SIMULAÇÃO E CONTROLO DE VEÍCULOS AUTÓNOMOS TERRESTRES

Renato Batista Pereira

Mestrado em Engenharia Electrotécnica e de Computadores

Área de Especialização de Sistemas Autónomos

Departamento de Engenharia Electrotécnica

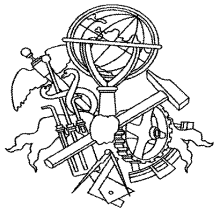
Instituto Superior de Engenharia do Porto

2009

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Disciplina de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores

Candidato: Renato Batista Pereira, Nº 1030375, 1030375@isep.ipp.pt

Orientação científica: Eduardo Alexandre Pereira da Silva, eaps@lsa.isep.ipp.pt



Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização de Sistemas Autónomos
Departamento de Engenharia Electrotécnica
Instituto Superior de Engenharia do Porto
12 de Dezembro de 2009

Dedico este Mestrado aos Meus pais, pois foram eles que me proporcionaram o acesso ao ensino superior

Agradecimentos

Agradeço a colaboração do meu orientador científico, Prof. Eduardo Alexandre Pereira da Silva e a todos os elementos e alunos do laboratório de Sistemas Autónomos do ISEP (LSA). Agradeço também a vontade e determinação dos meus pais de me proporcionarem a possibilidade de adquirir esta formação.

Em especial agradeço o apoio e colaboração da Ana Rita Barbosa e aos meus amigos de longa data no curso: Nuno Fonseca, Nelson Sousa, Ricardo Pinheiro, José Faria, Pedro Rocha, Rui, Gil, Jay, Fidel e Stephan.

Resumo

A implementação e venda de robôs autónomos tem sido um sector que nos últimos anos tem adquirido cada vez mais quota no mercado, nomeadamente no sector militar, agrícola e da vigilância. Como tal, tem sido também de grande importância a capacidade de implementar e testar robôs por parte das entidades que os fabricam. Uma das formas que tem garantido o sucesso do desenvolvimento de robôs é a simulação prévia dos mesmos antes que estes passem a fase de produção.

Sendo assim, o LSA como entidade de desenvolvimento de robôs autónomos, tem necessidade de adquirir um sistema que simule os robôs em desenvolvimento.

O trabalho desta tese consiste na realização de um sistema que simule robôs autónomos terrestres de forma que se possa observar o comportamento da cinemática, dinânica e hardware dos robôs em ambiente 3D. Esta aplicação de simulação pode mais tarde ser utilizada pelo laboratório para testar missões, validar alterações de estrutura, sensores, etc. Para além disso, com recurso ao simulador *Player/Stage/Gazebo* testar o robô LINCE e implementar algoritmos de controlo para o mesmo. Os algoritmos de controlo implementados baseiam-se em primitivas de controlo básico para serem utilizadas pelo sistema de navegação e gerar trajectórias complexas. Os algoritmos desenvolvidos nesta tese baseiam-se nas equações cinemáticas do veículo estudado.

Estes algoritmos depois de testados no simulador, poderão ser colocados no Hardware do robô. Desta forma consegue-se desenvolver algoritmos para determinado robô sem que este esteja operacional.

Palavras-Chave

Simulação, controlo, Gazebo.

Abstract

The sale and implementation of autonomous robots has been a sector that in recent years has gained increasing market share, as such, has also been of great importance to capacity to implement and test by the entities that make these robots. One way that has ensured the success of robotics' project consist in simulation of the robots, before, they pass through the design phase.

Thus, the LSA as producing agency has autonomous robots and they need to acquire a system that simulates robots developed and developing phase.

The work in this thesis is to implement a system that simulates autonomous terrain robots to simulate the kinematics, physics and hardware of the robot in 3D environment. This application of simulation can later be used by the laboratory for testing tasks, validate design changes, sensors, etc. Furthermore, using the simulator to test the robot LINCE and implement control algorithms for the same. The control algorithms implemented are based on basic control primitives to be used by the navigation system and generate complex trajectories. The algorithms developed in this thesis are based on the kinematic equations of the vehicle studied.

These algorithms when tested on the simulator can be placed on the hardware of the robot. Thus can be developed specific algorithms for robot which must be operational.

Keywords

Simulation, Control, Gazebo

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

AP - *Acess Point*

CAN - *Controller Area Network*

CANbus - *Controller Area Network Bus*

CCD - *Charge-coupled device*

CORBA - *Common Object Request Broker Architecture*

FLTK - *Fast Light ToolKit*

FLUID - *Fast Light User Interface Designer*

GLUT - *OpenGL Utility Toolkit*

GNOME - *GNU Object Model Environment*

GNU - *GNU is Not Unix*

GPL - *General Public Licence*

GPS - *Global Position System*

IMU - *Inertial measurement unit*

IR - *Infra Red*

*JNI*Ja - *Java Native Interface*

LGPL - *GNU Library General Public License*

LINCE - *Land Intelligent Cooperative Explorer*

LRFR - *Laser Range Finder Radar*

LSA - Laboratório de Sistemas Autónomos

MMC - *MultiMediaCard*

OA - *Obstacle Avoidance*

ODE - *Open Dynamic Engine*

OGRE - *Object-oriented Graphics Rendering Engine*

PC - *Personal computer*

PFC - P*otential-Field Control*

POMDP - *Partially observable Markov Decision*

PSG - *Player/Stage/Gazebo*

RAM - *Random Access Memory*

RC - *Radio Control*

RGV - *Red Green*

RIA - *Robotics Industries Association*

TCP/IP - *Transmission Control Protocol (TCP) and the Internet Protocol (IP)*

XML - *Extensible Markup Language*

Na actualidade temos vindo a observar-se um grande investimento no sector da robótica móvel em todo o mundo. Este crescimento deve-se da necessidade e interesse do mercado a nível mundial. Sabe-se que os robôs podem executar tarefas que de outra forma poderiam por em risco a vida dos seres humanos, por outro lado esse esforço também têm sido usado para melhorar a forma de tirar vidas no campo de batalha, mas como sempre, existe sempre os prós e contras em qualquer produto desenvolvido pelo ser Humano. De uma forma optimista, os robôs podem melhorar a qualidade de vida em geral de todos os habitantes do planeta se forem usados da forma correcta. Estamos perto de uma revolução tecnológica que terá o nome de *"Revolução da Robótica*".

Estamos perto de ter robôs a partilhar o nosso dia-a-dia nas tarefas domésticas, como companhia, como colegas de trabalho, como assistentes de todo o tipo de actividades. É fascinante o que pode acontecer, e de facto esta revolução que se aproxima é bem a demonstração do génio e da capacidade do Homem em fazer ciência e reinventar o seu futuro.

1.1 MOTIVAÇÃO

A necessidade de representar robôs em ambiente de simulação é uma prioridade para desenvolvimento de robôs na actualidade. Com a representação dos mesmos em ambiente controlado de simulação, pode-se distribuir de melhor forma as tarefas da equipa de desenvolvimento, repartindo assim os recursos humanos por controlo, navegação, hardware, Software e mecânica. A simulação de robôs em ambiente 3D pode também simular toda a missão para o qual o robô foi desenhado. Podem simular-se ambientes urbanos, não urbanos, *outdoor* ou *indoor*. Também assim pode-se simular os objectos com que este robô tem que interagir, como por exemplo, outros robôs, pessoas, obstáculos, etc. Desta forma, a equipa de projecto pode obter resultados do produto ainda em fase de desenvolvimento, o que pode promover a melhoramentos e em último caso concluir que determinado modelo de robô não é viável para determinada missão.

1.2 OBJECTIVOS

O objectivo principal deste projecto é realizar um ou vários algoritmos de controlo em ambiente simulado para o robô LINCE. O LINCE é um robô com um chassis de um jeep

de 4 rodas que foi desenvolvido pelo LSA (Laboratório de Sistemas Autónomos) como veremos nos capítulos seguintes. Para além de realizar o controlo do robô, este deve ser testado num simulador. Desta forma, esta tese concentra-se no desenvolvimento de um modelo do robô, de forma a simular o comportamento do mesmo em ambiente controlado.

Os objectivos podem dividir-se nos seguintes tópicos.

- ✓ Concepção ou utilização de uma plataforma de simulação robótica 3D.
- ✓ Desenvolvimento do modelo do robô em 3D.
- ✓ Desenvolvimento de um ou mais algoritmos de controlo para o robô e testa-los no simulador.

1.3 ESTRUTURA DA TESE

Esta tese está dividida em 9 capítulos. No Capítulo 1 (Introdução) é dado a conhecer a motivação para a realização desta tese. Para além disso, são definidos os objectivos a cumprir para este trabalho. No capítulo 2 (O robô e a sua constituição), é feito um estudo teórico das origens e evolução da robótica, bem como a actual constituição dos robôs modernos. No capítulo 3 (Estado de Arte), é feito um estudo do projecto *Marhes* e Metodologias de Controlo & Navegação. No capítulo 4 (Caso de Estudo) é feito um levantamento de informação acerca do robô de estudo. Neste capítulo houve a preocupação de dimensionar o modelo cinemático do robô e as características do mundo para mais tarde construir a simulação dos mesmos. O capítulo 5 (Escolha do Simulador) é feito um levantamento de informação de vários simuladores candidatos que possam satisfazer as necessidades do trabalho a implementar. No capítulo 6 (*Player/Stage/Gazebo project*) é feito um estudo da arquitectura de funcionamento do simulador escolhido no capítulo anterior. No capítulo 7 (Implementação) é demonstrado o processo de construção do modelo do robô e do mundo simulado. Também são explicados os princípios matemáticos no qual foram baseadas as primitivas de controlo para o robô LINCE. No capítulo 8 (Resultados da simulação) é feita uma demonstração dos resultados obtidos dos algoritmos desenvolvidos em simulação. No capítulo 9 (Conclusões) tirou-se as conclusões necessárias do trabalho efectuado. E por último encontramos as Referências documentais e Anexos.

2. O ROBÔ E A SUA CONSTITUIÇÃO

2.1 ORIGEM E EVOLUÇÃO DA ROBÓTICA

2.1.1 ROBÔ DE *HERON*

Para muitos os robôs têm origem no século XX, mas na realidade, já na antiguidade grega se faziam experiências com robôs. É claro que estes robôs não tinham um núcleo de silício nem eram alimentados a energia eléctrica. Quem descobriu detalhes sobre o autómato do século 1 D.C. foi o cientista da computação britânico *Noel Sharkey*, da Universidade de *Sheffield. Sharkey*. Ele vasculhou as obras teóricas de *Heron* de Alexandria, o lendário criador do autómato, e diz ter descoberto que esta é a primeira máquina guiada por um programa, tal como os computadores modernos, cujos registos chegaram até nós. Sem disco rígido ou memória RAM, a programação tinha de ser incorporada ao robô por meio de cordas, que eram enroladas em determinada sequência em torno dos eixos de suas rodas dianteiras.

O trigo ajudava a controlar a força motriz. Na parte de trás do autómato, a corda que estava enrolada em torno dos eixos ficava presa a um peso. Esse peso, por sua vez, ficava no alto de um tubo cheio de grãos de trigo. O tubo tinha um furo, do qual os grãos iam caindo devagar. Assim, o peso ia baixando cada vez mais, fazendo os eixos rodarem e o robô inteiro se mover.

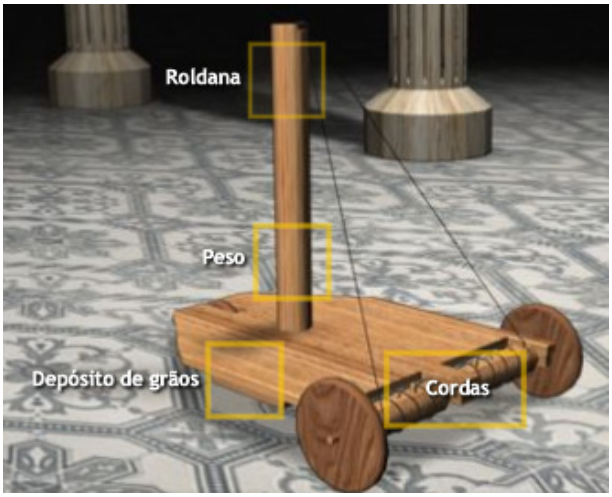


Figura 1 Robô de Heron

O grande catalisador do desenvolvimento dos autómatos actuais foi a Revolução Industrial. Com ela, foram desenvolvidos e aperfeiçoados dispositivos automáticos capazes de manipular e executar peças, permitindo a automatização da produção. Produzir em massa trouxe destaque à manipulação de objectos e acelerou o desenvolvimento de manipuladores. Os que mais ganharam importância e interesse foram as estruturas que eram montadas com segmentos e junções, colocados de forma linear, que lhe davam uma aparência de braços e pernas.

Neste mesmo período, em 1924, surgiu o primeiro modelo de robô mecânico. *Roy J. Wensley*, Engenheiro Electrotécnico da *Westinghouse,* desenvolveu uma unidade de controlo supervisionada. O dispositivo podia, utilizando o sistema telefónico, ligar e desligar ou regular remotamente qualquer coisa que estivesse ligado a ele. Três anos depois, ele criou o *Televox*, um pequeno robô com aspecto humano que conseguia executar movimentos básicos, de acordo com os comandos de seu operador. Os robôs ganharam ainda mais popularidade com o nascimento de *Willie Vocalite*, em 1930. *Willie* tinha o formato daqueles robôs que vemos nos filmes de ficção antigos.

Tinha 2 metros de altura e era feito de aço e da mesma forma que o *Televox* podia ligar, desligar e regular dispositivos conectados a ele. A grande diferença estava no facto de fazer tudo isto sob comandos de voz, fumava, sentava-se, ficava de pé, movia os braços e conversava com as pessoas reproduzindo frases gravadas em discos de 78 rotações. Foi a grande sensação da exposição Mundial de Chicago em 1933.

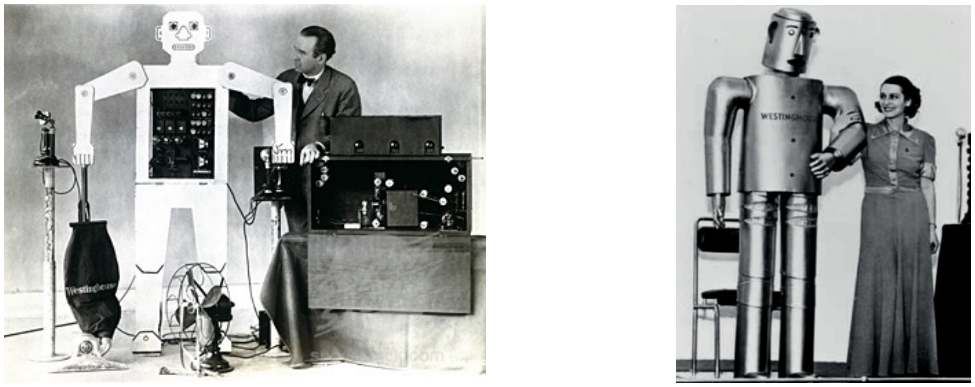


Figura 2 Televox e Willie Vocalite

2.1.2 ACTUALIDADE DA ROBÓTICA E O SEU FUTURO

A grande quantidade de recursos destinados ao desenvolvimento e pesquisa em robótica já começou a dar frutos, e certamente conduzirá a grandes acontecimentos no futuro. Várias ferramentas têm sido desenvolvidas e fontes de energia tem sido exploradas, para substituir o homem e ajuda-lo no seu trabalho. Actualmente o homem ainda é uma parte importante do sistema por ser responsável pela tomada de decisão.

O principal objectivo da robótica é liberar o ser humano de tarefas difíceis e de risco. Para atingir esta meta, muita pesquisa deve ser realizada na área da Inteligência Artificial, para que o robô possa por si só tomar decisões que hoje são feitas pelos humanos e identificar os objectos ao seu redor.

A robótica surgiu como resultado de intensa pesquisa na área de computadores e está num estágio precoce de desenvolvimento, pois as soluções robóticas actuais não respondem a todas as missões desejadas. Um exemplo disso é o robô mais sofisticado da actualidade (Asimo), para qualquer missão, este tem que ser previamente programado de forma a atingir o objectivo desejado. A próxima etapa depende do aparecimento de computadores que possam funcionar como cérebros. O cérebro do robô do futuro deverá ser um

computador sofisticado, rápido, com grande espaço de memória e capacidade de analisar situações complicadas. É necessário também que seja pequeno e tenha baixo consumo de potência.

2.1.3 O QUE É UM ROBÔ

Segundo a RIA, *(Robotics Industries Association)* "*um robô é um manipulador multifuncional reprogramável projectado para mover material, partes, ferramentas ou dispositivos especializados, através de diversos movimentos programados, para a execução de uma variedade de tarefas*". Mas como a tecnologia tem evoluído, a própria definição sofre alterações.

Na minha opinião, um robô é um dispositivo que possui a capacidade de executar uma tarefa para a qual foi projectado. Normalmente estamos habituados a entender que um robô é um dispositivo electromecânico idêntico ao ser humano (humanóide), mas, na realidade um robô pode ser mais que isso. Um robô para além de poder ser um dispositivo electromecânico pode também ser um dispositivo puramente mecânico ou um simples programa de computador.

Um robô móvel é uma combinação de elementos físicos (hardware e Mecânica) e computacionais (software) agrupados num conjunto de subsistemas que suportam locomoção (actuadores) para se deslocar no ambiente, percepção (sensores) para avaliar as propriedades do seu estado interno bem como o ambiente envolvente, raciocínio (controlo) de forma a usar a informação adquirida e planear a tarefa seguinte e aprender com a experiência passada bem como decidir e implementar acções e por último comunicação de forma a comunicar com o operador ou com outros robôs.

2.1.4 SISTEMA DE LOCOMOÇÃO

A locomoção é o mecanismo que permite que um robô se mova. As estratégias de locomoção presentes nos robôs móveis são função das aplicações e dos ambientes em que os robôs operam e resultam da necessidade de responder a qual é a velocidade máxima operar, se o robô irá subir escadas ou pavimentos irregulares, se irá robô sobrevoar o terreno, etc.

Com base na aplicação e consequentemente no tipo de ambiente em que se move, há quatro categorias de robôs móveis:

- ✓ Robôs terrestres
- ✓ Robôs aquáticos
- ✓ Robôs aéreos
- ✓ Robôs espaciais

2.1.5 ROBÔS TERRESTRES

Movem-se em contacto com o solo e são projectados para tirar partido da gravidade e do contacto físico com a superfície. Embora a maior parte dos robôs móveis terrestres tenham rodas, existem robôs que têm uma ou mais patas, lagartas ou uma combinação destes sistemas. O tipo das rodas e a sua disposição, o número de patas e o movimento coordenado entre elas e a disposição das lagartas definem o tipo de movimento e a liberdade para efectuar trajectórias mais ou menos complexas.

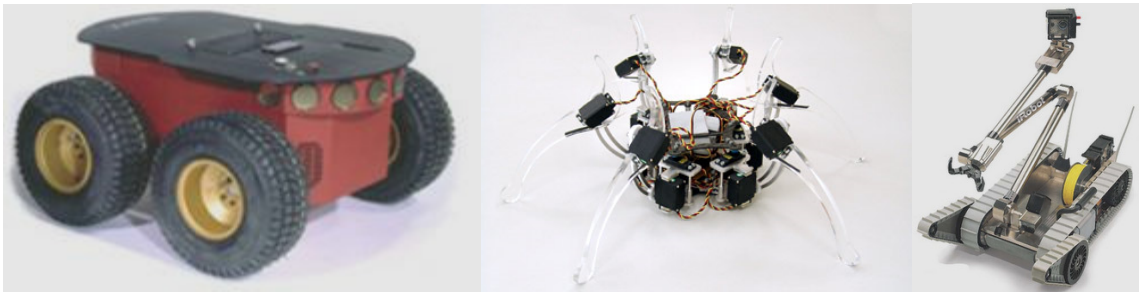


Figura 3 Robôs terrestres

2.1.6 ROBÔS AQUÁTICOS

Os robôs aquáticos funcionam à superfície, como navios autónomos de superfície (usados para fins de pesquisa oceanográfica) ou submersos. Por exemplo, o robô oceanográfico ROAZ do LSA, operado sem intervenção humana e com uma larga autonomia. È movido através de propulsores constituídos por motores eléctricos associados à hélice. A maioria dos robôs submarinos é movida do mesmo modo, sendo o controlo de orientação efectuado através de lemes horizontais ou verticais. Os robôs submarinos da actualidade estão dotados de propulsores com alimentação eléctrica por sistemas de baterias.

Figura 4 Robôs aquáticos

2.1.7 ROBÔS AÉREOS

Os robôs aéreos são semelhantes a aviões ou a pássaros e têm sido desenvolvidos helicópteros, dirigíveis, para quedas robotizados, todos a operar de modo autónomo ou semi-autónomo. A movimentação resulta, na maioria dos casos, do controlo de um ou mais motores associados a hélice (no caso de helicópteros ou dirigíveis) cujo nalguns casos, com a movimentação de superfícies de deflexão. Estes robôs têm feito surgir um particular interesse das forças militares, pois veículos com capacidade de voo e de pequenas dimensões podem ser perfeitos espiões, visto que o tamanho pode fazê-los invisíveis a radares inimigos. Para além do interesse militar este robôs têm particular interesse pela vigilância e pesquisa meteorológica.

Figura 5 Robôs aéreos

2.1.8 ROBÔS ESPACIAIS

Os robôs autónomos espaciais são veículos todo o terreno, capazes de recolher imagens e amostras da superfície dos astros por si explorados. Possui vários tipos de câmaras e antenas, para transmissão de dados, e um grande contentor que se destina ao armazenamento das amostras recolhidas. São transportados a bordo de sondas ou de naves espaciais e quando pousam na superfície dos astros, podem percorrer grandes distâncias a descoberta de novos dados.

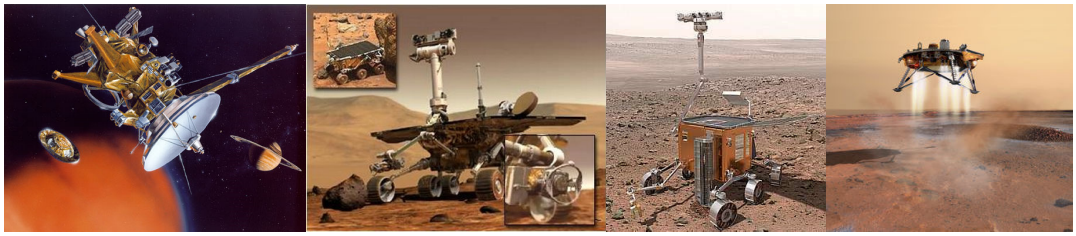


Figura 6 Robôs espaciais

2.2 SENSORES

A utilização de sensores permite que um robô possa interagir com o ambiente que o rodeia de uma forma flexível e inteligente. O uso da tecnologia dos sensores introduz nas máquinas um maior nível de inteligência para lidar com o seu meio. É igualmente objecto de uma pesquisa constante e intensa no campo da robótica. Um robô que possa sentir e ver como o homem, é mais fácil de treinar para a realização de tarefas complexas, e requer mecanismos de controlo menos rígidos e atentos que os das máquinas pré-programadas. Um sistema sensorial é também mais adaptável a uma maior variedade de tarefas, atingindo desta forma um maior grau de universalidade e que terá repercussões nos custos de produção e de manutenção, que serão menores.

2.2.1 SENSORES DIGITAIS

Os sensores digitais são aqueles cuja "saída" apresenta apenas dois estados possíveis, aos quais normalmente nos referimos como alto e baixo ou "1" e "0". São, por isso, muito simples de serem conectados à micro-controladores. Entre os sensores digitais, os mais utilizados em robótica são:

Sensores de toque, ou de presença: São constituídos basicamente por um *switch* (interruptor) mecânico com uma haste, que é posicionada de maneira que o sensor possa perceber a presença de um objecto ou de um obstáculo.

Sensores ópticos com foto-transístor: O foto-transístor é um tipo especial de transístor cuja polarização é determinada pela captação ou não de luz, geralmente são usados acompanhados de um emissor de luz, montado em oposição ao receptor onde está o foto-transístor, de forma a perceber a presença de qualquer coisa que obstrua o feixe de luz.

2.2.2 SENSORES ANALÓGICOS

Sensores analógicos são aqueles cuja saída pode assumir infinitos valores dentro de um determinado intervalo. Sendo assim, faz-se necessário algum tipo de conversão da saída destes sensores para que possam ser utilizados em projectos que envolvam electrónica digital. Os componentes responsáveis por isso, como se pode imaginar, recebem o nome de conversores analógicos/digitais (A/D). Eles traduzem o sinal analógico que recebem do sensor para um valor, dentro de um intervalo numérico com que irá corresponder a sua precisão de leitura. Existe uma grande variedade de sensores deste tipo, na realidade, quase todos os sensores existentes são deste género.

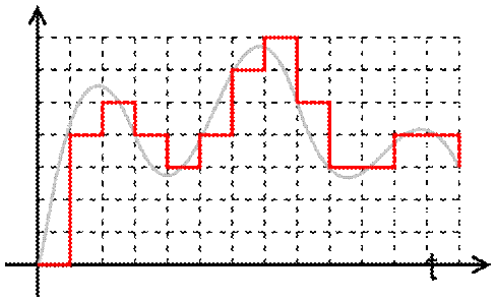


Figura 7 Ilustração de uma conversão de um AD

Na robótica os sensores são utilizados para percepção do próprio robô bem como percepção do mundo que rodeia o mesmo, isto leva a dividir e classificar os sensores como:

- ✓ Sensores internos
- ✓ Sensores externos

2.2.3 SENSORES INTERNOS

São usados para medir o estado do robô, posição, velocidade ou aceleração das juntas ou extremidades de um robô, manipulador, ou das rodas de um robô móvel.

2.2.3.1 ENCODERS

Os *Encoders* são transdutores de movimento capazes de converter movimentos lineares ou angulares em informações eléctricas que podem ser transformadas em informações binárias e trabalhadas por um programa que converta as informações passadas em algo que possa ser entendido como distância, velocidade e aceleração.

Os *encoders* possuem internamente um ou mais discos perfurados, que permitem a passagem de um feixe de luz infravermelho, gerado por um emissor que se encontra num dos lados do disco sendo captado por um receptor que se encontra do outro lado do disco, este, com o apoio de um circuito electrónico gera um pulso. Dessa forma a velocidade ou posicionamento é registada contando-se o número de pulsos gerados.

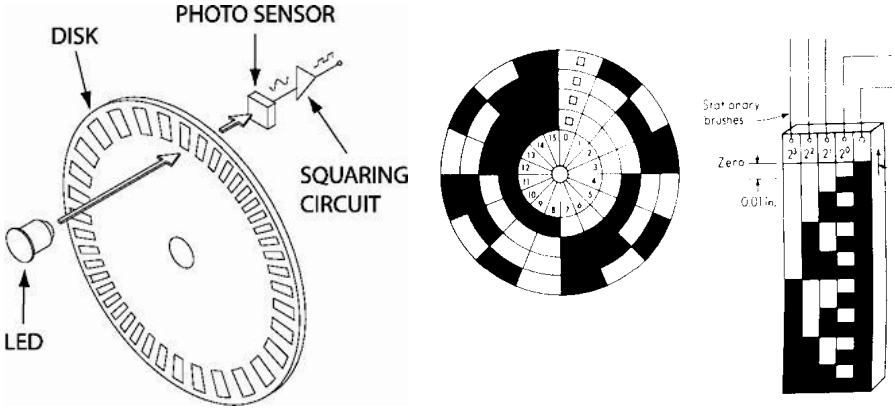


Figura 8 Princípio de funcionamento de um encoder

2.2.3.2 SENSORES DE POSICIONAMENTO ABSOLUTO, AMBIENTAIS E INÉRCIAS

Uma outra classificação agrupa os sensores pelo tipo de grandeza que avaliam. Existem sensores de posicionamento absoluto que avaliam a localização absoluta do robô, por exemplo os sistemas GPS (*Global Position System*), sensores inerciais que indicam componentes diferenciais da posição do robô, como por exemplo aceleração e velocidade.

2.2.3.3 FUSÃO SENSORIAL

A disponibilidade de um sensor para estimar a posição e a orientação de um robô móvel é fundamental para a execução de algoritmos de localização, mapeamento e navegação em qualquer aplicação de robótica móvel. Estes algoritmos dependem habitualmente da estimação da posição fornecida por um *odómetro* baseado em *encoders* acoplados às rodas do robô, a partir do qual reduzem o erro da estimação da posição através da fusão com outros sensores (ex. laser, visão stéreo, GPS, etc) e da utilização de filtros probabilísticos apropriados (Filtro de Kalman). Contudo, a estimativa fornecida pelo odómetro convencional está limitada a 3 graus de liberdade (dois de posição mais um de orientação), sendo apenas viável em aplicações em que o robô móvel se movimente num plano, em ambientes estruturados e com baixo deslizamento das rodas. A fusão sensorial permite minimizar os erros associados aos sensores permitindo assim utilizar os mesmos sistemas em ambientes não estruturados, nomeadamente em ambientes outdoor apresentando fortes desníveis ou perturbações no movimento das rodas do robô. Para o efeito, proceder-se-á normalmente à fusão sensorial da informação proveniente de três tipos de sensores: Odometria *(Encoder*), GPS (*Global Position System*) e IMU (*inertial measurement unit*). O primeiro tem a capacidade de medir o deslocamento das rodas e assim estimar a distância percorrida pelas mesmas. O segundo, com muito mais precisão, tem a capacidade de corrigir o efeito do escorregamento, deformação e deslizamento das rodas. O terceiro estima a rotação com um erro aceitável e a translação com um erro cumulativo ao longo do tempo. A fusão sensorial dos dados dos três sensores é efectuada através de um filtro estatístico (Filtro de Kalman) que se baseia na previsão da medida e correcção da previsão.

2.2.4 SENSORES EXTERNOS

São utilizados para monitorar o próprio robô e a sua relação com o mundo ao seu redor, bem como a realização da tarefa que lhe foi destinada.

2.2.4.1 ULTRA-SOM E LASER

Muitos robôs móveis, em particular robôs terrestres, usam sensores de ultra-sons para detectar obstáculos inesperados e desviarem-se deles. Também há automóveis comerciais dotados deste tipo de sensores para auxiliar o estacionamento em zonas apertadas. Os sonares (acrónimo de *Sound Navigation And Ranging*), como são vulgarmente conhecidos,

são sensores externos, activos que avaliam distâncias. O princípio de funcionamento é a avaliação do tempo de voo de uma onda acústica, gerada pelo sensor em rajadas curtas e que se propagam no meio ambiente. Quando existem obstáculos, a onda é reflectida e detectada pelo sensor. O tempo que entre emissão e recepção depende da velocidade de propagação do som no ar (em robôs terrestres) e é proporcional ao dobro da distância percorrida pela onda sonora.

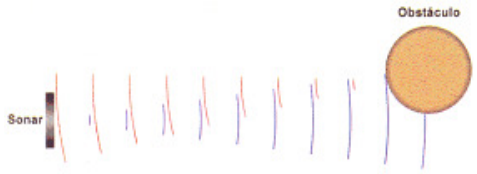


Figura 9 Princípio de funcionamento de um sonar

A semelhança dos sonares, também a maioria dos laser calcula a distância aos obstáculos pelo "tempo de voo" entre emissão e recepção de um feixe de luz. Na verdade um pulso de luz é gerado e transmitido com um determinado comprimento de onda, o desfasamento entre o sinal emitido e o sinal recebido é proporcional a distância percorrida pelo feixe de luz. O feixe de sonar pode ser aproximado como cónico, com largura da ordem dos 10º e 15º dependendo do sensor. O feixe de laser pode ser considerado como cilíndrico com um diâmetro da ordem dos milímetros.

Os emissores laser estão muitas vezes associados a mecanismos de varrimento que permitem cobrir uma larga área a frente dos robôs. Designam-se então como lasers scanner.

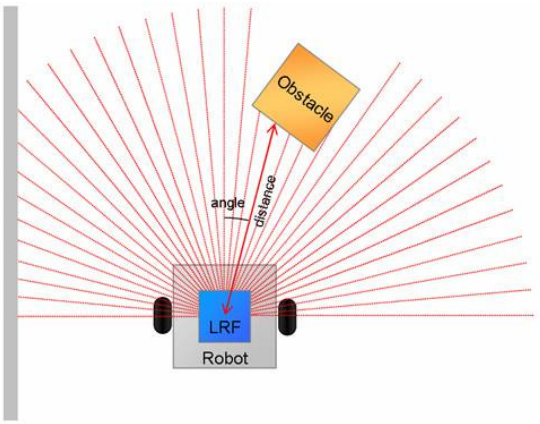


Figura 10 Princípio de funcionamento de um Laser scanner

2.2.4.2 VISÃO ROBÓTICA

Os robôs móveis operam num mundo que é difícil de caracterizar para os mesmos. Esta incerteza aparece na percepção e mapeamento do ambiente, no movimento dos manipuladores e objectos e no planeamento e execução de tarefas. Se por um lado um sistema sensorial simples é capaz de suprir parte das informações, ele é consequentemente limitado na sua habilidade de interpretar ambientes desconhecidos ou parcialmente desconhecidos. Recentemente tem aumentado o interesse sobre o desenvolvimento de sistemas multi-sensoriais que podem resolver simples ambiguidades entre os sensores e interpretar o ambiente mais precisamente. Um sistema de integração de visão equipado com um LRFR (*Laser RangeFinder Radar*) e uma câmara colorida CCD, ilustrados na figura seguinte, são utilizados para detectar os obstáculos e obter conhecimento sobre o ambiente para a navegação e tarefas de planeamento de rota.

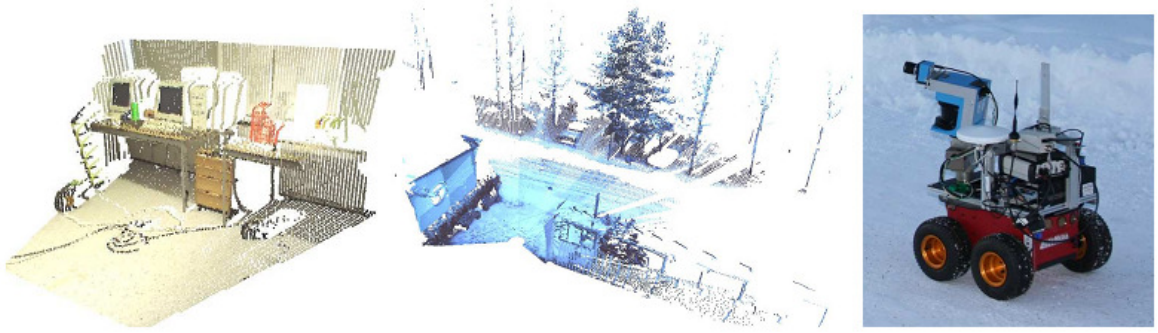


Figura 11 Fusão de imagem e Laser

2.3 ACTUADORES

Actuadores são dispositivos capazes de gerar uma força a partir de um líquido, gás ou energia eléctrica. O actuador recebe uma ordem de um controlador e fornece a força necessária para activar um elemento final de controlo, como por exemplo motores ou braços. Existem três tipos de actuadores:

- ✓ Hidráulicos
- ✓ Pneumáticos
- ✓ Eléctricos

Na robótica os mais utilizados são os eléctricos devido a natureza da fonte de energia de toda a electrónica ser eléctrica. Só em situações especiais se escolhem actuadores de outra natureza.

2.3.1 MOTORES DE PASSO

O motor de passo é um transdutor que converte energia eléctrica em movimento controlado através de pulsos, o que possibilita o deslocamento por passo, onde passo é o menor deslocamento angular.

Uma vantagem do motor de passos em relação aos outros motores é a estabilidade. Quando quisermos obter uma rotação específica de um certo grau, calcularemos o número de rotação por pulsos o que nos possibilita uma boa precisão no movimento.

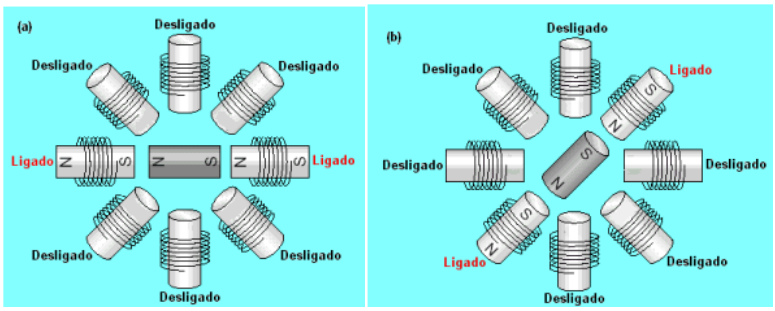


Figura 12 Princípio de funcionamento de um motor de passo

Estes motores são essencialmente usados para movimentos de precisão. Nomeadamente braços robóticos, movimentação de dispositivos e sensores.

2.3.2 MOTORES BRUSHLESS

A não existência de escovas nestes motores torna-os ideais para uma grande quantidade de aplicações, dada a não necessidade de manutenção. Além disso eles não produzem tanto ruído eléctrico, como os motores com escovas, podendo também operar em ambientes hostis. Nestes motores encontramos bobinas que são excitadas por um controlador de modo a produzir o movimento no sentido desejado. Os motores comuns deste tipo possuem três partes principais: estactor, rotor e sensores Hall. Num motor trifásico, o estactor possui

três bobinas. O rotor, por outro lado, consiste em um número par de imanes permanentes cujo tamanho e força determinarão o binário do motor.

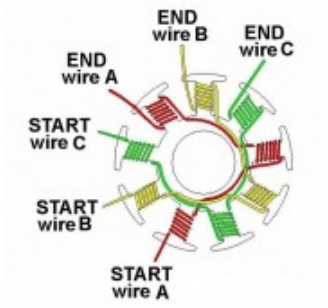


Figura 13 Bobinas de um motor brushless

A quantidade afecta o passo e o ripple. Mais pólos significam mais passos e menor ripple de torque. São utilizados de 1 a 5 pares de pólos nos motores comuns, chegando em certos casos a 8. Nestes motores as bobinas são fixas enquanto que o conjunto de imanes gira. Estes motores são mais leves que os motores com escovas, pois naqueles as bobinas ficam no rotor.

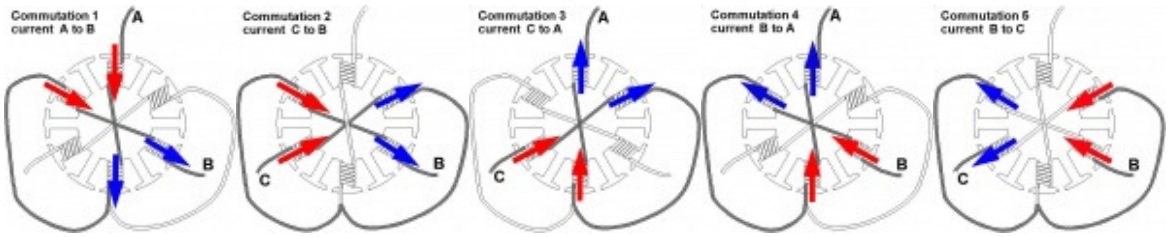


Figura 14 Princípio de funcionamento de um motor Brusless

Alterando a frequência da alimentação do motor podemos assim controlar de forma eficaz a rotação do motor conseguindo assim um motor trifásico de excelente performance. Estes motores são da maior parte das vezes utilizados na locomoção dos robôs visto serem motores robustos e capazes de grandes binários.

3. ESTADO DE ARTE

3.1 PROJECTO *MARHES*

O projecto Marher [14.] é uma plataforma de controlo multi-robótica de um conjunto de robôs de forma a se moverem e interagirem como uma equipa. Este projecto foi desenvolvido pelo *Oklahoma State University*. O objectivo deste projecto é ser utilizado em exploração, salvamento, mapeamento e transporte colectivo entre robôs.

Figura 15 Projecto *Marhes*

3.1.1 DESCRIÇÃO DA PLATAFORMA *MARHES*

Cada plataforma tem como estrutura mecânica um modelo *RC TXT-1*da *Tamiya.* Este modelo foi modificado para satisfazer as necessidades desta equipa de robôs. A estrutura foi equipada com um conjunto de sensores e actuadores de forma a construírem uma plataforma multifuncional autónoma e com capacidade de executar diversas missões.

Esta plataforma possui sensores e actuadores dispostos numa rede *CANBus*. Desta forma eles conseguem ter acesso a toda a informação da plataforma pela rede (Rede de sensores). O *hardware* implementado por este projecto pode ser vistos na imagem seguinte:

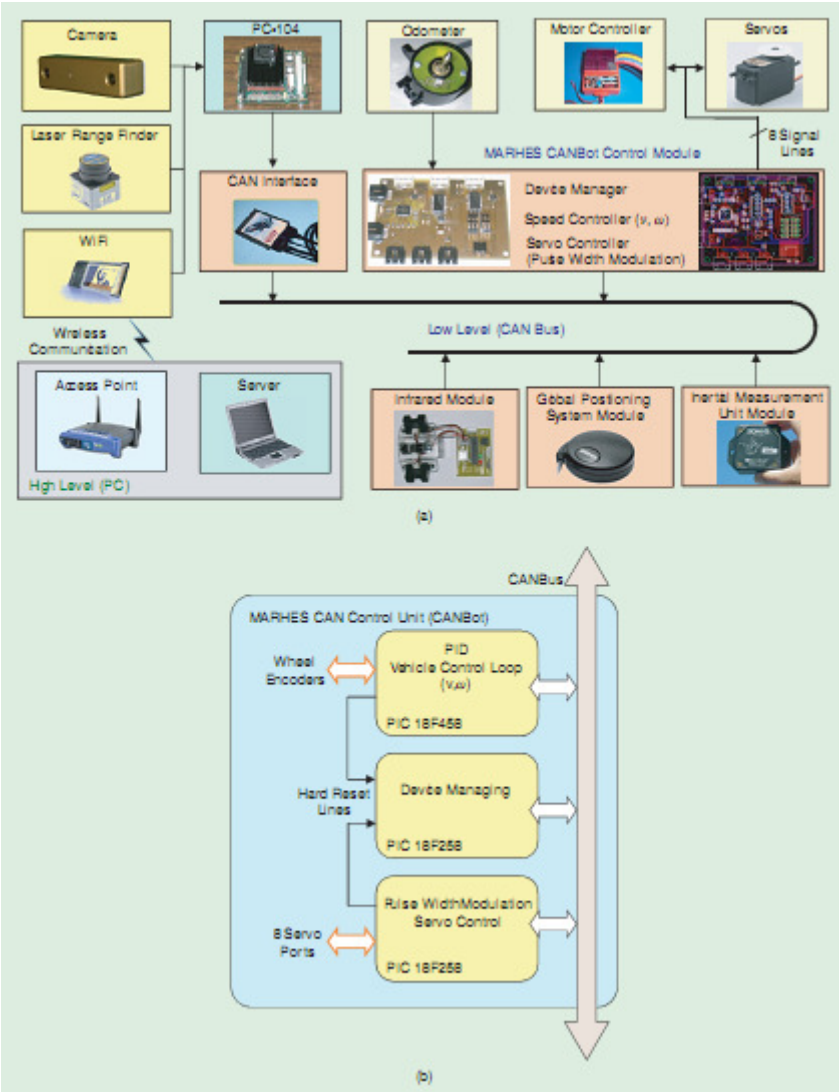


Figura 16 Rede de controlo (*CAN*). Modelo da arquitectura do *Hardware* do robô

Para realização do algoritmo de controlo este projecto utilizou o simulador *Gazebo*, um simulador 2D e 3D *open Source*. Com este simulador, conseguiu-se simular a carroçaria, sensores e actuadores do robô. Este simulador foi importante para desenvolver os algoritmos de visão-stéreo dos robôs, visto ser esta a principal forma de os robôs se localizarem entre eles.

O sistema de visão baseia-se em detecção de marcas em cada robô. A partir da detecção dessas marcas o algoritmo de visão pode calcular a posição e orientação do robô. Partilhando essa informação na rede pode assim melhorar-se a confiança da posição de toda a equipa de robôs através de triangulação de informação.

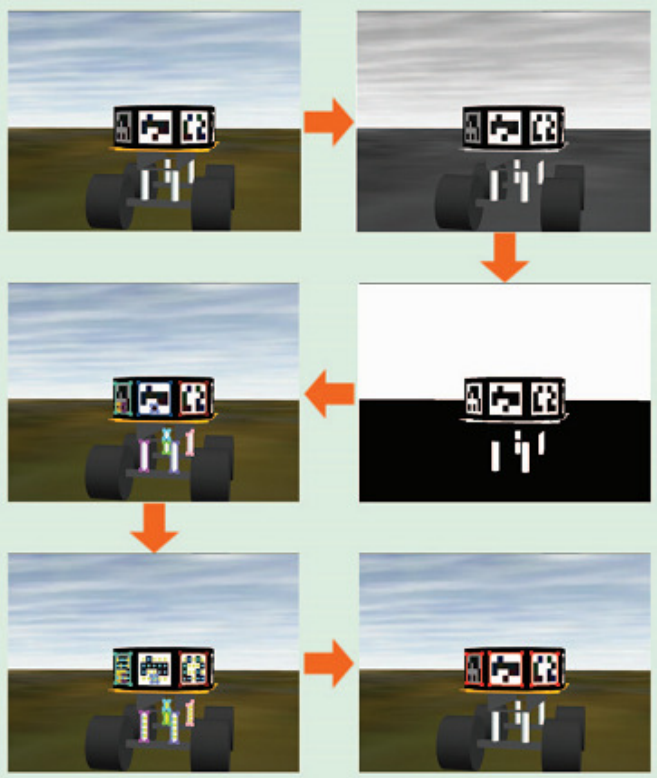


Figura 17 Sistema de Visão Stéreo

O PC das plataformas executa o *Player*, o programa servidor. Depois do *Player* estar executado, o robô é controlável através da rede. Um programa pode executar o controlo sobre a máquina local ou utilizando um computador remoto na rede.

3.1.2 CONTROLO DA PLATAFORMA *MARHES*

Esta secção descreve comportamentos básicos de controlo que são implementados nas plataformas. Esses comportamentos podem ser executados em paralelo ou sequencialmente ao criar rotas mais complexas pelo sistema de navegação. Os algoritmos são implementados em C++ e correm em sistema *Linux* no PC das plataformas. O *Player* é usado para fazer interface com os veículos e com o *hardware*. Assim, o comportamento implementado sofre interferências devido aos atrasos da rede devido aos requisitos temporais da rede *CANBus*, assim como os sensores e actuadores do sistema descritos anteriormente.

3.1.2.1 GO-TO-GOAL

O comportamento *go-to-goal* é baseado num algoritmo PFC. PFC usa um potencial atractivo, o que permite que os agentes se movam para um local específico. O PFC pode ser utilizado quando o objectivo passa por seguir: líder, *waypoint*, ou qualquer outra aplicação em que um veículo necessite de atingir um determinado ponto no espaço.

Este algoritmo é base de quase todos os algoritmos de controlo implementados por este sistema. No fundo quando o sistema de navegação calcula uma rota, este divide essa rota em vários *GOAL* (Objectivo) a atingir dividindo assim a missão em vários objectivos.

3.1.2.2 OBSTACLE AVOIDANCE (OA)

Obstacle Avoidance é um elemento básico para sistemas robóticos autónomos. O *OA* torna o funcionamento do robô seguro usando sonares ou sensores *IR* para detectar e evitar obstáculos.

3.1.2.3 GOAL-SEEKING

O *goal seeking* surge da combinação entre *OA* e o *goal-to-goal* usando como comportamento a arquitectura híbrida como podemos observar na figura seguinte. O objectivo surge do comportamento resultante entre condução do agente designado de *waypoint* com desvio de obstáculos. O comportamento é útil para viajar autonomamente. A figura seguinte mostra o controlo do *goal-seeking*.

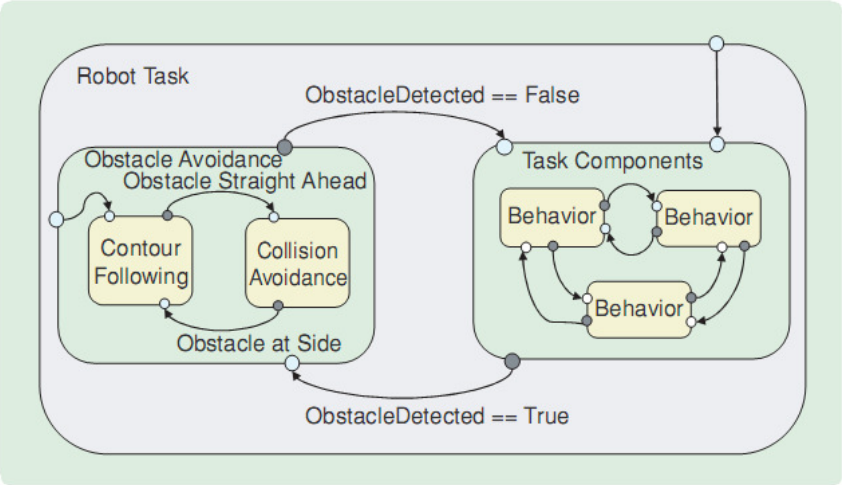


Figura 18 ***Obstacle Avoidance***

3.1.2.4 FORMATION CONTROL

Muitos sistemas naturais, tais como enxames, escolas, e os rebanhos exibem comportamentos estáveis de formação. Nestes sistemas, robôs seguem líderes distantes sem colidir com vizinhos. Em alguns domínios de aplicação, um grupo de agentes tem necessidade de se mover como uma estrutura rígida. Além disso, em situações concretas tais como a manipulação cooperativa, uma formação de destino deve ser estabelecida para uma determinada tarefa ou ambiente. Nestes casos, a reconfiguração dos agentes em formação é requerida. Controlar a formação é útil quando uma rede celular de agentes são obrigados a seguir uma trajectória prescrita e alcançar e manter uma formação desejada.

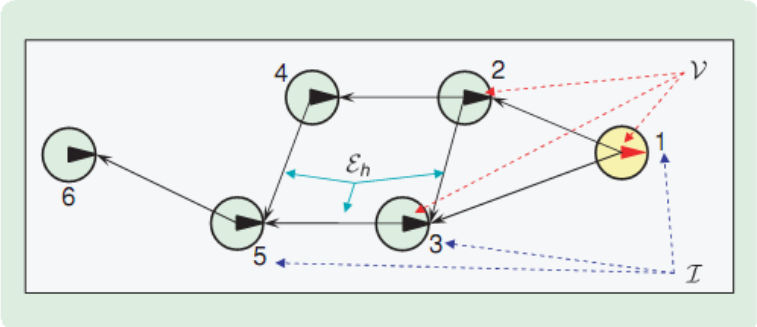


Figura 19 ***Formation Control***

3.2 METODOLOGIAS DE CONTROLO & NAVEGAÇÃO

3.2.1 ARQUITECTURAS DE CONTROLO

Uma arquitectura de robôs refere-se principalmente ao software e hardware que permite realizar o controlo de um robô. Uma placa controladora de um motor a correr um software de controlo, só por si, não constitui uma arquitectura. O desenvolvimento do código em módulos e a comunicação entre funções começa a definir uma arquitectura.

Os sistemas robóticos são complexos e difíceis de desenvolver. Eles integram múltiplos sensores, têm muitos graus de liberdade e é difícil trabalhar em tempo real. Os sistemas existentes actualmente seguem tipicamente arquitecturas robóticas para orientar a construção de dispositivos autónomos e de prestação de serviços computacionais (por exemplo, comunicações, processamento, etc.) para os subsistemas e componentes. Estas arquitecturas têm tendência de ser usadas em domínios específicos e têm falta de aptidão para uma ampla gama de aplicações. Por exemplo, uma arquitectura bem adaptada para operação remota tende a não ser fiável para supervisão ou controlo autónomo.

Uma tendência recente nas arquitecturas robóticas tem sido no comportamento base ou sistemas reactivos. Comportamento base refere-se ao facto de que esses sistemas apresentam diferentes comportamentos, alguns dos quais são emergentes. Estes sistemas são caracterizados por um acoplamento rápido entre os sensores e actuadores, minimizando a computação.

Outra tendência de arquitectura é caracterizada por uma mistura de controlo e fluxo de dados assíncrono e síncrono. Processos assíncronos são caracterizados como vagamente acoplados em eventos impulsionados sem tempo limite. Processos síncronos, em contraste, são fortemente acoplados, utilizam um relógio comum e exige uma execução em tempo real, ou seja, actuar instantaneamente.

3.2.2 RODNEY A. BROOKS, ARQUITECTURA HÍBRIDA

Tradicionalmente os sistemas de controlo de robôs móveis foram sintetizados com base nas informações de transformação dos sinais dos sensores e actuar nos actuadores. [5.]

Esta arquitectura adopta o processamento em paralelo das acções Sensor/Actuador, o que vai em contra aos tradicionais métodos hierárquicos como podemos ver:

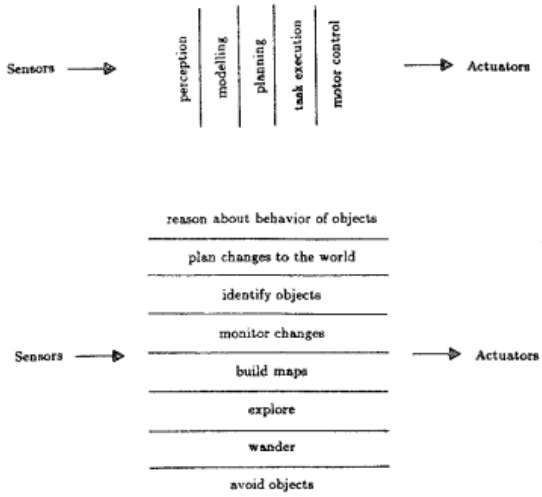


Figura 20 Modelo da arquitectura Híbrida

A ideia é atingir níveis de comportamentos construindo camadas de um sistema de controlo correspondentes a cada nível de competências e simplesmente adicionar uma nova camada de um conjunto existente "nível mais elevado de competências". Existe um sistema de controlo de mais baixo nível, nível zero. Os outros níveis são responsáveis por determinadas funções que se tornam de alto nível quanto mais se sobe no subsistema. Ao contrário das hierarquias tradicionais, esta pode actuar directamente nos actuadores pela camada mais elevada do subsistema.

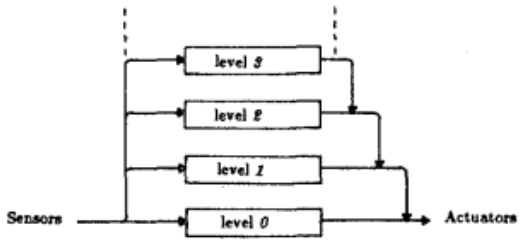
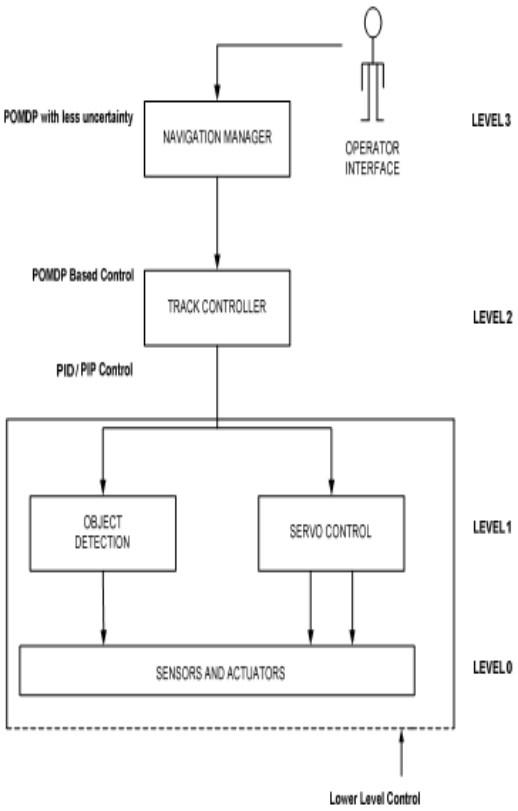


Figura 21 Modelo das camadas

3.2.3 RAHHE AGATE, ARQUITECTURA HIERÁRQUICA

A estrutura desta arquitectura é definida por camadas, em que cada camada interage com as camadas subjacentes entre elas. [25.]

Nível 3: A tarefa deste nível é de planear trajectórias. Este nível pode ser definido como módulo de planeamento de navegação. Este nível tem como objectivo concluir tarefas no tempo mais curto e menor gasto de energia. O algoritmo calcula a trajectória mais segura, contorna obstáculos mas, sem ter em conta as condições do terreno.



Nível 2: Este nível é responsável pela segurança das operações, precisão e movimentos do robô. Este nível recebe os comandos do nível acima. Pode dizer-se que este nível é uma fatia mais fina mas mais precisa do nível 3. O método de controlo usado neste nível é POMDP (*Partially observable Markov Decision*)

Nível 1: Este nível está dividido em 2 módulos, sendo eles, modo de detecção de objectos e modo de controlo de Servo.

Modo de detecção de Objectos:

O modelo sabe como reagir a diferentes tipos de obstáculos podendo assim assegurar o êxito da missão programada.

Modo de controlo de Servo:

Este modo encarrega-se da posição dos servomotores. Controla os servos de maneira a estes não ultrapassarem os graus de liberdade do robô.

Nível 0: Esta camada é a camada de hardware. São os circuitos responsáveis pela percepção sensorial e actuar nos actuadores.

3.2.4 CONTROLO DE ROBÔS MÓVEIS AUTÓNOMOS

As principais estratégias e características de controlo são as seguintes:

Deliberativa: " pensa muito, logo actua".

Reactiva: " Não pensa, reage".

Híbrida: " pensa e actua independentemente, em paralelo"

***Behavior Based*:** " pensa na forma em que actua".

Apresentação de uma breve descrição destas estratégias de controlo.

3.2.5 ESTRATÉGIA DELIBERATIVA

Esta estratégia usa um modelo centralizado do ambiente envolvente para planificar e gerir as acções sobre os robôs. O robô processa todas as informações disponíveis dos sensores e de toda a informação gravada internamente para gerar um plano de acção, isto é, planificar requer a verificação de todos os planos de acção até encontrar um plano com o qual o objectivo possa ser atingido. Planificar é um dos processamentos que requer mais tempo. Se o robô tem que reagir rapidamente então, planificar não é a forma mais prática/eficaz. Porém, se há tempo suficiente, planificar permite ao robô actuar estrategicamente.

Outra característica desta estratégia é a planificação ser feita sobre o mundo. Esse modelo tem que ser actualizado o mais rapidamente possível, já que este muda antes da resposta do robô, o plano será feito com informação desactualizada provocando erros de operação.

3.2.6 ESTRATÉGIA REACTIVA

Esta estratégia de controlo do robô segue um conjunto de regras do tipo condição/acção pré-programadas com o número de estados mínimo. O controlador destes sistemas mapeiam situações específicas com acções específicas. Assim, esses sistemas conectam directamente os sensores e os actuadores.

Existe uma característica importante dos sistemas reactivos puros. É que não usam nenhum sistema interno de representação do estado e não realiza planificação. Eles simplesmente reagem com a informação actual dos sensores. Isso permite que o robô responda de forma rápida as mudanças de estado, típicas nos ambientes não estruturados.

Habitualmente é difícil poder antecipar todas as possíveis situações que o robô pode encontrar. Em lugar disso, situações específicas resultam de acções específicas e essas acções são usadas para solucionar todos os outros casos. Assim, o programador pode reduzir efectivamente o espaço sensorial a somente as situações de entrada que interessam, mapeando estas com as acções apropriadas, simplificando assim, o sistema de controlo.

As limitações dessa estratégia são as seguintes:

- ✓ Os robôs não têm informação do ambiente.
- ✓ Não tem memória.
- ✓ Não tem habilidade de aprender.

As estratégias puramente reactivas têm provado efectivamente uma variedade de problemas que têm como características estarem bem definidos no momento do projecto mas não são inflexíveis nas suas acções devido a sua incapacidade de adaptação.

3.2.7 ESTRATÉGIA HÍBRIDA

A estratégia híbrida combina o melhor que tem as estratégias de controlo reactivo e deliberativo. A parte deliberativa trata das planificações enquanto que a parte reactiva encarrega-se das acções que exigem reagir rapidamente, como evitar obstáculos ou manter-se seguindo uma parede a uma distância fixa.

O desafio dessa estratégia de controlo é fazer trabalhar duas partes sincronizadas. A dificuldade destas sincronizações ocorre na razão das duas estratégias trabalharem com escalas de tempo diferentes. Assim, o sistema requer um componente intermediário entre os níveis de acção deliberativa e reactiva, por isso são chamados de sistema de três níveis ("*three-layer systems*").

Nas arquitecturas de controlo híbrido, o nível reactivo é controlado por um navegador ou sequenciador. Isso permite ao robô ter a capacidade de navegação sofisticada. Ao invés de somente seguir paredes e evitar obstáculos, os sistemas híbridos permitem ao sistema de controlo realizar comportamentos de forma sequencial. Um comportamento sequencial típico para navegar até uma determinada sala seria " seguir a parede, dar a volta no corredor entrar na sala". Este tipo de sequência é difícil de ser implementada num sistema puramente reactivo. Nos sistemas híbridos os comportamentos podem ser activados e desactivados para conseguir objectivos mais abstractos.

Ronald Arkin explica como as principais arquitecturas híbridas abordam a planificação:

> **Selecção**: o algoritmo faz a selecção de diferentes configurações do sistema de controlo reactivo de acordo com o que o robô precisa fazer.
>
> **Aconselhar**: o algoritmo sugere mudanças que o sistema reactivo pode ou não usar.
>
> **Adaptação**: o algoritmo modifica continuamente a configuração do sistema de controlo reactivo.
>
> **Adiamento ("*Postponing*"):** o algoritmo toma a decisão somente no último momento possível, permitindo que as informações sensoriais mais recentes afectem as decisões.

Estas estratégias têm provado efectivamente serem robustas em ambientes e tarefas nas quais modelos internos e planificações podem ser usados em tempo real sejam poucas ou suficientemente independentes da lógica de alto nível. Porém, se o mundo é muito variável, um sistema puramente reactivo pode ser uma solução melhor**.**

3.2.8 ESTRATÉGIA BASEADA NO COMPORTAMENTO "*BEHAVIOR BASED*"

O controlo baseado no comportamento é uma extensão dos sistemas de controlo reactivos. Pode-se dizer que este tipo de estratégia fica entre as estratégias puramente reactiva e deliberativa.

Segundo Mataric, a robótica baseada no comportamento integra os campos da inteligência artificial, engenharia e ciência cognitiva. Essa abordagem é uma metodologia para projectar agentes autónomos e robôs, sendo também uma espécie de arquitectura de agentes inteligentes. As arquitecturas fornecem uma estrutura e impõem limitações na forma em que os problemas de controlo de robôs são resolvidos. A metodologia baseada no comportamento impõe uma filosofia geral, inspirada na biologia, permitindo uma certa liberdade de interpretação. O seu objectivo é desenvolver métodos para controlar sistemas artificiais (geralmente robôs reais, mas também robôs simulados e outros agentes de software autónomos) e usar a robótica para modelar e entender melhor os sistemas biológicos (usualmente animais, desde insectos até humanos).

Fundamentalmente, um controlador desenvolvido utilizando essa estratégia é composto por um conjunto de componentes modulares denominados comportamentos. Um comportamento é uma regra de controlo que tem como missão atingir e manter um objectivo. Os comportamentos são executados em paralelo, e utilizando os dados de entrada dos sensores ou as informações de outros comportamentos, e os transformam em comandos de saída que são enviados aos actuadores do robô ou de outros comportamentos.

As bases dos conceitos da abordagem baseada no comportamento estão na robótica reactiva, especialmente na arquitectura "*Subsumption*". Na arquitectura "*Subsumption*" a tarefa é decomposta em módulos de comportamentos que trabalham em paralelo realizando tarefas específicas como evitar obstáculos e identificar objectos.

Porém, apesar de serem geralmente confundidos na literatura, a estratégia baseada no comportamento é muito mais poderosa do que a abordagem puramente reactiva. Enquanto os sistemas baseados no comportamento apresentam algumas característica dos sistemas reactivos e podem conter módulos (comportamentos) reactivos, o processamento da informação não é limitado a só perceber e reagir. Esses sistemas podem ter estados internos menores e aprendizagem ao longo prazo. Além disso, podem usar diferentes formas de representação interna e realizar processamento computacional nessas representações para decidir que acção executar. Como foi mencionado anteriormente, essas características constituem limitações na abordagem reactiva.

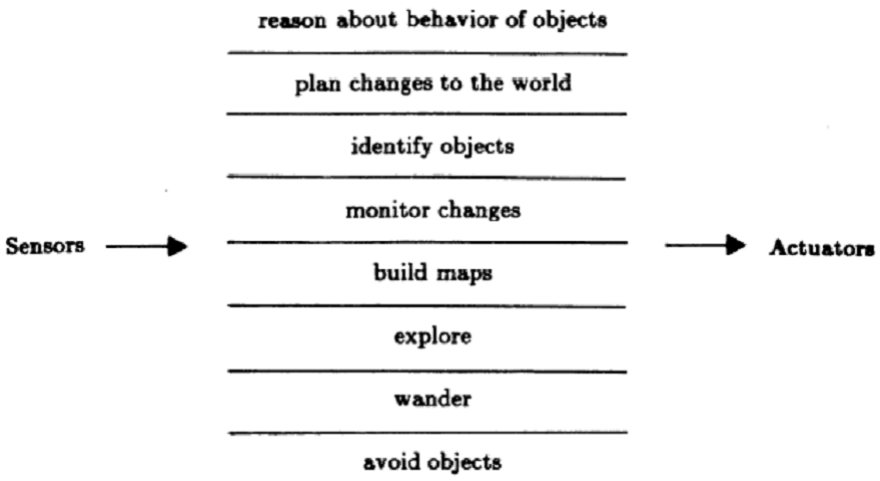


Figura 22 Decomposição de tarefas na arquitectura "*Subsumption*" (BROOKS, 1986)

O desafio do projecto para construir robôs com controlo baseado no comportamento é seleccionar a composição correcta de comportamentos e reflexos para gerar o comportamento global desejado. Usualmente não existe uma conexão visível entre as estruturas de comportamentos internos e o comportamento global produzido. Por exemplo, um robô de limpeza pode não ter um comportamento específico "limpar", ao contrário disso, a sua interacção com o mundo pode resultar em limpar salas. Isso é conhecido como o comportamento emergente e é característica destes sistemas.

3.2.9 *MOTOR – SCHEMA*

O *Motor – Schema* é uma metodologia desenvolvida para especificar e projectar sistemas robóticos baseados no comportamento ("*behavior based*"). Este método permite projectar comportamentos que actuam de forma paralela e distribuída para realizar acções robóticas inteligentes em resposta a um estímulo do ambiente. [24.]

Citando *Arkin*, um esquema é a unidade básica do comportamento a partir do qual acções complexas podem ser realizadas, ele consiste do conhecimento de como agir e perceber, bem como no processo computacional pelo qual o comportamento é implementado.

Na metodologia "*Motor-Schema*" um comportamento é constituído, basicamente, por dois componentes:

- ✓ Esquema perceptivo
- ✓ Esquema motor

O esquema perceptivo engloba as percepções. A filosofia da metodologia "*Motor-Schema*" para o desenvolvimento da percepção é conhecida como "*action oriented perception*". Esse princípio diz que todas as actividades da percepção sempre devem ser vistas do ponto de vista das necessidades motoras.

Assim, o esquema perceptivo fornece as informações do ambiente exclusivamente necessárias para aquele determinado comportamento, permitindo ao robô reagir rapidamente às mudanças do mesmo. Outras características importantes dos esquemas perceptivos é que podem ser definidos recursivamente, isto é, diferentes sub-esquemas perceptivos podem extrair pedaços de informação que serão subsequentemente processados por outro esquema perceptivo, gerando como resultado uma informação mais significativa para o comportamento.

O esquema motor representa a forma da actividade física. Usa a informação fornecida pelo esquema perceptivo para gerar o comando de controlo do robô. Esse comando define a forma que o robô deve responder ao estímulo percebido. A resposta do esquema motor é codificada em forma de vectores e usa o método de navegação em campos potenciais.

O método de campos potenciais foi apresentado por *Khatib* e *Krogh* como base de geração de trajectórias sem obstáculos para sistemas robóticos móveis e robôs manipuladores. Essa metodologia utiliza uma função potencial arbitrária para gerar um campo de forças que representa a velocidade e direcção com as quais o robô deve navegar pelo ambiente. Os objectivos são associados a campos de atracção e os obstáculos a campos de repulsão. O método consiste em construir campos potenciais separados para representar a relação entre o robô e cada objecto dentro do campo sensorial do robô. Esses campos são combinados, mediante superposição, para gerar um único campo potencial. A partir do campo potencial resultante, gera-se um campo de forças, representado por vectores com componentes de

magnitude (velocidade) e direcção, com o qual se pode planificar trajectórias de navegação para os robôs móveis e manipuladores robóticos. Dessa forma o robô pode ser visto como uma partícula num campo de forças. Uma característica importante desse método é que codifica a acção de forma contínua, isto é, não interessa quanto pequeno seja o elemento do ambiente, em cada ponto do espaço sempre se tem um vector de força associada a ele.

Existem cinco tipos de campos potenciais básicos, que podem ser combinados para gerar campos mais complexos.

Campo uniforme: Nesse caso a força tem a mesma direcção e magnitude em todo o campo. Dentro desse campo o robô, não importando onde nem em que posição esteja orientado, sentirá a necessidade de alinhar-se em direcção do campo e navegar com uma velocidade proporcional à magnitude da força. O campo uniforme é usado para gerar o comportamento " ir na direcção X".

Campo perpendicular: Esse campo é orientado perpendicularmente a algum objecto ou parede

Campo de atracção: Neste campo o objecto gera uma força de atracção na sua direcção. Esse campo geralmente é associado ao objectivo do robô.

Campo de repulsão: É o oposto do campo de atracção, nesse caso o objecto gera uma força de repulsão que faz o robô afastar-se dele; está associado aos obstáculos presentes no ambiente.

Campo tangencial: Esse campo é tangencial ao objecto. Pode-se pensar nele como sendo perpendicular às linhas radiais que se originam a partir do objecto. O campo tangencial pode girar em sentido horário ou anti-horário. Esses campos são úteis para dirigir o robô ao redor do obstáculo ou fazer que o robô investigue alguma coisa.

O esquema motor utiliza os campos potenciais para gerar a acção motora do comportamento, porém só calcula a contribuição de cada campo na posição instantânea onde o robô se encontra. Assim, não existe planificação de trajectória, a reacção do robô ao ambiente é calculada tão rápido quanto o processamento sensorial permite.

A figura 24 mostra um exemplo de navegação de um robô desenvolvido utilizando as metodologias "*Motor Schema*". O campo de força completo é apresentado somente para ilustrar a reacção do robô em cada ponto, mas o agente somente calcula a força na posição instantânea onde o robô se encontra. Na metodologia "*Motor-Schema*", diversos esquemas motores foram definidos:

"*Move-ahead*": Mover na direcção particular.

"*Move-to-goal*": Mover na direcção do objectivo detectado. Existem duas versões destes esquemas: balística e controlada.

"*Avoid-static-obstacle*": Afasta-se de obstáculos passivos.

"*Dodge*": Afasta-se de um obstáculo em movimento que esteja a aproximar-se.

"*Escape*": Afasta-se do ponto de intercepção calculado entre o robô e um predador a aproximar-se.

"*Stay-on-path*": Move-se em direcção ao centro de um caminho: estrada ou corredor.

"*Noise*": Move-se em direcção aleatória por um determinado tempo.

"*Probe*": Move-se em direcção a áreas abertas.

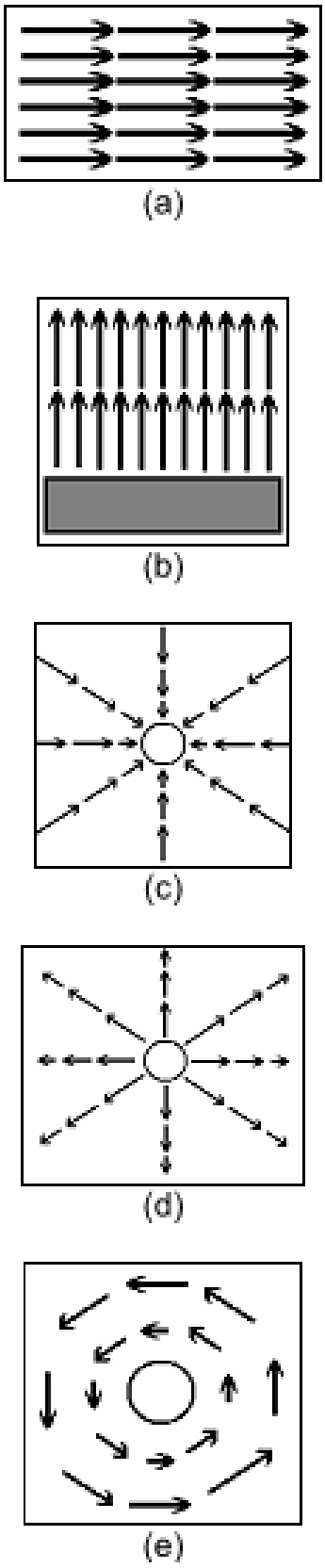


Figura 23 Campos potenciais básicos: (a) campo uniforme; (b) campo perpendicular; (c) campo de atracção; (d) campo de repulsão; e (c) campo tangencial

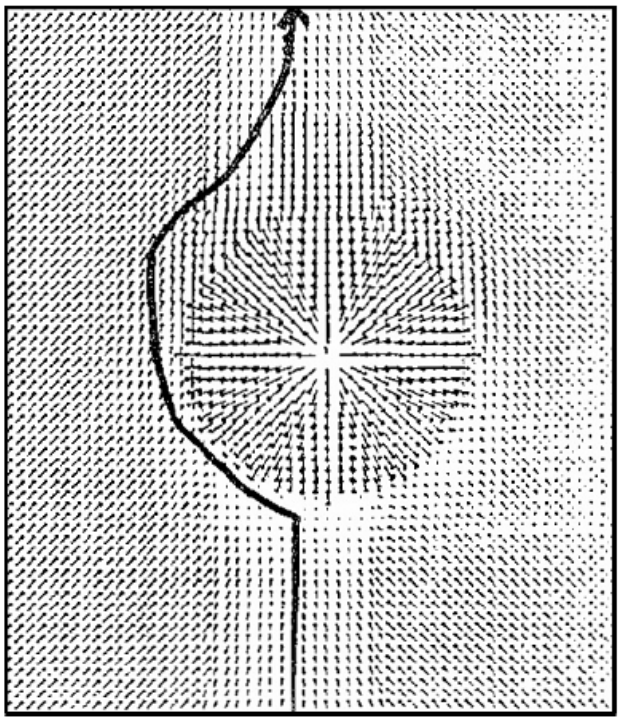


Figura 24 Navegação de um robô desenvolvido seguindo a metodologia "*Motor Schema*" (*ARKIN)*

"*Dock*": Aproxima-se de um objecto com uma determinada direcção.

"*Avoid-past*": Afasta-se de áreas visitadas recentemente.

"*Teleautonomy*": Permite que um operador humano forneça comandos de controlo no mesmo nível de qualquer outro esquema motor.

Estes esquemas motores são os blocos básicos para a navegação de robôs autónomos na metodologia "*Motor-Schema*".

A metodologia de campos potenciais fornece uma forma eficiente de planear trajectórias de navegação e que pode ser calculada em tempo real. Contudo, essa metodologia apresenta vários problemas. O mais característico é o fenómeno conhecido como mínimo local. Um mínimo local é um ponto do campo onde a resultante das forças é zero. Nesse ponto o robô não tem nenhuma força que gera movimento, assim, fica preso no ponto. Existem diversos métodos desenvolvidos para corrigir este problema. Em particular a metodologia "*Motor-Schema*" utiliza o esquema motor "*Noise*" que injecta um ruído aleatório no campo para evitar que o robô fique detido num ponto diferente do objectivo.

O método de coordenação de comportamentos da metodologia "*Motor-Schema*" é o somatório de vectores. Cada vector de saída de um esquema motor é multiplicado por um ganho associado e é somado aos outros vectores de saída. O resultado é um vector que engloba a contribuição de todos os comportamentos activos. Esse vector deve ser normalizado para garantir que seja executável pelo robô. O vector resultante normalizado é enviado ao robô para a sua execução. Esse processo de percepção/acção é repetido tão rápido quanto seja possível. A figura seguinte mostra o diagrama de blocos de um controlador desenvolvido usando a metodologia "*Motor Schema*"

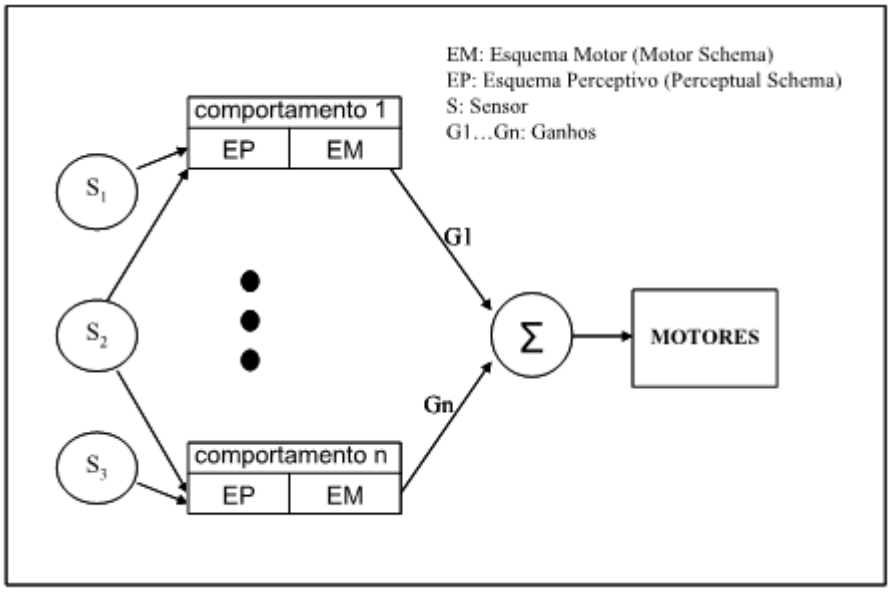


Figura 25 Diagrama de blocos de um controlador usando a metodologia "*Motor Schema*"

4. CASO DE ESTUDO

4.1 O ROBÔ LINCE

O LINCE (*Land INtelligent Cooperative Explorer*) é um veículo terrestre de 4 rodas com direcção Ackerman, semelhante a um *Monster Track* modelo RC TXT-1 da *Tamiya*. Este projecto do LSA privilegia a investigação em robótica móvel todo o terreno para exploração e salvamento em ambientes estruturados e não estruturados e em condições variáveis, tais como cenários urbanos, catástrofe ambientais ao ar livre ou em áreas pouco acessíveis.

Figura 26 LINCE (Land INtelligent Cooperative Explorer)

4.1.1 COMPOSIÇÃO FÍSICA DO LINCE

O LINCE tem uma estrutura mecânica do TXT-1 da *Tamiya*, um modelo RC de um *moster Truck*. Este modelo foi alterado de forma a suportar todo o equipamento electrónico, bem como baterias sensores actuadores e um computador. Os sensores e actuadores estão dispostos numa rede *CANBus* de forma a disponibilizar toda a informação do robô pela rede (*Network sensor*). O núcleo de processamento é um PC com uma *motherboard* da VIA com um processador de 1GHz, 1GB RAM e um MMC de 4GB como disco. Possui uma câmara direccionável da *logitec* para ser utilizada em *streaming* para tele-operação ou visão. Tem um *Accesses Point* (AP) *Wireless* e baterias recarregáveis. Contém um GPS para determinar a posição. Para além deste material o LINCE possui uma placa de gestão de energia e um controlador de motores e servos (*Pandora*) desenvolvidos pelo LSA (Laboratório de Sistemas Autónomos). A última aquisição do LSA para o LINCE foi um *Laser Range Finder, Hokoyo* para detecção e mapeamento de obstáculos.

4.1.2 ARQUITECTURA DE HARDWARE

O LINCE possui uma rede CAN onde todos os sensores e actuadores estão conectados. Desta forma consegue-se interligar todo o hardware de forma mais simples e prepara de melhor forma o hardware para novas alterações. Esta arquitectura baseia-se numa rede de sensores e actuadores como ilustra a seguinte imagem.

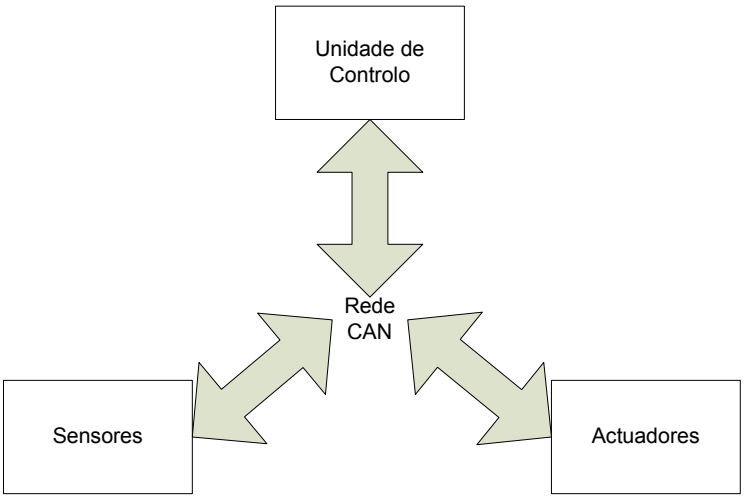
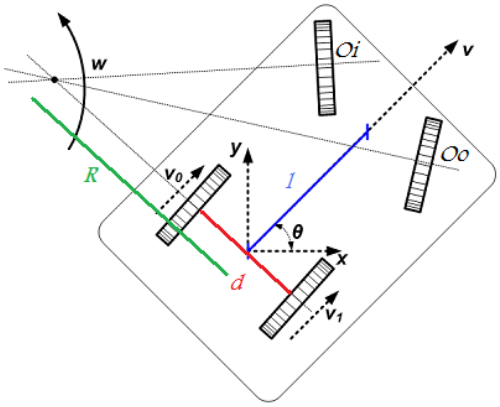


Figura 27 Arquitectura do Hardware

4.1.3 CONFIGURAÇÃO ACKERMAN

Esta configuração é não holonómica e é semelhante a um carro com quatro rodas. Duas das rodas são responsáveis pela tracção do veículo (rodas traseiras) e as outras duas pela direcção (rodas dianteiras). A roda direccional interior apresenta um ângulo superior à exterior, e percorre menor distância. Normalmente esta configuração recorre a um diferencial para as rodas de tracção visto que a roda exterior percorre um caminho maior que a roda interior. [18.]



$$O_i = \tan^{-1}\left(\frac{l}{R-\frac{d}{2}}\right)$$

$$O_o = \tan^{-1}\left(\frac{l}{R+\frac{d}{2}}\right)$$

$$\tan(O_{medio}) = \frac{l}{R}$$

$$R = \frac{l}{\tan(O_{medio})}$$

Figura 28 Configuração Ackerman

Esta configuração pode ser simplificada de forma a arredondar a resultante das rodas de direcção numa única roda de direcção frontal (roda virtual). Esta simplificação tem como nome configuração triciclo. No fundo, tem o mesmo resultado que a configuração Ackerman mas com uma roda frontal que vira com o factor resultante das duas rodas frontais de Ackerman.

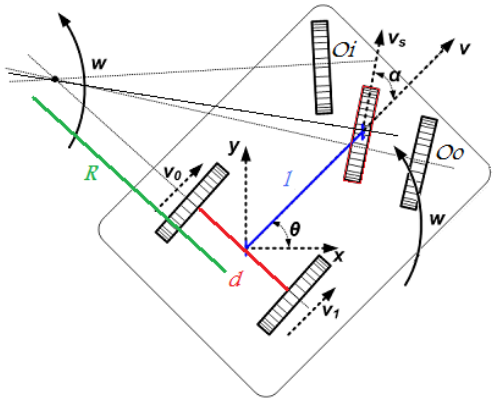


Figura 29 Configuração Ackerman simplificada a triciclo

Esta simplificação pode ser feita, pois a geometria desta configuração é trapeziana. Na figura seguinte está ilustrada a simulação do trapézio de Ackerman. Como podemos observar, o ângulo de curvatura das rodas em função do deslocamento da barra inferior do trapézio. Podemos assim concluir que os ângulos de curvatura de cada roda varia e adapta-se a cada raio de trajectória.

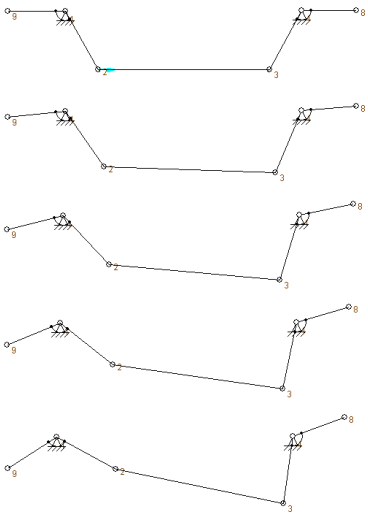


Figura 30 Simulação do trapézio de Ackerman

Assim desta forma, se for aplicado um servomotor que controle o deslocamento da barra inferior podemos controlar o ângulo de curvatura das duas rodas, desta forma estamos a gerar um ângulo de curvatura de uma roda virtual existente no meio do trapézio que é directamente proporcional ao ângulo de curvatura do servomotor.

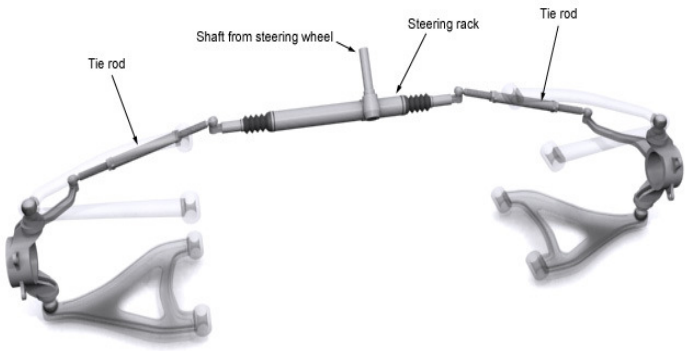


Figura 31 Resultado da geometria Ackerman com servomotor

4.1.4 CINEMÁTICA DO MODELO

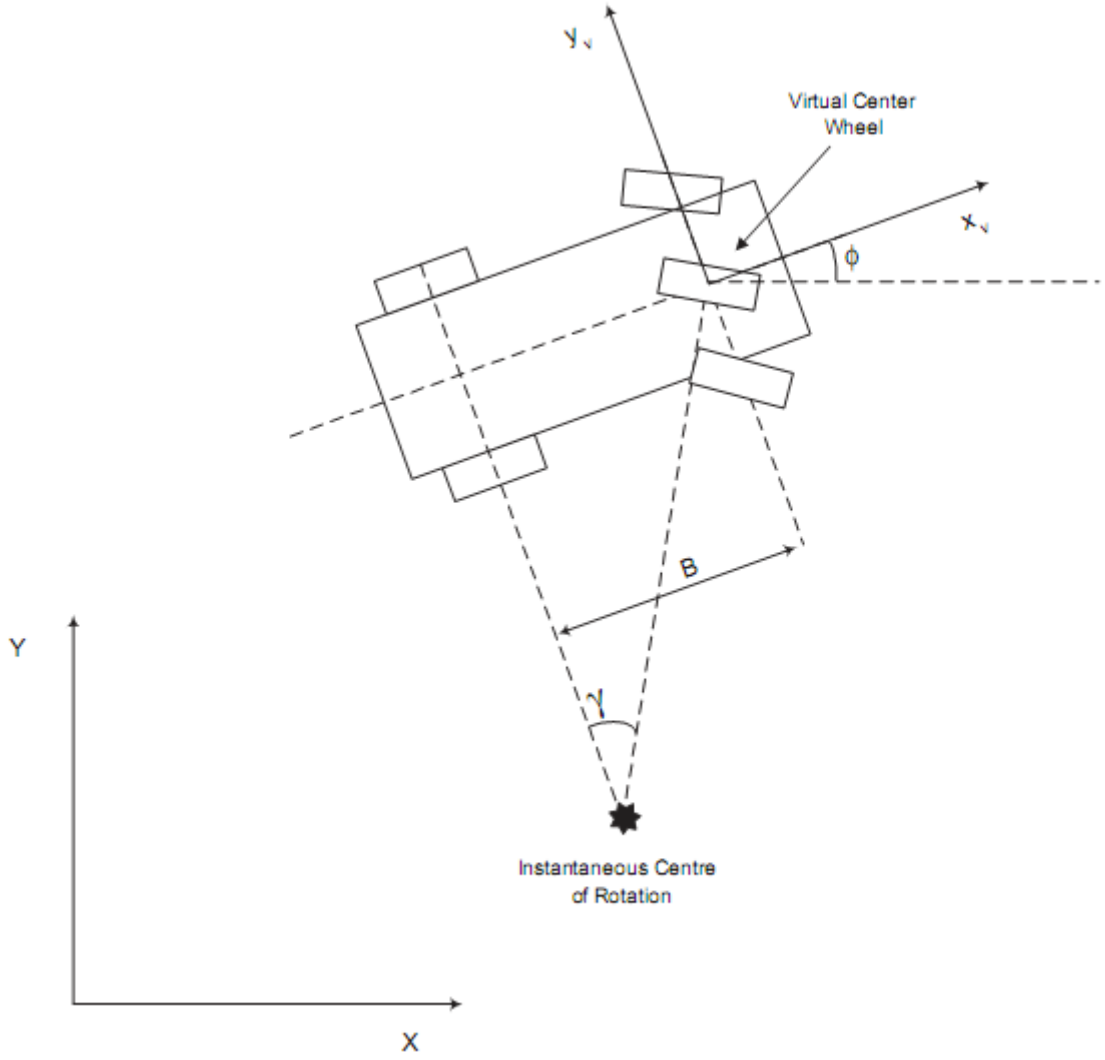


Figura 32 Modelo cinemático

O modelo cinemático permite de forma matemática antecipar o resultado da configuração em causa. A cinemática deste veículo tem como objectivo parametrizar o resultado do deslocamento do veículo no espaço bidimensional *X, Y* de forma a saber a sua posição, velocidade, aceleração, bem como a sua orientação *RPY (roll, pitch, yaw)* no mundo. Os parâmetros de controlo desta configuração são aqueles cuja variação permite alterar o estado do veículo. Estes parâmetros são a velocidade relativa ao solo $V(t) = R(t)\,\omega(t)$ e o ângulo de orientação das rodas frontais ϕ . [8.]

Portanto podemos concluir que o deslocamento e orientação desta configuração depende destes últimos parâmetros e o modelo cinemático é o seguinte:

$$\dot{X}(t) = R(t)\omega(t)\cos[\phi(t) + \gamma(t)]$$
$$\dot{Y}(t) = R(t)\omega(t)\cos[\phi(t) + \gamma(t)]$$
$$\dot{\phi}(t) = \frac{R(t)\omega(t)}{B}\sin[\gamma(t)]$$
$$\dot{R}(t) = 0$$

Figura 33 Equações de espaço de estado

Onde, (modelo cinemático)

$\dot{X}(t)$ - Variação do deslocamento em X

$\dot{Y}(t)$ - Variação do deslocamento em Y

$\dot{\phi}(t)$ - Variação do ângulo de curvatura

$\dot{R}(t)$ - Variação do raio da roda

E, (controlo)

$\omega(t)$ - Velocidade reactivamente ao solo (rad/s)

$\phi(t)$ - Ângulo de curvatura (rad)

Em controlo, a discretização é essencial visto que o modelo contínuo não é computacionalmente viável devido a natureza computacional. Qualquer computador executa uma operação num intervalo de tempo Δt e nunca de uma forma continua. Isto leva a necessidade de aproximar o modelo contínuo a um modelo discreto de forma a ser possível a sua implementação em ambiente computacional.

O resultado da discretização:

$$x(k+1) = x(k) + \Delta T R(k)\omega(k)\cos[\phi(k) + \gamma(k)]$$
$$y(k+1) = y(k) + \Delta T R(k)\omega(k)\sin[\phi(k) + \gamma(k)]$$
$$\phi(k+1) = \phi(k) + \Delta T\frac{R(k)\omega(k)}{B}\sin\gamma(k)$$
$$R(k+1) = R(k)$$

Figura 34 Equações de espaço de estado discreto

Onde, ΔT é o período de amostragem

4.1.5 SENSORES

Os sensores implementados nesta plataforma robótica adquirem relativa importância, pois são estes os responsáveis por realimentar o controlador de forma a orientar o veículo por um mundo gerado pelos sensores. O LINCE possui um conjunto de sensores odometricos, Laser, GPS e câmaras de forma a poder observar o estado do mundo e do próprio robô.

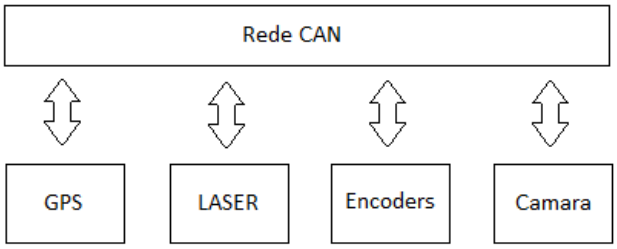


Figura 35 Arquitectura de ligação dos sensores

4.1.5.1 SENSORES ODOMETRICOS

O Lince possui encoders em todas as rodas. Estes encoders conseguem ler com relativa precisão o deslocamento angular de cada roda. Depois esta informação é trabalhada de forma a estimar e corrigir a odometria do robô ao longo do tempo.

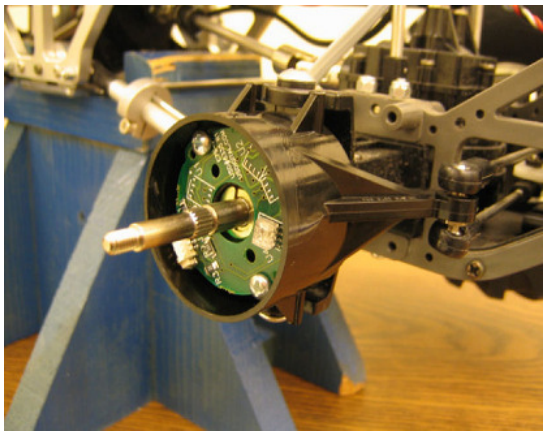


Figura 36 Encoders

4.1.5.2 GPS

O GPS permite que o LINCE obtenha a posição global em caso de deslocamento *outdoor* de forma a poder operar em ambientes exteriores. Este permite também realimentar a odometria conseguindo assim minimizar o erro acumulado por esta de forma a estimar e corrigir um posicionamento do robô com maior precisão

4.1.5.3 LASER

O Laser que o LINCE tem é o Hokoyo. Este laser possui um varrimento horizontal com abertura de 240º e alcance de 5m. Este sensor permite construir um modelo preciso do mundo exterior ao LINCE de forma que este consiga navegar neste mundo gerado.

Figura 37 R313-HOKUYO-LASER3

4.1.5.4 CÂMARA

A câmara permite ao LINCE ter visão robótica. A visão em robótica permite observar o mundo através das cores. O LINCE possui uma câmara da logitec e é direccionável. Esta câmara é principalmente usada para teleoperação, mas o objectivo é que possa ser usada para controlo autónomo em conjunto com algoritmos de visão.

4.2 O MUNDO

No mundo real como sabemos existem enumeras variáveis, condições e restrições, de tal forma que nem o mais avançado método de simulação consegue simular de forma realista e correcta o ambiente real. Contudo, existem enumeras condições que podemos estudar, nomeadamente:

- ✓ Acção da gravidade
- ✓ Cinemática
- ✓ Dinâmica
- ✓ Forças
    - o Força de atrito
- ✓ Momento de inércia
- ✓ Momento linear

4.2.1 ACÇÃO DA GRAVIDADE

Corpos com massa estão sujeitos a uma força de atracção entre eles, essa força é conhecida como força da gravidade e a lei da gravidade (descoberta por Newton).

A força gravitacional é proporcional ao produto das massas dos corpos e inversamente proporcional ao quadrado da distância entre os corpos.

$$F = G\frac{M_1 M_2}{R^2}$$

O peso de um corpo é a força com que ele é atraído gravitacionalmente pela Terra. Para um corpo na superfície da Terra (ou perto dela), o peso de um corpo de massa m é dado por:

$$P = m.g$$

Onde:

g é a aceleração da gravidade, que em módulo vale 9,81 m/s^2, mas é comum aproximar g para 10 m/s^2.

Quanto mais afastado estiver um corpo do centro da Terra menor será a força gravitacional exercida entre a Terra e o corpo. Assim, a força diminui quando a distância entre o corpo e o centro da Terra aumenta.

Esta consideração deve ser levada em conta quando projectamos robôs autónomos aéreos, visto que dependendo da altitude que voem, podem estar sujeitos a força gravítica mais baixa.

O peso de um corpo na Terra:

- ✓ Aumenta do equador para os pólos: por dois motivos:
- ✓ Achatamento nos pólos (a Terra não é uma esfera perfeita)
- ✓ Devido à acção da força centrífuga da rotação da Terra, que "empurra" os corpos para fora, reduzindo o seu peso (força que causa o achatamento polar entre outros efeitos naturais).
- ✓ Diminui quando a altitude do lugar aumenta

Dependendo das situações estes parâmetros podem influenciar o desempenho dos robôs autónomos. Os estimadores devem conhecer as alterações existentes da acção da gravidade pois por esse mesmo motivo podem geram erros de estimação o que pode levar a acidentes e perda do equipamento em causa.

Outra preocupação é como a aceleração não é constante em todo o planeta, os valores dos acelerómetros também não o são. Portanto, o projectista deve preocupar-se com esta realidade, mas sobretudo se justificar. Normalmente os veículos que justificam estas preocupações são aviões e submarinos.

4.2.2 CINEMÁTICA

Pode dizer-se que a modelação cinemática permite a descrição do movimento de um robô em termos da sua posição, velocidade e aceleração. A cinemática abstrai-se totalmente dos conceitos de massa, inércia e força ignorando-os.

4.2.2.1 POSIÇÃO

A posição de um robô depende da configuração da mecânica do mesmo. Contudo, independente deste pormenor, a posição pode ser estimada em função do movimento das rodas do robô.

Para isso, os sensores odometricos "encoders" contam os deslocamentos entre arcos de roda percorridos estimando assim de forma precisa o deslocamento da roda em função do solo.

$$x(k) = k.\overline{A}$$

Onde,

K é o número de tics do *encoder*.

$\overline{A}$ é o arco de roda de cada tic.

4.2.2.2 VELOCIDADE

A velocidade de um robô, depende da configuração da mecânica do mesmo como já foi dito anteriormente. Contudo independente deste pormenor, a velocidade pode ser estimada em função do movimento das rodas do robô.

Desta forma e com a leitura dos *encoders* obtendo a velocidade das rodas integrando a posição da mesma. Quando o sistema que recebe novo valor de posição, esta é subtraída ao valor anterior e dividida pelo período entre amostras "integração" calculando assim o valor da velocidade no instante entre $t_e t_{-1}$

$$\omega(\mathrm{k}) = \frac{x_k - x_{k-1}}{\Delta \mathrm{T}}$$

4.2.2.3 ACELERAÇÃO

A aceleração é obtida integrando a velocidade do robô. Para isso, o sistema tem que possuir mecanismo de memória de forma a guardar pelo menos o estado anterior da velocidade. Quando o sistema que calcula a aceleração recebe novo valor de velocidade, esta é subtraída ao valor anterior e dividida pelo período entre amostras "integração" calculando assim o valor da aceleração no instante entre $t_e t_{-1}$

$$a(k) = \frac{\omega_k - \omega_{k-1}}{\Delta T}$$

4.2.3 DINÂMICA

A dinâmica é o ramo da física que trata do movimento dos corpos sob acção de forças. Isto inclui cinemática, que é o estudo do movimento como vimos anteriormente. A dinâmica preocupa-se em estudar o movimento dos corpos com referência as forças que dão origem ao movimento, inércia e forças externas relacionando-as de forma a estudar o efeito das forças resultantes. Na dinâmica dos corpos rígidos, no caso de existir movimento de rotação, deve-se ter em linha de conta o momento de inércia.

4.2.4 FORÇAS

Uma força é uma acção física de carácter vectorial responsável pelas mudanças do estado de movimento dos robôs.

Além de forças aplicadas como a gravítica e a de tracção, podem actuar, num robô, forças de ligação que lhe restringem o movimento (tensões de fios, reacções de superfícies). As forças de ligação caracterizam-se por tomarem valores sempre dependentes das forças aplicadas, enquanto estas assumem valores que não dependem da acção das outras. Os valores das forças de ligação, que a priori são desconhecidos, podem ser determinados com base nos valores da força aplicada e no conhecimento do estado cinético do robô.

Para aplicar as Leis de Newton, devemos atender aos seguintes procedimentos:

- ✓ Identificar a partícula "robô"cujo movimento se pretende estudar.
- ✓ Verificar quais os corpos que, nas proximidades, exercem forças aplicadas e forças de ligação sobre o robô em estudo.
- ✓ Escolher um referencial com dois eixos, um tangente e outro normal à trajectória.
- ✓ Construir o diagrama de corpo livre do robô ou componentes em estudo, mostrando o referencial e todas as forças que actuam. Se intervier mais do que uma partícula, faz-se um diagrama independente para cada uma delas.
- ✓ Projectar, nos eixos do referencial escolhido, todas as forças que não tenham as direcções desses eixos.

Utilizar a equação vectorial $\vec{F}_{res} = m.\vec{a}$, na forma:

$$F_t = m \cdot a_t \qquad F_c = m \cdot a_c$$

Para isso é necessário calcular a resultante das forças segundo a tangente à trajectória, Ft, e aplicar $\vec{F}_t = m.\vec{a}_t$. Se Ft = 0, também at, = 0 (movimento uniforme);

se $F_t \neq 0, a_t \neq 0$ (movimento variado).

Quanto à resultante segundo a normal à trajectória, Fc, aplica-se $\vec{F}_c = m.\vec{a}_c$. Se Fc, = 0, também ac = 0 (trajectória rectilínea); se $F_c \neq 0$, também $a_c \neq 0$ (trajectória curvilínea).

4.2.5 FORÇA DE ATRITO

O atrito tem características relevantes em diversos sistemas de controlo. Em consequência do atrito podem ocorrer erros de seguimento, surgimento de movimentos de aderência e deslizamento indesejáveis.

Nos robôs, o atrito pode aparecer nas transmissões, nos motores, cilindros pneumáticos ou hidráulicos, sensores de posição, rolamentos, freios, contactos dos actuadores, etc. Assim, o atrito continua a ser um dos grandes problemas a ser entendido e estudado. Neste sentido, diversos trabalhos têm surgido com diferentes técnicas de compensação baseadas no modelo com o objectivo de reduzir os seus efeitos através de uma representação mais fiel do seu comportamento dinâmico. Através das técnicas baseadas no modelo é possível prever e compensar os efeitos do atrito, analisar estabilidade, prever ciclos limites, ajustar os ganhos dos controladores, etc.

As forças de atrito são muito importantes. Por um lado, provocam desgaste nas peças móveis das máquinas e são responsáveis pelo aumento de energia interna (as peças aquecem). Por outro lado, sem atrito não haveria transmissão do movimento por correias, os robôs não poderíamos caminhar, nem se mover por rodas ou qualquer tipo de tracção. Sem força de atrito não seria possível haver qualquer tipo de movimentação.

Chama-se de força de atrito dinâmico a força que surge entre as superfícies que apresentam movimento relativo de deslizamento entre si. A força de atrito dinâmico opõe-se sempre a este deslizamento, e actua nos corpos de forma a sempre contrariá-lo (tentar impedi-lo), mas nem sempre mostra-se oposta ao movimento observado do corpo.

$F_{a.c} = \mu_c.N$

$\boldsymbol{\mu_d}$ é o coeficiente de atrito dinâmico

$\boldsymbol{N}$ é a força normal à direção do movimento

A força de atrito estático é a que se opõe ao início do movimento entre as superfícies, ou ao atrito de rolamento de uma superfície sobre outra, por exemplo, um pneu.

$F_{ae\max} = \mu_e.N$

$\boldsymbol{\mu_e}$ é o coeficiente de atrito estático.

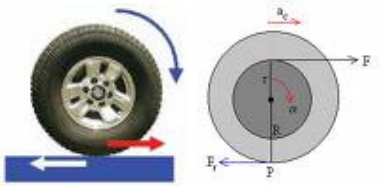


Figura 38 Força de atrito e a roda

4.2.6 MOMENTO DE INÉRCIA

Em Mecânica, o momento de inércia mede a distribuição da massa de um corpo em torno de um eixo de rotação. Quanto maior for o momento de inércia de um corpo, mais difícil será fazê-lo girar. Contribui mais para a elevação do momento de inércia a porção de massa que está afastada do eixo de giro. Um eixo girante fino e comprido, com a mesma massa de um disco que gira em relação ao seu centro, terá um momento de inércia menor que este. Sua unidade de medida, no SI, é quilograma vezes metro ao quadrado (kg·m²).

Por definição, o momento de inércia J de uma partícula de massa m e que gira em torno de um eixo, a uma distância r dele, é

$$J = mr^2$$

Se um corpo é constituído de *n* massas pontuais (partículas), o seu momento de inércia total é igual à soma dos momentos de inércia de cada massa:

$$J = \sum_{k=1}^{n} m_i r_i^2$$

É importante que o sistema robótico seja capaz de sobreviver, isto é, o robô deve medir a sua posição angular *roll*, *pitch* e *yaw* e medindo o momento de inércia, pode-se prevenir que o sistema entre em capotamento eminente.

4.2.7 MOMENTO LINEAR

Sendo o momento linear de uma partícula igual a $\vec{p} = m\vec{v}$, um sistema de n partículas terá momento linear $\vec{p} = sist$:

$$\vec{p}(sist) = \vec{p}_1 + \vec{p}_2 + \vec{p}_3 ..... + \vec{p}_n$$

Ou $\vec{p}(sist) = \sum_{i=1}^{n} m_1 + v_2$ sabendo que $\vec{p}(sist) = \sum_{i=1}^{n} m_1 + v_2 = m\,\vec{v}_{CM}$ então: $\vec{p}(sist) = m\,\vec{v}_{CM}$

Isto é, o momento linear de um sistema de partículas materiais é igual ao produto da massa de todo o sistema pela velocidade do seu centro de massa. Derivando, em ordem ao tempo, a expressão de $\vec{p} = sist$ e considerando a massa constante, fica:

$$\vec{p}'(sist) = m \cdot \vec{a}_{CM}$$

Comparando esta última expressão com $\sum_{i=1}^{n} \vec{F}_{ext} = m\,\vec{a}_{CM}$ chega-se á igualdade:

$$\vec{p}'(sist) = \sum_{i=1}^{n} \vec{F}_{ext}$$

A derivada, em ordem ao tempo, do momento linear de um sistema de partículas materiais é igual à soma de todas as forças exteriores ao sistema. Note-se que, nesta expressão, válida para qualquer tipo de sistema (corpos rígidos ou não), também não figuram as forças interiores. A Lei da Conservação do Momento Linear; já verificada experimentalmente para um sistema de duas partículas, vai ser deduzida para um sistema com qualquer número de partículas, a partir da última expressão. Assim, se a resultante das forças exteriores for nula, isto é, se $\sum_{i=1}^{n} \vec{F}_{ext} = \vec{0}$ (sistema isolado), então $\vec{p}(sist) = 0$. Onde se conclui que o momento linear do sistema, $\vec{p}(sist)$, **é constante**. Se houver alguma alteração no sistema (colisão, explosão) o momento linear inicial é igual ao final, $\vec{p}_i(sist) = \vec{p}_f(sist)$. O momento linear nos sistemas autónomos permite simular ou prever o acontecimento de colisões com corpos externos de forma a corrigir a trajectória após colisão.

5. ESCOLHA DO SIMULADOR

Os simuladores robóticos são na maior parte das vezes usados para criar algoritmos de controlo para um determinado robô sem necessidade de testar os mesmos no hardware, poupando assim custos e tempo de concepção. Em alguns casos, estas aplicações podem ser transferidas para o robô sem que haja qualquer tipo de modificações no próprio algoritmo. O termo simulador robótico pode referir-se a vários tipos de simulação. Por exemplo, em robótica móvel, aplicações com base em simular o comportamento permitem aos usuários criar mundos simples de objectos rígidos e fontes de luz para que os algoritmos criados possam interagir com esses mundos.

Um dos mais populares métodos de simulação robótica é modelização de ambientes 3D de um ou vários robôs. Este tipo de software de simulação robótica é capaz de simular o movimento real de um robô em um verdadeiro ambiente simulado. Alguns simuladores robóticos, utilizam um motor de física permitindo assim simular não só a cinemática mas também a dinâmica e colisões do robô em ambiente 3D. A utilização de um simulador robótico para o desenvolvimento de um algoritmo de controlo é bastante recomendado, independentemente do facto de o robô real estar ou não disponível. O simulador permite que algoritmos de controlo possam ser escritos e testados *off-line* de forma que quando a versão final do algoritmo esteja pronta possa ser testada no hardware real.

5.1 SIMULADORES ROBÓTICOS

Na actualidade existe uma grande variedade de simuladores robóticos. Contudo o utilizador tem que se orientar para o simulador onde obtém melhor proveito face ao produto que deseja alcançar, bem como em qual sistema operativo que este deve correr e que tipo de simulação deseja alcançar. Com uma pesquisa prévia desenhou-se uma lista de simuladores que se classificam como *Open-Source* e Comerciais.

5.1.1 SIMULADORES *OPEN SOURCE*

- ✓ *Blender*
- ✓ *OpenSim*
- ✓ *Simbad 3D Robot Simulator*
- ✓ *Breve*
- ✓ *Gazebo*

5.1.2 SIMULADORES COMERCIAIS

- ✓ *AnyKode Marilou*
- ✓ *WebotsPython e MATLAB.*
- ✓ *Microsoft Robotics StudioRoboLogix*

5.2 ESTUDO DOS SIMULADORES *OPEN SOURCE*

5.2.1 BLENDER

O *Blender* é um conjunto de ferramentas que permite a criação de vastos conteúdos 3D. Oferece funcionalidades completas para modelização, animação, pós-produção, criação e visualização de conteúdo 3D interactivo, com os benefícios singulares de portabilidade numa aplicação com cerca de 5MB. [3.]

Dirigido a profissionais e artistas da área da simulação, o *Blender* pode ser utilizado para criar visualizações de espaços tridimensionais, imagens estáticas, bem como vídeos de alta qualidade, incorpora ainda um motor 3D em tempo real, que permite também a criação de conteúdo 3D interactivo, para reprodução *stand-alone*.

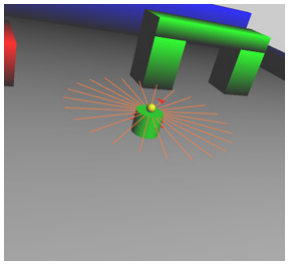
Originalmente desenvolvido pela empresa *'Not a Number' (NaN),* o *Blender* é agora desenvolvido como *Open Source* e o seu código fonte está disponível sobre a licença *GNU GPL.*

5.2.2 OPENSIM

O Projeto *OpenSimulator* é um Servidor de Mundos Virtuais que pode ser utilizado para criar e desenvolver Ambientes Virtuais em 3D. Ele vem sendo desenvolvido por vários parceiros. Pronto para uso, o *OpenSimulator* pode ser utilizado para criar um ambiente semelhante ao *Second Life*, capaz de rodar em modo *standalone* ou conectado à outras instâncias de *OpenSimulator* através da tecnologia de *grid* embutida. Ele também pode ser facilmente estendido para produzir aplicações interactivas em 3D mais especializadas. [21.]

O *OpenSimulator* é escrito em C#, e pode rodar sob o *runtime Mono* ou *Microsoft. NET* Devido à sua natureza limpa e modular é possível estender funcionalidades através de módulos *plug-in* para atender sua aplicação.

5.2.3 *SIMBAD 3D ROBOT SIMULATOR*



Simbad é um simulador 3D de robôs simulador em linguagem *Java* para fins científicos e educacionais. É essencialmente dedicado a programadores que querem uma simples base para estudar o comportamento de algoritmos de Inteligência Artificial, no contexto da Robótica Autónoma e agentes Autónomos. *Simbad* permite que os programadores escrevam os algoritmos de controlo do robô de forma a poderem modificar o ambiente e utilizar os recursos disponíveis dos sensores. É um produto aberto para testar ideias de controlo. O projeto *Simbad* está hospedado no *SourceForge*. O simulador é livre para usar e modificar de acordo com as condições da *GNU General Public Licence*. [26.]

5.2.4 *BREVE*

Breve é um simulador *open-source* que torna mais fácil construir simulações 3D de sistemas multi agentes e modelos de robôs. Utilizando *Python*, ou utilizando uma linguagem simples chamado *Steve*, pode definir-se o comportamento dos agentes em um mundo 3D e observar como eles interagem. *Breve* inclui simulação física e detecção de colisões, para que se possa simular de uma forma realista os robôs. Utiliza o motor *OpenGL* de forma a visualizar a simulação. *Breve* está disponível para *Mac OS X*, *Linux* e *Windows*.

5.2.5 *GAZEBO*

Gazebo é um simulador multi-robótico para ambienteis interiores e exteriores. É capaz de simular uma população de robôs, sensores e objectos, mas fá-lo num mundo tridimensional. Simula sensores de forma realista, objetos, colisões e dinâmicas. O *Gazebo* é normalmente usado em conjunto com o dispositivo *Player* (servidor). O *Player* fornece uma camada de rede através da qual os controladores do robô (clientes) podem interagir com o *hardware* do robô. Este quando utilizado com o *Gazebo*, dispõe de dados simulados, no lugar dos dados reais dos sensores. O *Gazebo* também pode ser controlado através de uma API de baixo nível C (*libgazebo*). Estas bibliotecas permitem que os programadores se concentrem mais a fundo nos algoritmos de controlo a implementar. [22.]

5.3 ESTUDO DOS SIMULADORES COMERCIAIS

5.3.1 *ANYKODE MARILOU*

O *AnyKode Marilou* permite desenhar cenários complexos. Este simulador permite criar e testar algoritmos para frotas de robôs, sem ter que investir em *hardware*, ou lidar com a quebra e recursos limitados. Permite reutilizar os modelos criados, sensores, actuadores e superfícies. È um simulador muito completo e permite enumeras linguagens de programação. [1.]

5.3.2 *WEBOTS*

Webots é um simulador utilizado para programar e simular robôs móveis. Com *Webots* o usuário pode projectar configurações robóticas complexas, com um ou vários robôs, sejam homogéneos ou heterogéneos num ambiente compartilhado. As propriedades de cada objecto, como forma, cor, textura, massa, atrito, etc, são escolhidos pelo utilizador. Uma grande escolha de sensores e actuadores simulados está disponível para equipar a cada robô. Os controladores do robô podem ser programados com o *built-in IDE*. O robô pode ser testado no comportamento fisicamente realista. O *Webots* é utilizado por mais de 650 universidades e centros de pesquisa em todo o mundo. Com *Webots*, pode-se tirar vantagem de uma tecnologia que tem sido desenvolvida pelo Instituto Federal Suíço de Tecnologia, em Lausanne, exaustivamente testado, bem documentado e mantidos por mais de 10 anos. [29.]

5.3.3 *MICROSOFT ROBOTICS STUDIO*

O *Microsoft Robotics Studio* pode ser usado com uma variedade de plataformas robóticas. De modo geral, basta seguir as instruções para configurar o *hardware* e os meios de comunicação com o PC executando *Windows XP* (ou Windows Vista). As informações a seguir podem ser úteis na configuração de robôs com suporte a tutoriais na pagina do fabricante. O *Microsoft Robotics* usa o Visual *Programming Language* como meio de programação de todo o ambiente de simulação. [16.]

5.4 CONCLUSÃO

Um dos requisitos para este trabalho foi encontrar um simulador 3D de robôs *Open Source* que corre numa plataforma *Linux*. Outro requisito foi que este mesmo fosse livre de escolher a linguagem de programação a adoptar. Para tal, o simulador escolhido foi o *Gazebo Project* pois para além de cumprir com os requisitos previamente estabelecidos, é também um dos melhores e mais utilizados simuladores de robôs da actualidade.

Simulador	***Linux***	***Open Source***	**Vasta game de linguagens de programação**	**Simulação 3D**	**Simulação de múltiplos robôs**
OpenSim	Sim	Sim	Não	Sim	Sim
Simbad	Sim	Sim	Não	Sim	Sim
Breve	Sim	Sim	Sim	Sim	Sim
Gazebo	Sim	Sim	Sim	Sim	Sim
Blender	Sim	Sim	Não	Sim	Não
AnyKode	Sim	Não	Sim	Sim	Sim
Webots	Sim	Não	Sim	Sim	Sim
MRSR	Não	Não	Sim	Sim	Sim

Tabela 1 Prós e contras dos simuladores propostos

No anexo A segue um guia de instalação do simulador, por favor consultar antes de avançar.

6. PLAYER/STAGE/GAZEBO PROJECT

O projecto *Player/Stage/Gazebo (PSG)* utiliza ferramentas de *software* livre para criar uma *framework* para pesquisas na área de robótica e sistemas de sensores. O projecto é composto por três módulos: um servidor (*Player*) e dois simuladores, sendo um para ambientes 2D (*Stage*) e outro para 3D (*Gazebo*).

Este projecto é capaz de simular uma população de robôs, sensores e objectos em ambiente interior ou exterior e fá-lo em um mundo virtual 2D ou 3D. Gera de uma forma realista o comportamentos de uma variedade de sensores, objectos, colisões e dinâmicas. [27.]

6.1 SERVIDOR *PLAYER*

O *Gazebo* é normalmente usado em conjunto com o *Player*, esta aplicação é um servidor que fornece uma *interface* de comunicação em rede (cliente), através da qual se pode aceder aos controladores do robô podendo interagir com o *hardware* do mesmo. Como

funciona em rede, os programas clientes comunicam com o servidor através de troca de mensagens, suportando vários protocolos, entre eles *TCP/IP*, *CORBA* e *JNI*, sendo mais comum a comunicação via *TCP/IP*. Quando utilizado com o *Gazebo*, o *Player* dispõe de dados simulados, no lugar dos sensores reais.

O *Gazebo* também pode ser controlado através de uma *API* (*Application Programming Interface*) em linguagem C/C++ de baixo nível (*libgazebo*). Estas bibliotecas permitem que os programadores se concentrem mais a fundo nos algoritmos de controlo a implementar.

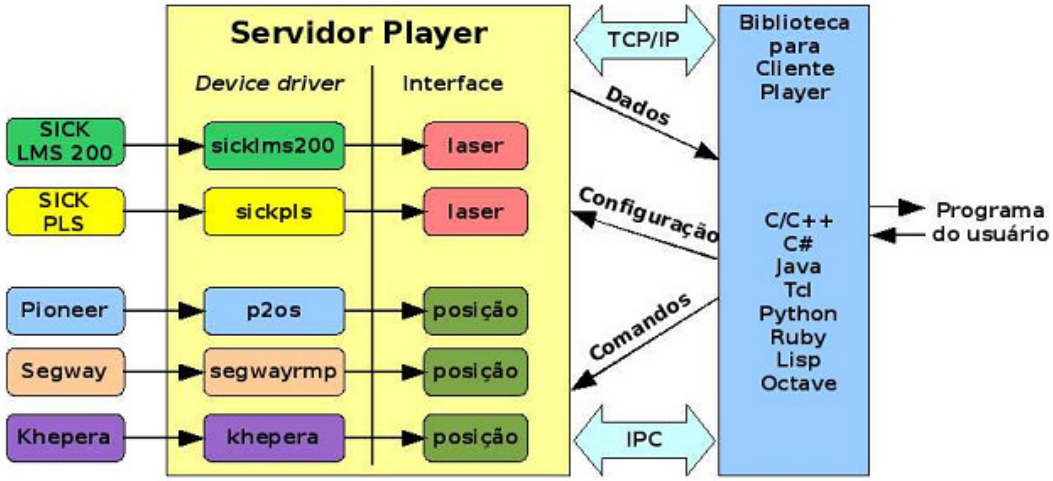


Figura 39 Modelo cliente servidor do *Player*

O servidor *Player* comunica com cada dispositivo específico do robô utilizando *device drivers* e disponibilizando aos seus clientes interfaces abstractas para tais dispositivos, isto é, para o servidor se comunicar com o robô e aceder aos seus dispositivos, é necessário um *driver* específico para aquele tipo de robô, no entanto, o programa cliente apenas necessita saber qual dispositivo aceder, como, por exemplo, sonar, laser e odometria. Com isso, a aplicação cliente não precisa ser reescrita para ser executada em outro robô. Tais *drivers* são mantidos como *plug-ins* para o *Player* e são seus arquivos de configuração que determinam quais *device drivers* serão usados. Os sensores e actuadores são tratados da mesma forma que dispositivos de *hardware* em sistemas operacionais *GNU/Linux*, na forma de ficheiros. Para ler os sensores, são abertos para leitura os ficheiros correspondentes aos mesmos, e para enviar comandos aos actuadores, são abertos para escrita os ficheiros correspondentes aos mesmos.

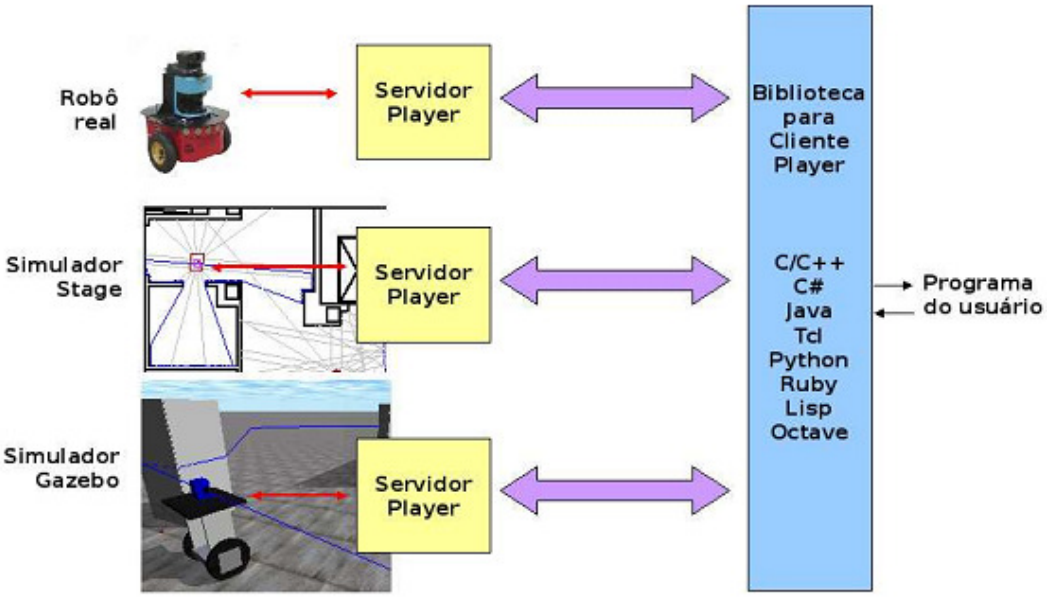


Figura 40 Modelo de abstracção do Hardware

O *Player* suporta vários tipos de *hardwares* e robôs, entre eles estão os robôs *Khephera* e *Pioneer* e também disponibiliza algoritmos para controlar e sentir eventos. A lista do *hardware* suportado pode ser encontrada no site do Projecto.

Ainda, por ser distribuído sob licença *GPL* e por sua arquitectura modular, torna-se fácil criar suporte a novos *hardwares*.

6.2 SIMULADOR 2D *STAGE*

O *Stage* faz parte do projecto *Gazebo*, este simulador permite também simular um ou uma população de robôs móveis, sensores e objectos num ambiente bidimensional "*bitmap*". Para além da diferença obvia entre o *Gazebo* (3D) e o *Stage* (2D), o segundo é mais utilizado para desenvolver algoritmos de controlo para populações de robôs de número elevado, visto que este é computacionalmente mais leve que o *Gazebo*.

Assim como o *Gazebo*, o *Stage* não simula dispositivos específicos, como, por exemplo, um robô *Pioneer*. Em vez disso, ele possui um modelo configurável de uma interface abstracta para o dispositivo, como "laser " e "posicionamento", os quais são configurados num ficheiro "*world*", que contém a descrição do ambiente a ser simulado. Sendo assim, pode-se criar uma *interface* que seja a mais próxima da usada por um robô sem ter que

codificar algo específico da arquitectura do mesmo. Os modelos de sensores e actuadores estão disponíveis ao *Player* através das suas *interfaces* padrões, como por exemplo, laser e sonar. Como o acesso aos sensores e actuadores é feito através das *interfaces* padrões dos mesmos, para a aplicação cliente não importa se o ambiente/robô é simulado ou real.

Vale ressaltar que os *device drivers* do *Player* são usados tanto pelo *Stage* quanto pelo *Gazebo*. Para se usar o *Stage* com o *Player* em vez de dispositivos reais, basta carregar o driver do *Stage* no *Player*. Por padrão, o *Stage* é executado em tempo-real, com os modelos sendo actualizados num intervalo fixo e configurável de tempo. Mas ele também pode ser executado em modo optimizado (*fast mode*), não esperando assim pelo relógio de tempo-real e, com isso, sendo executado frequentemente de modo mais rápido que quando operando em modo de tempo-real.

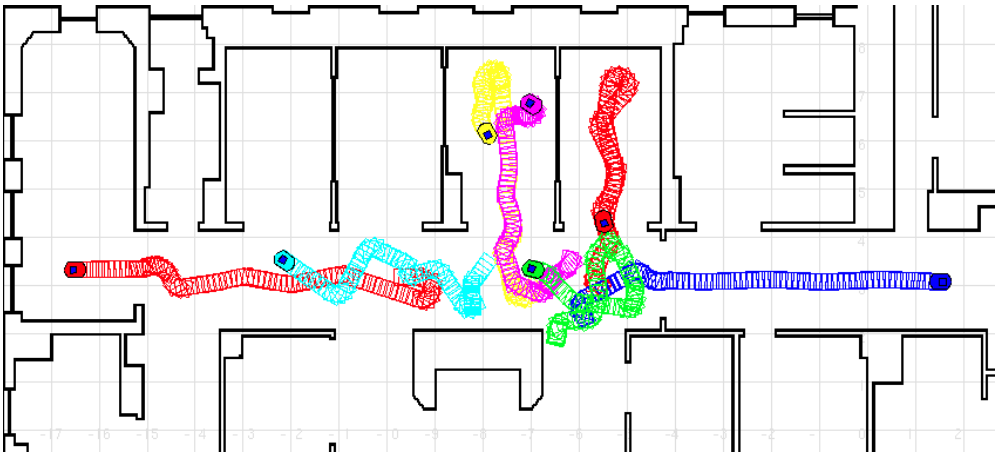


Figura 41 Simulação em Stage

6.3 SIMULADOR 3D *GAZEBO*

Gazebo é um ambiente tridimensional simulado para reproduzir a dinâmica de um ambiente que um robô pode encontrar. Neste ambiente, todos os objectos têm massa, velocidade, atrito e inúmeros atributos que permitem que se comportem como no mundo real, podendo ser puxados, empurrados, carregados, etc.

Na simulação, os robôs são compostos por estruturas dinâmicas conectadas por juntas. Forças lineares e angulares podem ser aplicadas às superfícies e às juntas para gerar locomoção e interacção com o ambiente.

O mundo é descrito por meio de paisagens, construções e outros objectos criados pelo usuário. Todos os aspectos da simulação são controláveis, desde as condições de iluminação até os coeficientes de atrito. Os controlos são feitos através da biblioteca do *Gazebo*, que oferece meios para ajustar vários atributos como a velocidade das rodas, leitura dos dados de um dispositivo de laser, manipulação de câmara e inserção de objectos do ambiente, em tempo de simulação.

Porém, nem tudo é simulado no *Gazebo*. Dentro dos tipos de objectos/eventos que não são simulados, encontram-se modelos físicos de solos, areia, relva, entre outras superfícies encontradas na natureza. Também, não são implementados objectos deformáveis e dinâmicas térmicas e de fluídos. Isto significa que também que o *Gazebo* é um projecto em fase de desenvolvimento visto que a ultima versão ainda é a versão 0.8.

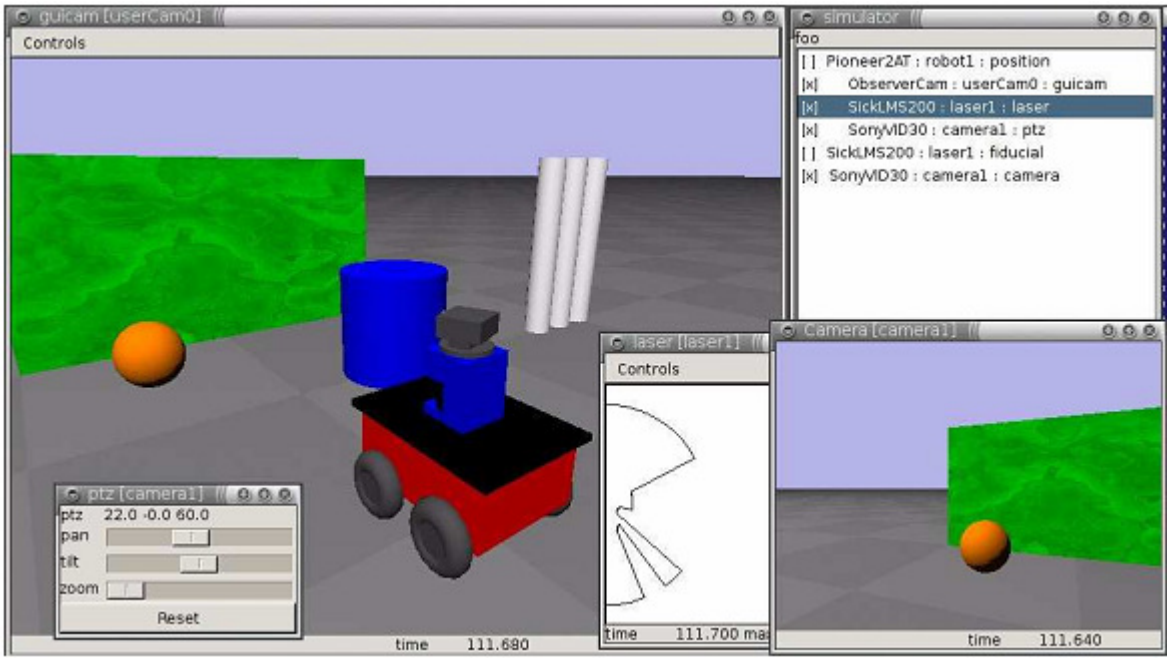


Figura 42 Interface gráfica do *Gazebo*

6.4 MOTOR GRÁFICO *OGRE*

O *Gazebo* possui um motor de processamento gráfico ***OGRE***. Este é responsável por toda a visualização do ambiente 3D. Este motor gráfico é *Open Source* é muito utilizado por programadores para desenvolvimento de animações e jogos 3D.

OGRE (*Object-oriented Graphics Rendering Engine*) é um motor gráfico *open-source* que funciona na maioria das plataformas existentes (*Windows, Linux e MAC OS*). A *interface* de programação oferecida pelo *OGRE* é escrita em C++, o que requer do programador um bom conhecimento da linguagem e de conceitos de orientação a objectos (abstracção, encapsulamento, polimorfismo). Actualmente, existem alguns *wrappers* para o *OGRE* em *Java*, *.NET* e *Python*, mas eles ainda estão em fase de desenvolvimento. O propósito do *OGRE* não é só ser um simples motor gráfico para Jogos, ele é um motor genérico que pode ser incorporado a bibliotecas de tratamento de entradas de dados, de processamento de som e as plataformas que disponibilizem algoritmos de inteligência artificial, compondo assim um kit de desenvolvimento mais completo que dê suporte ao desenvolvimento de jogos e simuladores 3D.

6.5 MOTOR FÍSICO *ODE*

O *Gazebo* possui um motor de física *ODE* (*Open Dynamic Engine*). Este é responsável por gerar toda a dinâmica dos objectos 3D do *OGRE*.

O ***ODE*** é um motor de física *Open Source* de elevado desempenho para simular corpos rígidos através de bibliotecas dinâmicas. É caracterizada como uma plataforma estável e é fácil de usar em *API* em C++. Possui avançados mecanismos de detecção de colisões e fricção. O *ODE* é útil para simular veículos, objectos em realidade virtual e ambientes virtuais. Actualmente, é utilizado em muitos jogos de computador e ferramentas de simulação 3D.

6.6 AMBIENTE GRÁFICO DA APLICAÇÃO *FLTK*

O ***FLTK*** (***F**ast **L**ight **T**ool**K**it*) é um kit de desenvolvimento de interfaces gráficas para a linguagem C++, disponível para diversos sistemas operacionais (*GNU/Linux* – através do sistema gráfico X11, *Windows, Mac OS X, OS/2 e Solaris*).

Através do *FLTK* é possível criar programas com interface gráfica independente das rotinas específicas do sistema operacional. O conjunto de elementos disponíveis é amplo, e inclui desde os mais comuns, como botões, caixas de texto, etiquetas, *sliders*, entre outros,

até menos usuais, por exemplo, *dials*, navegadores e relógios. Novos elementos são facilmente construídos/adaptados através de derivações dos pré-existentes.

O *FLTK* suporta gráficos 3D (comuns em jogos modernos e visualizações científicas) via *OpenGL*, bem como provê emulação à biblioteca *GLUT*. Embora o *FLTK* forneça um rico conjunto de componentes gráficos, visa a manter-se simples e modular. *Software* construídos com o *FLTK* tendem a ser eficientes em tempo e espaço (mesmo quando ligados estaticamente). Adicionalmente, é fornecida com o *FLTK* a ferramenta *FLUID*, que permite de maneira fácil e visual a construção de todo o esqueleto da interface gráfica. O *FLTK* é um *Software* Livre distribuído sob a licença *LGPL* (*GNU Library General Public License*), permitindo inclusive que os programas criados sejam comercializados sem distribuição do código fonte.

6.7 BIBLIOTECA *LIBXML*

LibXML é uma biblioteca para analisar documentos *XML*, visto que o código do mundo do *Gazebo* ser escrito em *XML*. É também a base para outra biblioteca que analisa estilo *XSLT*-1.0. É escrito na linguagem de programação C, e fornece uma conexão para C, C++, *XSH, C #, Python, Kylix / Delphi, Ruby, PHP5*, Pascal e outros. Ele pode ser acedido a partir do *Perl* usando o *XML*. Esta biblioteca foi originalmente desenvolvida para o projecto *GNOME*, mas que podem ser utilizada fora dela. O código *libXML* é portátil, uma vez que depende apenas de bibliotecas padrão *ANSI C*.

6.8 ARQUITECTURA DO SIMULADOR *GAZEBO*

O simulador usa um conjunto de ferramentas (abaixo ilustradas) para gerar um conjunto de objectos simulados. Para simular todo este conjunto divide-se todo o processo em partes, sendo elas, o modelo do mundo, modelo do robô e modelo dos sensores, bem como actuadores.

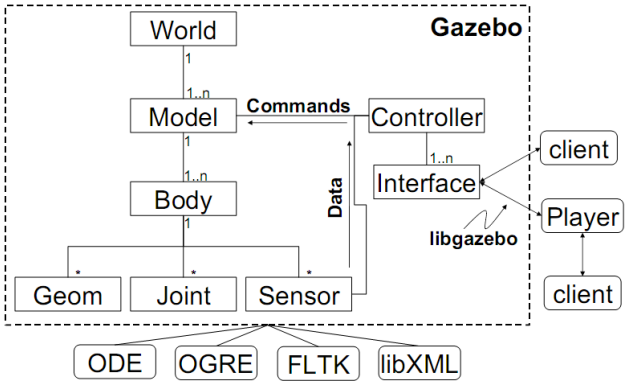


Figura 43 Arquitectura de funcionamento do *Gazebo*

6.8.1 *WORLD FILE*

O ficheiro do mundo contém uma descrição do mundo a ser simulado pelo *Gazebo*. Ele descreve o *layout* dos robôs, sensores, fontes luminosas, componentes de interface do utilizador, e assim por diante. Este ficheiro também pode ser utilizado para controlar alguns aspectos da simulação, como motores, força da gravidade ou intervalo de tempo da simulação. O ficheiro é escrito em *XML*, e pode, portanto, ser criado e modificado usando um editor de texto.

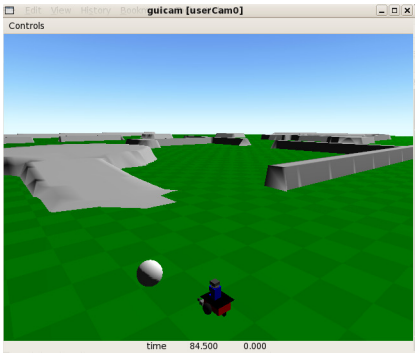


Figura 44 *Gazebo World*

6.8.2 MODELO DE UM ROBÔ

O Modelo forma o núcleo de simulação do *Gazebo.* Abrangem todas as entidades físicas e sensoriais. Uma vez que os modelos são tão importantes, devem ser facilmente criados e modificados. O *Gazebo* utiliza *XML* para definir todos os aspectos físicos de um modelo.

O *Gazebo* possui um conjunto de modelos que podem ser utilizados e alterados consoante o usuário assim entender. Este modelo carrega as características da dinâmica e cinemática de cada uma das configurações. O *Gazebo* possui de base configurações de robôs terrestres, aéreos, com direcção diferencial, *Ackerman* entre outros.

Para cada uma das configurações existe um controlador que define a forma de locomoção bem como forma de o controlar.

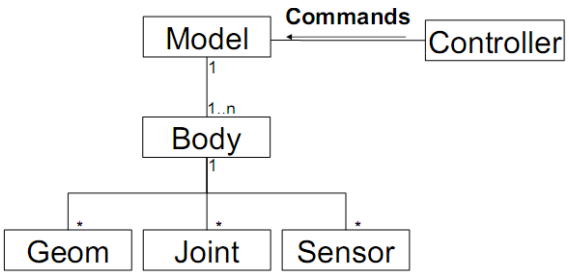


Figura 45 Modelo do robô

Todos os aspectos físicos de um modelo são definidos dentro de um ficheiro *XML*. Isto inclui a geometria, junções e sensores. O modelo é construído a partir das bibliotecas do *OGRE* (aspecto visual) e *ODE* (características físicas).

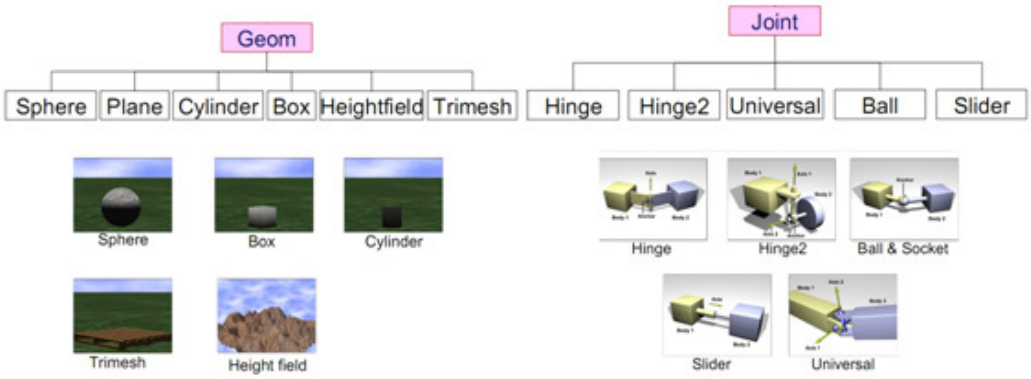


Figura 46 Modelo das geometrias *OGRE* e junções *ODE*

Os Sensores são acoplados ao corpo do robô dentro do ficheiro *World*. Estes são anexos na estrutura na posição e orientação desejada. Existem também controladores para inúmeros sensores como por exemplo, sonares, lazer, câmaras, entre outros.

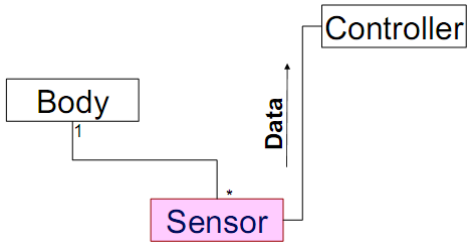


Figura 47 Acoplamento de sensores no corpo do robô

6.8.3 BIBLIOTECA *CLIENTLIB*

As bibliotecas de cliente C/C++ são geralmente as mais utilizadas pelos programadores para realização dos algoritmos de controlo e agentes de controlo.

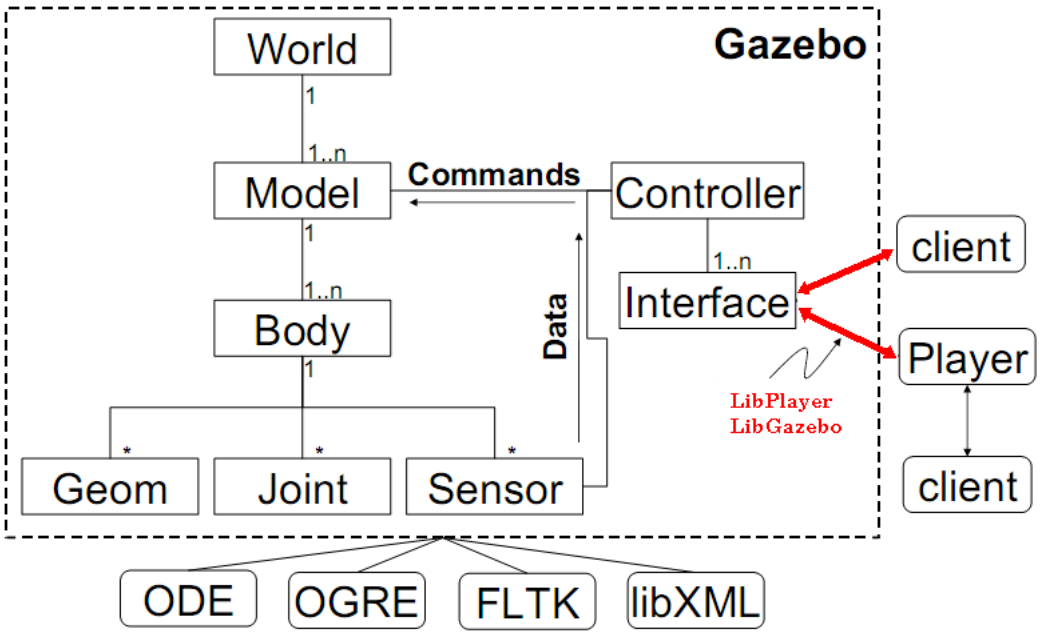


Figura 48 Conecção da biblioteca Libplayer e libgazebo

A biblioteca é construída sobre um "serviço de *proxy*" modelo em que o cliente mantém a leitura dos objectos que são *proxys* de serviços remotos. Cada tipo de *proxy* é

implementado como uma classe separada (laser, odometria, etc). O usuário cria um *PlayerClient* e usa-o para estabelecer uma ligação a um servidor *Player*.

Em seguida, os representantes dos dispositivos apropriados mantêm actualizada a informação do respectivo dispositivo, mantendo o cliente actualizado.

```
#include <stdio.h>
#include "playerc.h"

int main( int argc, const char **argv)
{
        int i;
        playerc_client_t *client;
        playerc_position_t *position;

        // Create a client object and connect to the server; the server must be running on "localhost" at port 6665
        client = playerc_client_create(NULL, "localhost", 6665);
        if (playerc_client_connect(client) != 0)
        {
                fprint f(stderr, "error: %s\n", playerc_error_str());
                return -1;
        }
        // Create a position proxy (device id "position:0") and susbscribe in read/write mode
        position = playerc_position_create(client, 0);
        if (playerc_position_subscribe(position, PLAYER_ALL_MODE) != 0)
        {
                fprint f(stderr, "error: %s\n", playerc_error_str());
        return -1;
        }
        // Enable the robots motors
        playerc_position_enable(position, 1);

        // Start the robot turning slowing
        playerc_position_set_cmd_vel(position, 0, 0, 0.1, 1);

        for (i = 0; i < 200; i++)
        {
                // Read data from the server and display current robot position
                playerc_client_read(client);
                print f("position : %f %f %f\n",
                position->px, position->py, position->pa);
        }

        // Shutdown and tidy up
        playerc_position_unsubscribe(position);
        playerc_position_destroy(position);
        playerc_client_disconnect(client);
        playerc_client_destroy(client);
}
```

Figura 49 Exemplo de API de controlo

7. IMPLEMENTAÇÃO

7.1 MODELIZAÇÃO DO ROBÔ

Para implementar o modelo do mundo e do robô, fez-se um estudo da estrutura mecânica como vimos no capítulo 4. Como vimos, o robô é uma estrutura mecânica de quatro rodas, e para o desenho 3D escreve-se os parâmetros dessa configuração num ficheiro de modelo *XML*, de forma a construir o modelo do robô recorrendo as bibliotecas do *OGRE.*

Observemos agora a estrutura central do robô:

Figura 50 Estrutura central do robô

7.1.1 ESTRUTURA FÍSICA DO ROBÔ

Para desenhar esta estrutura, dividiu-se o bloco em duas partes distintas. O chassis de baixo e o chassis de cima. Esta divisão aparenta ser mais lógica visto que o centro de massa do chassis se encontra na estrutura do chassis de cima onde se encontra parte da electrónica e baterias. Com recurso as bibliotecas do *ODE*, podemos dimensionar qual o peso desses blocos de forma a estrutura ter um comportamento dinâmico.

7.1.2 CHASSIS DE BAIXO

Caixa de dimensão :

$[x,y,z] = 0.3\ 0.1\ 0.1\ (m)$
$Massa = 1\ kg$

Figura 51 Estrutura de simulação do chassis de baixo

7.1.3 CHASSIS DE CIMA

Caixa de dimensão :

$[x,y,z] = 0.5\ 0.15\ 0.05(m)$
$Massa = 3\ kg$

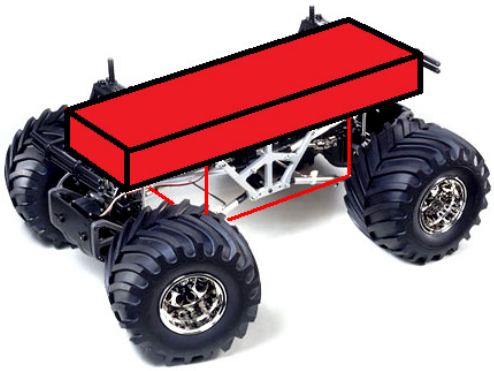


Figura 52 Estrutura de simulação do chassis de cima

7.1.4 BASE

A base é uma estrutura que separa a parte do chassis da parte do PC, esta estrutura é simplesmente uma chapa em metal que serve de plataforma a vários equipamentos de *hardware*.

Caixa de dimensão:

$[x,y,z] = 0.50\ 0.30\ 0.01(m)$
$Massa = 0.3\ kg$

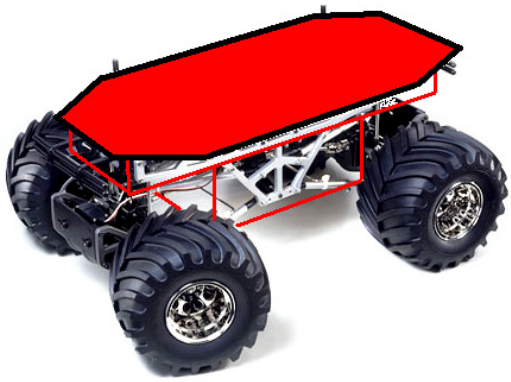


Figura 53 Estrutura da simulação da base

7.1.5 CAIXA DO PC

Caixa de dimensão:

$[x,y,z] = 0.20\ 0.25\ 0.08(m)$
$Massa = 1\ kg$

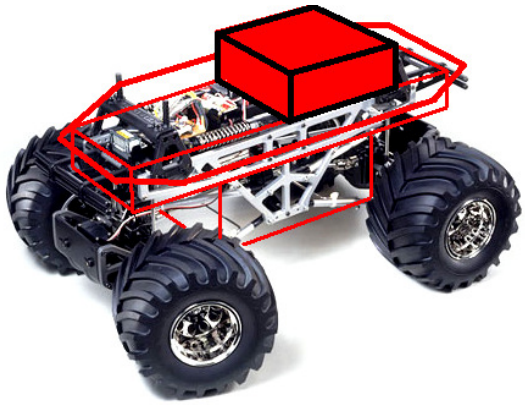


Figura 54 Estrutura de simulação do PC

7.1.6 CHASSIS

Para fechar este bloco como um corpo rígido, cria-se uma caixa em volta destes blocos, sem dinâmica ou forma. Apenas serve para dimensionar todo o corpo deste robô.

7.1.7 RODAS

As rodas do robô são cilindros no mundo da simulação. Neste contexto separou-se a roda em duas partes distintas para a visualização, o pneu e a jante. Na parte dinâmica considerou-se o conjunto como um bloco único.

Cilindro de dimensão :

$Raio = 0.075\,(m)$
$L\,\text{arg}\,urra = 0.1(m)$
$Massa = 0.2\ kg$

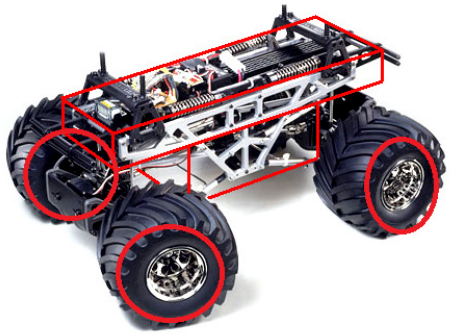


Figura 55 Estrutura de simulação das rodas

7.1.8 COMPORTAMENTO DAS RODAS

Neste robô, as rodas traseiras são as rodas de propulsão e giram em torno de um único eixo.

Já as rodas frontais rodam no mesmo sentido que as rodas traseiras e também sobre o eixo dos zz, de forma a virar o veículo numa dada direcção.

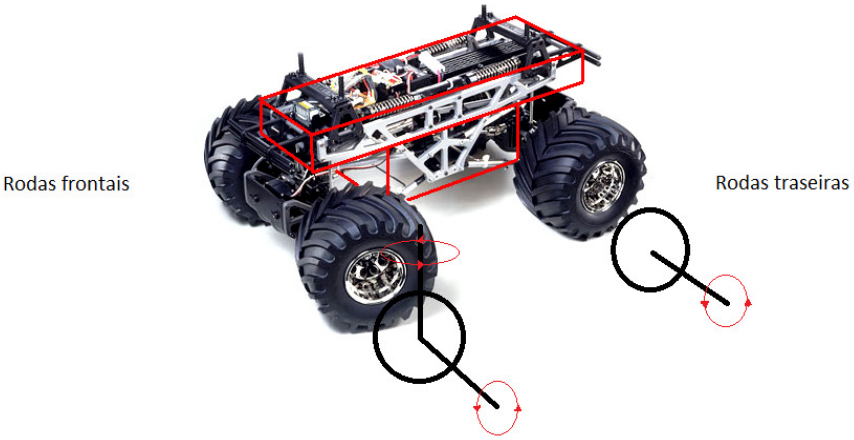


Figura 56 Configuração das rodas do robô

7.1.8.1 JUNÇÕES DAS RODAS

O *ODE* permite-nos criar vários tipos de ligação e configuração para vários tipos de rodas. Podemos criar junções para as rodas girarem livremente, em torno de um eixo, dois eixos e três eixos.

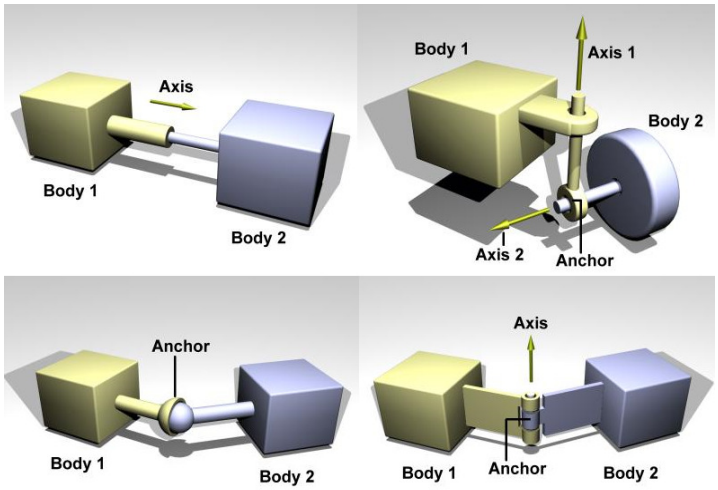


Figura 57 Os três tipos de junções do *ODE*

Com as configurações existentes, as configurações escolhidas para as rodas de trás são *hinge* e para as rodas da frente *hinge2*.

7.1.8.2 HINGE

Esta configuração foi feita para as rodas traseiras. Este configuração permite dar as rodas liberdade de revolução num eixo à escolha, tal como as rodas traseiras rodam.

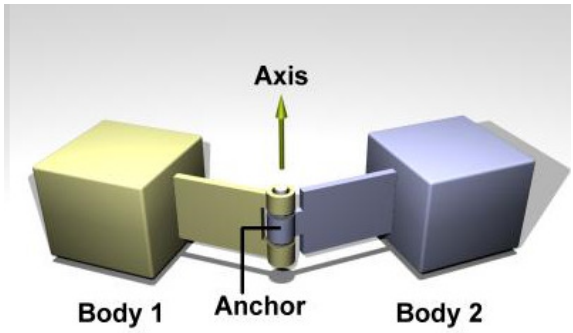


Figura 58 Estrutura de simulação da junção hinge

7.1.8.3 HINGE 2

Esta configuração foi feita para as rodas frontais. Este configuração permite dar as rodas liberdade de revolução nos dois eixos à escolha, tal como as rodas frontais rodam.

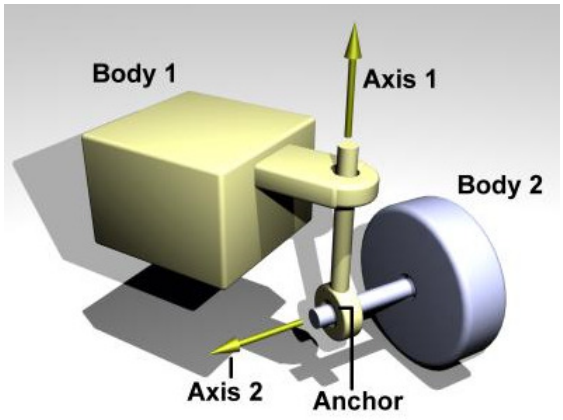


Figura 59 Estrutura de simulação da junção hinge 2

Na verdade esta configuração não funciona. Foi feito um esforço para tentar resolver este problema. O resultado foi realizar uma estrutura de direcções que contem duas junções do tipo *hinge*. Uma das junções fica a girar no eixo dos *ZZ* e a outra a girar no eixo dos *XX*.

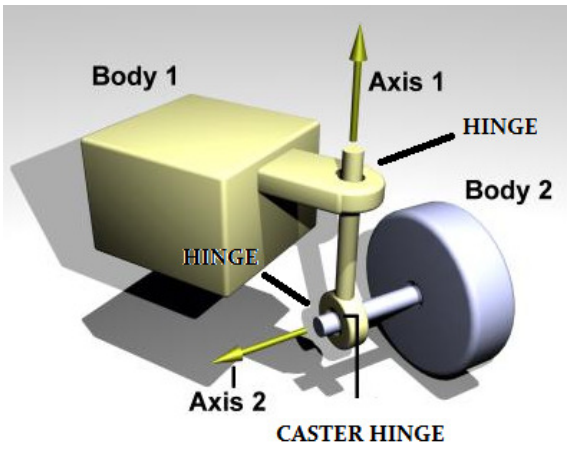


Figura 60 Alternativa ao Hinge2

7.2 ESTRUTURA DO MUNDO

O ficheiro de configuração do mundo *file.world* é um ficheiro onde é carregado um conjunto de parâmetros que irão caracterizar a estrutura física do mundo. Paredes, escadas etc. Neste ambiente, o mundo pode ser gerado e controlado como o utilizador bem entender.

7.2.1 CARACTERÍSTICAS GERAIS

A primeira configuração a ser feita diz respeito a aceleração da gravidade do simulador bem como o tempo de amostragem da simulação. Estes parâmetros serão os que irão dar maior importância ao resultado da simulação visto que a gravidade faz com que os objectos adquiram dinâmica e o período de amostragem será decisivo no produto final do factor simulador/realidade. Quanto menor este valor, mais próximo da realidade estamos.

StepTime	0.03
gravity	−9.8
cfm	10*e*-5
erp	0.3

Esta segunda configuração é a configuração do solo. O solo foi gerado como um plano horizontal com um tamanho de 1000 m por 1000 m (Valor facultativo), um quadrado perfeito. Para o solo, o simulador permite dar uma *skin* do tipo relva de forma a parecer mais realista.

Plano	
Tamanho	$1000 \times 1000\,(m)$

7.2.2 OBSTÁCULOS

A partir do momento que geramos o solo e a acção da gravidade, podemos introduzir neste mundo o robô e obstáculos. Para a simulação, introduzimos um cubo, e uma esfera de forma a validar este conjunto de hipóteses possíveis de obstáculos.

7.2.2.1 CAIXA

Caixa de dimensão:

$[x,y,z] = 0.5\ 0.5\ 0.5(m)$
$Massa = 1\ kg$

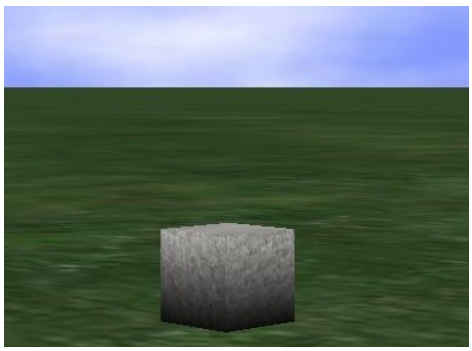


Figura 61 Estrutura de simulação de uma caixa

7.2.2.2 ESFERA

Esfera de dimensão:

$Raio = 0.2(m)$
$Massa = 1\ kg$

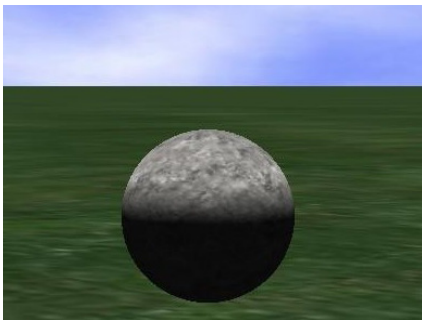


Figura 62 Estrutura de simulação de uma esfera

7.2.3 INTRODUÇÃO DO ROBÔ NO MUNDO

Neste momento temos o mundo do robô gerado. Agora falta a introdução do modelo gerado no ambiente virtual. Para isso faz-se um *include* do modelo no ficheiro *world* de forma a este robô fazer parte deste mundo.

```
<include embedded="false">
  <xi:include href="lince.model" />
</include>
```

7.2.4 ODOMETRIA

Depois da introdução do modelo do robô, agora é necessário configurar o controlador das rodas de forma a gerar a odometria do robô. O *Gazebo* possui uma biblioteca odometrica para veículos de quatro rodas. Este controlador permite obter leituras do movimento das rodas bem como a partir da aplicação cliente mandar comandos para virar e dar locomoção as rodas. Neste controlador, cada roda deve ser configurada. As rodas de traz como locomoção (*drive*) e as da frente como direcção (*steer*).

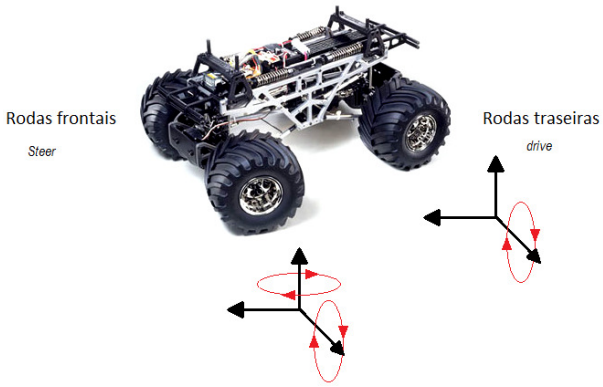


Figura 63 Configuração das rodas do LINCE no simulador

Este controlador supostamente devia retornar a posição do robô no mundo (odometria), no entanto não está implementado no código fonte. Por isso foi necessário implementar uma função que constrói a odometria do robô por integração do controlo efectuado.

```
////////////////////////////////////////////////////////////////////////////
// Update the data in the interface
void Steering_Position2d::PutPositionData()
{
  if (this->myIface->Lock(1))
  {
    // TODO: Data timestamp
    this->myIface->data->head.time = Simulator::Instance()->GetSimTime();

    this->myIface->data->pose.pos.x = this->odomPose[0];
    this->myIface->data->pose.pos.y = this->odomPose[1];
    this->myIface->data->pose.yaw = NORMALIZE(this->odomPose[2]);

    this->myIface->data->velocity.pos.x = this->odomVel[0];
    this->myIface->data->velocity.yaw = this->odomVel[2];

    // TODO ODOMETRI
    this->myIface->data->stall = 0;

    this->myIface->Unlock();
  }
}
```

Figura 64 Ausência de odometria no código fonte

7.2.5 IMPLEMENTAÇÃO DA ODOMETRIA

A odometria implementada baseia-se nas equações de cinemática do veículo. Estas são alimentadas pelas variáveis de controlo de forma a estimar a posição do veículo ao longo do tempo.

$$
\begin{aligned}
x(k+1) &= x(k) + \Delta T R(k)\omega(k)\cos[\phi(k) + \gamma(k)] \\
y(k+1) &= y(k) + \Delta T R(k)\omega(k)\sin[\phi(k) + \gamma(k)] \\
\phi(k+1) &= \phi(k) + \Delta T \frac{R(k)\omega(k)}{B}\sin\gamma(k) \\
R(k+1) &= R(k)
\end{aligned}
$$

Figura 65 Equações cinemáticas para implementação da odometria

7.2.6 LINCAGEM DE SENSORES NO ROBÔ

7.2.6.1 LASER

O *Gazebo* possui um conjunto de sensores que podemos utilizar para a simulação do robô. Para este estudo foi escolhido implementar o sensor *laser Hokoyo* que existe nas bibliotecas do simulador.

7.2.6.2 CÂMARA

O *Gazebo* permite simular vários tipos de câmaras, nomeadamente *RGV* e *BW*. Para simular a câmara do *LINCE,* introduzimos o modelo de uma câmara *RGV* de forma a conseguir obter uma imagem do ambiente simulado a cores.

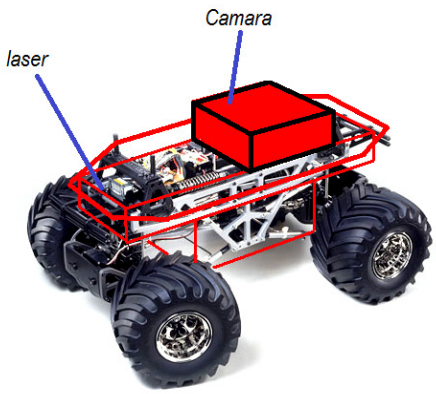


Figura 66 Anexo da câmara e do laser

7.3 MUNDO GERADO

7.3.1 MUNDO

Como podemos ver na seguinte imagem, aqui fica ilustrado o resultado da modelização do robô e do mundo. Este ambiente gerado apenas é constituído por 2 blocos, uma esfera e um quadrado. Seria possível gerar um mundo complexo, mas os objectivos da TESE passam por simular comportamentos. Numa implementação futura seria de interesse realizar um cenário de uma missão para testar um sistema de navegação elaborado pelo LSA de forma a testar a performance do robô antes de este efectuar determinada missão.

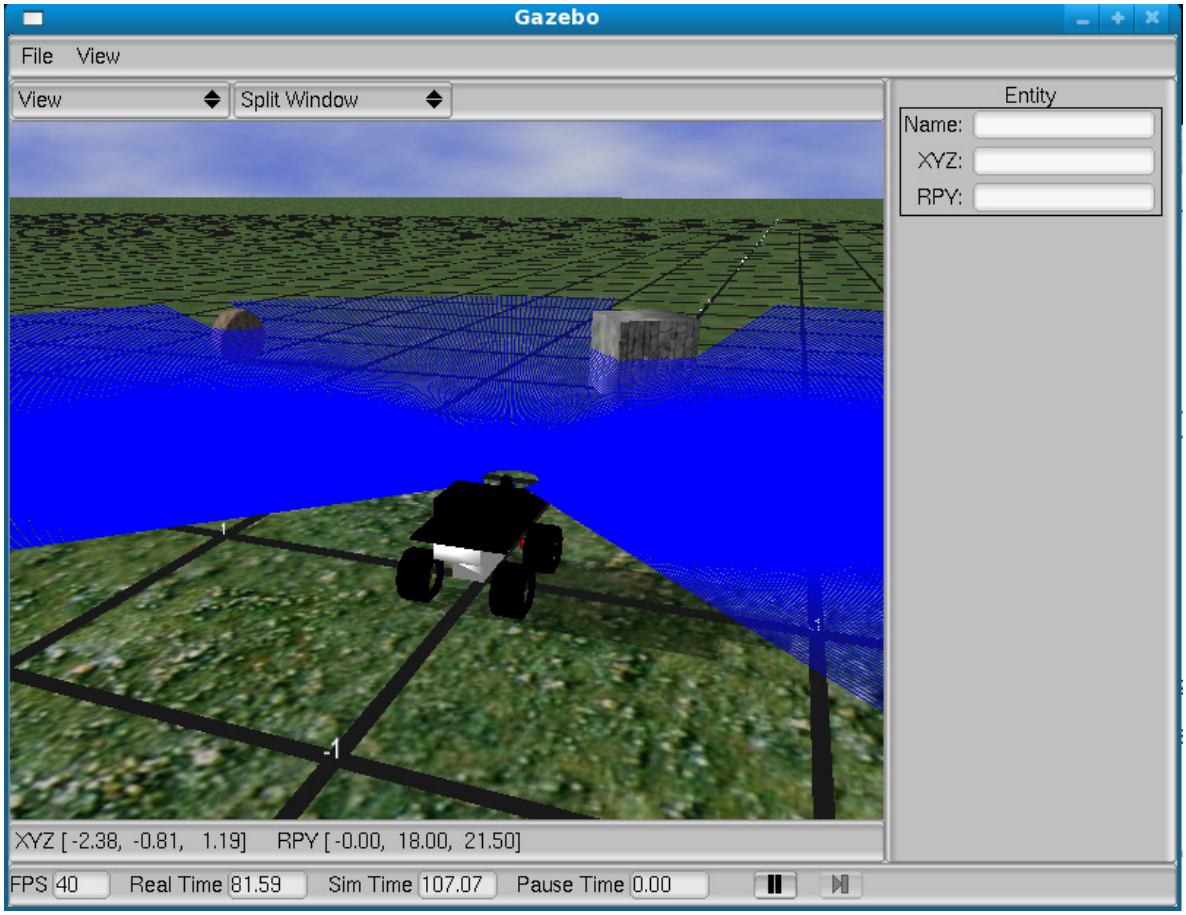


Figura 67 Visualização do mundo gerado pelo Gazebo

7.3.2 ROBÔ

Na seguinte imagem podemos ver com mais detalhe o robô no ambiente de simulação e ao lado vemos uma imagem do robô real.

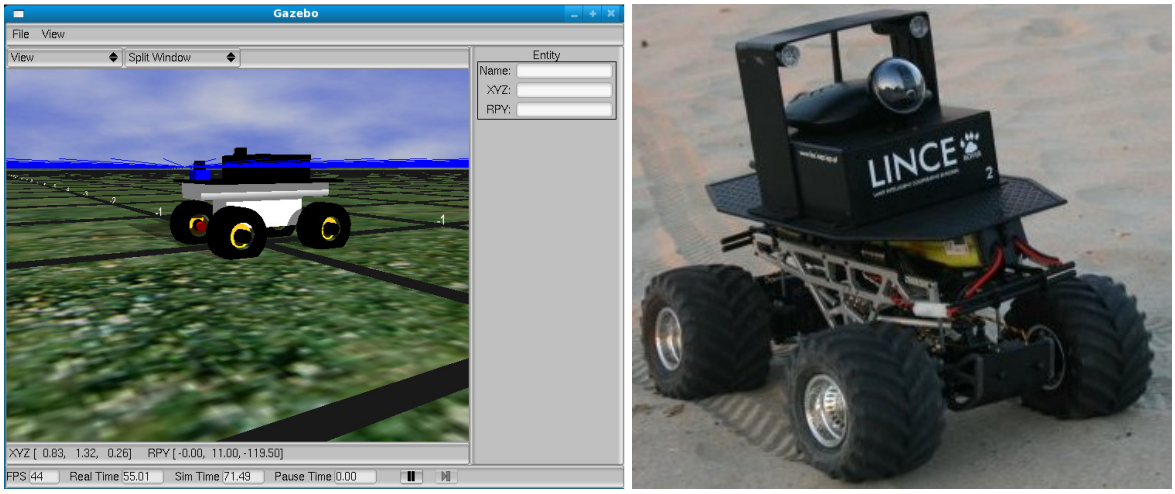


Figura 68 Visualização do robô simulado e real

Na seguinte imagem pode-se observa o conteúdo dos sensores simulados, nomeadamente o laser e câmara.

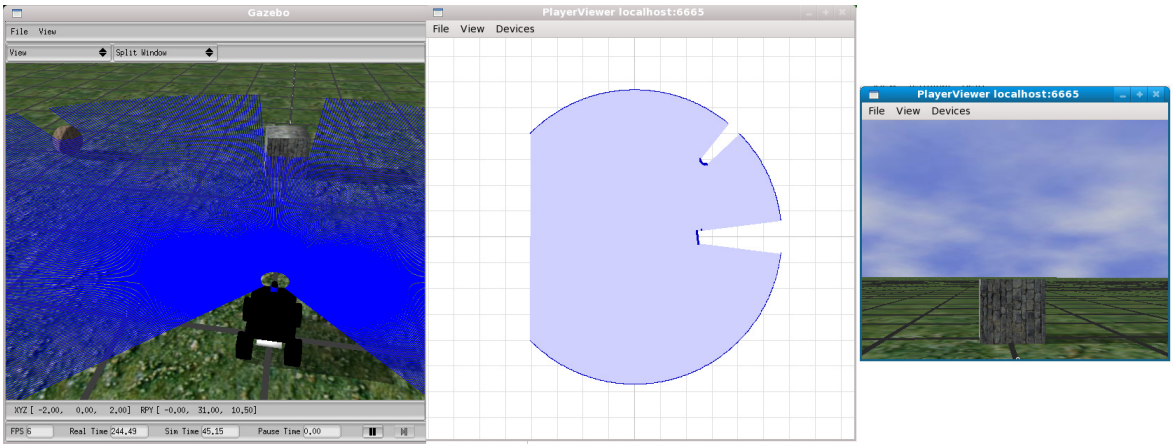


Figura 69 Laser e câmara

Também podemos observar o resultado da actividade do ODE. Usando o comando "playerjoy" podemos controlar o robô com o teclado (Tele-operação). No exemplo

seguinte, o robô é conduzido de forma a colidir com um obstáculo e como podemos ver, este altera a trajectória do robô.

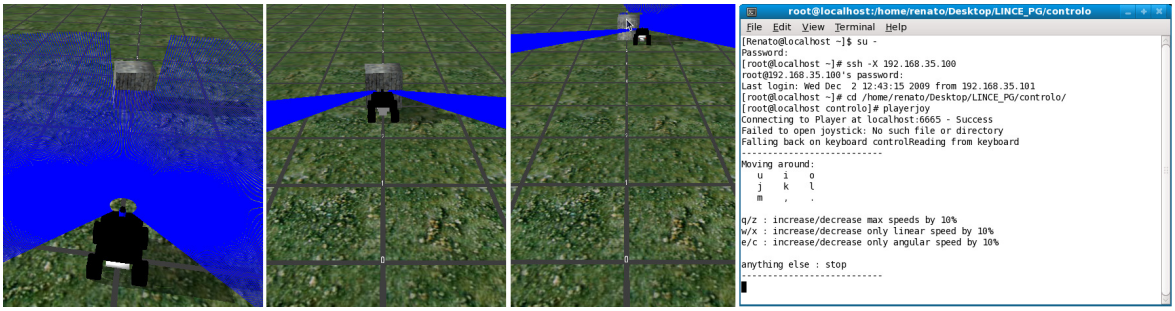


Figura 70 Movimento e colisão do robô usando o *playerjoy*

Como podemos observar, podemos concluir que o primeiro e segundo objectivo desta TESE foram cumpridos ate este ponto. O robô e o mundo estão simulados e de forma funcional. Para cumprir os objectivos da TESE falta desenvolver algoritmos de controlo e testá-los no simulador.

7.4 SIMULAÇÃO EM *STAGE*

Foi implementado também um modelo do LINCE em 2D (*Stage*). Esta simulação permitiu desenvolver os primeiros 2 algoritmos de controlo antes da implementação em *Gazebo*. Foi feita esta implementação ao mesmo tempo que se verificava os problemas de compilação e instalação do Gazebo. Os algoritmos implementados e validados em *Stage* foram o "*Ir para*" e "*composição de waypoints*".

Para isso, construiu-se o modelo do LINCE baseado no modelo cinemático utilizando a biblioteca *CarChassis.* Esta biblioteca implementa a cinemática do modelo do LINCE e assim simulou-se o modelo do mundo (Laboratório LSA) e do robô (LINCE) como podemos ver a seguir.

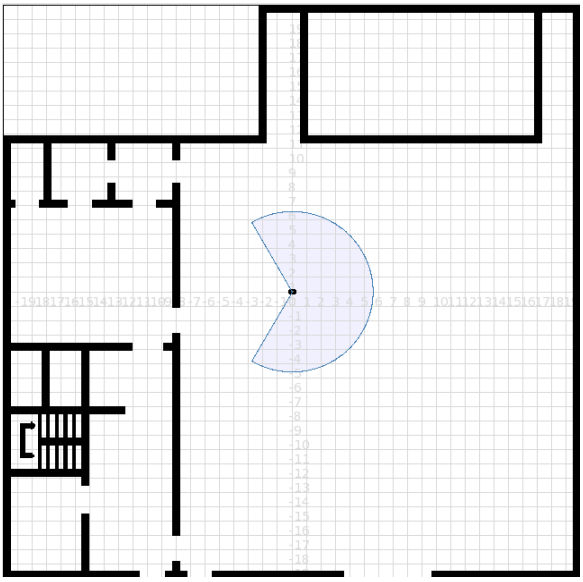


Figura 71 Simulação em *Stage*

7.5 ALGORITMOS DE CONTROLO

Para dar acção e movimento ao mundo implementado, desenvolveu-se uma *API* de comunicação com o simulador de forma a enviar e receber parâmetros do ambiente simulado. Desta forma conseguiu-se testar alguns algoritmos de controlo e provar que a simulação de robôs é uma ferramenta de implementação capaz e poderosa.

Temos que ter em conta que um caminho é uma composição de movimentos simples. Esta composição gera movimentos completos permitindo também descrever trajectórias que de outra forma seriam mais difíceis de implementar. Um exemplo disso seria executar uma manobra de contornar um obstáculo, o mecanismo de resposta poderia ser seguir para um ponto paralelo a trajectória e depois para o ponto de destino como mostra a figura seguinte.

Situação inicial:

Ir do ponto P_1 para o *Goal*

Situação final:

Ir do ponto P_1 para o ponto P_2, Ir do ponto P_2 para o *Goal*

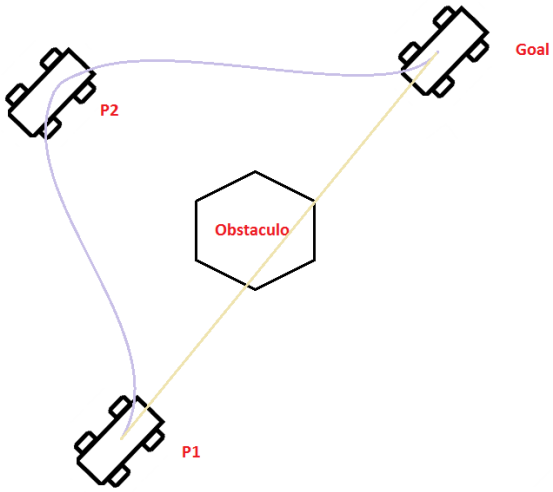


Figura 72 Contorno de obstáculos

A camada de controlo responsável pela geração da trajectória do robô é chamada de **Sistema de Navegação**. Este é responsável pela parte de navegação e gera as trajectórias do robô desde o local de origem até o local de destino da missão. Para isso, este usa as funções de controlo básicas de forma a traçar uma trajectória possível para cada missão.

Este mecanismo preocupa-se com encontrar o melhor caminho para o robô em determinada missão. Por exemplo, a seguinte missão destina o robô a mover-se da posição inicial até o ponto **P.** Na primeira opção poderia seguir-se em linha recta mas existem obstáculos, o que o gestor de missões faz é dividir o percurso em **N** objectivos de forma a cumprir a missão global.

Figura 73 Gestor de Missões

Para conseguir cumprir o objectivo da missão, o sistema de navegação elabora um caminho com funções de controlo básicas. Esses algoritmos de controlo de movimento implementam leis de controlo em que se pretende controlar os erros em teta ε_θ e distância ε_p levando-os a convergir para zero.

Os algoritmos de controlo implementados para um possível gestor de missões foram os seguintes:

- ✓ Ir para
- ✓ Fazer recta
- ✓ Seguir orientação
- ✓ Seguir *waypoints*
- ✓ Seguir rectas

7.6 DESCRIÇÃO DAS FUNCIONALIDADES DOS ALGORITMOS

7.6.1 IR PARA

Esta manobra permite que o *LINCE* percorra um caminho que una dois pontos. Os pontos são, posição actual do *LINCE* e a posição final. Assim o *LINCE* pode ir de uma posição actual para uma posição a marcar. Neste caso, o *LINCE* deve receber como parâmetro a posição final, pois a inicial já deve ser obtida pelos sensores de posição (odometria).

Depois de recebidos os parâmetros iniciais, o algoritmo deve calcular o ângulo de direcção que deve impor as rodas frontais do veículo, para isso usa as funções matemáticas desenvolvidas juntamente com o modelo cinemático do veículo.

Depois de calculado o ângulo a impor nas rodas frontais, devem ser feitas as correcções a direcção do veículo para que este possa convergir para o objectivo da missão. Entretanto o Estado do robô é actualizado e é feita nova actualização ao factor de correcção do veículo, até que este execute a missão com êxito.

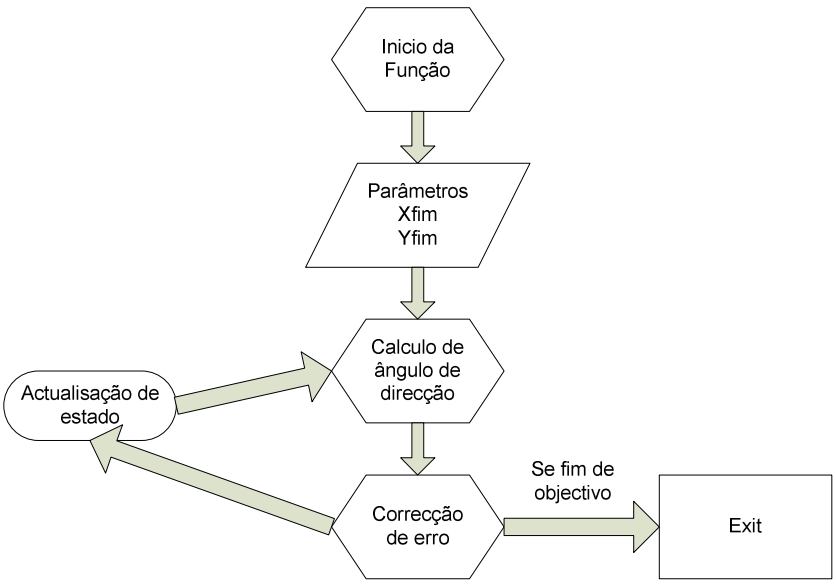
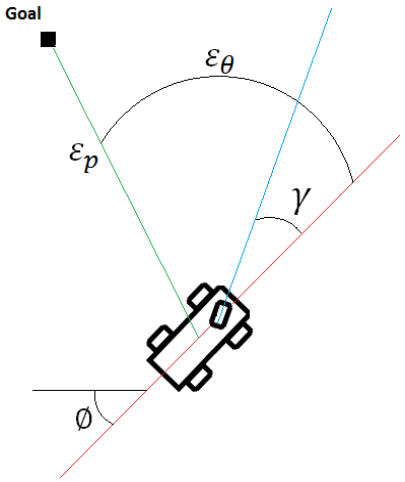


Figura 74 Fluxograma do algoritmo *Ir para*

7.6.1.1 EXPLICAÇÃO DETALHADA DO ALGORITMO

Como já vimos, esta função permite guiar o LINCE para um determinado ponto mesmo encontrando-se em qualquer posição e orientação. Para que isto seja possível, o algoritmo calcula periodicamente o ângulo que as rodas da frente devem ter, pois o controlo do veículo é em função do ângulo de curvatura e velocidade.



$$\theta_{recta} = \text{atan}\left(\frac{Y_{Goal} - Y_{\text{robo}}}{X_{Goal} - X_{\text{robo}}}\right)$$

$$\varepsilon_\theta = \theta_{recta} - \phi$$

$$\varepsilon_p = \sqrt{(X_{goal} - X_{robot})^2 + (Y_{goal} - Y_{robot})^2}$$

$$\left\langle \begin{cases} v = \dfrac{Max_{speed}}{R_{roda}}, & se \ |\varepsilon_p| \geq 0.1 \\ v = 0, & se \ |\varepsilon_p| \leq 0.1 \end{cases} \right\rangle$$
$$\gamma = \varepsilon_\theta$$

Figura 75 ***Ir para***

θ_{recta}	*Ângulo da recta*
ϕ	*Ângulo do robo*
γ	*Ângulo de direcção*
ε_θ	*Erro em Teta*
ε_p	*Erro em posição*

7.6.2 FAZER RECTA

Esta manobra permite que o *LINCE* execute segmentos de recta desde um ponto a outro da recta. O segundo ponto irá tornar-se o objectivo da missão *(Goal).* Para que o *LINCE* faça rectas, este deve receber um comando que tenha como parâmetros de entrada a posição inicial e final da mesma. Assim o *LINCE* aproxima-se de forma tangente à recta e permanece nesse caminho até atingir o objectivo da missão.

Depois de recebidos os parâmetros iniciais, o algoritmo deve calcular o ângulo de direcção que deve impor as rodas frontais do veículo, para isso usa as funções matemáticas desenvolvidas juntamente com o modelo cinemático do veículo.

Depois de calculado o ângulo a impor nas rodas frontais, devem ser feitas as correcções a direcção do veículo para que este possa convergir para o objectivo da missão. Entretanto o Estado do robô é actualizado e é feita nova actualização ao factor de correcção do veículo, até que este execute a missão com êxito.

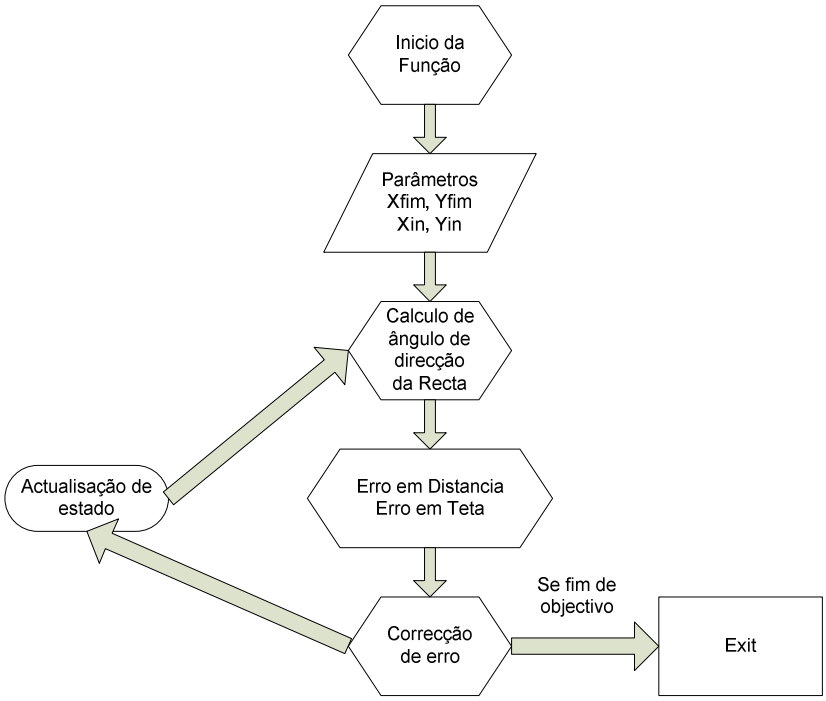


Figura 76 Fluxograma do algoritmo *fazer recta*

7.6.2.1 EXPLICAÇÃO DETALHADA DO ALGORITMO

Quando recebidos os parâmetros, o algoritmo calcula o ângulo da recta entre os 2 pontos introduzido, sendo o ultimo como já foi referido o objectivo da missão.

$$\theta_{recta} = a\tan\left(\frac{Y_{P2} - Y_{P1}}{X_{P2} - X_{P1}}\right)$$

Depois constrói-se a recta sob a equação geral para ser possível calcular a distância do robô a recta.

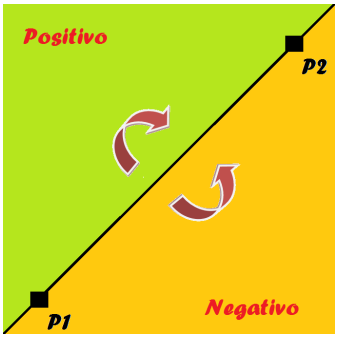
$$Ax + By + C = 0$$

$$A_{recta} = Y_{P1} - Y_{P2}$$
$$B_{recta} = X_{P2} - X_{P1}$$
$$C_{recta} = X_{P1} \times X_{P2} - Y_{P1} \times Y_{P2}$$

$$D_{recta} = \frac{\left| A_{recta} \times X_{robo} + B_{recta} \times Y_{robo} + C_{recta} \right|}{\sqrt{A_{recta}{}^2 + B_{recta}{}^2}}$$

Contudo, esta equação da distância entre ponto e recta danos o módulo da distância.

O importante seria saber qual a distância do ponto a recta e saber se o robô se encontra por baixo ou por cima da mesma de forma a aplicar a correcção da direcção no sentido correcto da recta.



$$b_{recta} = Y_{P2} - \left(\frac{Y_{P2} - Y_{P1}}{X_{P2} - X_{P1}} \right) \times X_{P2}$$

$$k_{\sin al} = Y_{robo} - \left(\frac{Y_{P2} - Y_{P1}}{X_{P2} - X_{P1}} \right) \times X_{robo} + b_{recta}$$

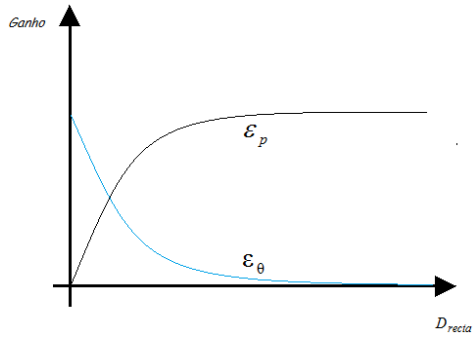
Figura 77 Posição relativamente a recta

θ_{recta} *Ângulo da recta*
ϕ *Ângulo do robo*
γ *Ângulo de direcção*
b_{recta} *b da recta*
$k_{\sin al}$ *coeficiente de posição relactivamente a recta*

O coeficiente $k_{\sin al}$ permite calcular se o robô se encontra por cima ou por baixo da recta, assim se este se encontrar na parte de baixo a correcção das rodas de direcção será no

sentido positivo $k_{\sin al} > 0$, se o robô se encontrar na parte de cima a correcção será no sentido negativo $k_{\sin al} < 0$.

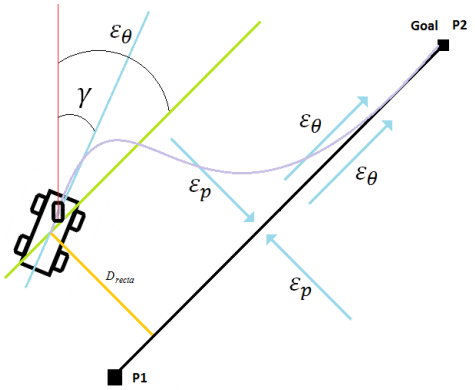
A estratégia de controlo implementada para o *"segue recta"* baseia-se no erro em teta ε_θ e erro em distância ε_p. Quando o robô se encontra perto da recta o erro em teta ε_θ tem maior peso, quando está longe da recta o erro em distância ε_p assume maior importância. Assim, os erros dependem de um factor logarítmico inversamente proporcional.



$$\varepsilon_\theta = G_\theta \times (\theta_{recta} - \phi)$$
$$\varepsilon_p = G_p \times (k_d - \phi)$$

Figura 78 Compensação do erro

A segunda parte da estratégia é a aproximação da recta. O algoritmo implementa uma aproximação perpendicular a recta, para isso o robô deve aproximar-se perpendicularmente a recta e quando estamos suficientemente perto da mesma o factor distância ε_p perde peso enquanto o factor teta ε_θ ganha.



$$\begin{cases} k_{\sin al} > 0, & k_d = \theta_{recta} - \dfrac{\pi}{2} \\ k_{\sin al} < 0, & k_d = \theta_{recta} + \dfrac{\pi}{2} \end{cases}$$

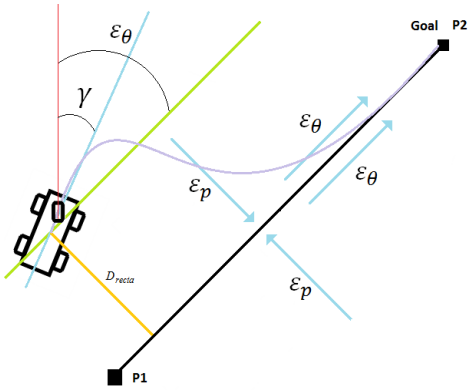
Figura 79 Ilustração da compensação do erro em distancia do "segue recta"

Para se fazer a aproximação perpendicular, quando o factor $k_{\sin al}$ é positivo, a aproximação é $k_{\sin al} > 0, \quad k_d = \theta_{recta} - \frac{\pi}{2}$, quando o factor $k_{\sin al}$ é negativo, a aproximação é $k_{\sin al} < 0, \quad k_d = \theta_{recta} + \frac{\pi}{2}$.

$$\varepsilon_\theta = e^{-D_{recta}} \times \left[\cos(\theta_{recta} - \phi) \times \vec{ux} + \sin(\theta_{recta} - \phi) \vec{uy} \right]$$

$$\varepsilon_p = \left(1 - e^{-D_{recta}}\right) \times \left[\cos(k_d - \phi) \times \vec{ux} + \sin(k_d - \phi) \vec{uy} \right]$$

$$\gamma = \varepsilon_\theta + \varepsilon_p$$



$$\gamma = \tan^{-1}\left(\frac{\left[\varepsilon_\theta + \varepsilon_p \right] \vec{uy}}{\left[\varepsilon_\theta + \varepsilon_p \right] \vec{ux}} \right)$$

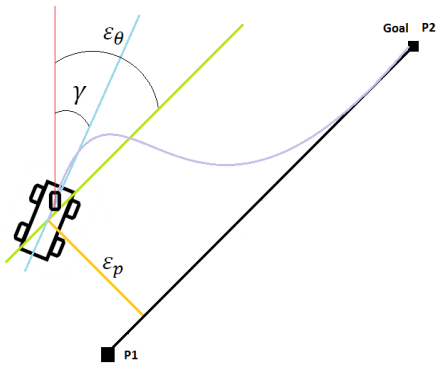
$$\varepsilon_\theta = e^{-D_{recta}} \times \left[\cos(\theta_{recta} - \phi) \times \vec{ux} + \sin(\theta_{recta} - \phi) \times \vec{uy} \right]$$

$$\varepsilon_p = \left(1 - e^{-D_{recta}}\right) \times \left[\cos(k_d - \phi) \times \vec{ux} + \sin(k_d - \phi) \times \vec{uy} \right]$$

$$\gamma = \tan^{-1}\left(\frac{e^{-D_{recta}} \times \sin(\theta_{recta} - \phi) + \left(1 - e^{-D_{recta}}\right) \times \sin(k_d - \phi)}{e^{-D_{recta}} \times \cos(\theta_{recta} - \phi) + \left(1 - e^{-D_{recta}}\right) \times \cos(k_d - \phi)} \right)$$

Figura 80 Ilustração da compensação do erro em distancia do "segue recta"

Concluindo, o controlo é efectuado somando as duas componentes de erro. Estas componentes combinadas geram o ângulo a aplicar no robô que desta forma converge para o objectivo da missão



(1)

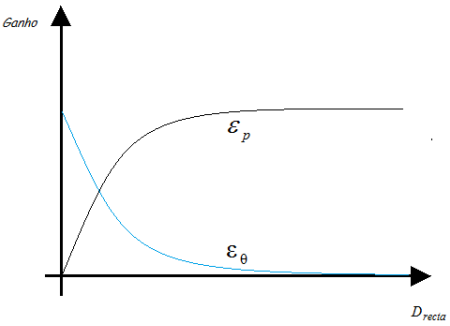
$$\theta_{recta} = \operatorname{atan}\left(\frac{Y_{P2} - Y_{P1}}{X_{P2} - X_{P1}}\right)$$

$$A_{recta} = Y_{P1} - Y_{P2}$$
$$B_{recta} = X_{P2} - X_{P1}$$
$$C_{recta} = X_{P1} \times X_{P2} - Y_{P1} \times Y_{P2}$$

$$D_{recta} = \frac{\left|A_{recta} \times X_{robo} + B_{recta} \times Y_{robo} + C_{recta}\right|}{\sqrt{A_{recta}^{\ 2} + B_{recta}^{\ 2}}}$$

$$b_{recta} = Y_{P2} - \left(\frac{Y_{P2} - Y_{P1}}{X_{P2} - X_{P1}}\right) \times X_{P2}$$

$$k_{\sin al} = Y_{robo} - \left(\frac{Y_{P2} - Y_{P1}}{X_{P2} - X_{P1}}\right) \times X_{robo} + b_{recta}$$



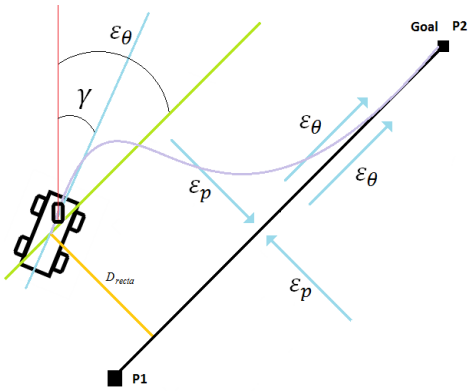
(2)

$$\begin{cases} k_{\sin al} > 0, & k_d = \theta_{recta} - \dfrac{\pi}{2} \\ k_{\sin al} < 0, & k_d = \theta_{recta} + \dfrac{\pi}{2} \end{cases}$$

$$\varepsilon_\theta = G_\theta \times (\theta_{recta} - \phi)$$
$$\varepsilon_p = G_p \times (k_d - \phi)$$

$$\varepsilon_\theta = e^{-D_{recta}} \times \left[\cos(\theta_{recta} - \phi) \times \vec{ux} + \sin(\theta_{recta} - \phi)\vec{uy}\right]$$
$$\varepsilon_p = \left(1 - e^{-D_{recta}}\right) \times \left[\cos(k_d - \phi) \times \vec{ux} + \sin(k_d - \phi)\vec{uy}\right]$$

$$\gamma = \varepsilon_\theta + \varepsilon_p$$



(3)

$$\gamma = \tan^{-1}\left(\frac{\left[\varepsilon_\theta + \varepsilon_p\right]\vec{uy}}{\left[\varepsilon_\theta + \varepsilon_p\right]\vec{ux}}\right)$$

$$\varepsilon_\theta = e^{-D_{recta}} \times \left[\cos(\theta_{recta} - \phi) \times \vec{ux} + \sin(\theta_{recta} - \phi) \times \vec{uy}\right]$$
$$\varepsilon_p = \left(1 - e^{-D_{recta}}\right) \times \left[\cos(k_d - \phi) \times \vec{ux} + \sin(k_d - \phi) \times \vec{uy}\right]$$

$$\gamma = \tan^{-1}\left(\frac{e^{-D_{recta}} \times \sin(\theta_{recta} - \phi) + \left(1 - e^{-D_{recta}}\right) \times \sin(k_d - \phi)}{e^{-D_{recta}} \times \cos(\theta_{recta} - \phi) + \left(1 - e^{-D_{recta}}\right) \times \cos(k_d - \phi)}\right)$$

$$\left\langle \begin{cases} v = \dfrac{Max_{speed}}{R_{roda}}, & se \ \ |\varepsilon_p| \geq 0.1 \\ v = 0, & se \ \ |\varepsilon_p| \leq 0.1 \end{cases} \right\rangle$$
$$\gamma = \tan^{-1}\left(\frac{e^{-D_{recta}} \times \sin(\theta_{recta} - \phi) + \left(1 - e^{-D_{recta}}\right) \times \sin(k_d - \phi)}{e^{-D_{recta}} \times \cos(\theta_{recta} - \phi) + \left(1 - e^{-D_{recta}}\right) \times \cos(k_d - \phi)}\right)$$

Figura 81 Diagrama geral do segue recta (1), Progressão do erro (2), Meio de convergência (3)

7.6.3 SEGUIR ORIENTAÇÃO

Esta manobra permite que o *LINCE* se oriente numa direcção. Assim o *LINCE* pode seguir numa dada direcção, desta forma o *LINCE* orienta-se para um ângulo que é introduzido nos parâmetros desta função.

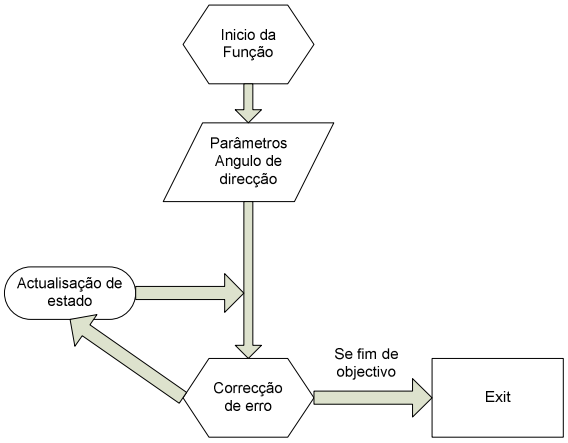


Figura 82 Fluxograma do algoritmo *Seguir orientação*

7.6.3.1 EXPLICAÇÃO DETALHADA DO ALGORITMO

Neste algoritmo, o LINCE orienta-se com um ângulo introduzido por parâmetro. Sendo assim o erro em teta a diferença entre parâmetro e ângulo das rodas de direcção. O LINCE orienta-se num dado ângulo e executa este movimento ate este se alinhar com o objectivo do algoritmo.

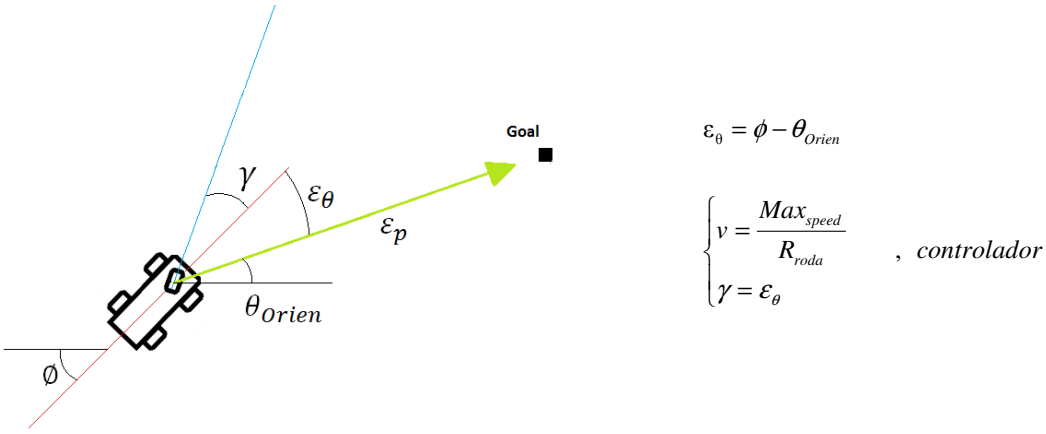


Figura 83 Ilustração do algoritmo "segue orientação"

7.6.4 SEGUIR *WAYPOINTS*

Para este tipo de composição de movimentos, o *LINCE* deve ser capaz de executar um conjunto de trajectórias que une os pontos recebidos por parâmetro. O objectivo deste tipo de composição passa por fazer com que o *LINCE* percorra um número de pontos de forma ordenada. Para tal este deve receber os *waypoints* num vector ordenado de forma a fazer com que este percorra o caminho que una os *waypoints*.

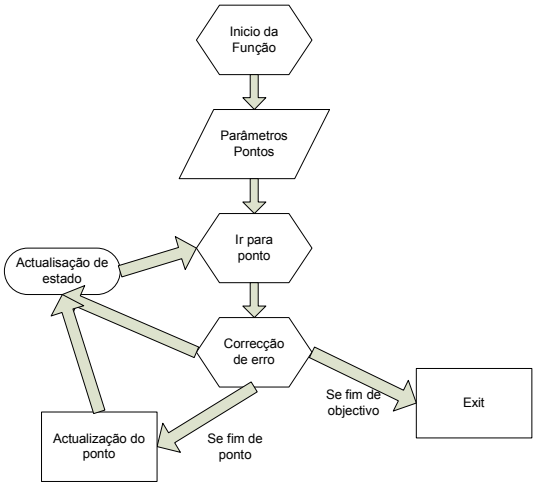


Figura 84 Fluxograma do algoritmo *seguir waypoint*

7.6.4.1 EXPLICAÇÃO DETALHADA DO ALGORITMO

Este algoritmo recebe um vector de pontos e para cada ponto realiza um "ir para". Quando atinge o primeiro ponto passa para o segundo e assim consecutivamente ate concluir todos os pontos.

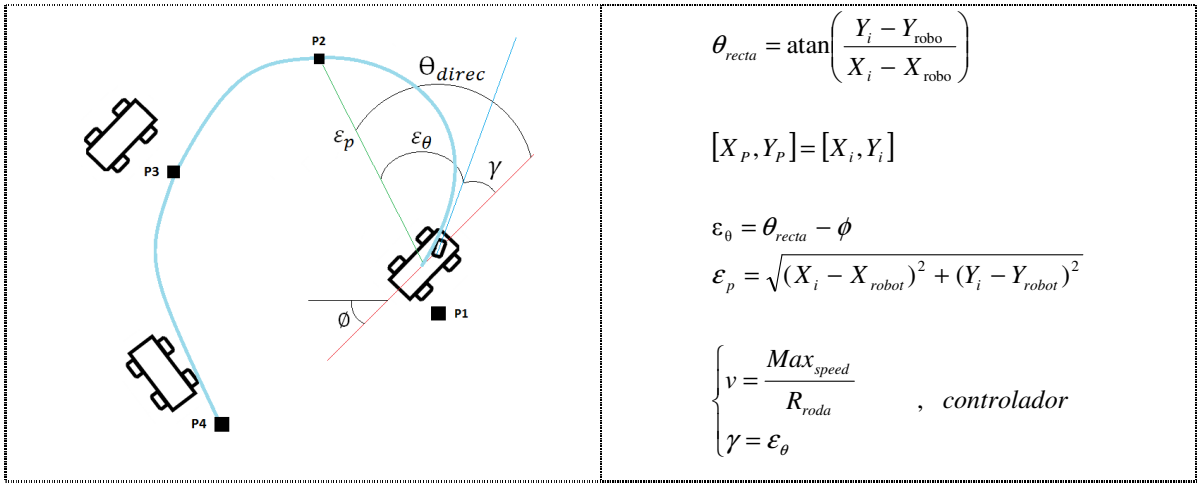


Figura 85 Ilustração da composição de waypoints

7.6.5 COMPOSIÇÃO DE RECTAS

Para este tipo de composição de movimentos, o *LINCE* deve ser capaz de executar um conjunto de trajectórias que une os pontos recebidos por parâmetro. O objectivo deste tipo de composição passa por fazer com que o *LINCE* percorra um número de pontos de forma ordenada sob trajectória rectilínea. Para tal este deve receber os *waypoints* num vector ordenado de forma a fazer com que este percorra caminhos que une os *waypoints*. O trajecto entre 2 *waypoints* deve ser feito segundo a função segue recta de forma a aproximar-se em distância e em teta de recta.

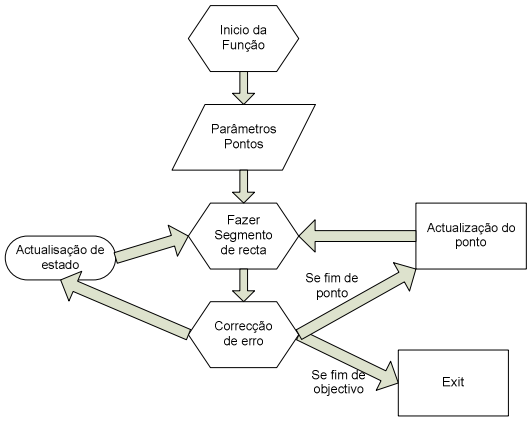
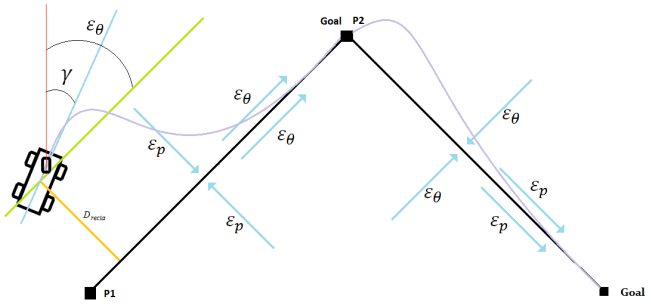


Figura 86 Fluxograma do algoritmo *composição de rectas*

7.6.5.1 EXPLICAÇÃO DETALHADA DO ALGORITMO

Este algoritmo recebe um vector de pontos e desde a posição inicial para a posição seguinte realiza a trajectória usando a função seguir recta. Desta forma o algoritmo realiza uma composição de rectas consecutivas até o ponto do objectivo *(Goal)*



$$\theta_{recta} = \operatorname{atan}\left(\frac{Y_{i1} - Y_{i2}}{X_{i1} - X_{i2}}\right)$$

$$[X_P, Y_P] = [X_i, Y_i]$$

$$\varepsilon_\theta = e^{-D_{recta}} \times \left[\cos(\theta_{recta} - \phi) \times \vec{ux} + \sin(\theta_{recta} - \phi) \times \vec{uy}\right]$$

$$\varepsilon_p = \left(1 - e^{-D_{recta}}\right) \times \left[\cos(k_d - \phi) \times \vec{ux} + \sin(k_d - \phi) \times \vec{uy}\right]$$

$$\gamma = \tan^{-1}\left(\frac{e^{-D_{recta}} \times \sin(\theta_{recta} - \phi) + \left(1 - e^{-D_{recta}}\right) \times \sin(k_d - \phi)}{e^{-D_{recta}} \times \cos(\theta_{recta} - \phi) + \left(1 - e^{-D_{recta}}\right) \times \cos(k_d - \phi)}\right)$$

$$\left\langle \begin{cases} v = \dfrac{Max_{speed}}{R_{roda}}, & se \ |\varepsilon_p| \geq 0.1 \\ v = 0, & se \ |\varepsilon_p| \leq 0.1 \end{cases} \right\rangle$$
$$\gamma = \tan^{-1}\left(\frac{e^{-D_{recta}} \times \sin(\theta_{recta} - \phi) + \left(1 - e^{-D_{recta}}\right) \times \sin(k_d - \phi)}{e^{-D_{recta}} \times \cos(\theta_{recta} - \phi) + \left(1 - e^{-D_{recta}}\right) \times \cos(k_d - \phi)}\right)$$

Figura 87 Ilustração do algoritmo composição de rectas

7.7 *API* DE CONTROLO

Esta API desenvolvida foi escrita na linguagem C. Foram usado as bibliotecas do *libplayerc* de forma a fazer a ligação ao servidor *player* e obter feedback dos actuadores e sensores do ambiente simulado. Esta API comunica com o servidor por *stack* TCP/IP usando também as bibliotecas do *libplayerc*.

A *API* de controlo desenvolvida permite escolher a função que o robô deve desempenhar. Depois de escolhida a função esta deve pedir os parâmetros respectivos para executar a função escolhida, depois, executa a função no simulador.

Enquanto a aplicação corre no simulador todos os parâmetros de leitura são actualizados e assim permite que se faça estimação e correcção da posição do robô no mundo gerado.

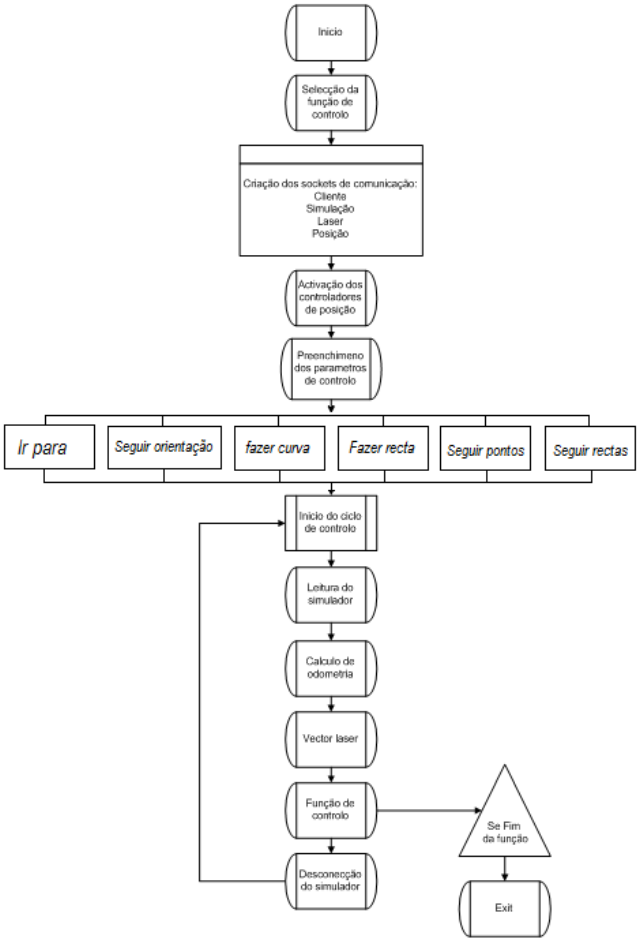


Figura 88 Fluxograma da *API* de controlo desenvolvida

7.7.1 CABEÇALHO DA *API* DE CONTROLO DEPOIS DE EXECUTADA

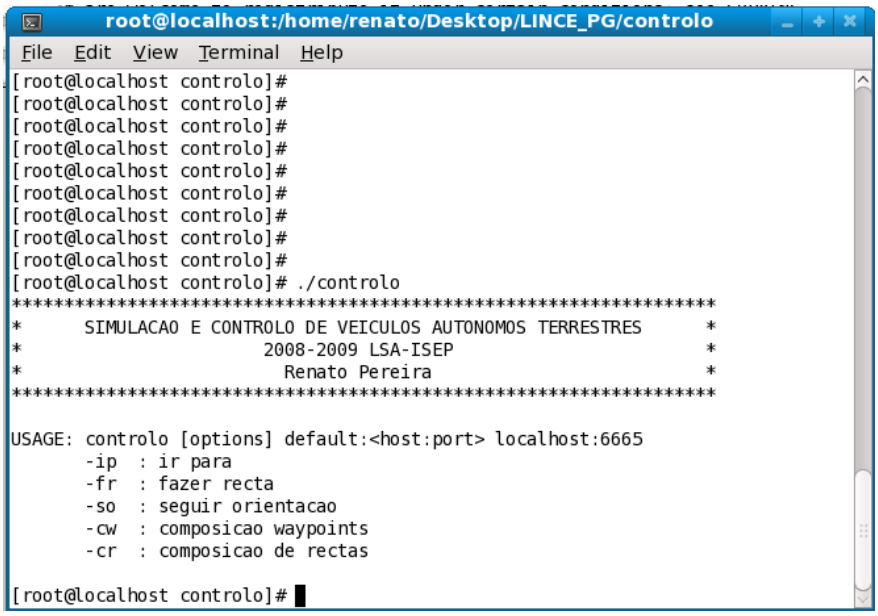


Figura 89 API de controlo

No cabeçalho da *API* de controlo podemos encontrar os parâmetros que o programa recebe de forma a escolher a função a implementar no simulador. As funções que este programa executa são:

- ✓ Ir para
- ✓ Fazer recta
- ✓ Seguir orientação
- ✓ Seguir *waypoints*
- ✓ Seguir *rectas*

Este programa comunica via *TCP-IP* com o *player* "Servidor" e este último com o *Gazebo* de forma a existir um *link* de comunicação entre *user* e simulador. Este mecanismo permite também de forma transparente abstrair o programador do ambiente simulado. Como já vimos nos capítulos anteriores.

8. RESULTADOS DA SIMULAÇÃO

8.1 IR PARA

Para esta simulação, o objectivo passa por fazer o veículo deslocar-se da posição inicial $x,y\,(0,0)$ para a posição final $x,y\,(10,10)$. Para isso, inicializou-se a API de controlo da seguinte forma: $./controlo-ip$ e quando pedidos, introduzimos os parâmetros $x,y\,(10,10)$ do *goal*.

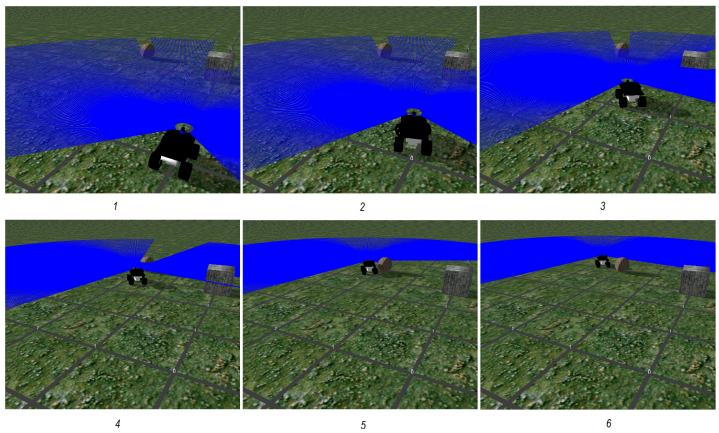


Figura 90 Visualização do resultado da simulação em Gazebo

Na imagem seguinte podemos observar a progressão do robo em posição X,Y em metros durante a trajectória efectuada.

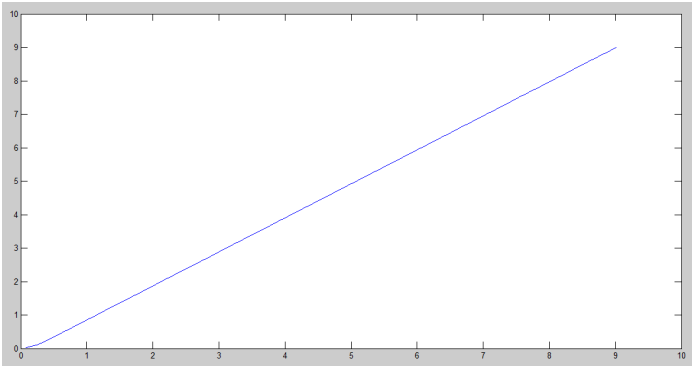


Figura 91 Progressão em posição X, Y

De seguida podemos observar a progressão do erro em teta do veiculo durante o percurso efectuado, onde o eixo dos Y representa o angulo em radianos e em X os pontos de simulação.

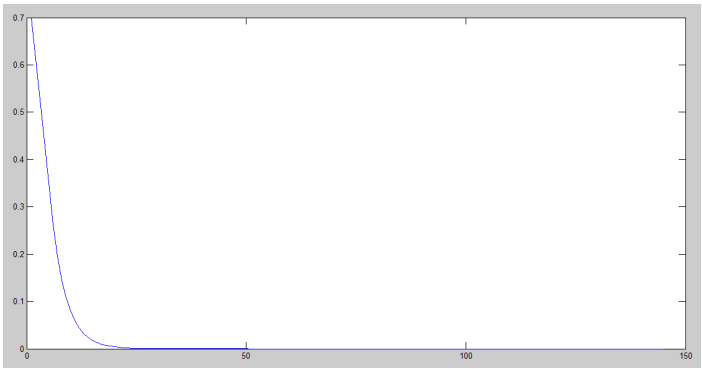


Figura 92 Progressão do erro em teta

Nesta situação podemos observar a progressão do erro em distância do veículo durante a trajectória efectuada, onde o eixo dos Y representa a distancia em metros e o eixo dos X os pontos de simulação.

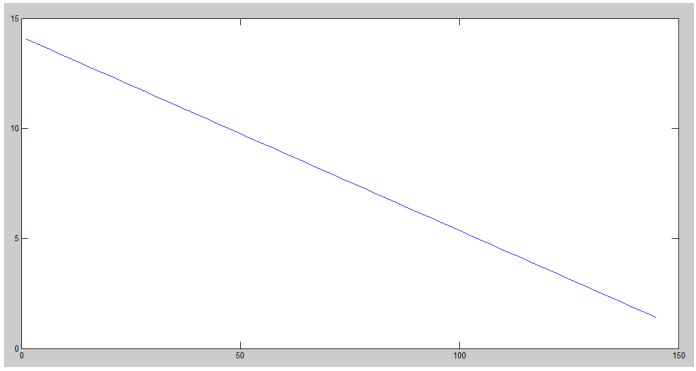
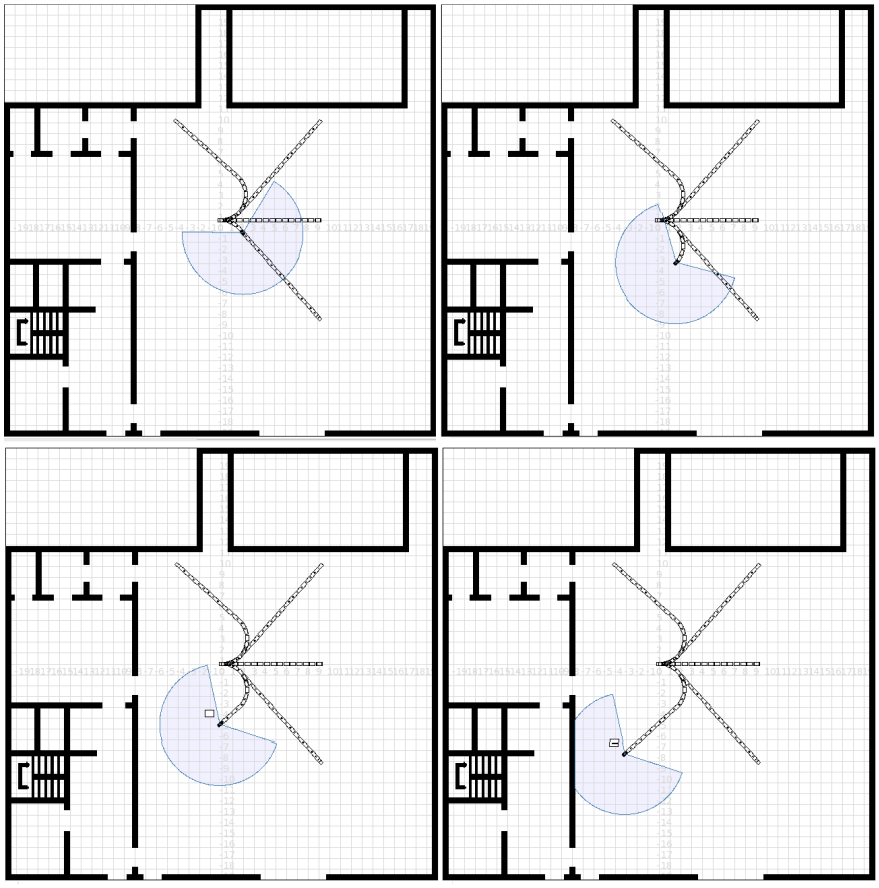


Figura 93 Progressão do erro em distância

A seguinte imagem ilustra a simulação em *Stage* 2D. Como podemos ver, testou-se o mesmo algoritmo para os seguintes pontos:



$x,y\ (-5,\ \ 10)$
$x,y\ (-5,-10)$
$x,y\ (\ 10,\ \ 10)$
$x,y\ (\ 10,\ \ \ 0)$
$x,y\ (\ 10,-10)$

Figura 94 Visualização do resultado da simulação do *Ir para* em Stage

Como podemos observar, o objectivo do algoritmo é fazer convergir o erro em teta e distância a zero. Desta forma podemos afirmar que o algoritmo converge para o objectivo da missão.

8.2 FAZER RECTA

Para esta simulação, o objectivo passa por fazer o veículo deslocar-se da posição inicial $x,y\,(0,0)$ e descrever um segmento de recta de coordenadas $x,y\,(0,5)$ e $x,y\,(10,10)$. Para isso, inicializou-se a API de controlo da seguinte forma: $./controlo-fr$ e quando pedidos introduzimos os parâmetros de $x,y\,(X1,Y1)$ e $x,y\,(X2,Y2)$, onde $x,y\,(X2,Y2)$ é o objectivo da missão.

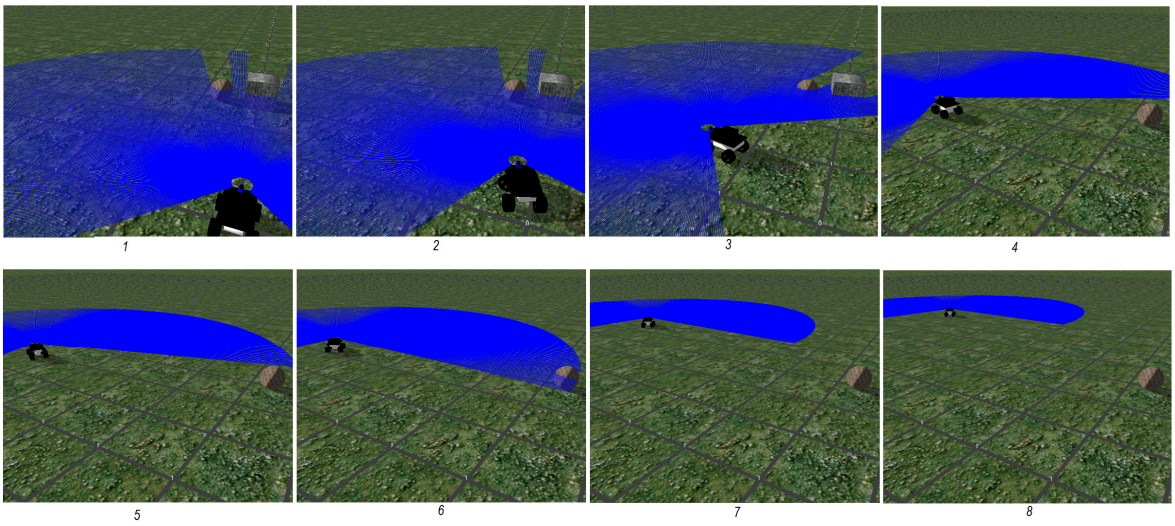


Figura 95 Visualização da simulação

Na imagem podemos observar a progressão do robo em posição X,Y em metros durante a trajectória efectuada.

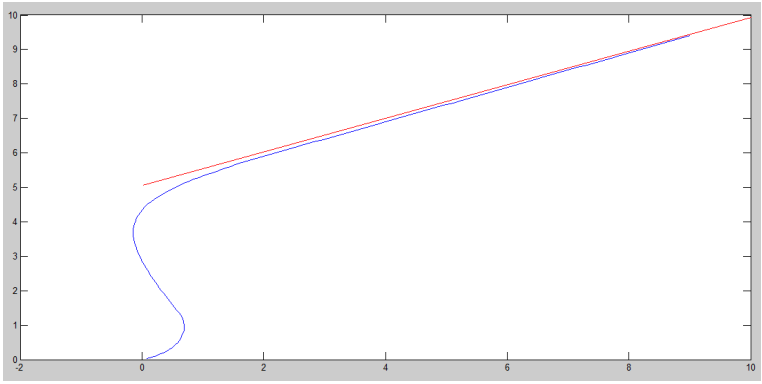


Figura 96 Progressão em posição X, Y

Na imagem seguinte podemos observar a progressão do erro em distância do veículo durante a trajectória efectuada, onde o eixo dos Y representam a distância e o eixo dos X o ponto de simulação.

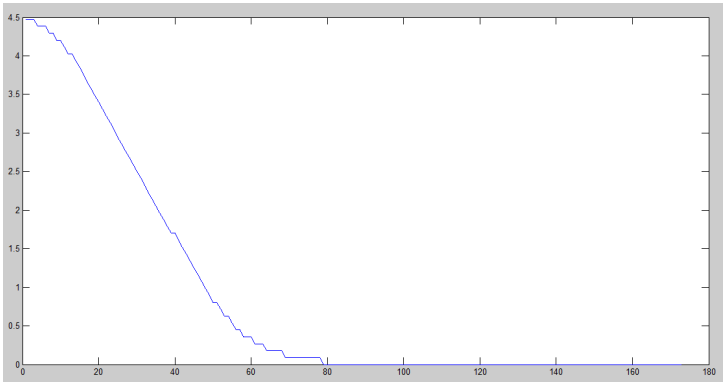


Figura 97 Distância entre robô e recta

Agora podemos observar o ganho resultante do erro em posição. O resultado deste ganho representa o peso que o factor distância tem em cada ponto de simulação.

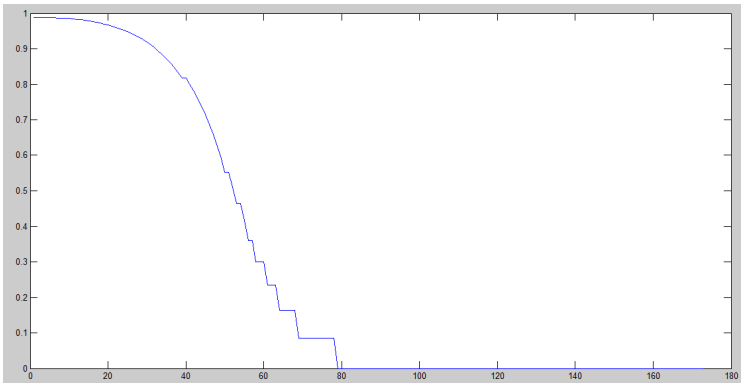


Figura 98 Erro em posição

Aqui podemos observar o ganho resultante do erro em teta. O resultado deste ganho representa o peso que o factor teta tem em cada ponto de simulação.

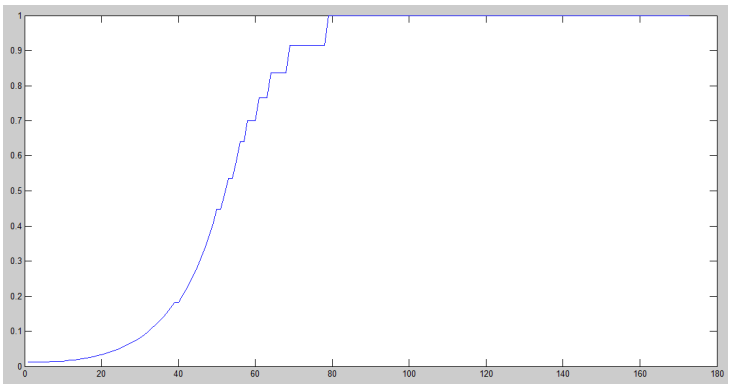


Figura 99 Erro em teta

Na imagem seguinte podemos observar o ângulo aplicado as rodas do robô. Este ângulo é o somatório dos ângulos do erro em teta e distância.

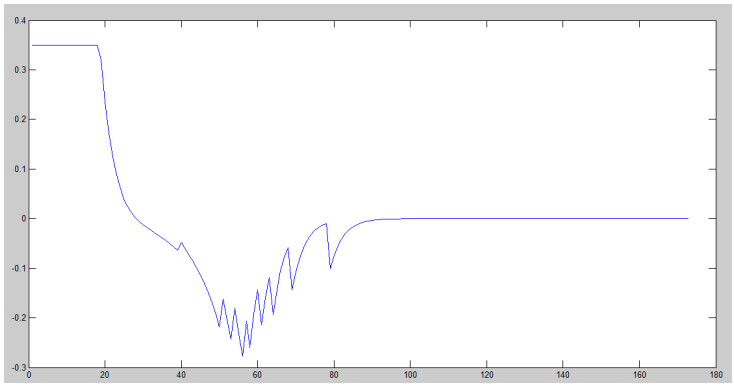


Figura 100 Ângulo aplicado a direcção

Como podemos observar, o objectivo do algoritmo é fazer convergir o erro em teta e distância a zero. Desta forma podemos afirmar que o algoritmo converge para o objectivo da missão.

8.3 SEGUIR ORIENTAÇÃO

Para esta simulação, o objectivo passa por fazer o veículo deslocar-se da posição inicial $x,y\,(0,0)$ e orientar-se segundo um ângulo $\theta_{\text{Re}\,ferencia}$. Para isso, inicializou-se a API de controlo da seguinte forma: $./controlo-so$ e quando pedidos, introduzimos o parâmetro $\theta_{\text{Re}\,ferencia}$ do objectivo da missão. Para a seguinte simulação, foi introduzido como parâmetro $\theta_{\text{Re}\,ferencia} = \frac{\pi}{2} rad$.

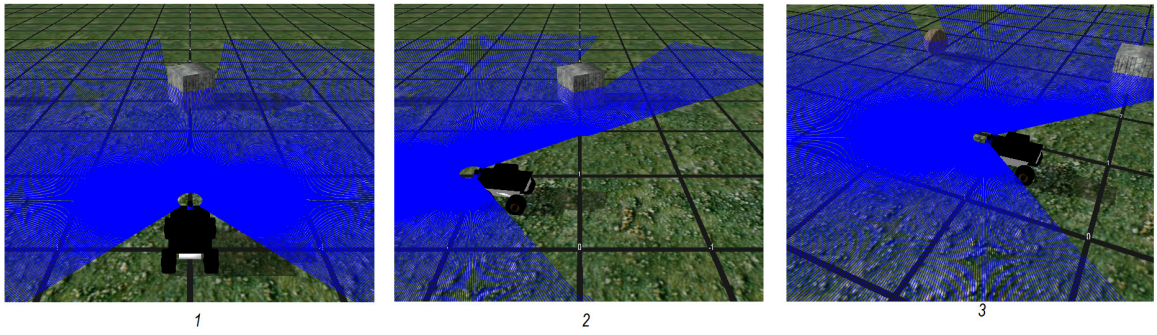


Figura 101 Visualização da simulação

Na imagem seguinte podemos observar a progressão do robo em posição X,Y em metros durante a trajectória efectuada.

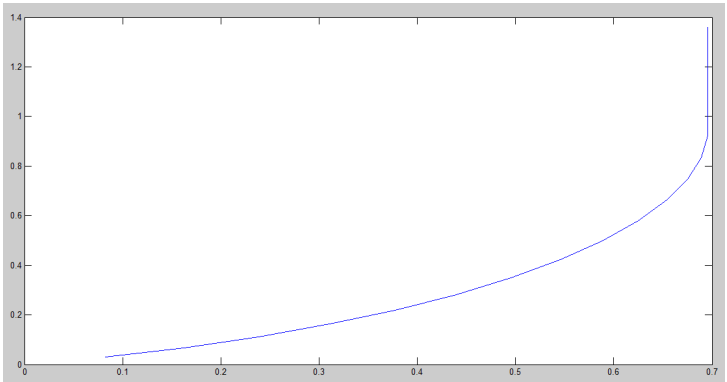


Figura 102 Progressão em posição X, Y

Seguidamente podemos observar a progressão do erro em teta do veiculo durante a tragectória efectuada, onde o eixo dos Y representa o angulo em radianos e wm X os pontos de simulação.

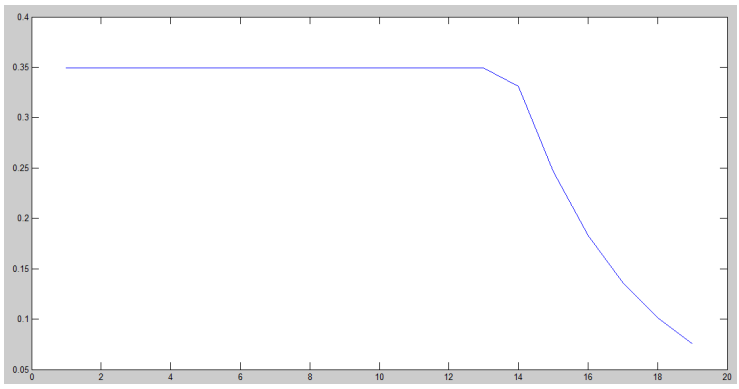


Figura 103 Progressão do erro em teta

Como podemos observar, o objectivo do algoritmo é fazer convergir o erro em teta a zero. Desta forma podemos afirmar que o algoritmo converge para o objectivo da missão.

8.4 COMPOSIÇÃO DE *WAYPOINTS*

Para esta simulação, o objectivo passa por fazer o veículo deslocar-se da posição inicial $x,y\,(0,0)$ para a posição final $x,y\,(5,0)$. Depois de este completar o primeiro objectivo passa a realizar o segundo, que neste caso é seguir para o ponto $x,y\,(5,5)$. O terceiro objectivo é deslocar-se até o ponto $x,y\,(0,5)$. O algoritmo só termina quando completar todos os objectivos. Para isso, inicializou-se a API de controlo da seguinte forma: $./controlo-cw$ e quando pedidos introduzimos os parâmetros x,y de todos os *waypoints* desejados.

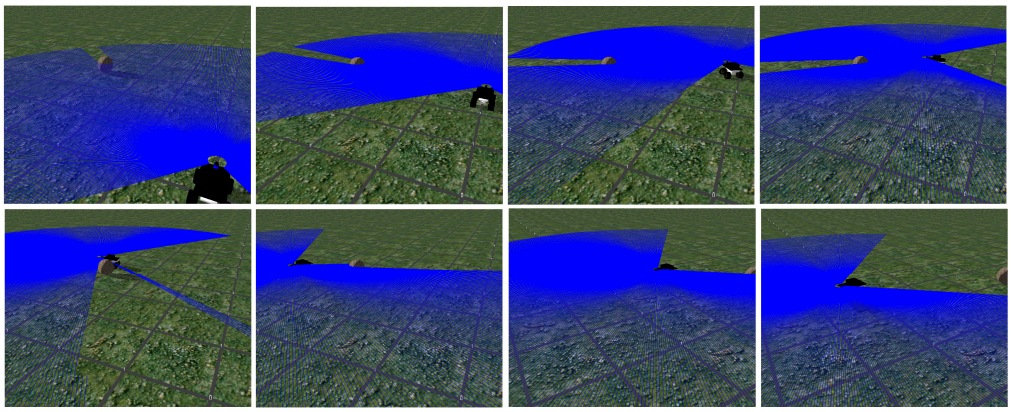


Figura 104 Visualização da simulação

Nesta imagem podemos observar a progressão do robo em posição X,Y em metros durante a trajectória efectuada.

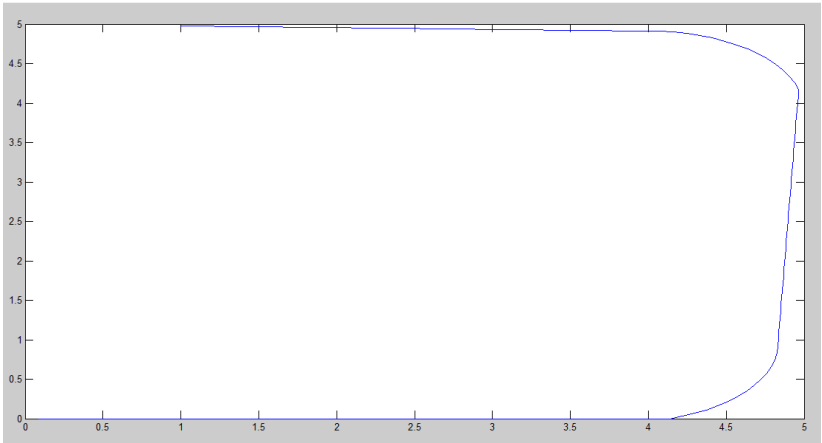


Figura 105 Progressão em posição X, Y

Aqui podemos observar a progressão do erro em teta do veiculo durante a tragectória efectuada. Cada um dos picos observados representam a mudança de objectivo de missão.

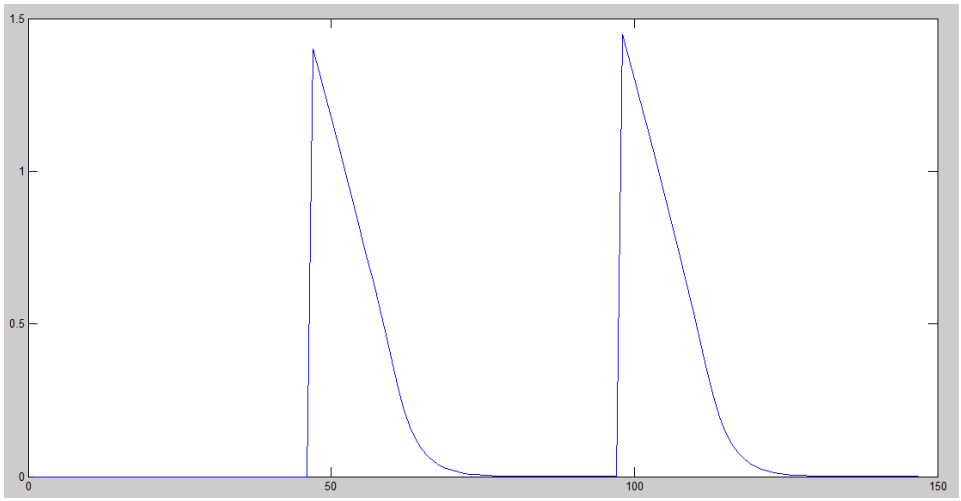


Figura 106 Progressão do erro em teta

Neste caso podemos observar a progressão do erro em distância do veículo durante a trajectória efectuada.

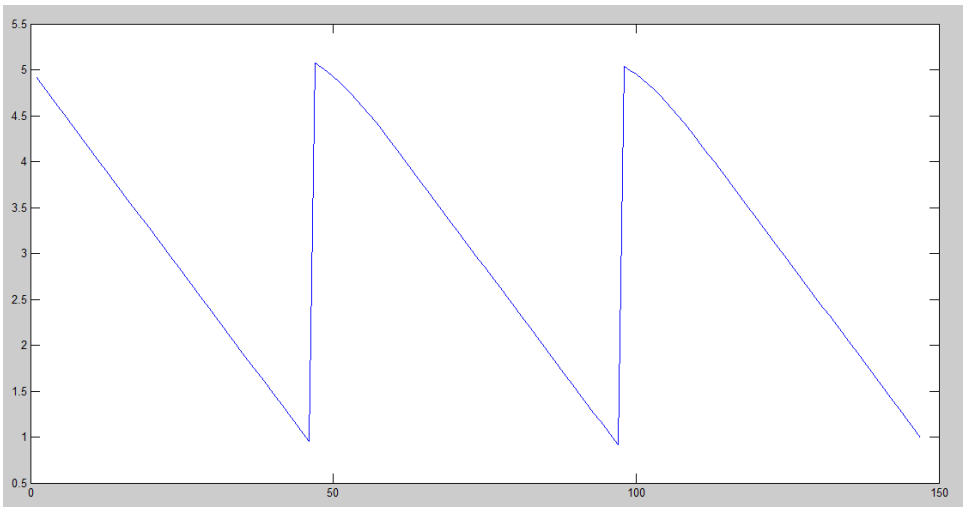


Figura 107 Progressão do erro em distância

A seguinte imagem ilustra a simulação em *Stage* 2D. Como podemos ver, testou-se o mesmo algoritmo para os seguintes pontos:

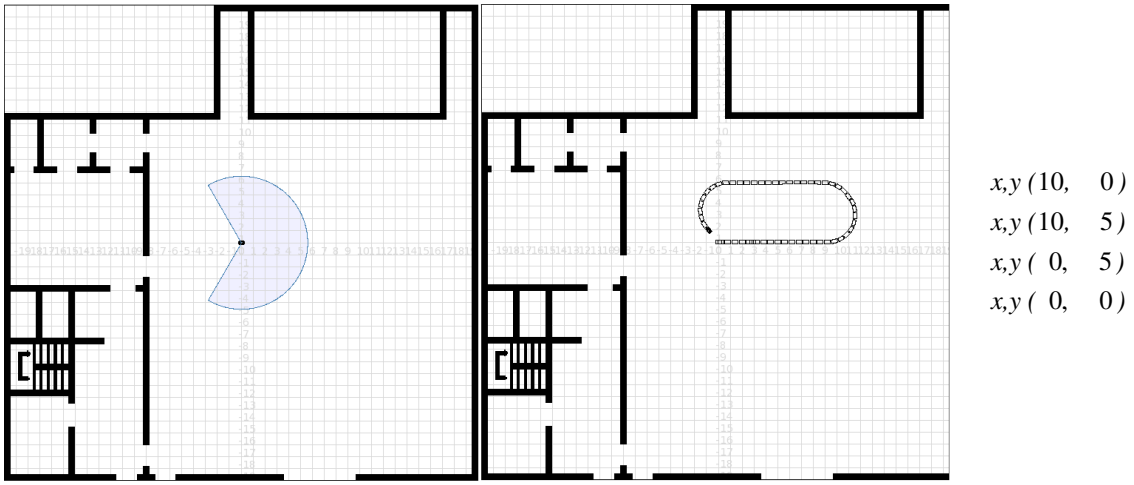


Figura 108 Visualização da simulação em Stage

Como podemos observar, o objectivo do algoritmo é fazer convergir o erro em teta e distância a zero. Desta forma podemos afirmar que o algoritmo converge para o objectivo da missão.

8.5 COMPOSIÇÃO DE RECTAS

Para esta simulação, o objectivo passa por fazer o veículo deslocar-se da posição inicial $x,y\,(0,5)$ para a posição final $x,y\,(5,5)$ descrevendo o segmento de recta segundo a função "*segue recta*". Depois de este completar o primeiro objectivo passa a realizar o segundo, que neste caso é seguir para o ponto $x,y\,(10,10)$ mantendo neste caso o segundo ponto da primeira recta como primeiro ponto da segunda recta. Para isso, inicializou-se a API de controlo da seguinte forma: $./controlo-cr$ e quando pedidos introduzimos os parâmetros x,y de todos os *waypoints* que compõem as 2 rectas.

Na imagem seguinte podemos observar a progressão do robo em posição X,Y em metros durante a trajectória efectuada.

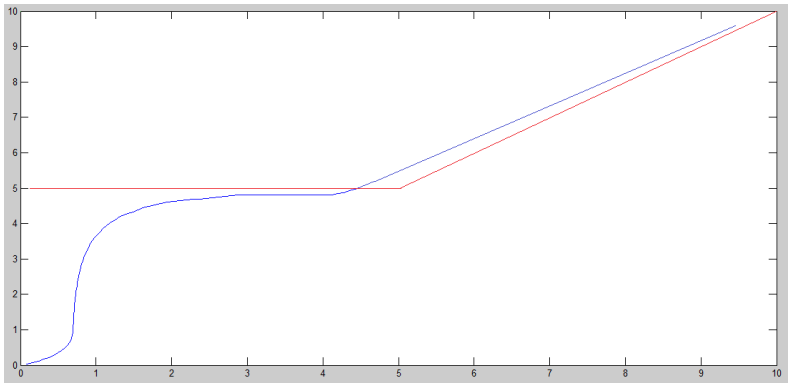


Figura 109 Progressão em posição X, Y

Seguidamente podemos observar a progressão do erro em distância do veículo durante a trajectória efectuada, onde o eixo dos Y representam a distância e o eixo dos X o ponto de simulação.

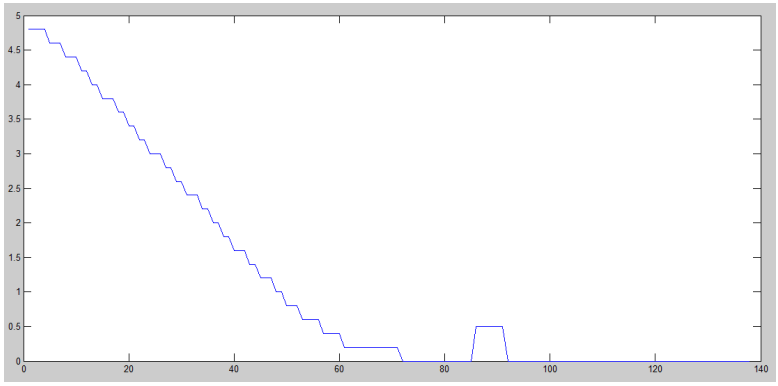


Figura 110 Progressão da distancio do robô a recta

Na seguinte imagem podemos observar o ganho resultante do erro em distância. O resultado deste ganho representa o ângulo que permite aproximar o robô a recta de uma forma perpendicular.

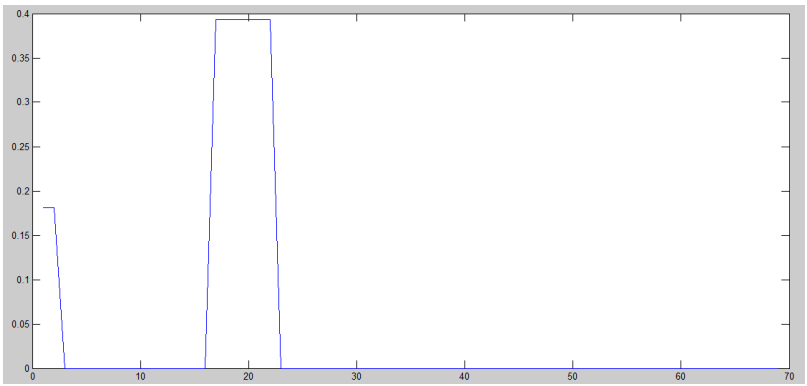


Figura 111 Evolução do erro em posição

Na seguinte imagem podemos observar o ganho resultante do erro em teta. O resultado deste ganho é o ângulo que permite alinhar o veículo com o ângulo da recta.

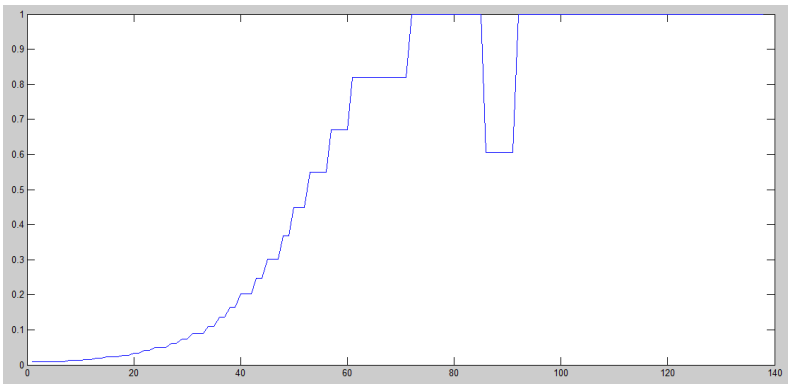


Figura 112 Evolução do erro em teta

Na imagem seguinte podemos observar o ângulo aplicado as rodas do robô. Este ângulo é o somatório dos ângulos do erro em teta e distância.

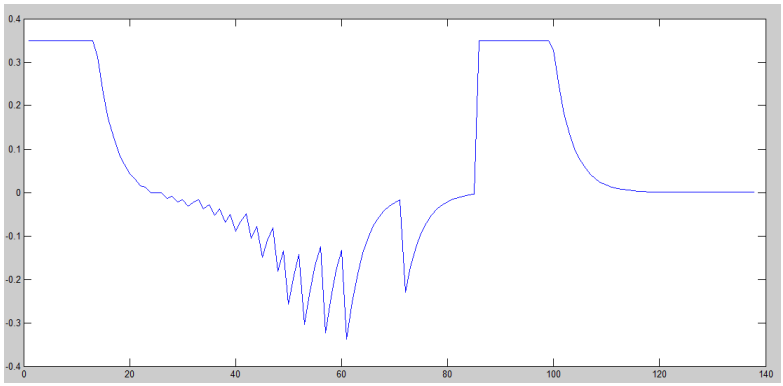


Figura 113 Evolução do ângulo aplicado as rodas

Como podemos observar, o objectivo do algoritmo é fazer convergir o erro em teta e distância a zero. Desta forma podemos afirmar que o algoritmo converge para o objectivo da missão.

9. CONCLUSÕES

Quando iniciado o estudo dos objectivos, foi feita uma pesquisa de vários simuladores de robôs existentes. Inicialmente o simulador *Gazebo* já era uma referência apontada para um dos possíveis candidatos a ser usado como material de estudo. Este foi escolhido por ser um simulador *OpenSource* e correr em Linux como foi referido no capítulo X. Contudo este simulador está ainda em fase de desenvolvimento, não existindo assim uma versão final.

Para instalar o simulador foi feita uma preparação do sistema operativo, pois para este simulador são necessárias várias bibliotecas e ferramentas para que mesmo corra de forma correcta, para além de ser necessárias algumas correcções no código fonte por inexistências de lincagens de várias bibliotecas, inexistências de variáveis, etc… Podemos observar o manual de instalação efectuado pelo candidato em anexo X, pois o manual de instalação publicado no site do simulador não contém a informação necessária para a instalação da versão do simulador utilizada, visto que no site do *Gazebo* apenas contém um guia de instalação para uma versão mais antiga deste simulador.

Para realização do modelo do robô foi feita uma pesquisa dos modelos ilustrados no site do simulador e por engenharia inversão foi desenhado o modelo do LINCE.

A versão do Gazebo utilizada é uma versão estável 0.8-pre3 e contem enumeras funcionalidades incompletas e com alguns *Bugs*. Aqui enuncio as funcionalidades mais relevantes para este projecto ainda por funcionar:

- ✓ Laser and Camera Interfaces

Apesar de estar incompleto pode-se obter imagens das câmaras simuladas, o problema é ocupar demasiado processador no computador (O pc fica muito lento).

O laser funciona correctamente, apenas não se consegue observar a reflectividade dos materiais.

- ✓ INS e GPS Interfaces

Não permite a utilização de sensores inércias, ainda está incompleto.

- ✓ Pose3d Interface

Não é possível retirar a posição real dos objectos simulados no simulador, incompleto

- ✓ Odometria para controlador steer não implementado

O controlador utilizado para controlar as rodas do veículo apenas funciona para enviar comandos para o simulador. Não existe retorno da posição do robô (por implementar). Para ultrapassar este problema foi implementada uma função Odometrica que integra o controlo efectuado, assim consegue-se obter a posição do robô no simulador através das equações de cinemática.

Apesar de tudo o simulador é capaz de simular a carroçaria do robô bem como o laser. Para além disso o ODE também simula o comportamento dinâmico do robô. Simula o comportamento das colusões, resvalamentos, etc…

O atrito simulado é um factor variável. Este faz com o comportamento do robô não seja sempre igual, dependendo do escorregamento efectuado pelas rodas do veículo simulado. O ideal seria obter a posição real do robô para que o algoritmo de controlo fizesse a

correcção do erro introduzido por este factor. Para isso seria útil neste curso uma cadeira orientada a programação por Objectos, pois o código fonte do simulador encontra-se escrito nesta linguagem.

Os algoritmos de controlo implementados foram feitos em linguagem *C*, utilizando as bibliotecas do *libplayerc* para fazer a conexão e tratamento de dados com o simulador. Esta API de controlo foi implementada a partir de uma API de exemplo fornecida pela equipa de desenvolvimento do Gazebo que está disponível no site do simulador.

Ao correr a API, verificou-se a inexistência da odometria no controlador de steer para veículos de 4 rodas como já foi referido. Assim foi necessário realizar esforços para realizar uma função que construísse a odometria do movimento a partir do controlo gerado pelas funções de movimento.

Os algoritmos de movimento implementados baseiam-se todos no modelo cinemático do robô. Os algoritmos implementados basearam-se nos mecanismos básicos para poder compor um conjunto de trajectórias ilimitadas. Os algoritmos desenvolvidos foram:

- ✓ Ir para
- ✓ Fazer recta
- ✓ Seguir orientação
- ✓ Composição de waypoints e de rectas

Estes algoritmos foram desenvolvidos respondendo a dois problemas, (Erro em teta e Erro em posição). Analisando a geometria de cada problema encontrou-se uma solução de forma a fazer convergir estes dois parâmetros a zero.

Em termos de conclusão, apesar de alguns algoritmos estarem incompletos, os objectivos foram cumpridos. Conseguiu-se simular o robô, sensores e actuadores, bem como comportamentos em modo de condução tele-operada. Mostrou-se ser possível criar várias configurações de mundos diferentes usando blocos de forma a construir uma missão

desejada. Os algoritmos de controlo testados convergiram para o objectivo para o qual foram desenhados.

A ferramenta está operacional e pode ser de grande utilidade para o laboratório. Esta ferramenta pode e deve ser utilizada para validar futuras alterações neste e noutros robôs deste laboratório. Desta forma consegue-se minimizar tempo e custos no desenvolvimento de aplicações robóticas autónomas.

Trabalhos futuros

Gostaria de acabar este trabalho de uma forma diferente. No meu entender, a odometria é um péssimo estimador de posição, pois está sujeito a erros enormes. Ao longo do tempo este acaba por dar uma posição do robô completamente errada da real. Seria útil a interface do Gazebo ter o GPS e INS a funcionar. Com isso poderia ser feito um estimador de posição fundindo GPS com INS e odometria. Desta forma teria um sistema de posicionamento muito mais eficiente que o que está implementado, maximizando o potencial do simulador bem como fiabilidade de resposta dos algoritmos desenvolvidos.

REFERÊNCIAS DOCUMENTAIS

[1.] *Annykode simulator*. (s.d.). Obtido de http://www.anykode.com/index.php

[2.] Author: Eric Berger, K. C. (s.d.). *ROS*. Obtido de http://www.ros.org/wiki/simulator_gazebo/Tutorials

[3.] *Blender*. (s.d.). Obtido de http://www.blender.org/

[4.] *Breve Simulator*. (s.d.). Obtido de http://www.spiderland.org/

[5.] Brooks, R. A. *A HARDWARE RETARGETABLE DISTRIBUTED LAYERED ARCHITECTURE FOR MOBILE ROBOT CONTROL.* MIT Artificial Intelligence Lab.

[6.] *Cyberbotics*. (s.d.). Obtido de Webot: http://www.cyberbotics.com/products/webots

[7.] DANIEL CRUZ, J. M. (2007). *A MULTIVEHICLE PLATFORM FOR RESEARCH IN NETWORKED EMBEDDED SYSTEMS.* University of Pennsylvania.

[8.] Durrant-Whyte, H. (2004). *Introduction to Estimation and Data fusion.* The University of Sydney.

[9.] *HECSSY*. (s.d.). Obtido de http://blog.csdn.net/hecssy/archive/2008/09/29/2998813.aspx

[10.] Koenig, N. a. (2004). *A. Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator.* IEEE International Conference on Robotics and Automation.

[11.] Koenig, N. (2007). Gazebo The Instant Expert's Guide. Munich Germany.

[12.] Kumar, V. (s.d.). *MAST Simulation* . Obtido de http://alliance.seas.upenn.edu/~kumar/wiki/index.php?n=Main.MASTSimulation

[13.] *Libplayerc Client Library Reference*. (s.d.). Obtido de http://playerstage.sourceforge.net/doc/Player-1.6.5/player-html/group__player__clientlib__libplayerc.php

[14.] *Marhes*. (s.d.). Obtido de Multi-Agent, Robotics, Hybrid, and Embedded Systems Laboratory: http://marhes.ece.unm.edu/index.php/Main_Page

[15.] Michel, O. W. (1998). *A Powerful Realistic Mobile Robots Simulator.* LNAI Springer-Verlag.

[16.] *Microsoft Robotics Developer Studio*. (s.d.). Obtido de http://msdn.microsoft.com/pt-pt/library/bb648760%28en-us%29.aspx

[17.] *Nabble*. (s.d.). Obtido de Free Forums + Other Embeddable Apps: http://old.nabble.com/Player-Stage-Gazebo-f4302.html

[18.] Oliveira, H. F. (2007). *Análise do Desempenho e da Dinâmica de Robôs Omnidireccionais de Três e Quatro Rodas.* Porto.

[19.] *Open Dynamics Engine*. (s.d.). Obtido de ODE: http://www.ode.org/

[20.] *Open Source 3D Graphics Engine*. (s.d.). Obtido de Ogre: http://www.ogre3d.org/

[21.] *Opensimulator*. (s.d.). Obtido de http://opensimulator.org/wiki/Main_Page

[22.] Page, G. S. (s.d.). *Gazebo*. Obtido de http://playerstage.sourceforge.net/gazebo: http://playerstage.sourceforge.net/gazebo

[23.] *Pyro stands for Python Robotics*. (s.d.). Obtido de http://pyrorobotics.org/?page=The_20Gazebo_20Simulator

[24.] Quiñones, D. I. (2007). *Controlo de robo autónomo movel para recolher lixo.* São Paulo.

[25.] Rahhe Agate, D. W. *Control architecture characteristics for intelligence in autonomous mobile construction robots.* In 23rd International Symposium on Automation and Robotics in Construction, ISARC2006, Japan.

[26.] *Simbad*. (s.d.). Obtido de Simbad Project Home: http://simbad.sourceforge.net/

[27.] Soares, A. (2007). *Arquitectura para criação de robos autónomos emocionais-cognitivos numa perspectiva situada.* Belo Horizonte.

[28.] SOARES, A. H. (2007). *ARQUITETURA PARA CRIAÇÃO DE ROBÔS AUTÔNOMOS EMOCIONAIS-COGNITIVOS NUMA PERSPECTIVA SITUADA.* Minas Gerais.

[29.] *Webot*. (s.d.). Obtido de Cyberbotics: http://www.cyberbotics.com/

Anexo A. Guia de Instalação do simulador

Em primeiro lugar é de salientar que o computador deve ter uma placa gráfica NVIDIA. Em segundo lugar devemos alterar os caminhos de (Bin, include e library_path).

```
export PATH = /usr/local/bin : $PATH
export CPATH = /usr/local/include : $CPATH
export LIBRARY_PATH = /usr/local/lib : $LIBRARY_PATH
```

Depois configurar o pkg-config path

```
Export PKG_CONFIG_PATH = /usr/local/lib/pkgconfig : $PKG_CONFIG_PATH
```

Depois instalar as seguintes dependências:

```
yum install fltk fltk - devel
yum install zziplib zziplib - devel
yum install freeimage freeimage - devel
rpm - ivh Cg - 2.0.0015 - 0.i386.rpm
yum install ois ois - devel
yum install ois ois - devel
yum install ogre ogre - devel
yum install ode ode - devel
yum install scons
yum install libtool - ltdl - devel
yum install gsl gsl - devel
yum install opencv opencv - devel
yum install libraw1394
yum install libraw1394 - devel
yum install libavc1394
yum install libavc1394 - devel
yum install libdc1394 libadc1394 - devel
yum install libjpeg libjpeg - devel
```

Python vinculos para Player/Stage:

```
mkdir - p /usr/local/lib/python2.5/
cd /usr/local/lib/python2.5/
ln - s /usr/lib/python2.5/site - packages
```

Add to /etc/bashrc:

```
export PYTHONPATH = /usr/local/lib/python2.5/site - packages : $PYTHONPATHs
```

Instalar o Player

```
tar xvzf player - 2.1.1.tar.gz
cd player - 2.1.
./configure
make
make install
```

Running:

```
player /usr/local/share/player/config/pioneer.cfg
```

Install Gazebo 0.8-pre3:

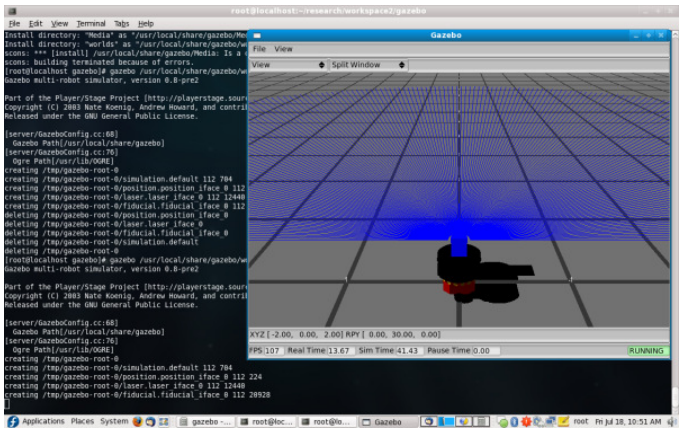
```
tar xvzf gazebo - 0.8 - pre3.tar.gz
cd gazebo - 0.8 - pre3
scons
scons install
```

Correndo Gazebo 0.8

```
gazebo < xmlfile >
```

Correndo um exemplo do gazebo project:

gazebo /usr/local/share/gazebo/worlds/pioneer2dx.world



Player & Gazebo juntos

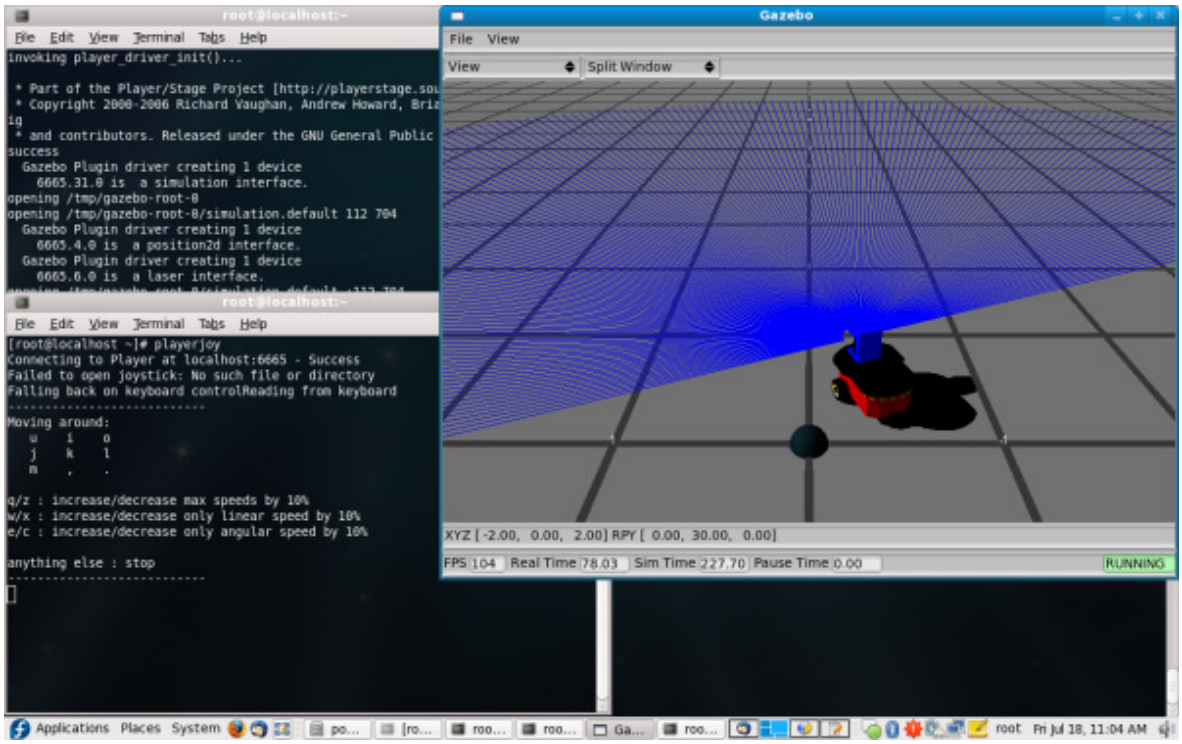
Primeiro:

player /path/to/your/gazebo_rev6886.cfg

Depois:

gazebo /usr/local/share/gazebo/worlds/pioneer2dx.world

Depois de ter-mos o servidor e o simulador a correr podemos controlar o robô com a aplicação playerjoy que permite controlar o robô com o joystick



Depois deste guia de instalação não é garantido que o simulador fique instalado correctamente. O ideal é ter paciência e observar bem o resultado das compilações para verificar e identificar os erros de compilação.