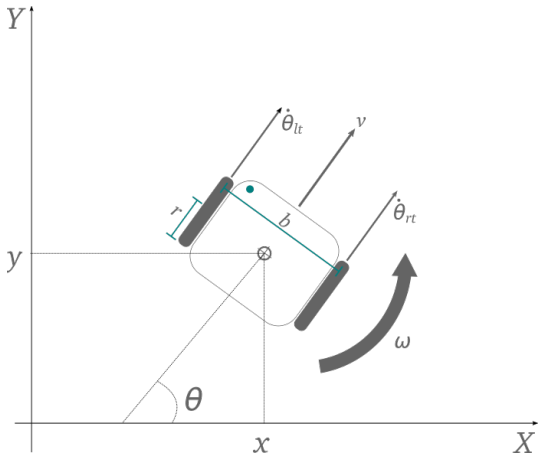


Figura 4.4: Robô diferencial

O modelo cinemático que define a *pose* do robô diferencial é dado pela equação 4.9.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{r}{2} cos(\theta) & \frac{r}{2} cos(\theta) \\ \frac{r}{2} sin(\theta) & \frac{r}{2} sin(\theta) \\ -\frac{r}{b} & \frac{r}{b} \end{bmatrix} \begin{bmatrix} \dot{\theta}_{lt} \\ \dot{\theta}_{rt} \end{bmatrix} \tag{4.9}$$

Em que r representa o raio da roda e b a distância entre rodas, de referir as entradas deste modelo $u = [\dot{\theta}_{lt}\ \dot{\theta}_{rt}]^T$ que corresponde ao vector de velocidades de cada roda do robô.

O modelo exposto na equação 4.9 é o modelo necessário para implementação, não é contudo natural nem perceptível as entradas serem definidas pelas velocidades de cada roda ($\dot{\theta}_{lt},\ \dot{\theta}_{rt}$) aquando do desenho de controladores. O modelo *Unicycle* exposto em 6.23 ultrapassa esta questão sendo mais intuitivo, com entradas definidas por $u = [v\ \omega]^T$.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{4.10}$$

A relação do modelo cinemático diferencial e modelo *Unicycle* é dada por:

$$\begin{aligned} v &= \frac{r}{2} * (\dot{\theta}_{rt} + \dot{\theta}_{lt}) \\ \omega &= \frac{r}{b} * (\dot{\theta}_{rt} - \dot{\theta}_{lt}) \end{aligned} \tag{4.11}$$

Ou partindo do controlo $u = [v\ \omega]^T$ a velocidade de cada roda é expressa pelas relações:

$$\begin{aligned} \dot{\theta}_{rt} &= \frac{2v + \omega b}{2r} \\ \dot{\theta}_{lt} &= \frac{2v - \omega b}{2r} \end{aligned} \tag{4.12}$$

4.2 Localização

A determinação da *pose* de um robô num determinado mapa é endereçado na robótica móvel pela Localização, sendo denominado como um problema de estimação *tracking* da posição. O conhecimento da localização global é fulcral para a interacção de um robô com o meio envolvente, determinando acções necessárias no controlo do movimento. A Localização é inclusivamente tido como um problema de resolução fundamental para que os sistemas sejam autónomos [84].

Os mapas, previamente conhecidos assumem um sistema de coordenadas global, com informação do posicionamento sobre diversos elementos que o compõem, nomeadamente obstáculos. O robô assume um sistemas de coordenadas local, sendo o localização o exercício de estabelecer referência entre estes dois sistemas de coordenadas [85].

A posicionamento global de um robô é usualmente estimado combinando sensores e acumulando informação do posicionamento no tempo. O *dead-reckoning* é a técnica de inferir o movimento do robô, recorrendo ao modelo cinemático ou dinâmico permite estimar as alterações de posição, que acumuladas no tempo determinam uma aproximação da *pose* do robô.

O movimento inferido pelo robô através de sensores é alvo de erro, o robô no seu movimento introduz igualmente erro (deslizamento das rodas a título de exemplo) e os modelos cinemáticos ou dinâmicos são por si aproximados. Torna-se portanto imperativo a inclusão de mais sensores no processo.

A *pose* tão pouco é obtida pela medida direta dos sensores, é inferida com base na informação destes e a fusão de todos estes dados colocam o exercício da localização global, um desafio interessante.

Conforme explanado no Estado de Arte 3, os algoritmos probabilísticos são uma abordagem para fusão sensorial e estimação da localização, sendo necessário a compreensão de alguns conceitos para melhor entendimento desta abordagem. As subsecções 4.2.1, 4.2.2, 4.2.3 e 4.2.4 introduzem conceitos necessários ao entendimento do Filtro de Kalman, opção tomada para implementação.

4.2.1 Conceitos probabilísticos e terminologia utilizada

O conceito subjacente à inferência probabilística é definir leis que traduzam eventos aleatórios. Vejamos o lançamento de um dado com seis eventos possíveis (1,2,3,4,5 e 6), seja x o evento concreto e X qualquer evento possível. A probabilidade de X assumir qualquer evento x é definida como:

$$p(X = x) \tag{4.13}$$

Para o caso é igual para qualquer evento, para o evento 1 a título de exemplo:

$$p(X = 1) = 1/6 \tag{4.14}$$

Os eventos podem ser discretos, como é caso o exposto pelo lançamentos do dado, ou contínuos assumindo um número infinito de valores possíveis. A probabilidade de cada evento x é descrita por uma distribuição de probabilidade, que é igualmente contínua

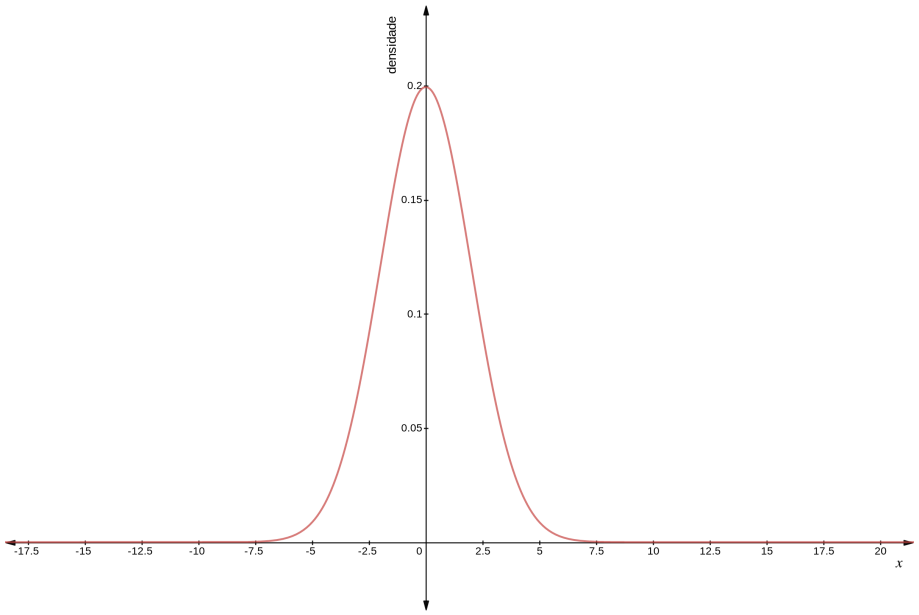


Figura 4.5: Função Densidade de probabilidade

ou discretos dependendo do tipo de eventos. Vamos avaliar apenas o exemplo contínuo, sendo esse o enquadrado neste trabalho.

A distribuição de variáveis aleatórias contínuas é descrita por uma função de densidade de probabilidade, PDF abreviado pela terminologia inglesa. A distribuição Normal é um exemplo de uma PDF, quando uma variável aleatória X acompanha uma distribuição normal é apelidada de gaussiana normal centrada em μ com variância σ^2.

$$X \sim \mathcal{N}(\mu, \sigma^2) \tag{4.15}$$

Com representação pela função Gaussiana:

$$f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{4.16}$$

A representação gráfica da função Gaussiana com $\mu = 0$ e $\sigma^2 = 4$ é apresentada na figura 4.5.

Numa função de densidade de probabilidade, a probabilidade de determinado x é obtida pelo integral da função de densidade entre $-\infty$ e x. Sendo 1 o integral de $-\infty$ a ∞ da função PDF, que corresponde à área da função:

$$\int f(x)dx = 1 \tag{4.17}$$

Na prática, isto permite obter a probabilidade de x ($p(x)$) estas confinada num intervalo $[a, b]$ sabendo que $p(a \leq x \leq b) = f(b) - f(a)$, respondendo à questão — Qual a probabilidade de x estar entre a e b?

Na robótica probabilística a informação dos sensores, do controlo e do estado do robô são igualmente tratados como variáveis aleatórias, que respondem a determinada lei de probabilidade.

Seja X e Y eventos possíveis independentes, a probabilidade de o evento x e y ocorrer é descrito por:

$$p(x, y) = p(x)p(y) \tag{4.18}$$

Assumindo que x e y são eventos dependentes, ou seja, ocorrer Y tem influência em *p(x)*, esta relação é denominada por probabilidade condicionada e descrita por:

$$p(x|y) = \frac{p(x, y)}{p(y)} \tag{4.19}$$

Podendo interpretar-se como a probabilidade de X ocorrer, sabendo que y ocorreu. Concretizando com um exemplo clássico, sabendo que foi retirada uma figura (evento y) de um baralho de cartas, qual a probabilidade dessa carta ser uma rainha de copas? O evento y condiciona a probabilidade de X visto que X e Y partilham eventos.

Outro teorema importante a considerar é o teorema da probabilidade total, que segue a definição da probabilidade condicionada, será melhor exposto a sua aplicação adiante neste trabalho, vamos contudo introduzir a sua definição para o caso contínuo:

$$p(x) = \int p(x|y)p(y)dy \tag{4.20}$$

O teorema de Bayes, ou regra de Bayes–Price (não menosprezando o trabalho de Price) é descrita por:

$$p(x|y) = \frac{p(y, x)p(x)}{p(y)} \tag{4.21}$$

O teorema relaciona a probabilidade condicionada de $p(x|y)$ com $p(y|x)$ e tem extrema utilidade na robótica.

Por terminologia filosófica, $p(x)$ e $p(y)$ são denominadas como distribuição de probabilidade *a priori*, são conhecidas previamente a determinado evento e em suma descrevem o conhecimento que é tido sobre X e Y, como a probabilidade de $1/6$ no exemplo do lançamento do dado. A probabilidade condicionada $p(x|y)$ e $p(y|x)$ são denominadas

como distribuição de probabilidades *a posteriori*, como o exemplo da Rainha de copas.

Vejamos a importância e aplicabilidade desta formulação matemática na robótica. Já foi exposto que as medidas de sensor não traduzem diretamente o estado, pelo que existe uma relação de independência do sensor em relação ao estado, se x é valor que se pretende estimar dos eventos y,em que y é determinada medida de um sensor, a probabilidade condicionada permite inferir $p(x|y)$ através do cálculo de $p(y|x)$ e das probabilidades *a priori* ($p(x)$ e $p(y)$).

Para consolidar o teorema, vejamos um exemplo prático. Seja $p(x)$ uma distribuição uniforme com igual probabilidade dos eventos a, b, c e d ocorrerem, $p(X = a) = 0,25$, $p(X = b) = 0,25$... Os eventos descrevem quatro localizações distintas e sabemos que um robô se encontra numa destas posições, $p(x)$ é a denominada distribuição de probabilidade *a priori*.

O robô tem acoplado um sensor, seja Z a medida desse sensor e abstraindo-nos da forma vamos assumir por simplicidade que o sensor nos informa sobre a localização do robô, mas incluindo incerteza na medida. O sensor reporta com 80% de certeza estar na localização certa e 10% para cada uma das localizações adjacentes, imaginando um mundo circular pelos eventos descritos. Para o exemplo do sensor medir a:

$$\begin{aligned} p(Z = a|X = a) &= 0.8 \\ p(Z = b|X = a) &= 0.1 \\ p(Z = c|X = a) &= 0.0 \\ p(Z = d|X = a) &= 0.1 \end{aligned} \tag{4.22}$$

Segundo a formulação do teorema de *Bayes–Price*, a probabilidade condicionada sabendo a medida do sensor é dada por:

$$p(X|Z) = \frac{p(Z, X)p(X)}{p(Z)} \tag{4.23}$$

O robô percorre o espaço e reporta pelo sensor estar na posições b, utilizemos o teorema de *Bayes–Price* para averiguar se o robô está efetivamente em b, tendo em conta a medida e incerteza do sensor. É necessário aplicar a regra em todas as localizações possíveis, vejamos o exemplo para a localização a.

$$p(X = a|Z = b) = \frac{p(Z = b, X = a)p(X = a)}{p(Z = b)} \tag{4.24}$$

Sabemos pelo modelo do sensor que $p(Z = b, X = a) = 0.1$, a distribuição *a prior* de $p(X = a) = 0.25$, ficando por calcular a $p(Z = b)$.

Voltando por momentos à formulação genérica de $p(x|y)$. O denominador da regra de *Bayes–Price* é independente de x, sendo $p(y)$ igual para qualquer $p(x|y)$, com somatório igual a um, por esse motivo é normalmente considerado como uma variável de normalização sendo habitual ver $p(y)^{-1}$ representado por η:

$$p(x|y) = \eta p(y|x)p(x) \tag{4.25}$$

Retomando o exemplo de calculo:

$$\begin{aligned} p(X = a|Z = b) &= \eta * 0.1 * 0.25 = 0.025\eta \\ p(X = b|Z = b) &= \eta * 0.8 * 0.25 = 0.2\eta \\ p(X = c|Z = b) &= \eta * 0.1 * 0.25 = 0.025\eta \\ p(X = d|Z = b) &= \eta * 0.0 * 0.25 = 0.0\eta \end{aligned} \tag{4.26}$$

Sabendo que o soma das probabilidade tem valor unitário, η pode ser obtido pela simples equação 4.27.

$$\begin{aligned} 0.025\eta + 0.2\eta + 0.025\eta &= 1 \\ \eta &= 4 \end{aligned} \tag{4.27}$$

A nova distribuição de probabilidade é agora descrita na tabela 4.1

Tabela 4.1: Distribuição de probabilidade

a	b	c	d
0.1	0.8	0.1	0.0

O que não é por si um resultado propriamente surpreendente, tenhamos em conta a distribuição dos eventos inicial uniforme, sendo que novas leituras do evento b aumentariam agora a probabilidade acima dos 80%, diminuindo a probabilidade do robô estar numa das localizações adjacentes.

4.2.2 Distribuição *Belief*

A distribuição de *Belief* é uma distribuição probabilística condicionada, que representa o estado previsto (x_t) recorrendo à informação do controlo $u_{1:t}$ e histórico das medidas dos sensores $z_{1:t-1}$:

$$\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t}) \tag{4.28}$$

Esta distribuição de probabilidade é no contexto dos filtros probabilísticos referida como *prediction*, considerando a previsão do estado em t com base na informação do estado anterior antes de incorporar as medidas do sensor.

4.2.3 Filtro de Bayes recursivo

O filtro de Bayes é a aplicação recursiva do teorema de Bayes, exemplo genérico e simplista de um filtro com abordagem probabilística. Executado a dois passos, *prediction* e *update*.

Na linha 2 do algoritmo 1 é realizada a previsão do estado sem informação da atual medida do sensor, esta é a fase denominada por *prediction*. O estado é previsto com base na previsão anterior do estado e entrada do controlo u_t.

De seguida na linha 3, a fase denominada por *update*, que inclui a medida do sensor. A probabilidade condicionada é exposta no formato já considerado na equação 4.25, o estado é agora estimado pelo produto da previsão $\overline{bel}(x_t)$ pela probabilidade de ter-se observado a medida z_t. Conforme o exemplo prático do Teorema de *Bayes* exposto na secção 4.2.1, a nova $bel(x_t)$ é calculada para cada hipotético estado e o resultado normalizado após obtenção da variável η.

Algorithm 1 Filtro de Bayes recursivo

1: **for** all x_t **do**
2: $\quad \overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx$
3: $\quad bel(x_t) = \eta p(z_t|x_t) \overline{bel}(x_t)$
4: **end for**
5: **return** $bel(x_t)$

4.2.4 Filtros Gaussianos

Os filtros Gaussianos são uma categoria de filtros de Bayes com implementação da estimação recursiva do estado, inspirados no teorema de *Bayes*. Os filtros Gaussianas representam a previsão (*Belief*) por uma distribuição normal multivariada, uma extensão a mais dimensões da distribuição Gaussiana referida em 4.16, ilustrada em 4.5.

A definição da distribuição multivariada é expressa por:

$$f(x|\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)} \tag{4.29}$$

Sendo x um vector de variáveis normalmente distribuídas e μ o vector das médias.

A função é completamente definida por μ e pela covariância $\sum$, pelo que assume a seguinte notação:

$$X \sim \mathcal{N}(\mu,\, \Sigma) \tag{4.30}$$

A covariância (Σ) é uma matriz quadrada, simétrica e positiva, em que o número de elementos da matriz depende quadraticamente do número de elementos do vector x.

A representação da função multivariada é igualmente unimodal, não adaptada a cenários de localização global com múltiplas hipóteses de estado, mas amplamente utilizada na robótica para *traking* da posição. O Filtro de Kalman é um exemplo de filtro recursivo Gaussiano, deriva de todos os conceitos referidos e é dos mais bem sucedidos para *traking* da *pose*. Será exposto na capítulo seguinte.

4.3 Filtro de Kalman

O Filtro e Kalman (KF) representa o estado por uma distribuição normal multivariada de média μ e covariância Σ. Se o estado representar a *pose* de um robô, a *pose* expectável do robô seria em μ_t, com incerteza quantificada por Σ_t. Uma vantagem considerável do Filtro de Kalman é a sua recursividade na estimação do estado previsto μ_t, dependendo unicamente do estado anterior μ_{t-1}, a ausência de história desde $t = 0$ é fator preponderante na robótica móvel minorando espaço de memória necessário e capacidade de processamento.

O Filtro de Kalman é aplicado a sistemas lineares, a versão com aplicação a sistemas não lineares conforme o sistema em estudo neste trabalho é denominado por Filtro de Kalman Estendido (EKF). A mecânica do algoritmo do KF é em tudo similar ao EKF e serve como ponto de partida para compreensão do algoritmo desenvolvido por *Rudolph Emil Kalman*.

4.3.1 Algoritmo do Filtro de Kalman

O filtro desenvolve-se em duas fases, conforme a secção 4.2.3, *prediction* e *update*, recursivamente e inicializado com condições iniciais de estado $x_0 \rightarrow x_{t-1}$ e covariância

$\Sigma \rightarrow \Sigma_{t-1}$.

Abordando diretamente o algoritmo exposto no algoritmo 2.

Algorithm 2 Filtro de Kalman

1: $\overline{\mu}_t = A_t\mu_{t-1} + Bu_t$
2: $\overline{\Sigma}_t = A_t\Sigma_{t-1}A_t^T + R_t$
3: $K_t = \overline{\Sigma}_t H_t^T (H_t\ \overline{\Sigma}_t H_t^T + Q_t)^{-1}$
4: $\mu_t = \overline{\mu}_t + K_t(z_t - H_t\overline{\mu}_t)$
5: $\Sigma_t = (I - K_t H_t)\overline{\Sigma}_t$
6: **return** μ_t, Σ_t

O estado é representado pelo vector $\mu_t = [x_1, x_2, ..x_n]^T$ e assume dimensão n. O controlo é realizado pelo vector $u_t = [u_1, u_2...u_m]^T$ com dimensão m e as observações representadas pelo vector $z_t = [z_1, z_2...z_k]^T$ com k medidas observadas.

Embora com nomenclatura algo complexa numa fase inicial, o filtro apresenta uma estrutura relativamente simples, com atualização dos parâmetros inicializados em $t-1$, μ_{t-1} e Σ_{t-1}, no output μ_t e Σ_t, recorrendo à informação do controlo u_t e medidas observadas z_t. Todas as restantes varáveis do filtro são calculadas pelo algoritmo, ou estabelecidas previamente.

Analisando a estrutura do algoritmo, expondo o significado das variáveis empregues e relação com a teoria explanada.

1. A primeira e segunda linha do algoritmo representam a fase *prediction*. São aqui calculados o estado e covariância prevista, com formato $p(x_t|u_t, x_{t-1})$. Representa a previsão pela distribuição de probabilidade de *belif* ($\overline{bel}(x_t)$ conforme formulação da equação 4.28) para o estado ($\overline{\mu}_t$) e covariância ($\overline{\Sigma}_t$).

   A_t e B_t são matrizes lineares e representam a transição do estado, são modelos que traduzem a alteração do sistema dinâmico a cada iteração (entre $t-1$ e t). A_t adota dimensão n x n e a matriz B_t dimensão n x m, já expostas na subsecção do Modelo em espaço de estados da subsecção 4.1.2.

   Embora os modelos implementados neste trabalho não sejam lineares, serão elucidativos sobre o significado das matrizes A_t e B_t.

   R_t é a matriz com propósito de incluir ruído no modelo do modelo do sistema que não sendo ideal, é alvo de diversos erros que são assim representados. O ruído é tido como gaussiano de média nula no Filtro de Kalman, pelo que apenas é

necessário obter conhecimento da sua variância. A matriz R_t é uma matriz diagonal de dimensão m x m, tendo na sua diagonal as variâncias dos ruídos que impactam cada variável de estado. A estimação da matriz de covariância é definida de forma algo empírica numa primeira fase, mas carece necessariamente de testes práticos para afinação considerando a quantidade de ruídos do sistema impossíveis de serem estimados previamente.

2. Na terceira linha do algoritmo é calculado o Ganho de Kalman. O ganho é utilizado como sistema de pesagem, define o ajuste do estado previsto incorporando a observação, sendo utilizado na seguinte fase do algoritmo aquando da estimação do estado.

   Nesta fase surgem como novas variáveis a matriz H_t e Q_t. A matriz H_t, de dimensão k x n representa o modelo de observação, ou seja, a relação entre as medidas do sensor e o estado. A matriz Q_t é a inclusão de ruído nas observações do sensor, sendo Q_t a matriz de covariância de dimensão n x n em tudo semelhante, na forma, ao exposto para a matriz R_t.

3. Segue-se a quarta linha do algoritmo, onde é obtido a saída do filtro com estimação do estado μ_t. O processo estocástico da inovação é aqui calculado, pela diferença entre a a observação mensurada e observação prevista $(z_t - H_t\overline{\mu}_t)$. As propriedades estatísticas da sequência da inovação reflectem o desempenho do filtro.

   A média do estado previsto é basicamente manipulada proporcionalmente ao Ganho de Kalman e pela inovação.

4. A linha 5 e última do algoritmo, procede ao cálculo do erro da covariância da nova *posterior belif* com base na alteração do ganho de Kalman. A covariância representa o erro da estimação, a diferença entre a estimação e o real valor do estado desconhecido. Conforme referido em 4.30, o resultado do filtro é expresso pela distribuição multivariada normal definida por μ e Σ, sendo portanto comum avaliar a covariância e estado aquando da implementação do filtro.

4.4 Filtro de Kalman Estendido (EKF)

O filtro de Kalman requer modelos lineares, mas o universo é ditado pela não linearidade e as equações do movimento na robótica móvel não são excepção. A aplicabilidade do filtro é na sua forma possível, recorrendo contudo à linearização dos modelos. O EKF

sintetiza a nova implementação do filtro para modelos não lineares, baseado na probabilidade da transição do estado e medidas expressas respectivamente pelas funções g e h.

$$x_t = g(u_t, x_{t-1}) + \epsilon_t \tag{4.31}$$

$$z_t = h(x_t) + \delta_t \tag{4.32}$$

O algoritmo EKF pode divergir sendo este um problema real e carece de redobrada atenção, a informação do estado inicial é preponderante para convergência do filtro.

Avaliando comparativamente o algoritmo em relação ao KF, a função g que define o modelo do sistema substitui a matriz A_t e B_t, a função h que estabelece o modelo de observação substitui a matriz H_t.

4.4.1 Linearização

A projecção das funções Gaussianas sobre a função do modelo de sistema (g) e observação (h), são distorcidas pelas não linearidades dos modelos e de tal forma que a resultante perde a sua forma gaussiana.

O processo de linearização aproxima por uma função linear, tangente a g no ponto médio da função gaussiana. O processo permite por aproximação para pequenas transições do estado, utilização de uma função linear em detrimento da função original g. Parte do princípio que sistemas não lineares, comportam-se linearmente numa vizinhança restrita. Processo análogo para a função h.

Existem diversas técnicas para Linearização de funções, o EKF recorre ao primeiro termo da expansão do Taylor, também denominado como Jacobiano. Concretizando a derivação para entendimento da aproximação em questão.

Seja (u_0, x_0) um ponto local da função g que assume um desvio face ao ponto local.

$$(u_0, x_0) \rightarrow (u = u_0 + \delta u, x = x_0 + \delta x) \tag{4.33}$$

O movimento imposto pela variação δx é obtido pela sua derivada, recordando que x_t expressa a função g conforme a equação 4.31.

$$\dot{\delta x_t} = \dot{x} + \dot{x_0} \tag{4.34}$$

O termo $\dot{x_0}$ é uma constante pelo que a sua derivada é nula, resultando em $\dot{\delta x} = \dot{x}$, substituindo 4.33 em 4.31, $\dot{\delta x}$ é expresso por:

$$\dot{\delta x} = g(u_0 + \delta u, x_0 + \delta x) \tag{4.35}$$

A expansão de Taylor permite reescrever esta função numa sucessão de termos:

$$= g(u_0, x_0) + \frac{\partial g}{\partial u}(u_0, x_0)\delta u + \frac{\partial g}{\partial u}(u_0, x_0)\delta x + h.o.t \tag{4.36}$$

Ignorando os termos de maior ordem (h.o.t) da sucessão, admissível por ausência de alterações abruptas de δx e δu, e considerando a função no ponto local $g(u_0, x_0) = 0$, o primeiro e segundo termo da sucessão representam as matrizes A e B:

$$\underbrace{\frac{\partial g}{\partial u}(u_0, x_0)}_{A}\ \delta x + \underbrace{\frac{\partial g}{\partial u}(u_0, x_0)}_{B}\ \delta x \tag{4.37}$$

O mesmo processo para obtenção da matriz H, sendo esta a derivada parcial

$$\underbrace{\frac{\partial h}{\partial x_0}}_{H}\ \delta\, x \tag{4.38}$$

4.4.2 Algoritmo do filtro de Kalman Estendido (EKF)

Algorithm 3 Filtro de Kalman Estendido

1: $\overline{\mu}_t = g(u_t, \mu_{t-1})$
2: $\overline{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
3: $K_t = \overline{\Sigma}_t H_t^T (H_t\ \overline{\Sigma}_t H_t^T + Q_t)^{-1}$
4: $\mu_t = \overline{\mu}_t + K_t(z_t - h_t(\overline{\mu}_t))$
5: $\Sigma_t = (I - K_t H_t)\overline{\Sigma}_t$
6: **return** μ_t, Σ_t

Abaixo é listado o significado de cada variável para auxílio à leitura do algoritmo recordando nomenclatura já introduzida e apresentado nova.

$\overline{\mu}_{t-1}$: Média do estado da iteração anterior

$\overline{\Sigma}_{t-1}$: Matriz de Covariância da iteração anterior

u_t: Entrada do sistema iteração atual

z_t: Observação atual

$\overline{\mu}_t$: Estado previsto

$\overline{\Sigma}_t$: Matrix de covariância prevista

$g(u_t, \mu_{t1})$: Função do sistema não linear, recebe como entrada a entrada do controlo e estado da iteração anterior

$h(\overline{\mu}_t)$: Sistema não linear da observação, recebe como entrada o estado previsto

G_t: Matrix Jacobiana do modelo do sistema

H_t: Matrix Jacobiana do modelo de observação

Q_t: Matrix ruído da observação

R_t: Matrix ruído do modelo

K_t: Ganho de Kalman

μ_t: Estado médio atualizado

Σ_t: Matriz de covariância atualizada

Como é passível de ser observado, a forma do algoritmo mantém-se inalterada relativamente ao previamente exposto KF no algoritmo 2. A mecânica do filtro é a mesma, havendo apenas inclusão dos modelos não lineares e respetivos Jacobianos em substituição das matrizes lineares.

Capítulo 5

Projeto

Esta secção apresenta a arquitectura de sistema, apresentando os diferentes módulos necessários para capacitação do robô para tarefas em modo autónomo.

A arquitetura de sistema está decomposto em vários módulos, cada um responsável pela gestão de um subsistema do robô. Encontra-se organizado por módulos, de modo a promover uma menor complexidade e ao mesmo tempo permitir construir uma solução de uma forma incremental. Esta é uma estratégia de arquitectura *standart* [86] na robótica denominada por *bahavior-based* introduzida por Ronald C Arkin em [87].

Os diferentes módulos operam concorrentemente reagindo à interacção com o ambiente real, ou simulado. Os diferentes componentes partilham informação segundo o paradigma de comunicação *publisher – subscriber*, as mensagens são comunicadas assincronamente via *broadcast*, os módulos "interessados" subscrevem a mensagem e recebem uma cópia.

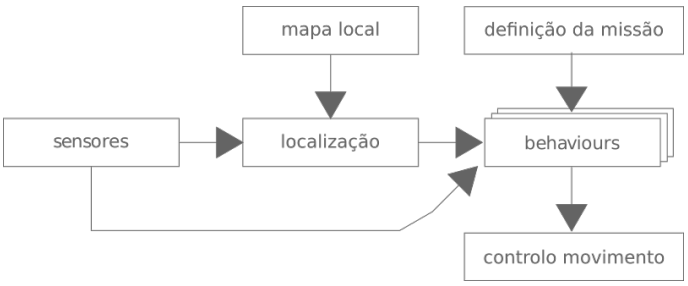


Figura 5.1: Arquitetura do sistema

O sistema global é um trabalho colaborativo em desenvolvimento por diferentes pessoas, na figura 5.2 apresentam-se destacados os componentes a serem desenvolvidos nesta dissertação e os restantes que fazem parte do sistema geral.

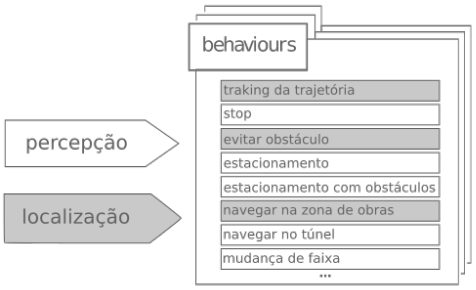


Figura 5.2: Arquitetura com os diversos *behaviors*

O robô opera segunda o processo descrito pela figura 5.3, uma malha fechada entre a perceção e o controlo que recorre à informação do mapa e instruções da missão a executar.

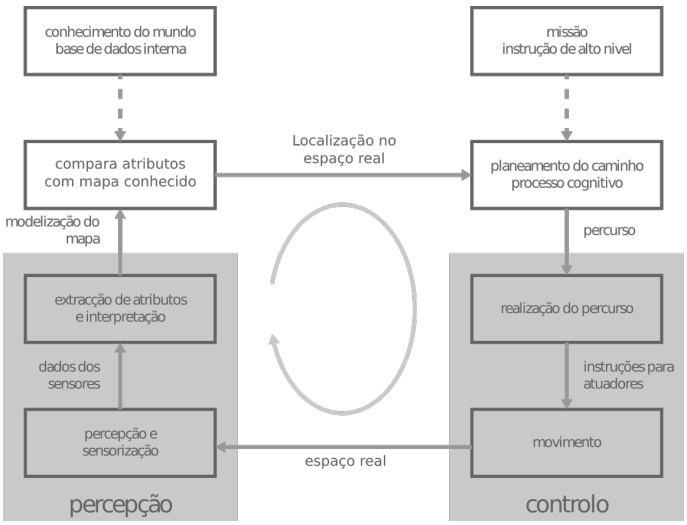


Figura 5.3: Arquitetura do sistema

A informação das missões define essencialmente os *behaviors* necessários. A título de exemplo, se o objetivo é realizar a primeira manga da prova, o *behavior avoid obstacle* é desabilitado, sabemos que nesta missão a pista está livre de qualquer obstáculo. Estas diferentes configurações permitem melhorar a performance do sistema ajustando-o para a necessidade, esta abordagem apresenta vantagens reduzindo complexidade quando não é necessário, pontos de falha, consumo energético entre outras.

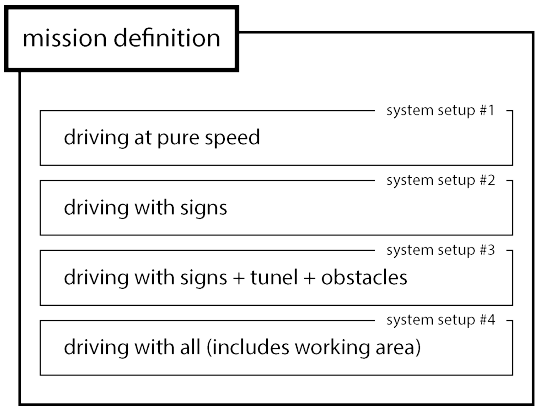


Figura 5.4: Definição das missões

O sistema proposto integra *frameworks*, ferramentas e bibliotecas que permitem o escalamento da solução, propiciando novos desenvolvimentos e validações, de forma célere.

A solução tem foco na prova FNR-ADC, mas é solução possível para o estudo de diversos tópicos da robótica móvel, nomeadamente no controlo de movimento, na localização, visão computacional ou na aplicação de redes neuronais para a condução autónoma.

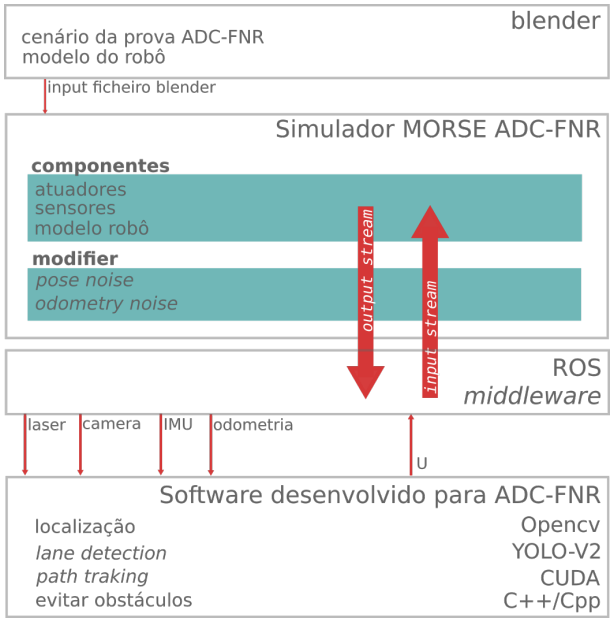


Figura 5.5: Arquitetura de alto nível

O simulador MORSE disponibiliza diferentes *middlewares* de comunicação, neste trabalho o ROS [44] foi tido como escolha pelas já referidas vantagens e presença marcada

na comunidade.

O simulador comunica através da *framework* ROS com a solução de software desenvolvido. A linguagem de programação utilizada foi o C++/CPP, tendo sempre em consideração uma estrutura de código reutilizável e boas práticas de programação. Para as soluções apoiadas no sistema de visão, o projeto é suportado pela biblioteca OpenCV, e como rede neuronal foi integrado no ambiente do ROS o YOLO-v2. O YOLO-v2 é a ferramenta *state-of-the-art* para o reconhecimento de objetos, mas para performance em tempo-real implica o processamento paralelo e com isso a biblioteca CUDA com hardware da Nvidia, como é caso a placa gráfica utilizada neste trabalho ou o sistema embebido (Jetson-Tk1) presente na solução real do robô que não é escopo deste trabalho apresentar.

A arquitetura da figura 5.5 resume a interação e ferramentas utilizadas no desenvolvimento deste projeto.

Com maior detalhe e exposto na figura 5.6, a arquitetura detalhada de *software* desenvolvido neste projeto. Os sensores utilizados, LiDAR Hokuyo, câmara e odometria alimentam diferentes blocos com intuito final na definição da ação de controlo para o robô. A visão é utilizada na identificação de *landmarks* do cenário, observações para o sistema de localização. O controlo não é reactivo ao sistema de visão e determinada regra no seguimento ou geração de faixa, é suportado na informação do mapa conhecido. A localização é parte fundamental do sistema e é alimentada pelos dados da odometria e observações da visão.

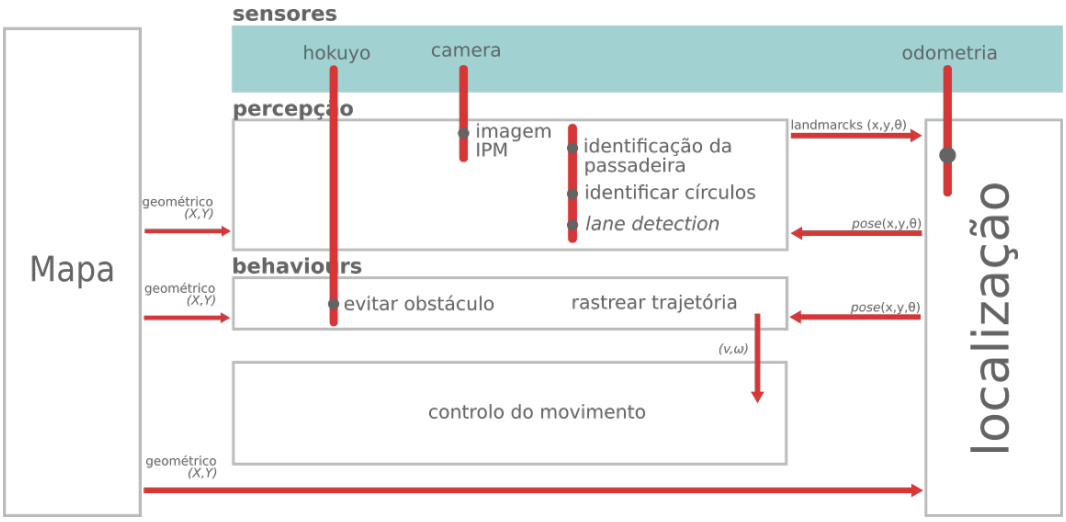


Figura 5.6: Arquitetura detalhada

O mapa gerado é geométrico e definido por coordenadas cartesianas, resume-se à

informação dos pontos referência no referencial fixo Mundo (X_{ref}, Y_{ref}) para a trajetória a ser percorrida, assim como das *landmarks* a serem identificadas. O mapa alimenta a camada de perceção, o sistema de localização e *behaviors*.

Os *behaviors* determinam as ações a serem realizadas pelo controlo, seguir a trajetória definida e na presença de obstáculo identificado pelo sensor *laser scan* Hokuyo, manobra para contorno do obstáculo.

Esta página foi intencionalmente deixada em branco.

Capítulo 6

Implementação

Este capítulo descreve os desenvolvimentos realizados na resposta aos desafios e objetivos definidos.

O ambiente de simulação da ADC-FNR desenvolvido no Blender para o simulador MORSE é detalhado em termos de funcionalidades, características e propriedades definidas. O ambiente de simulação serve de ferramenta para todos os desenvolvimentos subsequentes detalhados neste capítulo: módulo de localização, *path following*, *path tracking* e *collision avoidance*.

6.1 Simulador

6.1.1 Cenário de Simulação

A cenário da prova de condução autónoma do FNR detalhado na figura 7.1, foi inteiramente desenhado no Blender cumprindo com todas as especificações da competição. O cenário inclui o desenho das faixas de rodagem, zona de estacionamento, obstáculos, túnel, zona de trabalho delimitada por cones e painéis semafóricos.

O aspeto visual tem um papel fundamental na simulação, particularmente na robótica com recurso a sensores de visão, onde o mundo adquirido pelas câmaras pode pode ser suficientemente realista para testar algoritmos [88] e aplicá-los com sucesso na realidade. Para gerar um cenário mais realista, o mesmo inclui texturas, efeitos de luz (externa e de teto), sombras, e outros objetos em redor da pista.

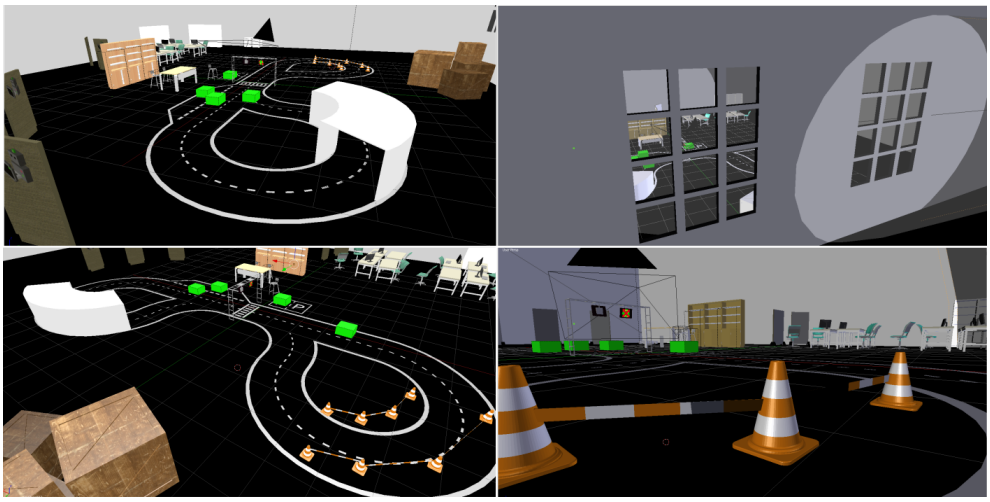


Figura 6.1: Ambiente de simulação da ADC-FNR.

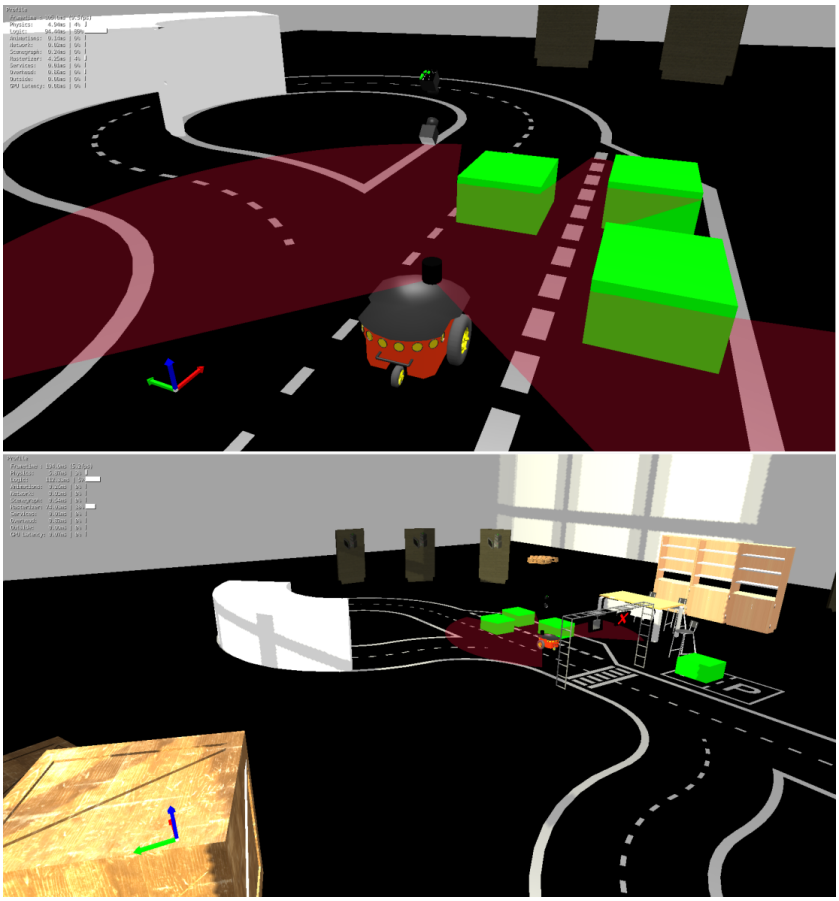


Figura 6.2: Ambiente de simulação incluindo robô Pionner.

O Morse disponibiliza um conjunto alargado de robôs, atuadores e sensores. Neste trabalho a simulação fez uso da plataforma Pionner 3-DX [89]. Este robô apresenta um corpo rígido com restrições físicas para um comportamento mais realista. As rodas giram à medida que o robô se movimenta e gozam de propriedades físicas, como a colisão com objetos. Nos capítulos seguintes as características físicas do Pionner 3-DX servem de referência para os modelos utilizados.

6.1.2 Game Engine (GE) - Logic Bricks

A prova ADC detalhada no capítulo 2 é composta por diferentes desafios com diferentes configurações do cenário.

A utilização das funcionalidades do Logic Bricks permite conciliar num mesmo ficheiro Blender todas as diferentes configurações que caracterizam as mangas prova, com alteração interactiva em tempo de simulação. A interface gráfica Logic Bricks é a ferramenta que permite configurar alterações das propriedades dos objetos e definir comportamentos.

A interface está dividida entre sensores, controladores, atuadores e suporta desenvolvimento via *scripts* na linguagem de programação *python*. O desenvolvimento de *scripts* foi testado e avaliado no modo GE do Blender, mas constatado sem justificação encontrada que o mesmo não funciona no MORSE.

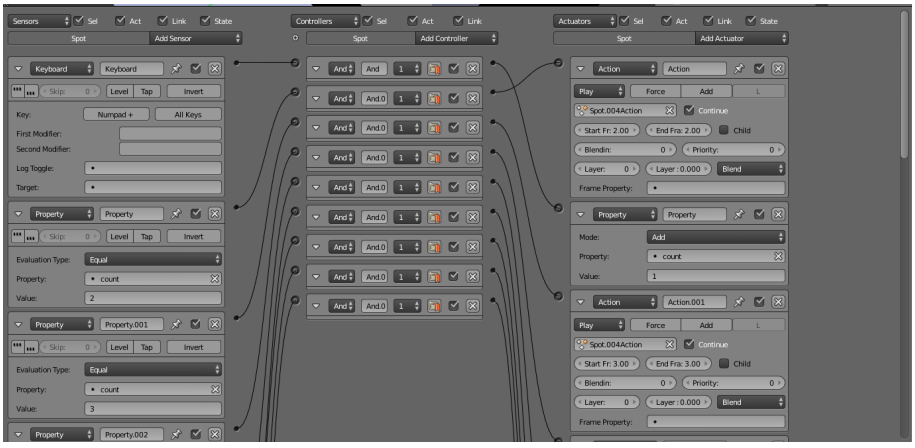


Figura 6.3: Interface de programação Logic Bricks.

O túnel, obstáculos e zona de trabalho podem ser inseridos e removidos do cenário da competição utilizando teclas predefinidas, em tempo de execução da simulação. Seis diferentes sinais nos painéis semafóricos são controlados via teclado, com comutação simultânea nos dois monitores.

A iluminação é composta por diferentes tipos de luzes, dois holofotes têm diferentes níveis de energia com ajuste simulando a entrada de luz externa ao edifício. A iluminação de teto é fixa e presente de forma distribuída. Todas as iterações com o simulador são descritas na tabela 6.1.

Tabela 6.1: Interação do utilizador com ambiente de simulação.

objeto	tecla(s) do teclado	ação
túnel	"7"e "4"	movimenta o túnel
obstáculo	"8"e "5"	movimenta o obstáculo
zona de trabalho	"9"e "6"	movimenta a zona de trabalho
sinais	esquerda/direita/seta para cima "p", "x"e "n"	altera o sinal dos monitores
holofotes	"+"	altera nível de energia dos holofotes

6.1.3 Física

O motor de simulação do MORSE pode ser utilizado em diferentes contextos, no teste e verificação de sistema robóticos, com relativo ou alto nível de abstração [88].

Os objetos desenvolvidos como *mesh* no Blender podem assumir propriedades de física ajustáveis, como massa e frição. Esta configuração é realizada no painel *Properties>>Physics*. Existem outras configurações utilizadas que incrementam o realismo do cenário, como os objetos estáticos, nos quais o robô colide sem que estes se movimentem (estrutura metálica dos sinais luminosos, paredes, armários, entre outros).

Outra classe de objetos foram definidos como obstáculos, exemplo das caixas verdes, cones sinalizadores da zona de trabalho, cadeiras, sujeitos a forças e colisões. A sua configuração é realizada em *Properties>>Physics* no *tab Collision* e comportam-se de acordo com a sua forma e corpo rígido, com massa e frição reagindo realisticamente a situações de colisão com o robô.

Os robôs, sensores e atuadores estão igualmente definidos como *mesh* com configuração das suas propriedades físicas. Estes elementos são utilizados pelo motor de física Bullet Physics Engine para realisticamente representar o comportamento e interação do robô no ambiente de simulação.

6.1.4 Builder Simulação

O simulador MORSE assenta numa estrutura modular, baseada numa biblioteca de componentes. Estes componentes estão particionados em dois ficheiros: um ficheiro Blender com representação visual do componente (uma *mesh*) e um *script* em *Python* que define a classe do objeto com o conjunto de métodos que define a sua funcionalidade. Cada componente pode registar serviços, acessíveis fora do motor de simulação através de *middleware's*.

Existem três tipos de componentes no MORSE: Robôs, sensores e atuadores. Todos estes componentes derivam de uma classe abstracta com o mesmo nome do simulador, MORSE.

O *script* em Python inicializa a simulação instanciando objetos da classe MORSE, gerando a importação de todos os *sub-packages* do *package* MORSE.

Através deste *script* é inicializado o ambiente de simulação e nele foram definidos todos os sensores, atuadores, o *middleware* de comunicação ROS, o robô e o cenário desenvolvido da prova ADC-FNR. Abaixo está exposto parte deste *script*.

```
    from math import pi
    from morse.builder import *
    from morse.modifiers.abstract_modifier import AbstractModifier
```

O robô é instanciado e suporta dois tipos de transformação, translação e rotação. O atuador *MotionVWDiff()* inicializado é associado ao robô, sendo através deste atuador que o robô recebe ordens de movimento (v ,ω)

```
# Inicializa o rob
my_robot = SegwayRMP400()
my_robot.translate(0.0, 0.0, 0.0)
my_robot.rotate(0.0, 0.0, 0)
motion =MotionVWDiff()
my_robot.append(motion)
```

Os sensores são igualmente instanciados, transladados ou rodados face ao referência associados, o robô.

```
# Inicializa o laser scan Hokuyou
    hokuyo = Hokuyo()
    hokuyo.translate(x=0.455,y=0,z=0.3)
    hokuyo.rotate(0,0,0)
    hokuyo.frequency(10)
    (...)
```

O *Builder* script pode modificar ou adicionar propriedades nos sensores e atuadores, que se repercute no motor de jogo. O Hokuyo, abaixo definido.

```
# Define propriedades do laser scan
    hokuyo.properties(Visible_arc = True)
    hokuyo.properties(resolution = 0.8)
    hokuyo.properties(scan_window = cal)
    hokuyo.properties(laser_range = 3.0)
    hokuyo.properties(layers = 1)
    hokuyo.properties(layer_separation = 0.8)
    hokuyo.properties(layer_offset = 0.25)
```

A adição ao *midleware* de comunicação ROS com definição da *frame* e tópico onde o sensor publica a informação.

```
# Define midleware de comunica  o
  my_robot.append(hokuyo)
  hokuyo.add_interface('ros', frame_id='laserScan_frame', topic="my_robot/↩
      hokuyo")
```

No mundo real os sensores e atuadores são afetados de ruído. Para aproximação da simulação aos constrangimentos do Mundo real, o MORSE disponibilidade *Modifiers* que permitem interferir diretamente na informação obtida pelos sensores, foi adicionado ruído à informação obtida pela odometria, entre outros sensores.

```
    (...)
    my_odometry.alter('Noise', pos_std=1, rot_std=1, _2D="True")
    (...)
```

Identificado na fase de desenvolvimento a impossibilidade por erro na inicialização do sensor acelerómetro, carece da correção do *script accelerometer.py* localizado no *package* da distribuição do MORSE, de forma a permitir a publicação via *strem* com *time-stamped*.

```
    from geometry_msgs.msg import TwistStamped
    (...)
    twist = TwistStamped()
```

Por fim, é inicializado o ambiente de simulação.

```
# Ambiente da prova ADC-FNR
    env = Environment('FNR')
```

6.2 Localização

A percurso da prova FNR exposto no capítulo 2 apresenta dimensões consideráveis,

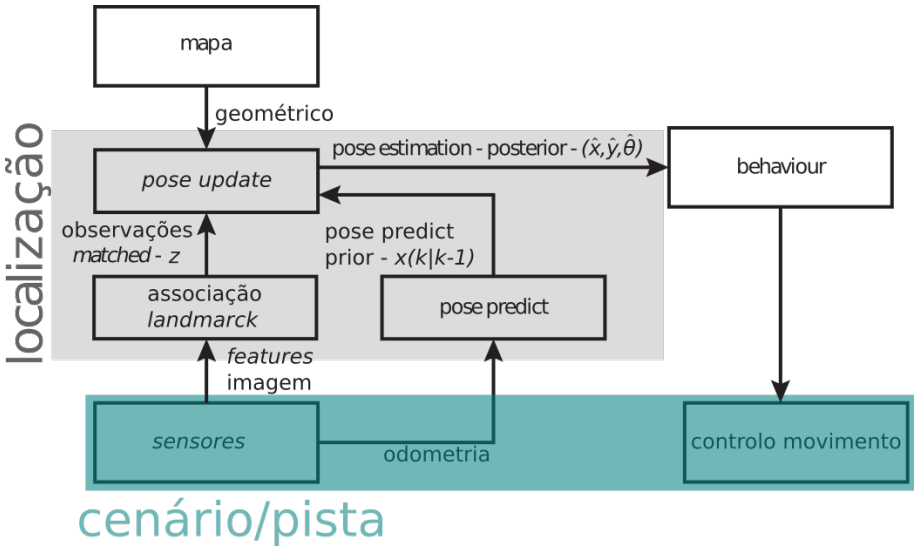


Figura 6.4: Arquitetura sistema de localização.

sendo os robôs convidados a realizar diversas voltas à pista numa mesma prova. Esta exigência, exclui a possibilidade de um sistema de localização global com recurso a *dead-reckoning*.

O sistema de localização implementado tem por base um filtro EKF. O filtro EKF conforme exposto na secção 4.3 é dividido essencialmente em duas partes, *Prediction* com base no modelo do sistema e *Update*, que recorre ao modelo e medidas das observações obtida pelos sensores.

No *Prediction*, o EKF implementado recorre ao modelo cinemático do robô diferencial para modelo do sistema, com entradas no controlo (u_t) suportadas pela odometria do robô. O estado estimado do robô representa a *pose* num sistema 2D, localização global no mapa $\mu_t = [x_t, y_t, \theta_t]^T$.

No *Update*, o módulo de perceção com base na visão alimenta o filtro de Kalman com observação de *landmarks*. A prova FNR é particularmente difícil na definição de *landmarks*, para além de espelhada o que pode gerar ambiguidade, é parca em objetos verticais fixos. O sistema implementado é baseado nas observações da passadeira e de círculos.

A figura 6.4 descreve a arquitura do sistema de localização implementado.

Adiante são expostas todas as fases de implementação do filtro, descriminando estratégias e desenvolvimentos realizadas para alcançar a funcionalidade do mesmo.

6.2.1 Modelo do sistema

O modelo cinemático do robô diferencial é tido como modelo do sistema, detalhado na subsecção 4.1.5 e descrito pela equação 4.9. Como entrada do controlo (u_t), o modelo recorre à velocidade de cada roda sendo esta informação obtida pelos dados da odometria.

O função do modelo de sistema $g(u_t, x_{t-1})$ é por conseguinte:

$$\overline{\mu}_t = g(u_t, \mu_{t-1}) = \mu_{t-1} + \begin{bmatrix} \frac{r}{2} cos(\theta_{t-1}) & \frac{r}{2} cos(\theta_{t-1}) \\ \frac{r}{2} sin(\theta_{t-1}) & \frac{r}{2} sin(\theta_{t-1}) \\ -\frac{r}{b} & \frac{r}{b} \end{bmatrix} \begin{bmatrix} \dot{v}_{lt} \\ \dot{v}_{rt} \end{bmatrix} \Delta t \tag{6.1}$$

A função $g(u_t, x_{t-1})$ estima o novo estado $\overline{\mu}_t$ com base no estado anterior e informação de controlo, corresponde à primeira linha do algoritmo de Kalman detalhado em 3.

A segundo passo do filtro que culmina com a fase *prediction* explicado em 4.3, obtêm a Covariância prevista do modelo do sistema $\overline{\Sigma}_t$, que carece do Jacobiano do modelo do sistema. A matriz Jacobiana G_t é calculada conforme a equação 6.2.

$$G_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial \mu_{t-1}} = \begin{bmatrix} \frac{\partial x_t}{\partial x_{t-1}} & \frac{\partial x_t}{\partial y_{t-1}} & \frac{\partial x_t}{\partial \theta_{t-1}} \\ \frac{\partial y_t}{\partial x_{t-1}} & \frac{\partial y_t}{\partial y_{t-1}} & \frac{\partial y_t}{\partial \theta_{t-1}} \\ \frac{\partial \theta_t}{\partial x_{t-1}} & \frac{\partial \theta_t}{\partial y_{t-1}} & \frac{\partial \theta_t}{\partial \theta_{t-1}} \end{bmatrix} \tag{6.2}$$

A matriz Jacobiano do modelo de sistema fica assim definda sendo:

$$G_t = \begin{bmatrix} 1 & 0 & -\dot{v}_{lt} \frac{r}{2} sin(\theta_{t-1}) - \dot{v}_{rt} \frac{r}{2} sin(\theta_{t-1}) \\ 0 & 1 & \dot{v}_{lt} \frac{r}{2} cos(\theta_{t-1}) + \dot{v}_{rt} \frac{r}{2} cos(\theta_{t-1}) \\ 0 & 0 & 1 \end{bmatrix} \tag{6.3}$$

Assim é garantida a multiplicação $G_t \Sigma_{t-1} G_t^T$ presente neste passo do algoritmo, ficando apenas em falta a adição da incerteza no modelo do sistema através da matriz R_t.

Seja $R_t = V_t \, M_t \, V_t^T$ em que M_t é a matriz de variâncias associadas às entradas do controlo, definida por:

$$M_t = \begin{bmatrix} \alpha_1 \, \dot{v}_{lt}^2 + \alpha_2 \, \dot{v}_{rt}^2 & 0 \\ 0 & \alpha_1 \, \dot{v}_{lt}^2 + \alpha_2 \, \dot{v}_{rt}^2 \end{bmatrix} \tag{6.4}$$

Os parâmetros α_1 e α_2 são variáveis que incorporam o ruído inerente de cada robô, impossível de se prever requerendo afinação manual por via de teste.

A matriz M_t carece agora de mapeamento para o espaço de estado do robô, com recurso a matriz Jacobina $V_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial u_t}$ pela multiplicação $V_t \, M_t \, V_t^T$. A matriz Jaco-

biana V_t coloca no espaço de estado a covariância das entradas, sendo a mesma definida por:

$$V_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial u_t} = \begin{bmatrix} \frac{\partial x_t}{\partial \dot{v}_{lt}} & \frac{\partial x_t}{\partial \dot{v}_{rt}} \\ \frac{\partial y_t}{\partial \dot{v}_{lt}} & \frac{\partial y_t}{\partial \dot{v}_{rt}} \\ \frac{\partial \theta_t}{\partial \dot{v}_{lt}} & \frac{\partial \theta_t}{\partial \dot{v}_{rt}} \end{bmatrix} = \begin{bmatrix} \frac{r}{2} cos(\theta_{t-1}) & \frac{r}{2} cos(\theta_{t-1}) \\ \frac{r}{2} sin(\theta_{t-1}) & \frac{r}{2} sin(\theta_{t-1}) \\ -\frac{r}{2} & \frac{r}{2} \end{bmatrix} \Delta t \qquad (6.5)$$

Definida a matriz R_t, fica estabelecida a primeira fase *prediction* do filtro.

Até ao momento foi definido o modelo do sistema e covariância do modelo do sistema, covariância que depende apenas da odometria pelo que tenderá a aumentar no tempo, aumentando a incerteza da localização do robô. Na secção seguinte serão definidas as seguintes fases do filtro, com inclusão das observações que tenderá a diminuir o valor da covariância e melhor definir a posição do robô.

6.2.2 Modelo de Observação

Encontra-se nesta secção descrito o modelo de observação e forma definida para identificação das *landmarks*.

6.2.2.1 Observação de circunferências

A circuito da prova ADC é constituída por duas circunferências com raio de 3 m face à faixa de rodagem lateral direita, o centro de cada uma destas circunferências constituem *landmarks* a serem identificadas.

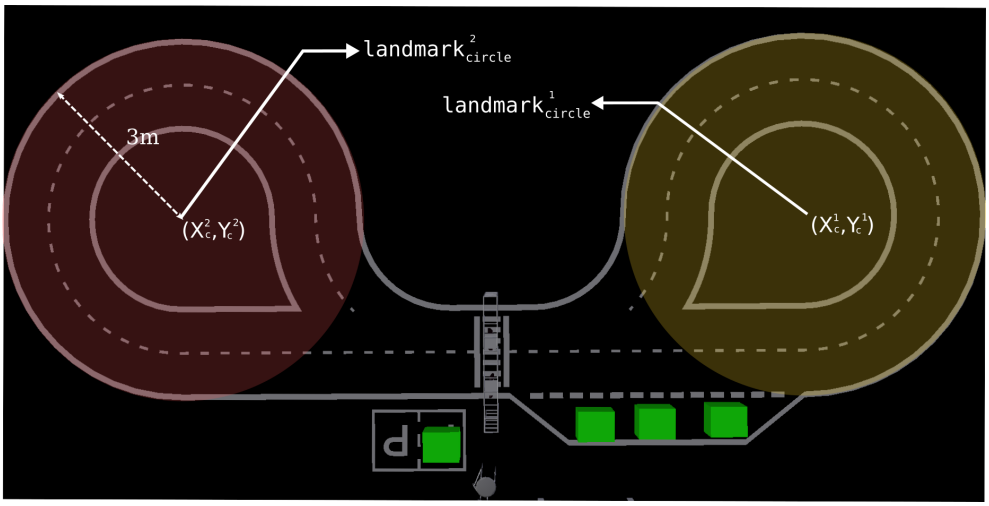


Figura 6.5: Representação das *landmarks* circunferência.

A obtenção da coordenada do centro de cada circunferência X_c^k, Y_c^k é realizada com base na seguinte estratégia:

1. Subscrição da imagem IPM
2. Identificação de faixas de rodagem e definição de faixa direita
3. Projecção de círculos face a 3 pontos da faixa direita
4. Validação de círculo identificado via critérios de raio e posição
5. Associação de observação à *landmark* do mapa com base na *pose* do robô por relação de proximidade.

A identificação de faixas de rodagem e processo IPM serão sucintamente abordados para contextualização da origem dos dados da observação das *landmarks*. O processo IPM permite por via dos dados de calibração da câmara (matriz dos parâmetros intrínsecos e extrínsecos), projectar a imagem capturada pela câmara em perspectiva e com isso, dispor de uma imagem planar do cenário.

A ausência de perspectiva facilita o processo de identificação e *tracking* das faixas de rodagem, o conhecimento dos parâmetros de calibração da câmara permitem, no plano do solo e assumindo que o mesmo é planar, relacionar cada ponto da imagem como determinada distância ao robô.

A imagem planar IPM é obtida pela matriz de transformação M_{IPM}:

$$M_{IPM} = K\,[R^T|t]\,T\,S \tag{6.6}$$

Em que:

- K: Matriz de intrínsecos obtida no processo de calibração do grupo ótico.
- R: Matriz de rotação da câmara em relação ao solo, posição do padrão de calibração aquando da calibração do grupo ótico. Obtida do processo de calibração.
- t: Vector de translação da câmara em relação ao referencial do padrão de calibração.
- T: Matriz de translação para correcção do centro da imagem.
- S: Matriz escalar para redimensionamento da imagem.

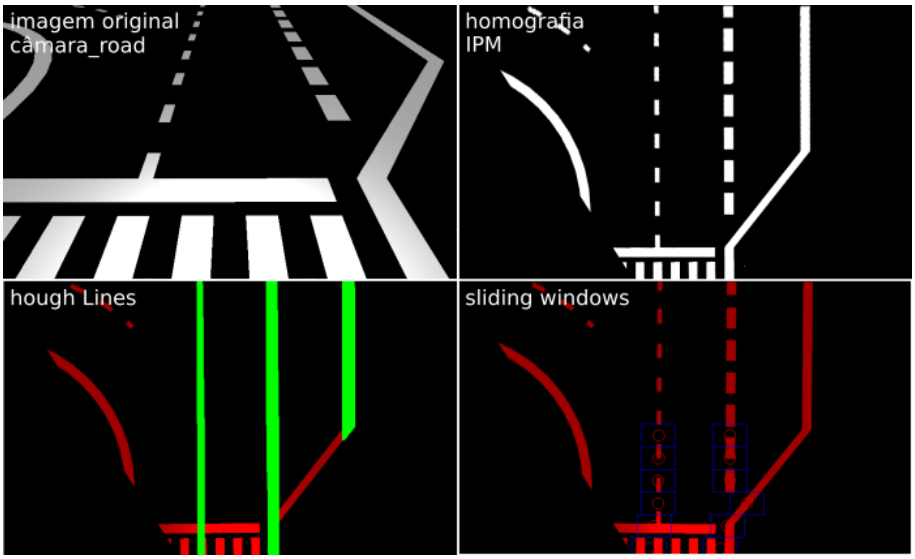


Figura 6.6: Processo para identificação de faixas de rodagem.

Excluída a perspectiva da imagem, cada *pixel* da nova imagem planar corresponde a uma distância que difere da realidade por um fator de escala. De ressalvar que esta aproximação é apenas verdade para *pixeis* do plano transformado, que no cenário exposto é o solo. Nenhum *pixel* correspondente a objetos acima do solo na imagem em perspectiva têm igual relação de distancia na imagem transformada.

O processo de identificação de faixas, parte de uma primeira iteração para identificação das linhas na imagem IPM, com critérios de verticalidade e comprimento, utilizando para isso algoritmo *houghlines*. Esta primeira iteração serve unicamente o propósito de inicialização do processo *sliding windows*, definindo as faixa à direita e esquerda do robô. Após posicionamento das "janelas" nas faixas identificadas, a porção da imagem de cada uma dessas "janelas" é processada para acompanhamento da faixa a cada *frame*, definido o ponto central de cada faixa identificada.

O processo é resumido pela sequência de imagens da figura 6.6.

O posicionamento relativo de três pontos da faixa direita de rodagem é obtido a cada *frame*, avaliando se os três pontos constituem parte de uma circunferência de raio $3\,m \pm \alpha$, que indicam a presença do robô numa das circunferências da pista. A identificação de circunferência é realizada recorrendo ao método exato seguidamente descrito.

Seja $A(x_1, y_1), B(x_2, y_2), C(x_3, y_3)$ três pontos de uma circunferência ilustrados na figura 6.7, em que m_1 e m_2 são os declives dos segmentos de reta entre os pontos A, B e C.

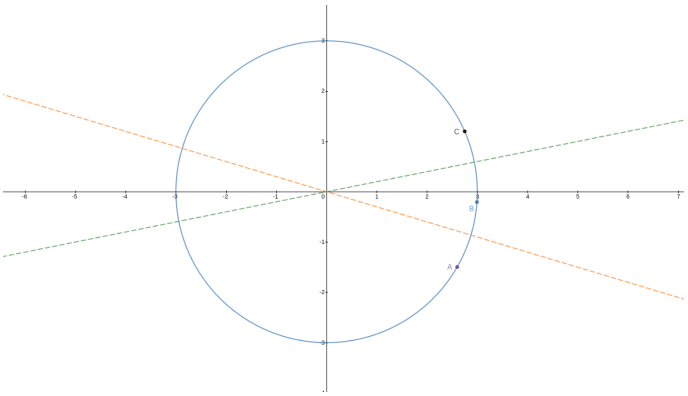


Figura 6.7: Projecção de circunferências

A coordenada X_c do centro da circunferência é obtida pela intercepção das bissetrizes perpendiculares aos segmentos de reta que interligam os pontos A, B e C.

$$X_c = \frac{m_1 m_2 (y_1 - y_3) + m_2 (x_1 + x_2) - m_1 (x_2 + x_3)}{2\,(m_2 + m_1)} \tag{6.7}$$

Sendo Y_c:

$$Y_c = -\frac{1}{m_1}\left(X_c - \frac{x_1 + x_2}{2}\right) + \frac{y_1 + y_2}{2} \tag{6.8}$$

Com informação das coordenadas do centro de circunferência (X_c, Y_c), o raio (r) é simplesmente obtido pela distância do centro de circunferência a qualquer ponto, A, B ou C. Exemplificando para o ponto A.

$$r = \sqrt{(X_c - X_A)^2 + (Y_c - Y_A)^2} \tag{6.9}$$

Os *pixeis* da imagem são relacionados com o referencial do robô, em distâncias. Seja p um *pixel* da imagem IPM de coordenadas (u_k, v_k), T e S matrizes enunciadas na equação 6.6, na imagem IPM a distância de p ao referencial da câmara é obtida por:

$$p_{cam} = T\,S\,p \tag{6.10}$$

Que difere do referencial do Mundo pela soma das transladações da câmara ao referencial do robô ($T_{cam2robo}$) e do robô ao referencial do Mundo ($T_{Mundo2Cam}$):

$$p_{Mundo} = T_{Mundo2Cam} + T_{cam2robo} + p_{cam} \tag{6.11}$$

Cada *pixel* dos pontos A, B e C é convertido para distâncias (x, y) referenciadas ao referencial do robô, pela equação 6.10 e 6.11, sendo em relação a este referencial estimado o centro da circunferência pela equação 6.7 e 6.8, este centro de circunferência é a *landmark* doravante designada por $landmark_{circle}$.

6.2.2.2 Observação da Passadeira

A identificação da passadeira recorre igualmente ao sensor de visão, com inclusão de rede neuronal YOLO-V2 para processamento de imagem e identificação da passadeira presente na imagem. O projeto inclui o YOLO[90] como um *package* embebido no *workspace* do ROS, um nó desse *package* é responsável pela subscrição da imagem IPM e identificação da passadeira.

O processo de aprendizagem da rede neuronal foi realizado com um elevado número de imagens capturadas em diversas condições de visualização da passadeira, garantindo amostra representativa. A preparação do *setup* de imagens foi auxiliado pela aplicação Yolo-mark [91], facilitando o exercício de gerar um ficheiro de texto para cada imagem, constituído pelo *id* do objeto a identificar (podem ser vários objetos por imagem) e as coordenadas do posicionamento deste na imagem.

O Yolo realiza a divisão da imagem por regiões e prevê *bounding boxes* com probabilidade associada sobre os objetos identificados. A informação da identificação do objeto, probabilidade e localização do mesmo é neste trabalho publicada num tópico do ROS.

Dois vértices da passadeira são tidos como landmarks e doravante denominados por $landmark_{crosswalk}$.

6.2.2.3 Associação de *landmarks*

Existe necessidade de associação de cada *landmark* identificada à sua respectiva *landmark* previamente conhecida do mapa. Sendo clara a divisão entre passadeira e círculo, permanecem duas possibilidades para cada uma destas sendo que associações erradas levariam a aumentos abruptos da inovação com respetivo aumento na incerteza do posicionamento.

O processo de associação é realizado por relação de proximidade da *landmarks* ao robô, comparando a observação com a referência lista de *landmarks* reais no referencial do mundo. Seja Obs_m^k o mapa das *landmarks*.

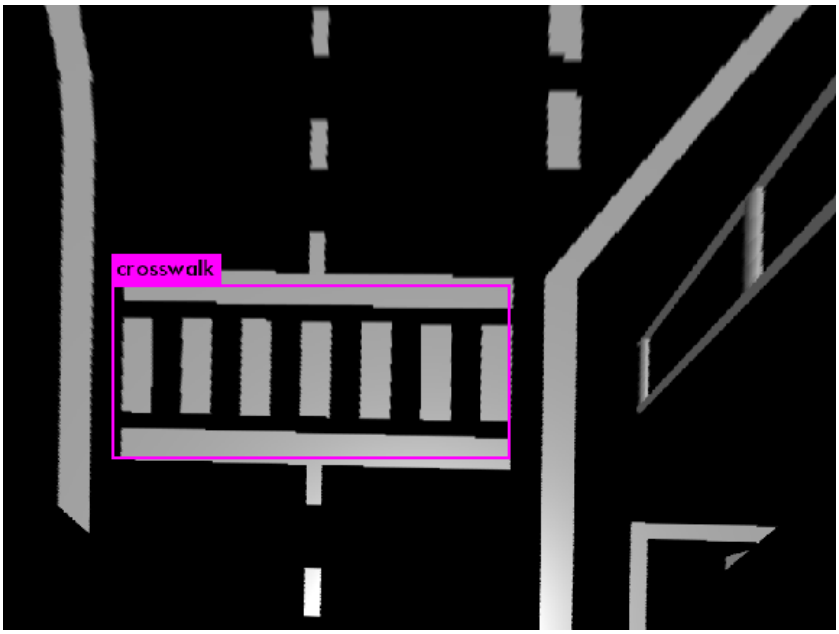


Figura 6.8: Identificação de passadeira

Algorithm 4 Associação de Landmarcks

```
1: procedure ASSOCIAÇÃO_LANDMARCK(raio, Obs_r, Obs_m^k, μ_t)
2:     if (3 − α < raio < 3 + α) then
3:         dist_1 = (μ_t + R Obs_r) − Obs_m^1
4:         dist_2 = (μ_t + R Obs_r) − Obs_m^2
5:         if dist_1 ≤ dist_2 then
6:             Obs_m = Obs_m^1
7:         else
8:             Obs_m = Obs_m^2
9:         end if
10:    end if
```

6.2.2.4 Modelo de Observação - Implementação no EKF

As observações descritas em 6.2.2.2 e 6.2.2.1 resultam numa medida (x, y) do robô à *landmark*. Seja $Obs_r = (x_r, y_r)$ a coordenada de uma *landmark* no referencial do robô e $Obs_m = (x_m, y_m)$ essa coordenada no referencial do Mundo.

$$Obs_m = \mu_t + R\,Obs_r \tag{6.12}$$

$$\begin{bmatrix} x_m \\ y_m \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \end{bmatrix} + \begin{bmatrix} cos(\theta_t) & -sin(\theta_t) \\ sin(\theta_t) & cos(\theta_t) \end{bmatrix} \begin{bmatrix} x_r \\ y_r \end{bmatrix} \tag{6.13}$$

A observação da *landmark* no referência do robô é convertida para o mundo recorrendo à matriz de rotação e ao estado atual do robô $u_t = [x_t\, y_t]^T$

O posicionamento real conhecido das *landmarks* é referenciado ao Mundo, a projeção dessas *landmarks* para o referencial do robô é conseguido pela simples manipulação da equação 6.15.

$$Obs_r = R^{-1}\left(Obs_m - \mu_t\right) \tag{6.14}$$

$$\begin{bmatrix} x_r \\ y_r \end{bmatrix} = \begin{bmatrix} cos(\theta_t) & sin(\theta_t) \\ -sin(\theta_t) & cos(\theta_t) \end{bmatrix} \begin{bmatrix} x_m - x_t \\ y_m - y_t \end{bmatrix} \tag{6.15}$$

Esta transformação resulta ser o modelo de observação definido por $\overline{z}_t = h(\overline{u}_t)$. A Observação prevista assume o robô na posição do estado previsto, e com informação da posição real no mundo da *landmark* antecipada a observação do sensor. Responde de alguma forma à questão, que medidas serão tomadas pelo sensor na observação da *landmark* assumindo que o robô está no estado previsto?

$$\overline{z}_t = h(\overline{\mu}_t) = R^{-1} \begin{bmatrix} x_m - \overline{x}_t \\ y_m - \overline{y}_t \end{bmatrix} \tag{6.16}$$

Sendo H a matriz Jacobiana da observação, conforme descrito na equação 4.38

$$H_t = \frac{\partial h(\overline{\mu}_t, m)}{\partial \mu_t} \tag{6.17}$$

$$H_t = \begin{bmatrix} -cos(\overline{\theta}_t) & -sin(\overline{\theta}_t) & -x_m\, sin(\overline{\theta}_t) + \overline{x}_t\, sin(\overline{\theta}_t) + y_m\, cos(\overline{\theta}_t) - \overline{y}_t\, cos(\overline{\theta}_t) \\ sin(\overline{\theta}_t) & -cos(\overline{\theta}_t) & -x_m\, cos(\overline{\theta}_t) + \overline{x}_t\, cos(\overline{\theta}_t) - y_m\, sin(\overline{\theta}_t) + \overline{y}_t\, sin(\overline{\theta}_t) \end{bmatrix} \tag{6.18}$$

O ruído da inovação ($H_t\ \overline{\Sigma}_t H_t^T + Q_t$) é fruto da multiplicação da covariância estimada $\overline{\Sigma}_t$ pelo Jacobiano da Matriz de observação, com adição de ruído pela matriz Q_t. A matriz Q_t é uma matriz diagonal composta pela variância dos dos parâmetros de ruído (σ_x, σ_y), a serem manualmente ajustados aquando da calibração do filtro.

$$Q_t = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix} \tag{6.19}$$

O Ganho de Kalman, o próximo passo do algoritmo EKF pode agora ser obtido:

$$K_t = \overline{\Sigma}_t H_t^T (H_t \, \overline{\Sigma}_t H_t^T + Q_t)^{-1} \tag{6.20}$$

A fase do *update* culmina com a estimação da média da *pose*, estado estimado, que inclui a observação real do sensor z_t (distancia (x, y) do robô à passadeira ou centro da circunferência):

$$\mu_t = \overline{\mu}_t + K_t(z_t - h_t(\overline{\mu}_t)) \tag{6.21}$$

Por fim. o cálculo da erro da covariância:

$$\Sigma_t = (I - K_t H_t)\overline{\Sigma}_t \tag{6.22}$$

6.3 Path following e Path tracking

Nesta secção são apresentados os modelos implementados dos controladores *Path following* e *Path tracking*, a explicação é precedida pela definição de *path* gerado como referência ao controlo.

6.3.1 Pontos de referência - *waypoints*

A prova ADC é definida por duas faixas de rodagem, abordaremos neste controlo o percurso de uma dessas faixas. O percurso geométrico da prova encontra-se bem definido não sendo portanto necessário proceder ao planeamento de um caminho a percorrer, interessa contudo definir o caminho a ser realizada garantindo as restrições não-holonómicas do robô diferencial.

Foram definidas referências (x_{ref}, y_{ref}) para definição do caminho geométrico a ser percorrido pelo robô, a figura 6.9 ilustra a interligação dos pontos referência gerada pelo Rviz, o posicionamento do pontos de referência garantem as condições não holonómicas do robô.

Contrariamente ao problema de estabilização num ponto, em que o controlo acarreta a correção definição da orientação final pretendida, o controlo num *path following* pode simplesmente assumir-se como um seguidor de referências.

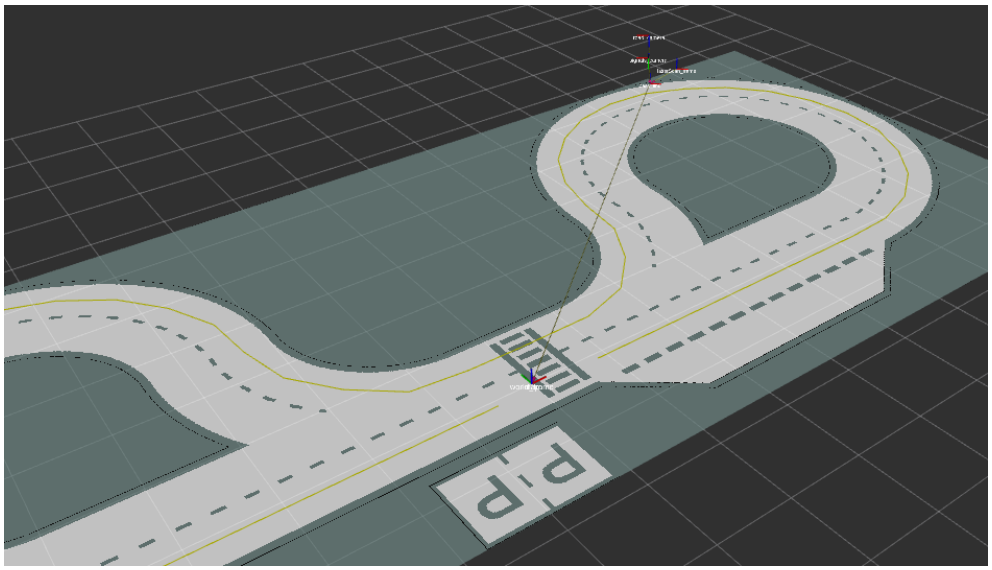


Figura 6.9: Path de referência.

Os controladores abaixo implementados seguem as referências do percurso definidas no espaço Cartesiano, diminuindo o erro relativo da *pose* do robô ao ponto referência (x_{ref}, y_{ref}), o que automaticamente corrige a orientação do robô no decorrer do percurso.

Os controladores são implementados com base no modelo de sistema referido em 6.23, recordando.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{6.23}$$

6.3.2 Path following - Controlador PID

O primeiro controlador implementado assenta na correção do erro da orientação do robô em relação ao ponto referência, por via de um controlador PID. A velocidade linear é assumida como constante $v \neq 0$.

Assumindo a posição ou estado do robô sendo $\mu = [x\ y\ \ \theta]^T$, o ângulo α representa a diferença angular entre a *pose* do robô e o ponto de referência do trajeto (x_{ref}, y_{ref}).

$$\alpha = \arctan\left(\frac{x_{ref} - x}{y_{ref} - y}\right) - \theta \tag{6.24}$$

A equação acima resulta contudo em condições indesejáveis para o controlo, com

ângulos a transitarem abruptamente em determinadas condições, o controlo angular deve restringir-se entre $[-\pi, \pi]$ evitando fenómenos indesejáveis de perda total de controlo sobre a plataforma.

O ângulo α é traduzido num erro (e) angular que define a variável de entrada para o controlador PID. Utilizando a função $arctan2$ a diferença é restrita a $[-\pi, \pi]$:

$$e = \arctan 2 \left(\frac{x_{ref} - x}{y_{ref} - y} \right) - \theta \tag{6.25}$$

A diferença com a atual orientação do robô pode ainda resultar no erro $-\pi \leq e \geq \pi$, pelo que a condição abaixo assegura a restrição desejada:

$$\text{Se } e \geq \pi \longrightarrow e = e + -2\pi$$
$$\text{Se } e \leq \pi \longrightarrow e = e + 2\pi$$

Este erro é assim a resultante referência para o controlador PID.

$$\omega(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{\partial e(t)}{\partial t} \tag{6.26}$$

Os parâmetros do PID (K_p, K_i e K_d) foram ajustados empiricamente via testes realizados no simulador implementado.

6.3.3 Trajectory Tracking - linearização por feedback

O sistema de *tracking* implementado adota o método de linearização por feedback e rastreamento assimptótico da trajetória, esta explicação é parte do estudo realizado e presente em [92] e [93].

O rastreamento assimptótico de uma trajetória carece da combinação de uma ação do comando *feedforward* e do controlo e minimização do erro via *close-loop*. O erro é obtido em função do estado e referência para *tracking*.

Seja (x, y) a posição do robô e (x_d, y_d) o ponto referência a ser atingido pelo robô, com $t \in [0, T]$, considerando o modelo da equação 6.23:

$$\theta = \arctan 2 \left(\frac{\dot{y}}{\dot{x}} \right) + k\pi, \qquad k = 0, 1 \tag{6.27}$$

Conforme referido na equação 6.25 o ângulo deve ser restrito entre $[-\pi\, \pi]$.

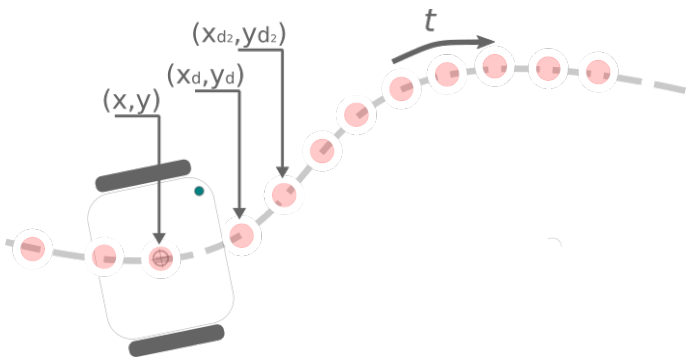


Figura 6.10: Trajetória gerada por pontos referências

O controlo *feedforward* obtém as acções de entrada do controlo considerando a referência pretendida (x_d, y_d) e modelo cinemático inverso, são estes:

$$v_d(t) = \pm\sqrt{\dot{x}_d^2(t) + \dot{y}_d^2(t)} \tag{6.28}$$

O sinal de $\pm v_d(t)$ determina o sentido do movimento do robô, sentido positivo no deslocamento em frente e negativo na inversão e deslocação para trás.

A equação 6.28 derivada em função do tempo define a velocidade angular $\omega_d(t)$

$$\omega_d(t) = \frac{\pm\ddot{y}_d(t)\dot{x}_t(t) - \ddot{y}_d(t)\dot{x}_t(t)}{\dot{x}_t(t) + \dot{y}_t(t)} \tag{6.29}$$

A condição de trajetória gerada em função das entradas *feedforward* $[v_d(t)\ \omega_d(t)]$ é apenas garantida para a condição de dupla derivada do trajeto em $[0, T]$. Neste trabalho a primeira e segunda derivada do ponto (x_d, y_d) é obtida pela diferença em relação ao tempo da posição do robô (x, y) e referência seguinte (x_{d2}, y_{d2}).

6.3.3.1 Controlo linear

O controlo via linearização tangencial para *traking* de uma trajetória pode expressar-se pelo erro entre o estado e a referência, no referencial do robô:

$$\begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_d - x \\ y_d - y \\ \theta_d - \theta \end{bmatrix} \tag{6.30}$$

A malha de controlo deve forçar a aproximação do robô à referencia, diminuindo o erro. Considerando o modelo cinemático 6.23 relacionado com a equação 6.30 resulta na nova equação da cinemática:

$$\begin{bmatrix} \dot{e_1} \\ \dot{e_2} \\ \dot{e_3} \end{bmatrix} = \begin{bmatrix} cos(e_3) & 0 \\ sin(e_3) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_d \\ \omega_d \end{bmatrix} + \begin{bmatrix} -1 & e2 \\ 0 & -e1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{6.31}$$

Resultando em:

$$v = v_d\, cos(e_3) - u_1 \tag{6.32}$$

$$\omega = \omega_d - u_2 \tag{6.33}$$

Em que $[u_1,\ u_2]$ são as entradas da realimentação em $close-loop$, v_d e w_d a velocidade linear e angular resultantes do controlo *feedforward*.

Reescrevendo a equação 6.31 em função das entradas de velocidade da malha *close-loop* $[u_1,\ u_2]$:

$$\begin{bmatrix} \dot{e_1} \\ \dot{e_2} \\ \dot{e_3} \end{bmatrix} = \begin{bmatrix} 0 & \omega_d & 0 \\ -\omega_d & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} + \begin{bmatrix} 0 \\ sin(e_3) \\ 0 \end{bmatrix} v_d + \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{6.34}$$

Linearizando a equação 6.34 para um ponto de operação em que $e_1 = e_2 = e_3 = 0$ e $u_1 = u_2 = 0$ resulta no modelo linear:

$$\Delta\dot{e} = \begin{bmatrix} 0 & \omega_d & 0 \\ -\omega_d & 0 & v_d \\ 0 & 0 & 0 \end{bmatrix} \Delta e + \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \Delta u \tag{6.35}$$

A equação 6.35 apresenta-se já definida no modelo espaço de estados na forma $\Delta\dot{q} = A\Delta q + B\Delta u$, a controlabilidade enunciada em 4.5 é obtido por:

$$rank(B,\ AB,\ A^2B) = 3 \tag{6.36}$$

O sistema é controlável apenas na condição de ambas as entradas $[v_d,\ w_d]$ serem não nulas.

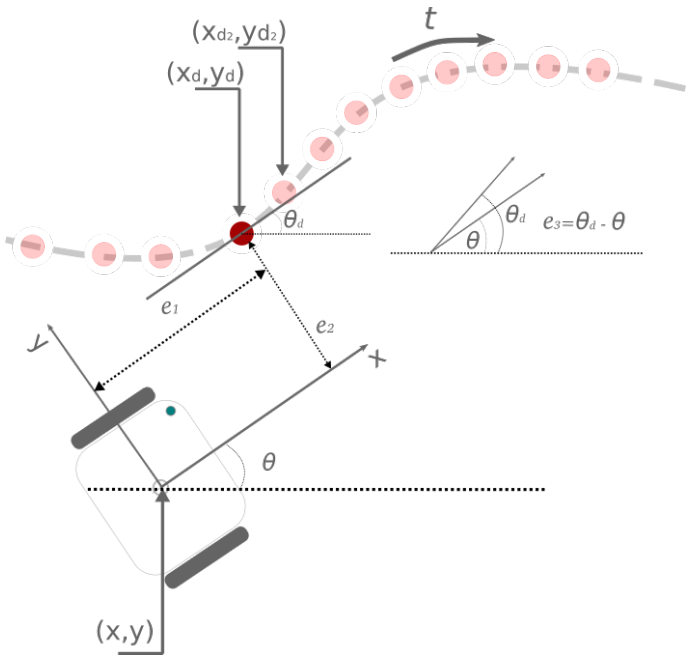


Figura 6.11: Erro do robô relativo ao ponto referência da trajetória

A lei de controlo linear em malha fechada, para minimização do erro representado na figura 6.11, é definida por:

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} -k1 & 0 & 0 \\ 0 & -sign(v_d)k2 & -k3 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} \tag{6.37}$$

Os ganhos adequados ao controlador podem ser determinados pela comparação do polinómio característico real e desejado, o polinómio assume a seguinte forma:

$$(s + 2\zeta\omega_n)(s^2 + 2\zeta\omega_n s + \omega_n^2) \qquad \zeta, \omega_n > 0, \tag{6.38}$$

Em que ζ representa um coeficiente de amortecimento desejado $\in$ (0,1) e ω_n a frequência característica, parâmetros a serem definidos.

O polinómio característico *close-loop* da realimentação do estado da equação 6.37 é:

$$det(sI - A + BK) = s^3 + (k1 + k3)s^2 + (k1k3 + k2\,v_d + w_d^2)s + k1k2\,v_d + k3\omega^2 \tag{6.39}$$

Igualando coeficientes do mesmo expoente de s da equação 6.38 e 6.39 resulta:

$$\begin{aligned} k1 + k3 &= 4\zeta\omega_n \\ k1k3 + k2\,v_d + \omega_d^2 &= 4\zeta^2\omega_n^2 + \omega_n^2 \\ k1k2\,v_d + k3\,\omega_d^2 &= 2\zeta\omega_n^3 \end{aligned} \tag{6.40}$$

Resolvendo temos que :

$$\mathrm{k1} = \mathrm{k3} = 2\zeta\omega_n, \qquad k2 = \tfrac{\omega_n^2+\omega_d^2}{|v_d|} \tag{6.41}$$

O ω_n deve assumir um valor superior à velocidade angular ω_d e tendo em conta que o $k2 \to \infty$ quando $v_d \to 0$, é por conveniência restrito o aumento de k2 pela introdução de um ganho proporcional, $k2 = g|v_d(t)|$. Desta forma a frequência característica assume a seguinte igualdade:

$$\omega_n(t) = \sqrt{\omega_d^2(t) + g\,v_d(t)^2}$$

Resultando nos ganhos:

$$\begin{aligned} k1 = k3 &= 2\zeta\sqrt{\omega_n^2(t) + g\,v_d^2(t)} \\ k2 &= g\,|v_d(t)| \qquad g > 0, \end{aligned} \tag{6.42}$$

Os ganhos anulam-se quando as velocidades linear e angular tendem para zero, neste ponto o robô não é controlável. Se a opção dos ganhos for tida em função das equações 6.41, o controlador é não linear e variante no tempo com pólos bem definidos e constantes. Contudo, o sistema permanece sendo variante no tempo pelo que a estabilidade local não é garantida.

O controlador não linear variante no tempo, em função da entradas do controlo é definido por:

$$\begin{aligned} v &= v_d\,cos(\theta_d - \theta) + k1\,[cos(\theta)(x_d - x) + sin(\theta)(y_d - y)] \\ \omega &= \omega_d + k2\,sign(v_d)\,[cos(\theta)(x_d - x) + sin(\theta)(y_d - y)] + k3\,[\omega_d - \omega] \end{aligned} \tag{6.43}$$

6.4 Collision Avoidance

Conforme explicado no capítulo 2 os controladores são assumidos como *behaviors*, a transição entre *behaviors* depende do mundo percecionado pelo robô. O robô desconhece fundamentalmente o mundo e reage conforme programado. Esta abordagem minimiza a necessidade de planeamento prévio e é adaptável a cenários dinâmicos, conforme ocorre na prova ADC com a colocação de obstáculos em posições não previamente conhecidas do percurso.

Esta secção descreve o *collision avoidance* implementado.

6.4.1 Collision Avoidance - Implementação

O robô percorre o percurso definido pelo mapa e na presença de um obstáculo, transita para o *behavior collision avoidance*, retomando o *path following* ou *path tracking* após contornar o obstáculo.

Para identificação de obstáculos na trajetória do robô foi adicionado um sensor à plataforma, o LiDAR Hokuyo.

A detecção de obstáculo foi definida partindo da dimensão mínima dos obstáculos que se pretendem identificar e distância máxima do sensor (parte frontal do robô) ao obstáculo definido para detecção. A identificação é assim realizada numa condição mínima de um obstáculo de $obs_{max} = 15\,cm$ a uma distância máxima de $l_{max} = 2\,m$, considerando obstáculos numa abertura angular restrita a $\pm 20^{\circ}$ relativa ao referência x do robô.

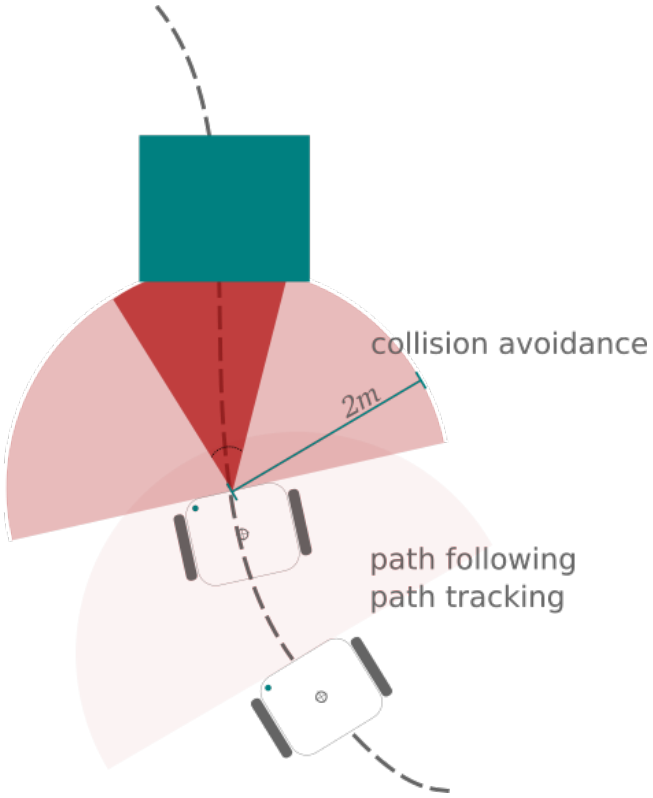


Figura 6.12: Transição de *behaviour*.

Estas condições definem por aproximação a identificação de obstáculo com abertura angular $\phi = \arctan(\frac{0.15}{2}) = 4.29°$, que permite determinar o número de lasers mínimo consecutivos com retorno inferior a l_{max}, $num_{lasers} = \phi / resolução_{laser}$.

Na condição de obstáculo identificado o laser com distância mínima (l_{min}) é referência para os passos seguintes. Informação da distância e id do feixe laser $[1, 2, \ldots n]$ é obtida na leitura do sensor, sendo o ângulo do referencial do robô ao obstáculo:

$$\alpha_{\text{obs}} = \phi_{\text{mínimo}} + \phi_{incremento} * id_{l_{min}}$$

O obstáculo fica assim definido em função do robô por via de coordenadas polares $[l_{obs}\ \alpha_{\text{obs}}]$.

A comutação desajustada entre *behaviors* é um comportamento indesejável e pode inclusivamente derivar numa condição de Zeno, onde o sistema realiza um número infinito de transições num determinado tempo finito [94]. Aquando da identificação do obstáculo, um novo *behavior* é induzido de forma a que o obstáculo seja contornado, o robô não se

limita a uma manobra evasiva, desvia-se e persegue o obstáculo até determinada condição de transição para outro *behavior*.

Este comportamento é simplesmente alcançado por um desvio angular ($\beta = \pm\pi/2$) relativo ao obstáculo (α_{obs}), mantendo esta diferença angular, o robô contorna o obstáculo a determinada distância. O sentido pelo qual é contornado o obstáculo ($\pi/2$ ou $-\pi/2$) é definido com base na identificação da faixa a ser percorrida pelo robô.

De forma a suavizar a nova trajetória gerada, a diferença angular β foi estabelecida em função da distância ao obstáculo, utilizando a função exponencial.

$$\beta = \frac{\frac{\pi}{2}}{e^{a*l_{obs}}} \tag{6.44}$$

A parâmetro de ajuste a define a atenuação da curva, a figura 6.13 demonstra curvas para diferentes valores de a.

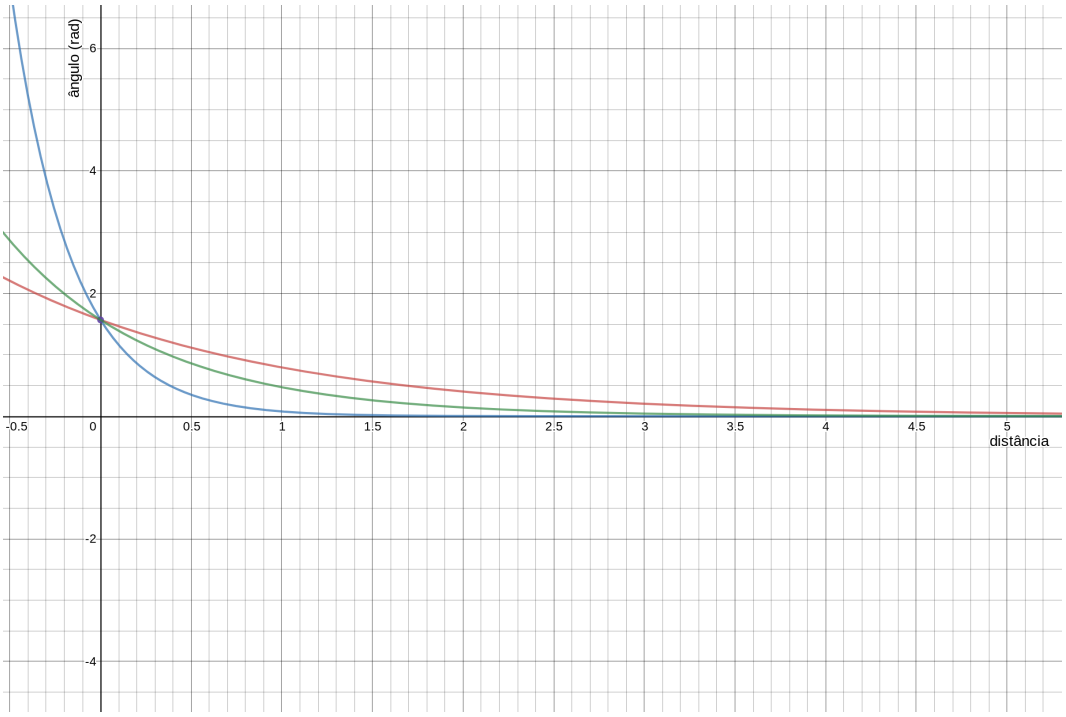


Figura 6.13: Função exponencial aplicada ao cálculo de α_{obs}.

Impõem-se uma condição de saída deste *behavior*, de outra forma e uma vez identificado um obstáculo, o mesmo seria contornado infinitamente.

O ângulo entre o robô e a referência do *path* (α) é adquirido no decorrer da manobra evasiva do obstáculo, através da equação 6.24. A informação obtida pelo sensor LiDAR

é avaliada numa abertura angular ($\pm\gamma$) em função do ângulo (α).

$$id^k = \frac{(\alpha \pm \gamma) - \phi_{\text{mínimo}}}{\phi_{incremento}}$$

Na ausência de obstáculo na abertura angular definida, a condição de transição de *behavior* ocorre e o robô prossegue percorrendo o percurso abandonando o *behavior obstacle avoidance*.

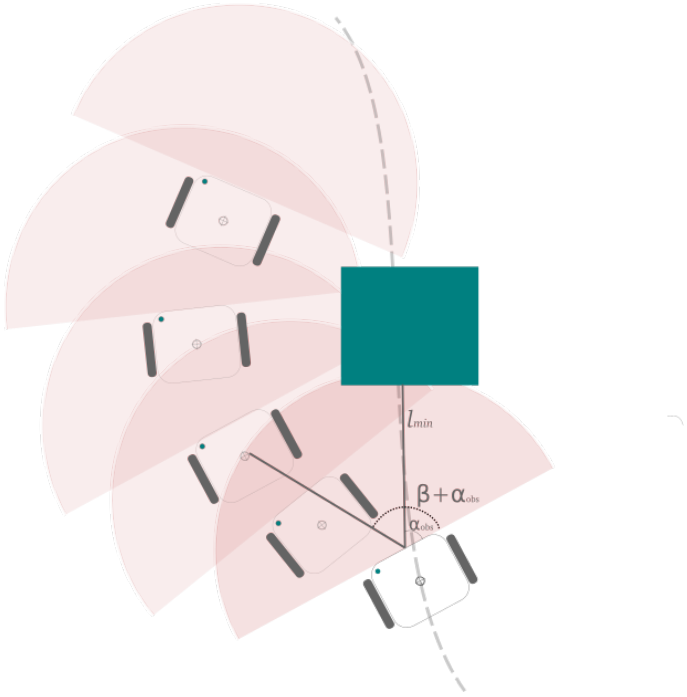


Figura 6.14: Representação do momento da transição de *behavior*.

Esta metodologia permite adotar um sistema *collision avoidance* sem conhecimento prévio do posicionamento, dimensão ou características do obstáculo, podendo o mesmo apresentar forma circular, convexa ou não convexa.

Capítulo 7

Resultados

Este capítulo apresenta os resultados alcançados nas diferentes frentes abordadas desta dissertação. Detalha a solução de simulação desenvolvida, focando aspetos mais relevantes de conceção e funcionalidade. O sistema de Localização dá continuidade, exposto de forma particionada pelos diferentes desenvolvimentos e resultados que permitiram a sua implementação, culminado com análise da *performance* do filtro EKF.

Neste capítulo também são expostos os resultados alcançados no *path following* e *path tracking*. O capítulo é finalizado com os resultados do *collision avoidance*, apresentando exemplo de manobras evasivas no contorno de obstáculos.

7.1 Simulador

O objetivo de modelação e funcionalidade como simulador de todas as partes constituintes da prova ADC-FNR foi atingido, conforme já previamente ilustrado. O cenário foi também disponibilizado via package ROS *map_server* como mapa com representação no Rviz. A ferramenta Rviz do ROS servirá de suporte à apresentação dos resultados alcançados nesta dissertação.

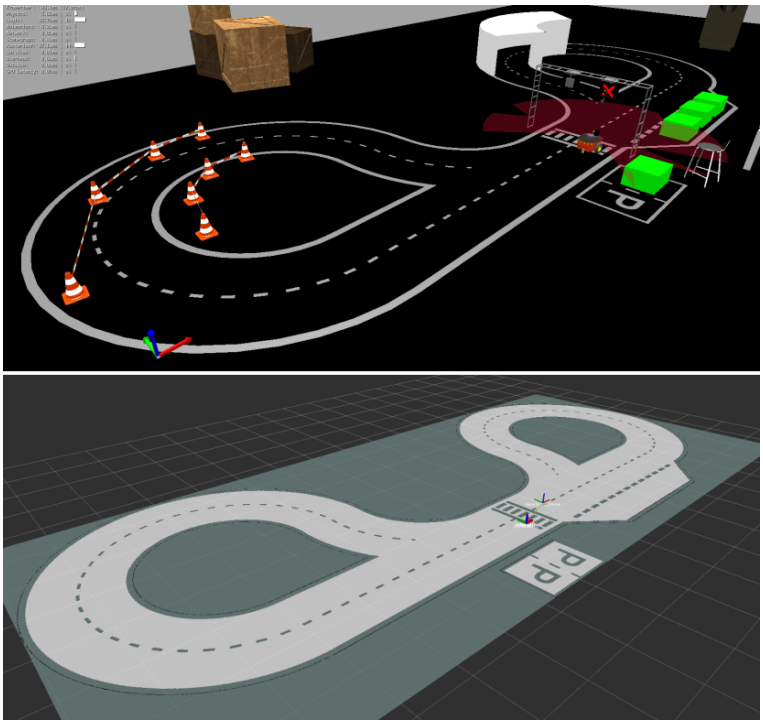


Figura 7.1: Ambiente de simulação da ADC-FNR e mapa Rviz.

A dinâmica introduzida através do Logic Bricks permite a manipulação dos diferentes sinais e objetos da prova (túnel, obstáculo e zona de obras) em tempo de execução do simulador, permitindo célere interação na configuração do cenário da prova aliado à missão pretendida. A figura 7.2 ilustra alguns destes comportamentos.

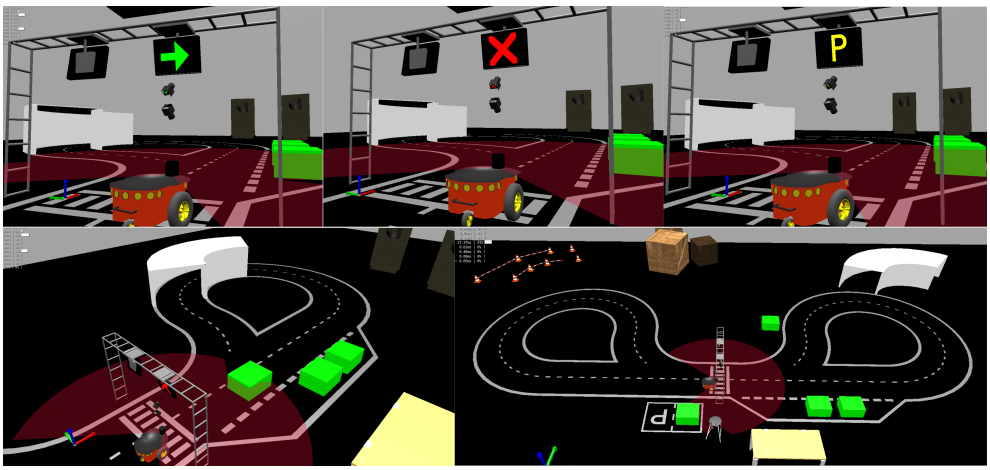


Figura 7.2: Dinâmica de configuração do cenário da ADC-FNR.

A inclusão de física, objetos externos, texturas e condições de iluminação ajustável, conferiram ao simulador grande aproximação à realidade, promovendo robustez nas soluções desenvolvidas. A confrontação aquando da simulação para cenários e condições não previstas, promove o refinamento das soluções.

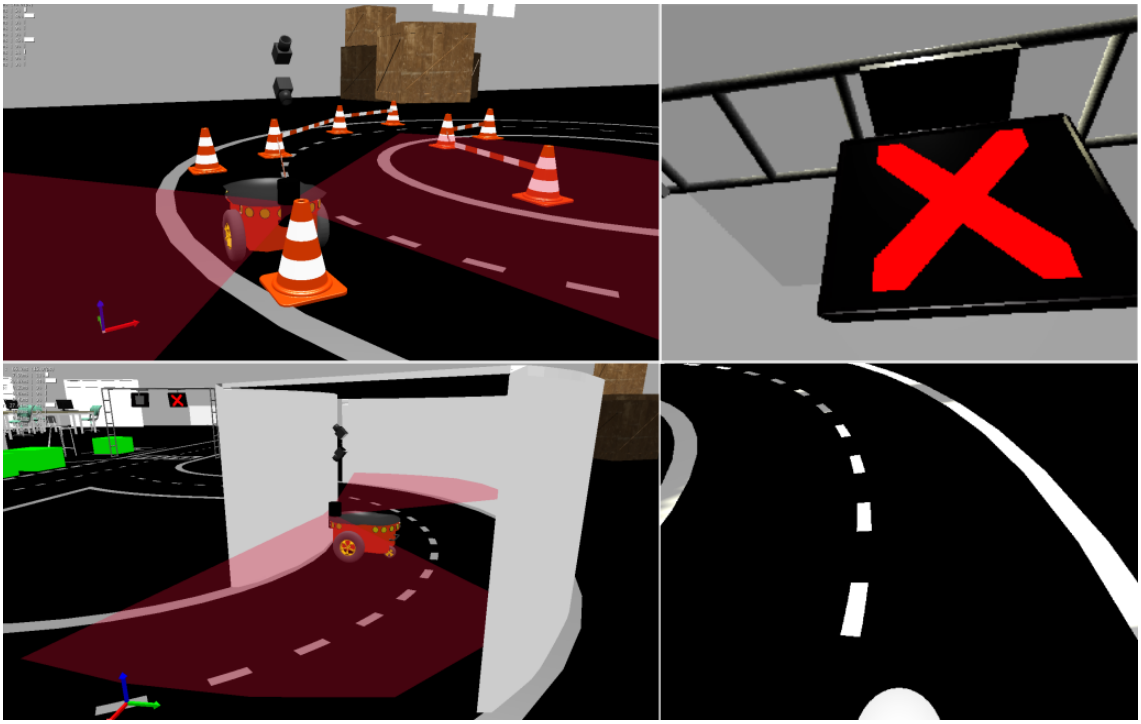


Figura 7.3: Condições de simulação, efeitos da iluminação e colisão com obstáculos.

7.2 Localização

O sistema global representado via *rqt_ graph* é ilustrado na figura 7.4 e resume a estrutura e dados necessários ao sistema de localização.

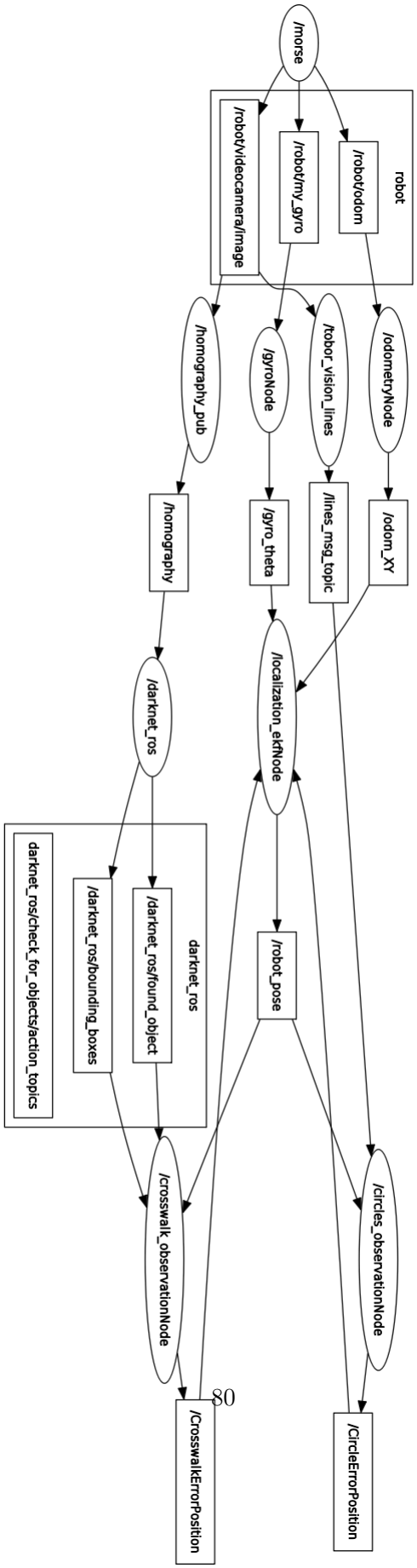


Figura 7.4: Diagrama do sistema de localização.

Numa primeira camada estão presentes os sensores, a odometria, *gyro* e câmara de visão. A odometria e *gyro* são combinados fornecendo informação do deslocamento linear e angular do robô, que alimenta o modelo do sistema no *predict* do filtro. A câmara adquire imagem da pista em perspectiva conforme já ilustrado na figura 6.6, convertida para imagem IPM que alimenta a identificação de faixas de rodagem e passadeira.

As observações (distâncias $(x_r\,, y_r)$ do robô às *landmarks* no referencial do robô) obtidas pelos sistemas de perceção, são publicadas nos tópicos */CircleErrorObservation* e */CrosswalkErrorObservation.*

A informação é combinada no *node /localization_ ekfNode*, onde o algoritmo EKF está implementado, o qual publica o estado estimado na localização do robô.

7.2.1 Observação das Landmarks

A figura 7.5 apresenta os resultados da simulação de diferentes aproximações do robô Pioneer-3dx à passadeira, com probabilidade associada à presença de passadeira presente na imagem. É passível de constatar total ausência de falsos positivos e consistência na identificação em diferentes testes realizados. Esta simulação permitiu definir *threshold* de identificação válido, o vértice do canto inferior direito à marcha do robô é tido como landmarck relacionado ao robô por $(x_r\,, y_r)$ e projetado para o mundo conforme equação 6.13. O YOLO-V2 é exigente computacionalmente, o sistema foi contudo capaz de proceder à identificação conforme figura 7.5 com *frame rate* de 20 fps, utilizando computador Intel Core i7-7700HQ - CPU 2,8 GHz (x8) e GPU 1050TI (768 CUDA cores).

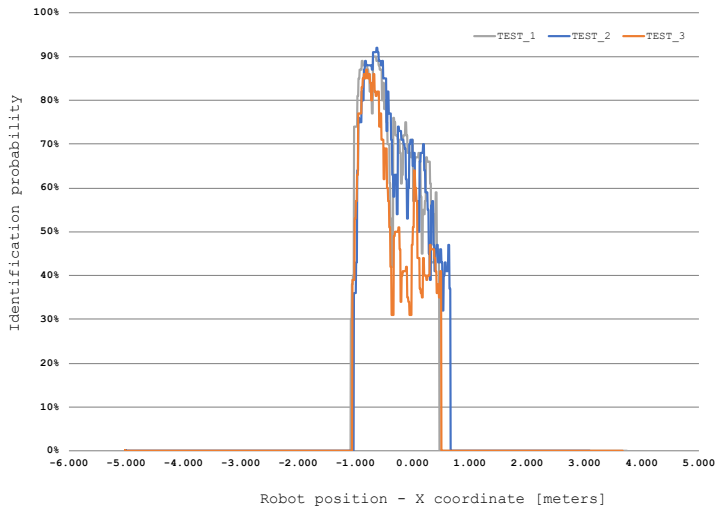


Figura 7.5: Análise performance identificação passadeira pelo YOLO-V2

A identificação de circunferências e projeção de landmarck foi alcançada conforme ilustrado na figura 7.6, que representa visualmente através do *package maker* do ROS a circunferência projetada pela curvatura identificada na faixa lateral direita da pista.

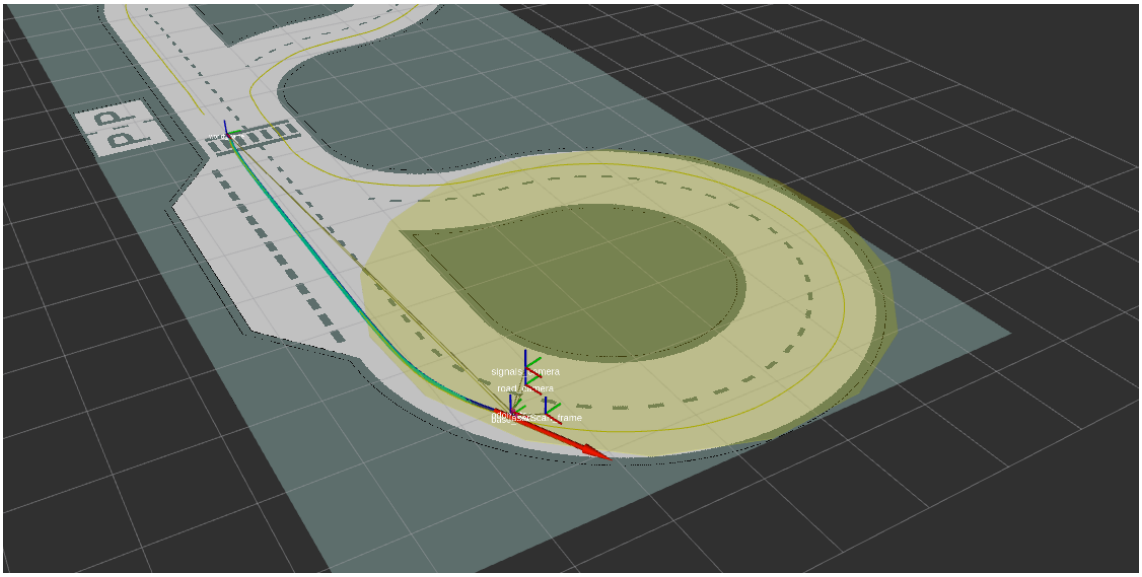


Figura 7.6: Observação de Landmark.

O processo apresenta contudo incerteza significativa não passível de refinamento no decorrer da dissertação. Alternativamente e para avaliação do desempenho do sistema de localização, as observações serão simuladas. As observações são publicadas em tópicos ROS por condição de proximidade do robô à *landmark*, de forma análoga à realidade e em igual formato ao assumido na explicação prévia.

A figura 7.7 apresenta o *path* real do robô numa volta ao percurso com identificação de cada *landmark* adquirida no referencial do robô, projetada para o referencial do mundo.

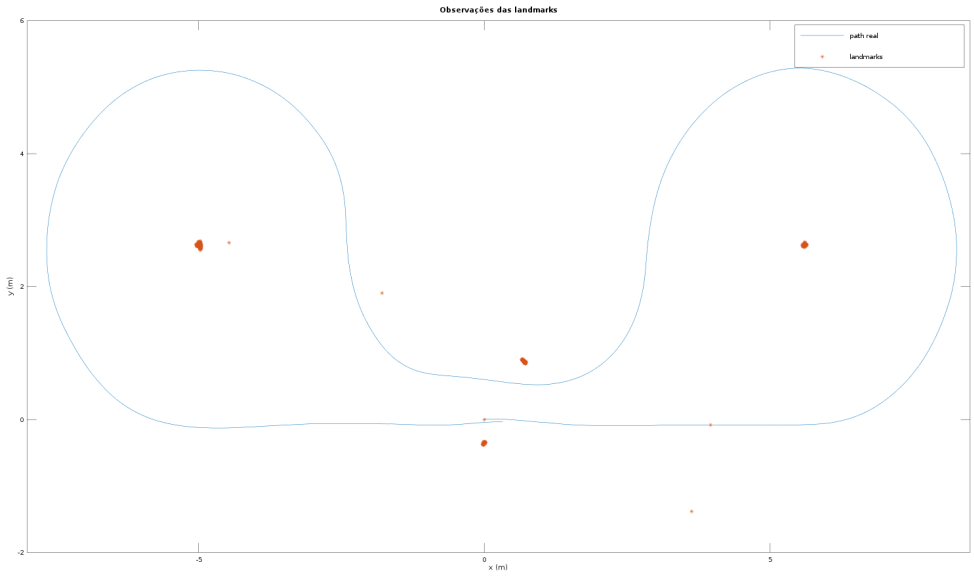


Figura 7.7: Observações e path real do robô

São visíveis na figura 7.7 algumas falsas observações não coincidentes com nenhuma *landmark* e erro significativo de algumas observações, *outliers* a serem descartados no *update* do filtro por definição de uma *gate* de aceitação.

Para maior aderência à realidade, foi adicionado ruído Gaussiano de média nula e desvio padrão $\sigma = 5\,cm$ às observações. A figura 7.8 ilustra a projeção no mundo das novas observações numa volta ao percurso.

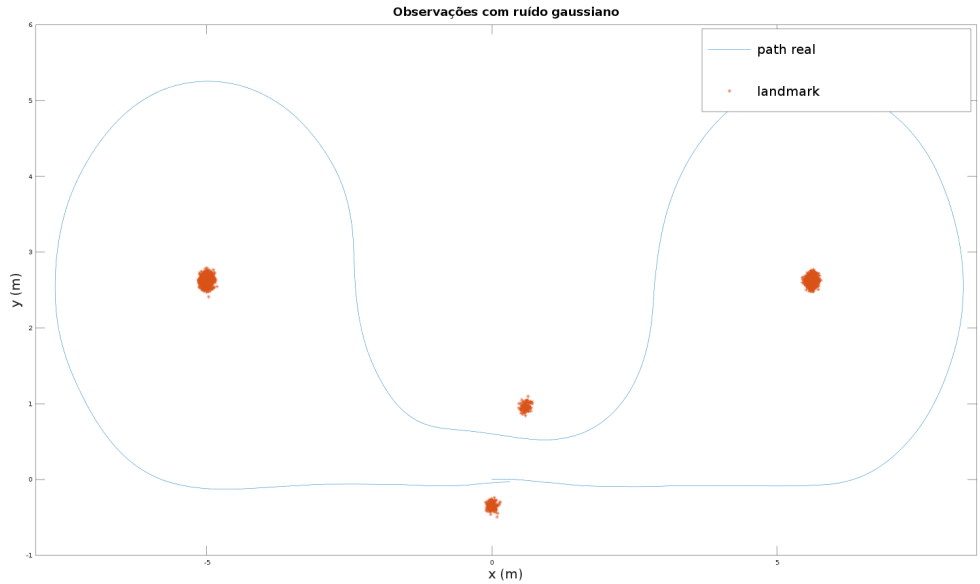


Figura 7.8: Observações com ruído gaussiano e path real do robô

As observações são assim descritas pela equação 7.1:

$$\begin{bmatrix} x_m(k) \\ y_m(k) \end{bmatrix} = \begin{bmatrix} x(k) \\ y(k) \end{bmatrix} + \begin{bmatrix} cos(\theta(k)) & -sin(\theta(k)) \\ sin(\theta(k)) & cos(\theta(k)) \end{bmatrix} \begin{bmatrix} x_r(k) \\ y_r(k) \end{bmatrix} + \begin{bmatrix} r_x^i(k) \\ r_y^i(k) \end{bmatrix} \tag{7.1}$$

Em que $r_k^i(k) = [r_x^i(k), r_y^i(k)]^T$ descreve o ruído introduzido no modelo por erros e incerteza na observação e $\overline{\mu} = [x(k), y(k), \theta(k)]^T$ o estado real do robô passível de se obter neste exemplo de simulação.

7.2.2 Modelo do Sistema

De forma a avaliar a aproximação do modelo há realidade e melhor percepcionar os erros do sistema e incerteza do modelo, o estado previsto é avaliado comparativamente ao verdadeira estado, algo raramente disponível em condições reais.

A figura 7.9 apresenta o *path* resultante de uma volta ao percurso com estado previsto pelo modelo do sistema representado a azul e *path* real a verde.

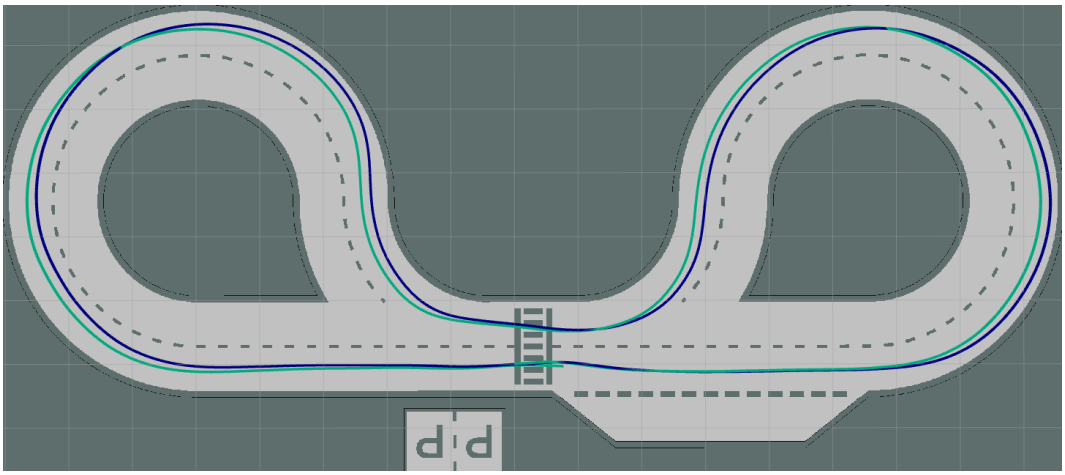


Figura 7.9: Path do modelo do sistema.

A figura 7.10 representa o erro sobre cada coordenada (x e y) entre o *path* real e previsto.

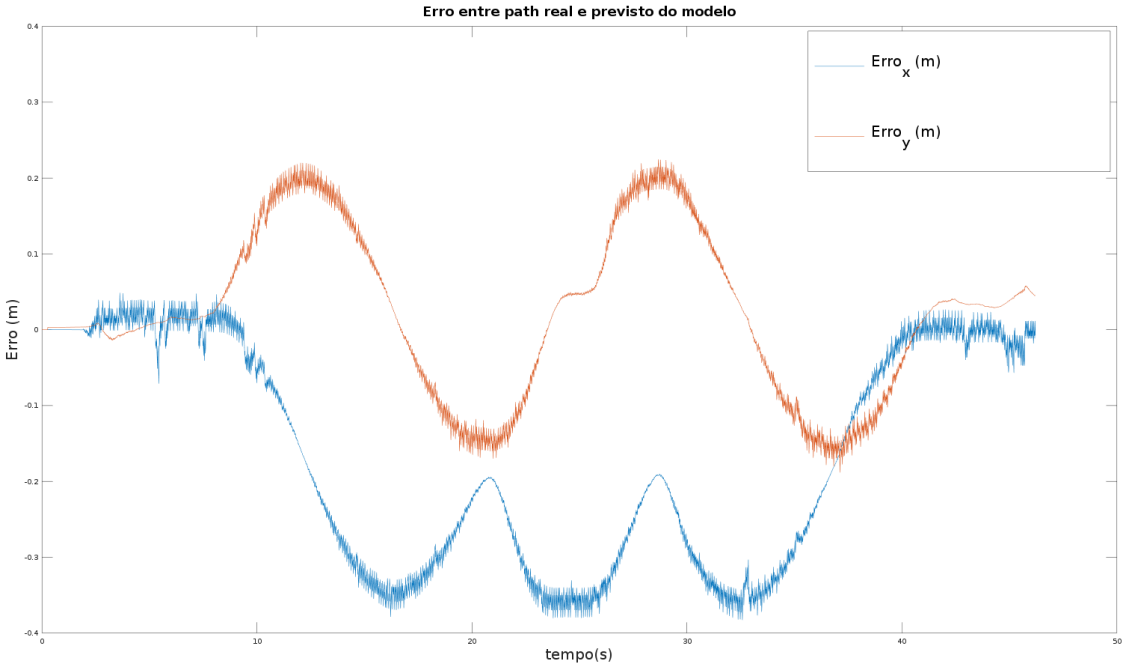


Figura 7.10: Erro entre *path* real e previsto pelo modelo de cada coordenada.

O *path* resultante é consistente em todo o percurso embora apresente erro com momentos de incerteza elevada no posicionamento. O erro predominante ocorre aquando

diferenças de orientações do robô, por desvios laterais com impato significativo no movimento longitudinal (sobre x). Na pior condição verificada, o estado previsto difere 34 cm em x e 20 cm sobre y relativamente ao *path* real.

A previsão do modelo retrata condições reais de incerteza no posicionamento do robô, pela presença de erros sistemáticos frequentes tais como a diferença no raio da rodas, incerteza na distância entre rodas e deslizamento.

7.2.3 Inicialização e análise da performance do filtro EKF

Na ausência de observação, a previsão é assumida como o estado estimado. Se novas observações surgirem publicadas nos tópicos */CircleErrorObservation* ou */CrosswalkErrorObservation*, são associadas a uma *landmarck* do mapa e o estado é estimado pelo *update* do filtro. O filtro é inicializado com a posição inicial do robô a cada início de prova, estado estimado $\mu(0\,|\,0) = [0, 0, 0]^T$ e covariância $\Sigma(0\,|\,0)$ com diagonal das variâncias associadas ao ruído do sistema:

$$\sqrt{\Sigma(0\,|\,0)} = \begin{bmatrix} 0.02 & 0 & 0.0 \\ 0.0 & 0.02 & 0.0 \\ 0.0 & 0 & 0.05 \end{bmatrix} \tag{7.2}$$

Esta inicialização é crucial para a convergência do filtro.

A sequência da inovação ($v(k)$) é um indicador da performance e consistência do filtro, $v(k)$ a diferença entre a observação $z(k)$ e a observação prevista $h_k(\overline{\mu}(k)$:

$$v(k) = z(k) - h_k(\overline{\mu}(k))) \tag{7.3}$$

O filtro foi sintonizado com base na análise da sequência de inovação, os erros introduzidos foram ajustados garantido mais de 95% das observações enquadradas na *gate* de aceitação com desvio padrão $2\sigma = 10cm$. A figura 7.11 e 7.12 representam a inovação para cada variável de estado observada, numa volta ao percurso da prova ADC.

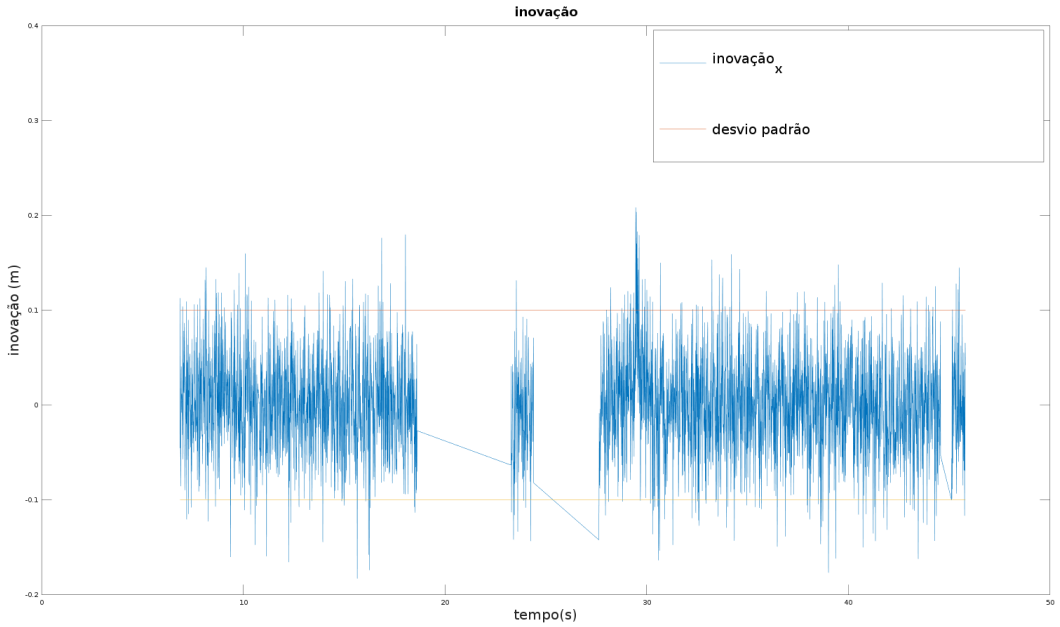


Figura 7.11: Sequência de inovação e desvio padrão em x

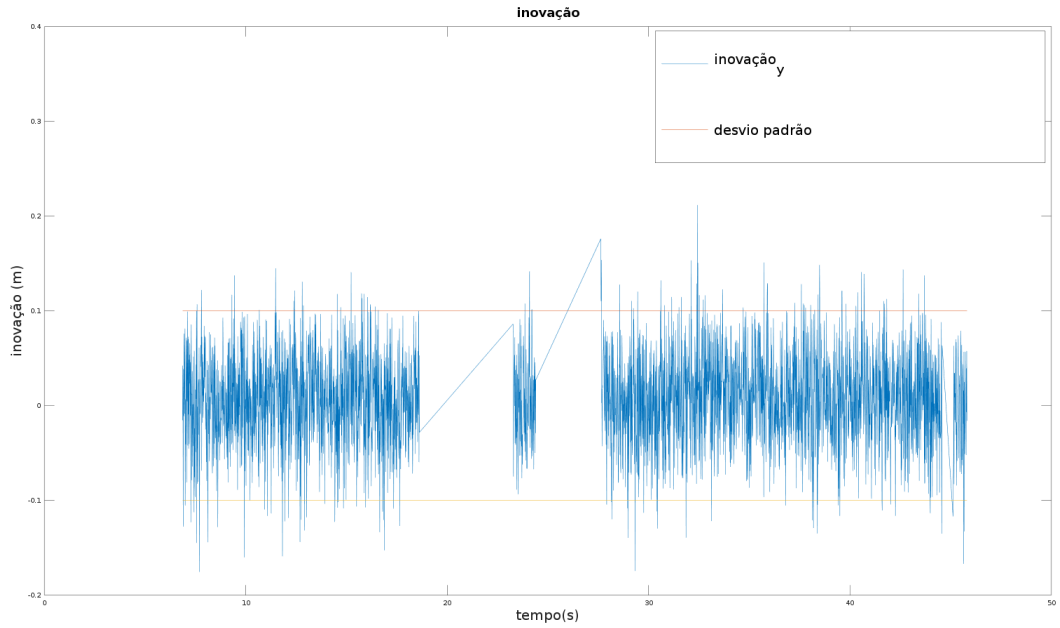


Figura 7.12: Sequência de inovação e desvio padrão em y

Conforme é passível de ser observado na figura 7.11 e 7.12, existem quatro momentos no percurso de uma volta à pista em que o robô não tem acesso a observações, sendo o

estado estimado atualizado com base na previsão do estado.

Considerando o início de prova junto à passadeira e sentido CCW do percurso, os primeiros quatro metros percorridos correspondem aos 7 s iniciais do gráfico. O segundo momento ocorre na transição entre a observação da primeira circunferência e aproximação à observação da passadeira, intervalo 18.6 s a 23.2 s. Ao transpor a passadeira e até observação da *landmarck* da segunda circunferência, intervalo de 24.4 a 27.6 s. Por fim, os instantes prévios à identificação da passadeira no final de uma volta ao trajeto, instante entre 44.5 s e 45.1 s.

Para consistência do filtro a sequência da inovação deve assumir média nula, sem enviesamento e representar ruído branco. A magnitude da sequência de inovação está claramente enquadrada na variância 2σ para ambos casos, tendo inclusive a maioria das inovações enquadramento entre 1σ o que comprova o correto *tuning* do filtro. O comportamento do resíduo é igualmente branco e consistente sem enviesamento.

A figura 7.14 ilustra uma volta ao percurso da prova ADC-FNR. Embora praticamente imperceptível devido a sobreposição, a estimação obtida pelo filtro de Kalman Estendido acompanha em grande parte do percurso o *path* real realizado pelo robô.

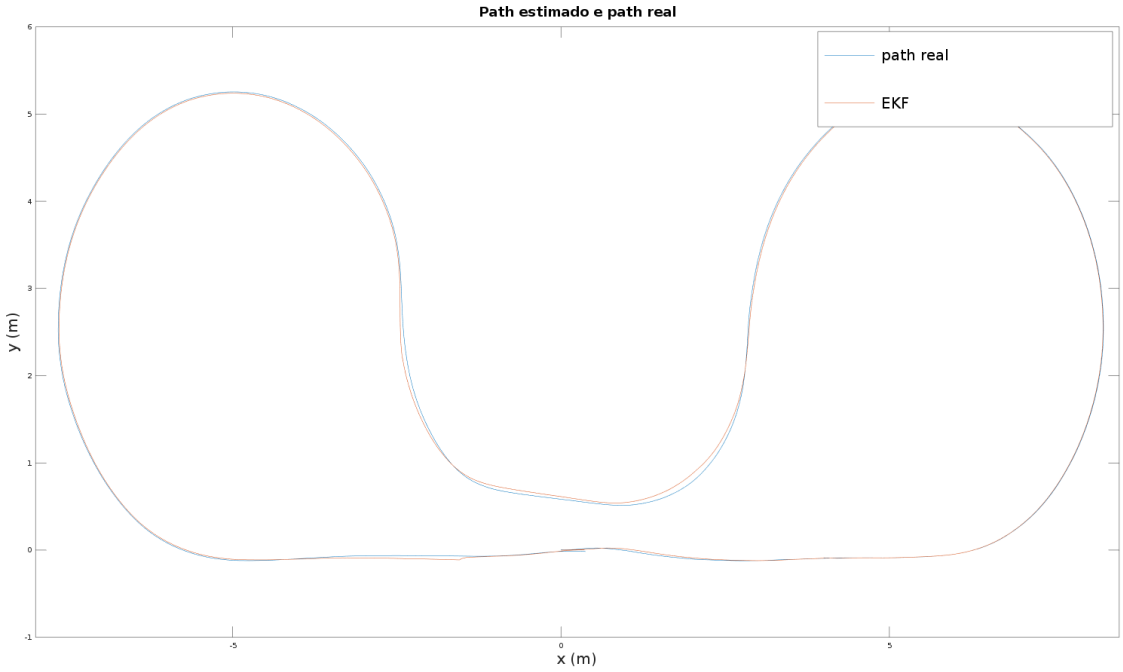


Figura 7.13: *Path* real e *Path* resultante da estimação do Filtro EKF

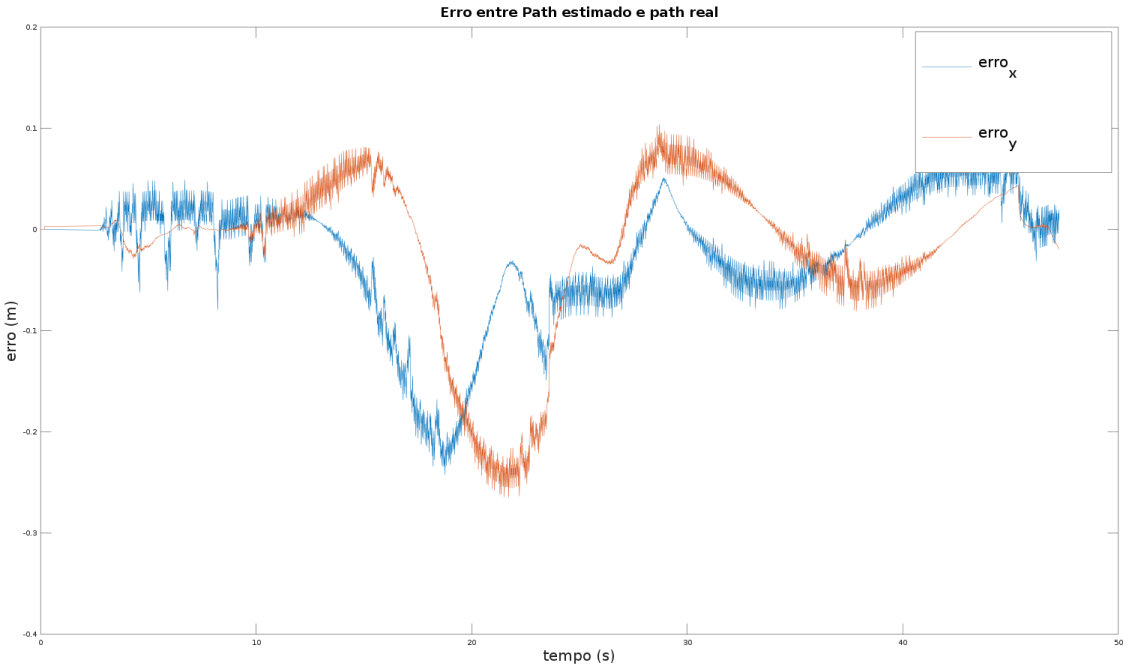


Figura 7.14: Erro da estimação face ao *path* real

O resultado da estimação acompanha de forma inequívoca o *path* real confirmando a *performance* do filtro, existindo contudo momentos de maior incerteza. Os intervalos previamente descritos com atualização do estado pelo *predict* demarcam-se do restante trajeto, é visível o erro relativo ao *path* real com consequente aumento na incerteza do estado pela ausência de observações para correção. Em simultâneo ocorre outro efeito penalizador, as curvas mais acentuadas do trajeto estão presentes em momentos que o robô não observa nenhuma *landmark*. Conforme já avaliado na análise do modelo do sistema, o erro da previsão aumenta aquando alterações de orientação do sistema.

Na fase final do percurso é igualmente possível constatar o aumento gradual do desvio lateral da estimação, com atualização pouco suave do estado sensivelmente a $x = -1.5\,m$ da passadeira, momento em que o robô corrige o estado pela observação desta *landmark*.

7.3 Path following e Trajectory Tracking

O controlador PID foi definido empiricamente e facilmente ajustado por via de testes com recurso ao simulador, avaliando estabilidade e resposta para diferentes velocidades lineares do robô. A definição de ganhos resultou em $Kp = 1.6$, $Ki = 0.01$ e $Kd = 0.4$.

A controlo foi testado para diferentes condições de velocidade linear, registando-se o

tempo realizado numa volta ao percurso. As figuras de 7.15 a 7.18 ilustram o caminho real executado pelo robô comparativamente ao percurso definido como referência a percorrer (*path* definido a amarelo na imagem).

- Definido $v_{linear} = 0.5\,m/s$ com $t_{volta} = 86.9\,s$

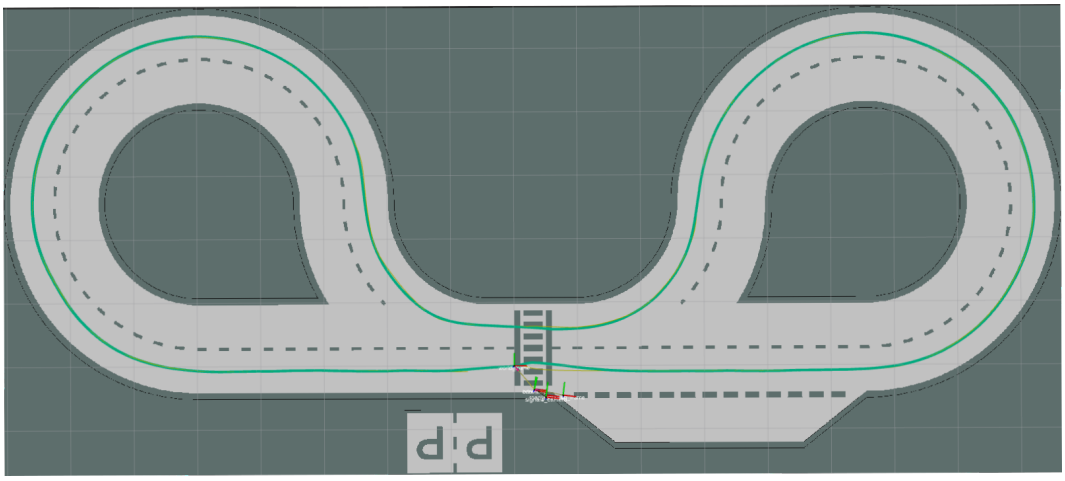


Figura 7.15: Controlador PID com v = 0.5 m/s e tempo de volta 86.9 s.

- Definido $v_{linear} = 1.0\,m/s$ com $t_{volta} = 44.2\,s$

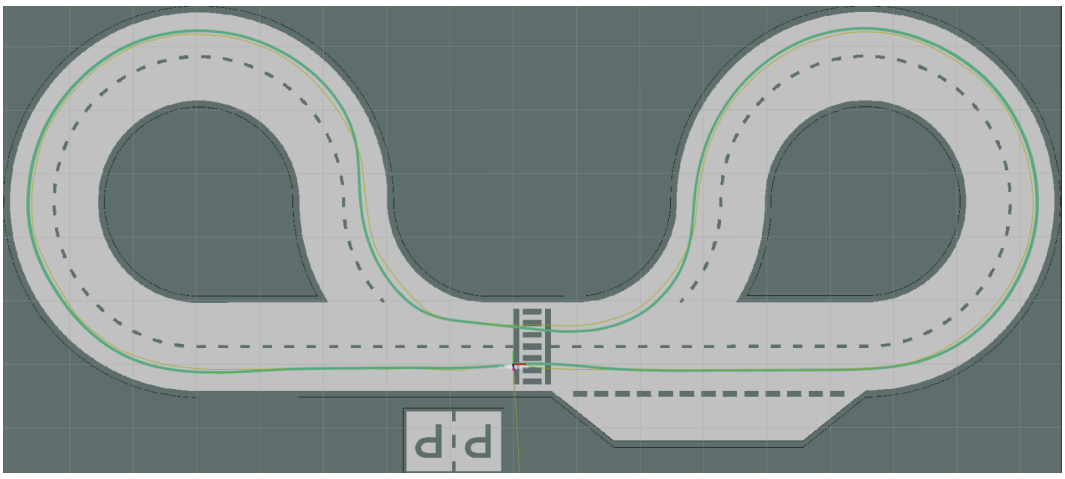


Figura 7.16: Controlador PID com v = 1.0 m/s e tempo de volta 44.2 s.

- Definido $v_{linear} = 1.5\,m/s$ com $t_{volta} = 30.4\,s$

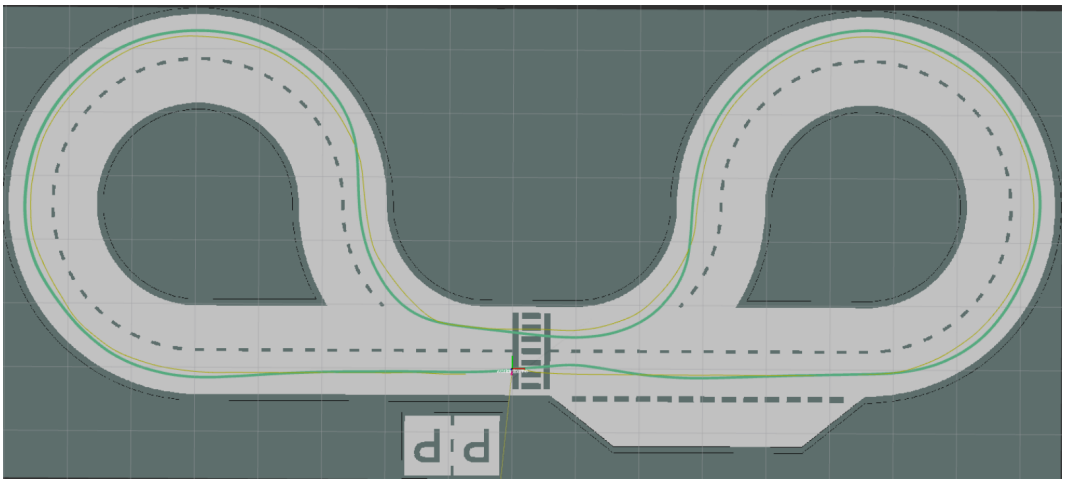


Figura 7.17: Controlador PID com v = 1.5 m/s e tempo de volta 30.4 s.

- Definido $v_{linear} = 2.0\,m/s$ com $t_{volta} = 23.5\,s$

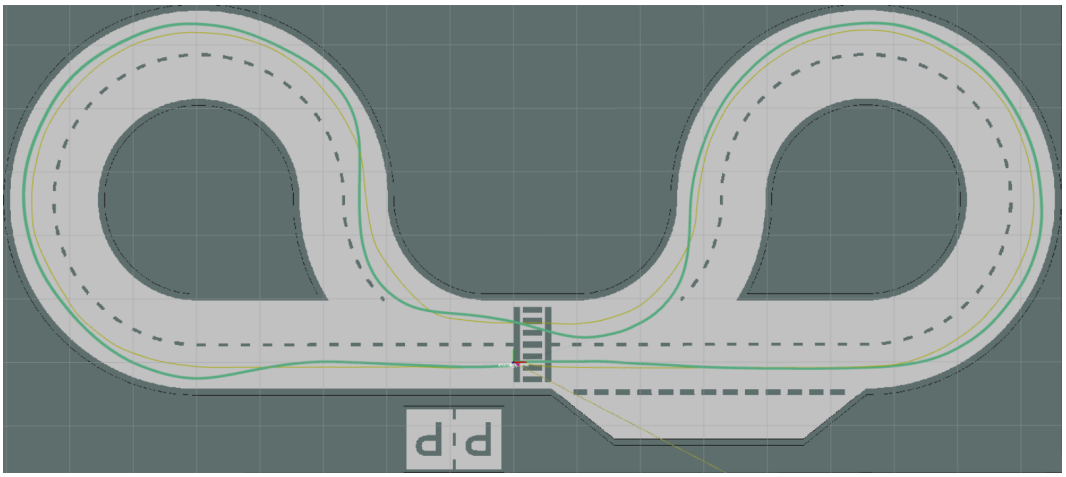


Figura 7.18: Controlador PID com v = 2.0 m/s e tempo de volta 23.5 s.

O comportamento é estável e bem definido para velocidades até $1.5\,m/s$. Embora com resposta atempada e sem perda do controlo sob a plataforma, velocidades acima deste limiar apresentavam grande desvio face à trajetória definida e respostas bruscas pouco susceptíveis de serem aplicadas na realidade.

O controlador com linearização por *feedback* revelou ser de sintonização complexa. Foi contudo possível estabelecer uma relação de ganhos estável, que embora resulte num ligeiro desvio face ao trajeto real, goza de variações de velocidade alternando

comportamento em reta e curva. A figura 7.19 apresenta o resultado alcançado.

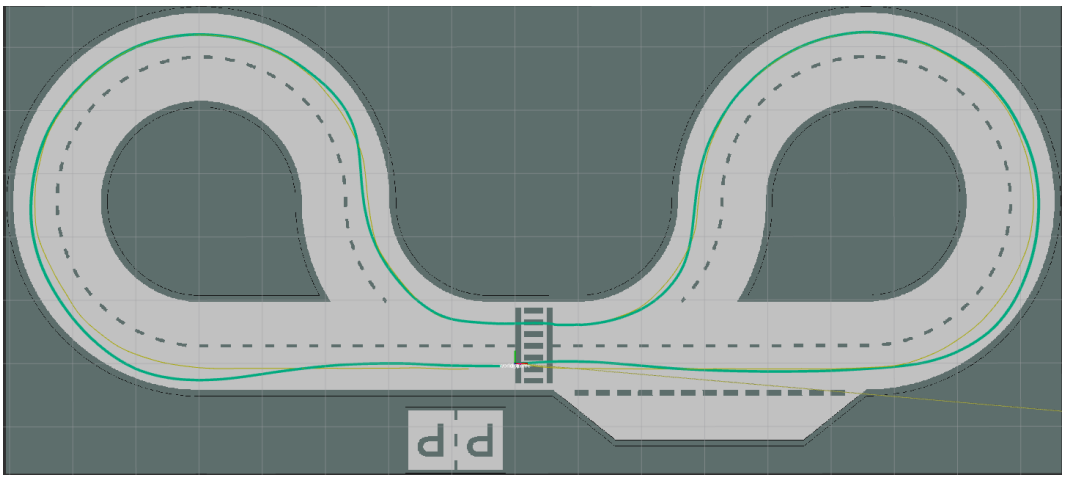


Figura 7.19: Controlador linearização por *feedback*, velocidade variável com tempo por volta 50.3 s.

7.4 Collision Avoidance

Conforme exposto na secção 6.4, o *collision avoidance* actua no controlo angular do robô definindo referência através da equação 7.4:

$$\beta = \alpha_{obs} + \frac{\pi}{2} * \frac{1}{e^{a*l_{obs}}} \tag{7.4}$$

Em que α_{obs} e l_{obs} representam respectivamente o ângulo e distância do obstáculo relativo ao robô, informação adquirida através do sensor laser Hokuyo. O parâmetro a traduz a inclinação da curva exponencial na aproximação ao obstáculo, foi definido após algumas experiências como 0.9.

O *collision avoidance* comporta-se como um *behavior* independente com função única de evitar e contornar os obstáculos. A figura 7.20 reforça esta noção estando em falta condição de transição de saída deste *behavior*, originando o comportamento ilustrado.

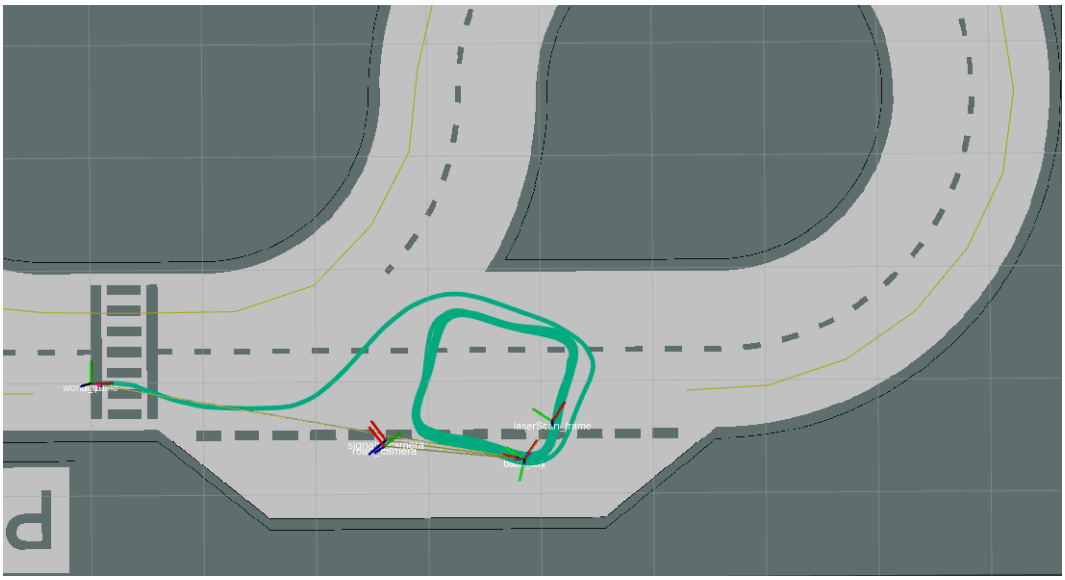


Figura 7.20: Collision avoidance sem transição de *behavior*.

A figura 7.21 apresenta o comportamento alcançado na evasão a obstáculo presente na faixa de rodagem, cenário típico presente na prova ADC. A figura apresenta sobreposição do *path* resultante desta manobra com robô em fase de execução da mesma.

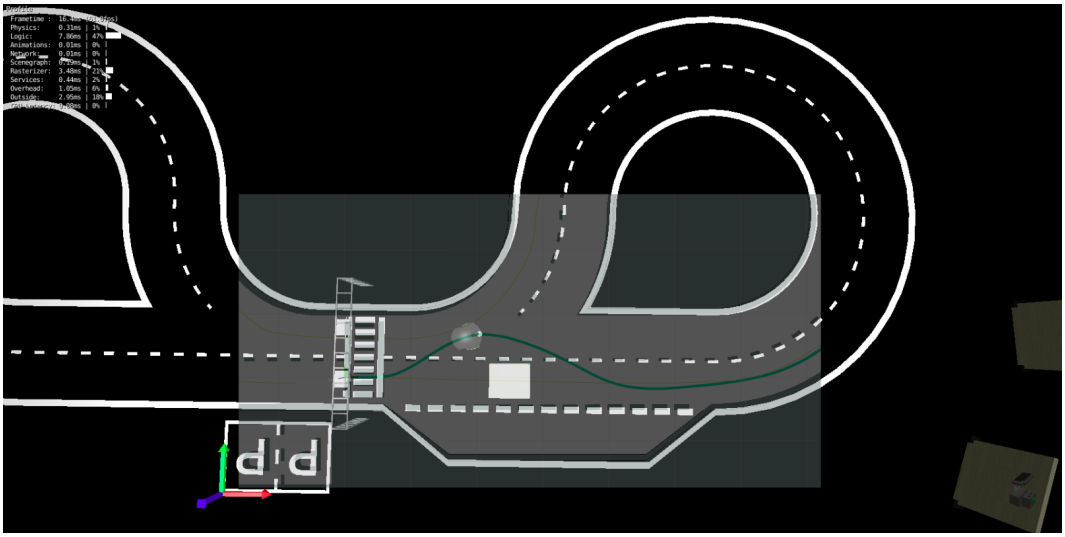


Figura 7.21: Collision avoidance

A figura 7.22 demonstra manobra de evasão sobre diversos obstáculos presentes na

faixa de rodagem. Após identificação do obstáculo na trajetória do robô, o *behavior* para manobra de evasão não é condicionado pelo formato, dimensão ou tipo de obstáculo. Esta valência permite que este *behavior* seja utilizado nos mais variados cenários, com utilidade em contextos ADC-FNR.

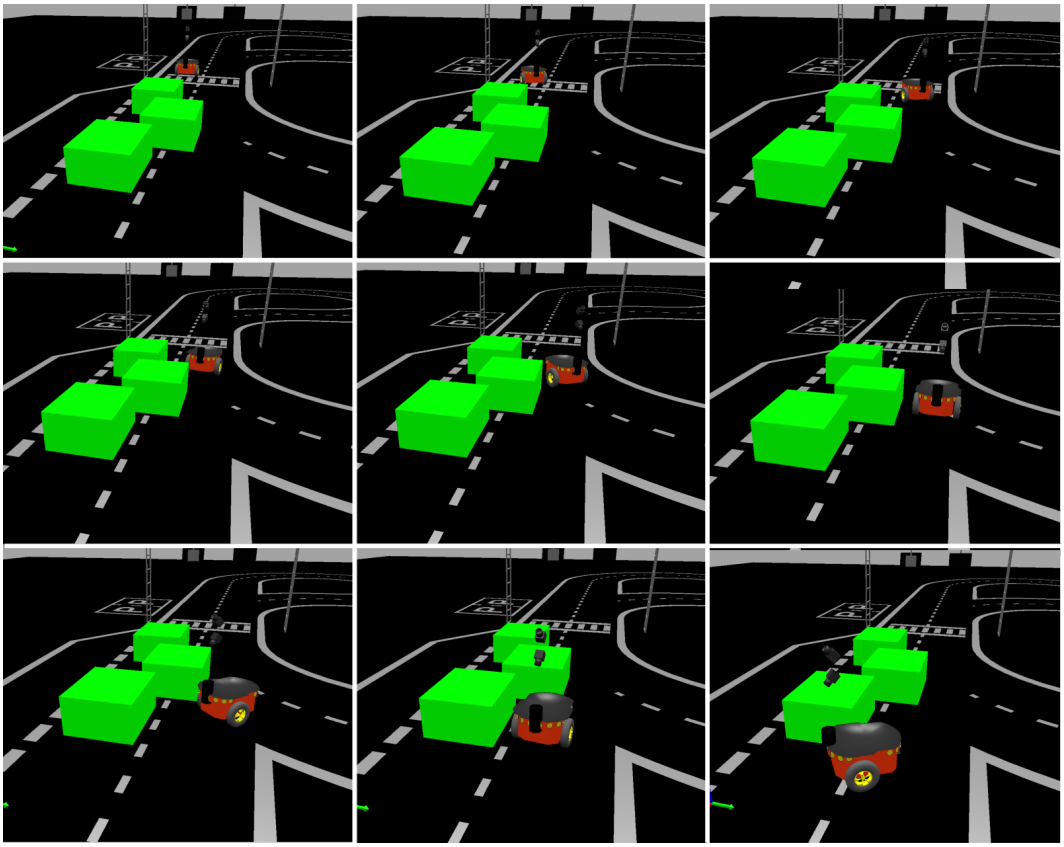


Figura 7.22: Collision Avoidance com diversos obstáculos.

Capítulo 8

Conclusões

O trabalho realizado nesta dissertação providencia à comunidade um conjunto de ferramentas e soluções enquadradas nos desafios propostos da Prova de Condução Autónoma, do Festival Nacional de Robótica.

Foi proposto e desenvolvido um ambiente de simulação com elevado detalhe e funcionalidade, uma solução que permite melhor capacitar interessados a competir nesta prova. O simulador promove o estudo e validação de desenvolvimentos para os desafios desta competição, endereçando tópicos da robótica como a Localização, Controlo e Visão Computacional.

O realismo alcançado em termos visuais, a interação e dinâmica do simulador cumpre amplamente os objetivos delineados. Parte do trabalho desta dissertação e com destaque para o ambiente de simulação, foram alvo de publicação científica na conferência RIE 2019 [95] no artigo **Teaching Robotics with a Simulator Environment Developed for the Autonomous Driving Competition**, disponível em [96].

Está presente nesta dissertação a exposição dos conceitos teóricos subjacente aos filtros probabilísticos, a explicação prossegue para o algoritmo do filtro de Kalman linear (KF) e estendido (EKF). O modelo do sistema é parte integrante do módulo de localização, o modelo diferencial e relação com modelo *unicycle* é apresentado neste trabalho e traduz a cinemática do robô Pioneer-2dx utilizado em simulação.

A implementação do módulo de localização encontra-se detalhado neste trabalho e determinou um conjunto de outros desenvolvimentos e acções necessárias. O modelo de observações em particular, exigiu:

- Calibração do sistema de visão;
- Implementação de algoritmo para gerar imagem IPM;

- Algoritmo para identificação das faixas de rodagem;
- Relacionar objetos da imagem (pixeis) com distâncias ao robô no Mundo (m);
- Inclusão de rede neuronal YOLO-v2;
- Preparação de setup e treino da rede neuronal para identificação da passadeira.

O módulo das observações carece contudo de desenvolvimentos adicionais para refinamento do processo e diminuição da incerteza na projeção das *landmarks* observadas. Em alternativa, as observações foram simuladas com incerteza na medida pela inclusão de ruído Gaussiano.

A *performance* do filtro é avaliada na estimação da posição do robô, com grande aproximação ao estado real o que valida a estratégia e implementação realizada.

Duas abordagens para controlo do movimento do robô são endereçadas para execução do percurso, um *Path Tracking* e *Path following* avaliados sobre diferentes condições de velocidade.

O módulo *Collision Avoidance* foi demonstrado perante diferentes configurações de obstáculos, testado com ausência de colisões e resposta adequada na transição entre *behaviors*. A solução é robusta e viável para implementação prática. Para aplicação deste controlador na zona de obras da prova ADC, o módulo de perceção do LiDAR carece de desenvolvimentos adicionais que se endereçam para trabalhos futuros.

Para trabalho futuro, prevê-se a portabilidade das soluções endereçadas nesta dissertação numa solução real de robô, avaliando comportamento em condições reais comparativamente à simulação.

Bibliografia

[1] Dean A. Pomerleau. ALVINN: an autonomous land vehicle in a neural network (Technical Report CMU-CS-89-107). pages 305–313, 1989.

[2] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[3] Xilin Zhou, Ziran Hu, Yu Wang, Jiahao Zhang, and Shouwen Yao. Autonomous-driving vehicle test technology based on virtual reality. *The Journal of Engineering*, 2018(16):1768–1771, 2018.

[4] Susan M. Paddock. Autonomous Vehicles Cannot Be Test-Driven Enough Miles to Demonstrate Their Safety; Alternative Testing Methods Needed | RAND.

[5] Nidhi Kalra and Susan M. Paddock. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice*, 94:182–193, 2016.

[6] Yurong You. arXiv : 1704 . 03952v4 [cs . AI] 26 Sep 2017 Virtual to Real Reinforcement Learning for Autonomous Driving. (October), 2017.

[7] Olivier Michel. Webots^{TM}: Professional mobile robot simulation. *Journal of Advanced Robotics Systems*, 1(1):39–42, 2004.

[8] Marc Freese, Surya Singh, Fumio Ozaki, and Nobuto Matsuhira. Virtual robot experimentation platform v-rep: A versatile 3d robot simulator. In Noriaki Ando, Stephen Balakirsky, Thomas Hemker, Monica Reggiani, and Oskar von Stryk, editors, *Simulation, Modeling, and Programming for Autonomous Robots*, volume 6472 of *Lecture Notes in Computer Science*, pages 51–62. Springer Berlin Heidelberg, 2010.

[9] E. Folgado, M. Rincón, J.R. Álvarez, and J. Mira. A multi-robot surveillance system simulated in gazebo. In José Mira and JoséR. Álvarez, editors, *Nature Inspired Problem-Solving Methods in Knowledge Engineering*, volume 4528 of *Lecture Notes in Computer Science*, pages 202–211. Springer Berlin Heidelberg, 2007.

[10] N. Koenig and A Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154 vol.3, Sept 2004.

[11] W. Yao, W. Dai, J. Xiao, H. Lu, and Z. Zheng. A simulation system based on ros and gazebo for robocup middle size league. In *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 54–59, Dec 2015.

[12] G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan. Modular open robots simulation engine: Morse. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 46–51, May 2011.

[13] Gilberto Echeverria, Séverin Lemaignan, Arnaud Degroote, Simon Lacroix, Michael Karg, Pierrick Koch, Charles Lesire, and Serge Stinckwich. Simulating complex robotic scenarios with morse. In Itsuki Noda, Noriaki Ando, Davide Brugali, and JamesJ. Kuffner, editors, *Simulation, Modeling, and Programming for Autonomous Robots*, volume 7628 of *Lecture Notes in Computer Science*, pages 197–208. Springer Berlin Heidelberg, 2012.

[14] A. Dias, J. Almeida, N. Dias, E. Silva, and P. Lima. Simulation environment for multi-robot cooperative 3d target perception. *SIMPAR 2014 – 4th International Conference on Simulation, Modeling , and Programming for Autonomous Robots, Springer-Verlag Lecture Notes in Computer Science*, 2014.

[15] S. Carpin, M. Lewis, J. Wang, S. Balakisky, and c. Scrapper. USARSim: a robot simulator for research and education. In *2007 IEEE International Conference on Robotics and Automation*, pages 1400–1405, 2007.

[16] DARPA. Welcome to Grandchallenge. `https://www.grandchallenge.org/`. Accessed: 2019-11-20.

[17] Audi autonomous driving cup. `https://www.audi-autonomous-driving-cup.com/`. Accessed: 2019-01-10.

[18] Open zeka marc. `https://openzeka.com/marc/`. Accessed: 2019-01-10.

[19] CARLA AD Challenge. `https://carlachallenge.org/`. Accessed: 2019-03-18.

[20] Festival Nacional de Robótica. `http://www.sprobotica.pt/index.php?option=com_content&view=article&id=108&Itemid=62`. Accessed: 2019-11-20.

[21] B Kim, Baehoon Choi, Euntai Kim, and Kwangwoong Yang. Indoor localization using laser scanner and vision marker for intelligent robot. *Control, Automation and . . .*, (2):1010–1012, 2012.

[22] Eric Royer, Maxime Lhuillier, Michel Dhome, and Jean Marc Lavest. Monocular vision for mobile robot localization and autonomous navigation. *International Journal of Computer Vision*, 74(3):237–260, 2007.

[23] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. Springer, 2016.

[24] Valter Costa, Rosaldo Rossetti, and Armando Sousa. Simulator for Teaching Robotics, ROS and Autonomous Driving in a Competitive Mindset. *International Journal of Technology and Human Interaction*, 13(4):19–32, 2017.

[25] Iqbal Sarker, Md Faruque, Ujjal Hossen, and Atikur Rahman. A survey of software development process models in software engineering. *International Journal of Software Engineering and its Applications*, 9:55–70, 01 2015.

[26] Olaf Gietelink, Jeroen Ploeg, Bart De Schutter, and Michel Verhaegen. Development of advanced driver assistance systems with vehicle hardware-in-the-loop simulations. *Vehicle System Dynamics*, 44(7):569–590, 2006.

[27] K. Athanasas, C. Bonnet, H. Fritz, C. Scheidler, and G. Volk. Valse- validation of safety-related driver assistance systems. In *IEEE IV2003 Intelligent Vehicles Symposium. Proceedings (Cat. No.03TH8683)*, pages 610–615, June 2003.

[28] Thomas Bokc, Markus Maurer, and Georg Farber. Validation of the Vehicle in the Loop (VIL); A milestone for the simulation of driver assistance systems. (Vil):612–617, 2007.

[29] Mechanical Simulation. `https://www.carsim.com/`. Accessed: 2019-07-01.

[30] UNREAL. What is Unreal Engine 4. `https://www.unrealengine.com/en-US/`. Accessed: 2019-11-20.

[31] Unreal Engine. VehicleSim Dynamics by Mechanical Simulation in Code Plugins - UE4 Marketplace. `https://www.unrealengine.com/marketplace/en-US/slug/carsim-vehicle-dynamics`. Accessed: 2019-11-20.

[32] Şükrü Yaren Gelbal, Santhosh Tamilarasan, Mustafa Rıdvan Cantaş, Levent Güvenç, and Bilin Aksun-Güvenç. A connected and autonomous vehicle hardware-in-the-loop simulator for developing automated driving algorithms. *2017 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2017*, 2017-January:3397–3402, 2017.

[33] Shengchao Zhao and Lijing Zhu. Cruise Control System Based on Joint Simulation of CarSim and Simulink. *OALib*, 05(07):1–8, 2018.

[34] Ehsan Khalili, Jafar Ghaisari, and Mohammad Danesh. Control and analysis of the vehicle motion using sliding mode controller and Carsim software. *2017 5th International Conference on Control, Instrumentation, and Automation, ICCIA 2017*, 2018-January(2):1–5, 2018.

[35] Yang Ying and Ogunleye Odunayosolomon. Research on Adaptive Cruise Control Systems and Performance Analysis Using Matlab and Carsim. *2017 5th International Conference on Mechanical, Automotive and Materials Engineering, CMAME 2017*, pages 249–253, 2018.

[36] MATLAB - MathWorks - MATLAB & Simulink. https://www.mathworks.com/products/matlab.html.

[37] Simulink - Simulation and Model-Based Design - MATLAB & Simulink. https://www.mathworks.com/products/simulink.html.

[38] Ümit Özgüner, Keith Redmill, Scott Biddlestone, Ming Feng Hsieh, Ahmet Yazici, and Charles Toth. Simulation and testing environments for the DARPA urban challenge. *Proceedings of the 2008 IEEE International Conference on Vehicular Electronics and Safety, ICVES 2008*, pages 222–226, 2008.

[39] Reinhold Behringer, William Travis, Rob Daily, David Bevly, Wilfried Kubinger, Wolfgang Herzner, and Victor Fehlberg. RASCAL - An autonomous ground vehicle for desert driving in the DARPA grand challenge 2005. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2005:644–649, 2005.

[40] Gazebo. `http://gazebosim.org/`. Accessed: 2019-10-16.

[41] USARSim. USARSim download | SourceForge.net. `https://sourceforge.net/projects/usarsim/`. Accessed: 2019-11-20.

[42] MORSE. Overview — The MORSE Simulator Documentation. `https://www.openrobots.org/morse/doc/stable/morse.html`.

[43] V-REP. Robot simulator V-REP: create, compose, simulate, any robot. - Coppelia Robotics. `http://www.coppeliarobotics.com/`. Accessed: 2019-10-06.

[44] ROS.org | Powering the world's robots. `https://www.ros.org/`. Accessed: 2019-10-04.

[45] Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. YARP: Yet Another Robot Platform. *International Journal of Advanced Robotic Systems*, 3(1):8, mar 2006.

[46] Gazebo. Gazebo. `http://gazebosim.org/`. Accessed: 2019-11-20.

[47] VehicleSim Dynamics by Mechanical Simulation in Code Plugins - UE4 Marketplace. `https://www.unrealengine.com/marketplace/en-US/slug/carsim-vehicle-dynamics`. Accessed: 2019-05-01.

[48] Willow Garage. `http://www.willowgarage.com/`. Accessed: 2019-05-07.

[49] Kristin L. Nichol. Influenza vaccine for healthy working adults [1] (multiple letters). *Journal of the American Medical Association*, 285(3):290–292, 2001.

[50] SDF Home. `http://sdformat.org/`. Accessed: 2019-05-08.

[51] Lenka Pitonakova, Manuel Giuliani, Anthony Pipe, and Alan Winfield. Feature and performance comparison of the V-REP, Gazebo and ARGoS robot simulators. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10965 LNAI:357–368, 2018.

[52] Game Creation — blender.org. `https://www.blender.org/features/game-creation/`. Accessed: 2019-05-08.

[53] Gilberto Echeverria, Séverin Lemaignan, Arnaud Degroote, Simon Lacroix, Michael Karg, Pierrick Koch, Charles Lesire, Serge Stinckwich, Gilberto Echeverria, Séverin Lemaignan, Arnaud Degroote, Simon Lacroix, and Michael Karg. Simulating complex robotic scenarios with MORSE To cite this version : HAL Id : hal-00988475 Simulating complex robotic scenarios with MORSE. 2014.

[54] Bullet Real-Time Physics Simulation | Home of Bullet and PyBullet: physics simulation for games, visual effects, robotics and reinforcement learning.

[55] Webots: robot simulator. `https://cyberbotics.com/{#}webots`. Accessed: 2019-05-16.

[56] Francisca Rosique, Pedro J. Navarro, Carlos Fernández, and Antonio Padilla. A Systematic Review of Perception System and Simulators for Autonomous Vehicles Research. *Sensors*, 19(3):648, 2019.

[57] Bonn Bonn. Monte Carlo Localization: Ef cient Position Estimation for Mobile Robots Dieter Fox, Wolfram Burgard , Frank Dellaert, Sebastian Thrun. *Intelligence*, (Handschin 1970), 1999.

[58] Abdul Muis and Rizky Prasetya. Implementation of landmark-based localization on wall-based environment for mobile robot with limited sensor resources. *IEEE Region 10 Annual International Conference, Proceedings/TENCON*, pages 1050–1054, 2011.

[59] Martin Barczyk. Invariant EKF Design for Scan Matching-aided Localization. pages 1–10.

[60] Shuqiang Zhao, Jason Gu, and Yongsheng Ou. IRobot Self-localization Using EKF. *2016 IEEE International Conference on Information and Automation (ICIA)*, (61273335):801–806, 2016.

[61] Amitava Chatterjee and Fumitoshi Matsuno. A Neuro-Fuzzy Assisted Extended Kalman Filter-Based Approach for Simultaneous Localization and Mapping (SLAM) Problems. 15(5):984–997, 2007.

[62] Mohammed Faisal, Mansour Alsulaiman, Ramdane Hedjar, Mansour Zuair, Hamdi Altaheri, and Mohammed Zakariah. Enhancement of mobile robot localization using extended Kalman filter. 8(11):1–11, 2016.

[63] Luka Tesli and Škrjanc Gregor. EKF-Based Localization of a Wheeled Mobile Robot in Structured Environments. pages 187–203, 2011.

[64] Juzhong Zhang and Kai Zhao. 3D localization and pose tracking system for an indoor Autonomous Mobile Robot. *2015 IEEE International Conference on Mechatronics and Automation, ICMA 2015*, pages 2209–2214, 2015.

[65] Xiaoyue Hou and Tughrul Arslan. Monte Carlo localization algorithm for indoor positioning using Bluetooth low energy devices. *2017 International Conference on Localization and GNSS, ICL-GNSS 2017*, pages 1–6, 2018.

[66] Cyrill Stachniss. On the Position Accuracy of Mobile Robot Localization based on Particle Filters Combined with Scan Matching. (Mcl):3158–3164, 2012.

[67] Dieter Fox. Adapting the sample size in particle filters through KLD-sampling. *International Journal of Robotics Research*, 22(12):985–1003, 2003.

[68] Andrea Censi. An ICP variant using a point-to-line metric.

[69] Monte Carlo. Mobile Robot Localization Based on Particle Filter. *Proceeding of the 11th World Congress on Intelligent Control and Automation*, (1):3089–3093, 2014.

[70] Joydeep Biswas and Manuela Veloso. WiFi localization and navigation for autonomous indoor mobile robots. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 4379–4384, 2010.

[71] Matthias Rapp, Markus Hahn, Thom Markus, Jurgen Dickmann, and Klaus Dietmeyer. Semi-Markov Process Based Localization Using Radar in Dynamic Environments. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2015-October:423–429, 2015.

[72] H. Ohta, N. Akai, E. Takeuchi, S. Kato, and M. Edahiro. Pure pursuit revisited: Field testing of autonomous vehicles in urban areas. In *2016 IEEE 4th International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA)*, pages 7–12, Oct 2016.

[73] W. Wang, T. Hsu, and T. Wu. The improved pure pursuit algorithm for autonomous driving advanced system. In *2017 IEEE 10th International Workshop on Computational Intelligence and Applications (IWCIA)*, pages 33–38, Nov 2017.

[74] T. Taniguchi, L. Eciolaza, and M. Sugeno. Tracking control for a non-holonomic car-like robot using dynamic feedback linearization based on piecewise bilinear models. In *2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 2465–2471, July 2014.

[75] S. Gao, Y. Li, and R. Song. Cooperative multiple nonholonomic robots control for moving-target circular formation using backstepping design and tracking differentiator. In *2017 Chinese Automation Congress (CAC)*, pages 7606–7611, Oct 2017.

[76] Nguyen Gia Minh Thao, Duong Hoai Nghia, and Nguyen Huu Phuc. A pid backstepping controller for two-wheeled self-balancing robot. In *International Forum on Strategic Technology 2010*, pages 76–81, Oct 2010.

[77] R. Hafner and M. Riedmiller. Neural reinforcement learning controllers for a real robot application. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 2098–2103, April 2007.

[78] M. Riedmiller, M. Montemerlo, and H. Dahlkamp. Learning to drive a real car in 20 minutes. In *2007 Frontiers in the Convergence of Bioscience and Information Technologies*, pages 645–650, Oct 2007.

[79] P. Xia and Y. Li. The control of two-wheeled self-balancing vehicle based on reinforcement learning in a continuous domain. In *2017 32nd Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pages 1084–1089, May 2017.

[80] Z. Li, C. Yang, C. Su, J. Deng, and W. Zhang. Vision-based model predictive control for steering of a nonholonomic mobile robot. *IEEE Transactions on Control Systems Technology*, 24(2):553–564, March 2016.

[81] Z. Li, J. Deng, R. Lu, Y. Xu, J. Bai, and C. Su. Trajectory-tracking control of mobile robot systems incorporating neural-dynamic optimized model predictive approach. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 46(6):740–749, June 2016.

[82] Tiago P Nascimento, Carlos Eduardo Trabuco Dórea, and Luiz Marcos G Gonçalves. Nonlinear model predictive control for trajectory tracking of nonholonomic mobile robots: A modified approach. *International Journal of Advanced Robotic Systems*, 15(1):1729881418760461, 2018.

[83] C. Sun, X. Zhang, Q. Zhou, and Y. Tian. A model predictive controller with switched tracking error for autonomous vehicle path tracking. *IEEE Access*, 7:53103–53114, 2019.

[84] I. J. Cox. Blanche-an experiment in guidance and navigation of an autonomous robot vehicle. *IEEE Transactions on Robotics and Automation*, 7(2):193–204, April 1991.

[85] D. Thrun, S., Burgard, W., Fox. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[86] David Kortenkamp, Reid Simmons, and Davide Brugali. Robotic systems architectures and programming. In *Springer Handbook of Robotics*, pages 283–306. Springer, 2016.

[87] Ronald C Arkin, Ronald C Arkin, et al. *Behavior-based robotics*. MIT press, 1998.

[88] Gilberto Echeverria, Nicolas Lassabe, Arnaud Degroote, and Séverin Lemaignan. Modular open robots simulation engine: MORSE. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 46–51, 2011.

[89] Pionner 3-DX platform — The MORSE Simulator Documentation. `https://www.openrobots.org/morse/doc/stable/user/robots/pioneer3dx.html`. Accessed: 2019-09-28.

[90] YOLO: Real-Time Object Detection. `https://pjreddie.com/darknet/yolo/`. Accessed: 2019-01-10.

[91] GitHub - AlexeyAB/Yolo_mark: GUI for marking bounded boxes of objects in images for training neural network Yolo v3 and v2.

[92] Raphael Grech and Simon G Fabri. Trajectory Tracking of a Differentially Driven Wheeled Mobile Robot in the Presence of Obstacles. (1):1–6.

[93] Alessandro De Luca, Giuseppe Oriolo, and Marilena Vendittelli. Control of Wheeled Mobile Robots: An Experimental Overview.

[94] Andrew Lamperski and Aaron Ames. Lyapunov-like conditions for the existence of zeno behavior in hybrid and lagrangian hybrid systems. *Proceedings of the IEEE Conference on Decision and Control*, 01 2007.

[95] Munir Merdan, Wilfried Lepuschitz, Gottfried Koppensteiner, and Richard Balogh, editors. *Robotics in Education*, volume 457 of *Advances in Intelligent Systems and Computing*. Springer International Publishing, Cham, 2017.

[96] David Fernandes, Francisco Pinheiro, André Dias, Alfredo Martins, Jose Almeida, and Eduardo Silva. Teaching Robotics with a Simulator Environment Developed for the Autonomous Driving Competition. pages 387–399. Springer, Cham, apr 2020.