Robô Reativo Simples

Simple Reative Robot

Miguel Ribeiro, Pedro Oliveira
Robótica., Faculdade de Engenharia, Universidade do Porto
Porto, Portugal
ei11144@fe.up.pt, ei11123@fe.up.pt

Resumo — **Este documento descreve o funcionamento de um robô reativo simples capaz de se deslocar e resolver um labirinto. Esta implementação foi realizada sob a plataforma ROS e com o uso do simulador STDR.**

Palavras Chave - Robô; Reativo; ROS; STDR.

Abstract — **This document describes the operation of a simple reactive robot able to move around and solve a maze. This implementation was performed under the ROS platform and using the simulator STDR.**

Keywords - Robot; Reactive; ROS; stdr.

I. INTRODUÇÃO

Para a implementação do robô reativo foi usada a plataforma de desenvolvimento ROS [1]. Esta plataforma é *opensource* e ideal para desenvolvimento de software para robôs. Este sistema, oferece uma grande abstração o que facilita bastante no processo de construção do *hardware* e controlo de baixo nível. O ROS usa uma comunicação *pear-to-pear*, utilizando um protocolo de comunicação síncrono RPC. Uma das principais vantagens do ROS é o poder da reutilização de investigação e desenvolvimento realizada por outros. Assim podem atuar numa espécie de *framework* de processos, também chamados de nós, que podem ser usados de forma independente por outros desenvolvedores. Desta forma, quem desenvolve na plataforma ROS beneficia de vários nós que interagem entre si.
Outra estrutura importante são os tópicos, que são responsáveis por escutar as mensagens trocadas entre nós.

II. OBJECTIVOS

O principal objetivo deste projeto, é a criação de um robô puramente reativo capaz de resolver um qualquer labirinto, detetando e evitando todos os obstáculos durante o seu percurso desde o inicio até à saída.

III. SIMULADOR

Após a realização de um estudo das tecnologias existentes, concluímos que o desenvolvimento do programa deveria ter por base o uso do pacote "stdr simulator".

O STDR Simulator implementa uma arquitetura cliente-servidor. Cada nó pode ser executado numa máquina diferente e comunicar usando interfaces ROS. O STDR Simulator, também fornece uma interface gráfica desenvolvida em QT, para fins de visualização. A GUI, não é necessária para o simulador para executar as suas funcionalidades, contudo pode ser utilizada utilizando ferramentas de linha de comando fornecidos com o pacote.

Os pacotes disponíveis no ambiente são:

- **stdr_server:** implementa sincronização e coordenação de funcionalidades de strd_simulator.
- **stdr_robot:** fornece o robô e a implementação de sensores.
- **stdr_parser:** executa o *parse* de ficheiros yaml e xml.
- **stdr_gui:** fornece a interface GUI em QT.
- **stdr_msgs:** usada para mensagens, serviços e ações.
- **stdr_lauchers:** executa ficheiros.
- **stdr_resources:** fornece descrição do robô e dos sensores.
- **stdr_samples:** fornece códigos simples para demonstração das funcionalidades.

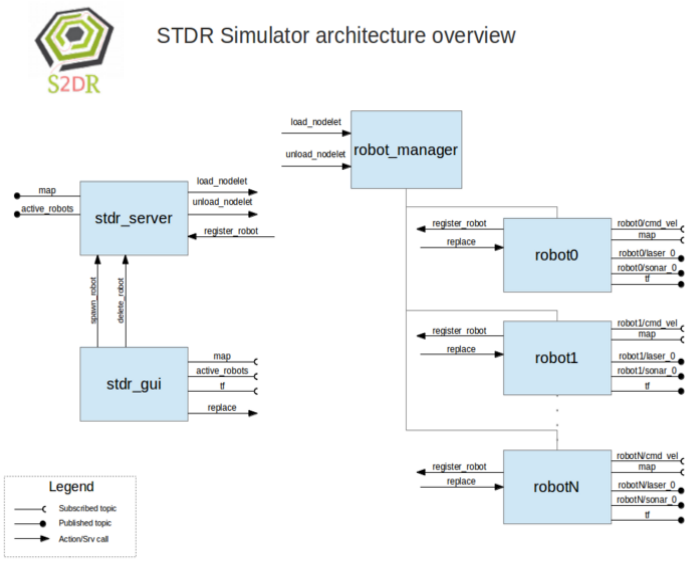

Fig.1 – Arquitectura STDR SImulator

IV. ALGORITMO USADO

O algoritmo usado adota uma estratégia puramente reativa. Segundo Ronald C. Arkin (1995), esta metodologia permite uma resposta rápida e flexível do sistema. É caracterizado por uma estrutura baseada em blocos de comportamento com diferentes prioridades e por não guardar qualquer informação sobre o estado do mundo e reagindo apenas às informações do presente. Para além disso, o tipo de arquitetura em que se baseia o algoritmo denomina-se *subsumption architecture* (Brooks 1986). Isto é, o comportamento do robô está dividido em diferentes camadas que implementam um objetivo especifico.
No seu conjunto, o controlo do robô é composto por uma serie de camadas, com diferentes níveis, e que pressupõe uma ordem de prioridade e subsunção.
Para evitar um obstáculo é usado um algoritmo de contorno de paredes, mantendo esta estratégia até que encontre a saída do labirinto.
Este método consegue resolver diferentes tipos de configurações.

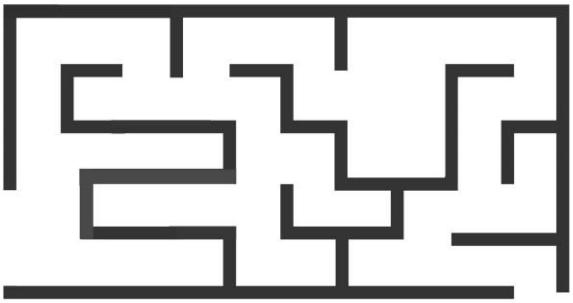

Fig.2 – Exemplo de labirinto resolvido pelo robô

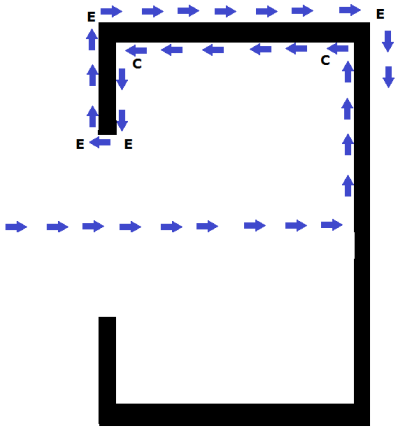

Fig.3 – Exemplo de um obestáculo contornavél.

A figura3 representa um obstáculo complexo que pode ser evitado com o algoritmo desenvolvido.
O robô começa por se deslocar em frente. Quando verifica que há uma parede à sua frente, roda no sentido contrário ao dos ponteiros do relógio até encontrar uma parede do seu lado direito (cerca de 90º). Depois, continua em frente. Nos pontos C (e de forma generalizada, em todos os cantos) o mecanismo é semelhante ao descrito anteriormente. No caso dos pontos E, que representam esquinas, o robô posiciona-se de forma a afastar-se o suficiente para conseguir rodar no sentido dos ponteiros do relógio e seguir em frente sem bater na esquina.
No entanto, este algoritmo não consegue resolver um tipo especifico de labirintos. Como não é guardada nenhuma informação sobre o contorno da parede (mesmo o próprio facto de a estarmos a contornar), torna-se difícil evitar que o robô entre em ciclos, a não ser que consiga obter informação do objetivo possível de abortar o contorno e que não o faça recomeçar a contornar a mesma parede de novo.

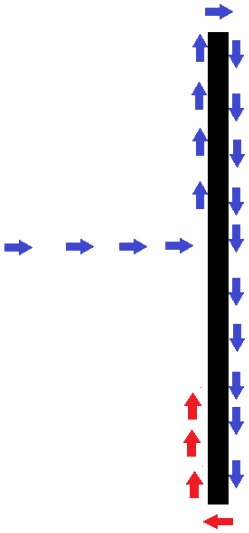

Fig.4 – Exemplo de obestáculo que consuz a ciclo.

A figura 4 exemplifica o caso em que o algoritmo desenvolvido entra em ciclo. Seja considerada uma parede isolada e sem ligações em frente ao ponto de inicio. Quando o robô se aproxima de P1, começa a contornar a parede. No entanto, como em nenhum momento do contorno, o robô obtém informação do objetivo, acaba por entrar em ciclo. Mais uma vez como não é guardada nenhuma informação é impossível, com esta abordagem evitar este tipo de obstáculos.
O problema descrito podia ser evitado caso se implementasse um algoritmo mais complexo, o que acabaria por tornar o robô menos reativo.

V. CONCLUSÕES

A solução apresentada, baseia-se numa abordagem puramente recativa e num algoritmo de contorno de paredes. Apesar de resolver uma inúmera quantidade de labirintos existe pelo uma pequena situação em que o robô falha a sua tarefa.
De forma genérica, quando no contorno da parede não se obtém a informação necessária para terminar, o robô acaba por entrar num ciclo.
Assim, a evolução natural deste algoritmo seria, alterar a abordagem puramente reativa e guardar informação que permitisse detetar e tentear mitigar os problemas encontrados.

REFERÊNCIAS BIBLIOGRÁFICA

[1] Arkin, R. (1995). Reactive robotic systems. The handbook of brain theory and neural networks.

[2] Brooks, R. (1986). A robust layered control system for a mobile robot. Robotics and Automation, IEEE Journal of.

[3] Luís, P., Martins, B., Almeida, P., e Silva, V. (1999). Deteccão de configurações de obestáculos perigosas: aplicação no rob^o EnCuRRalado.

[4] Technical overview, available at http://wiki.ros.org/ROS/Technical%20Overview

[5] J. Faust, turtlesim, available at http://wiki.ros.org/turtlesim

[6] D. Lu, M. Fergunson, map_server, available at http://wiki.ros.org/map_server

[7] M. Tsardoulias, C. Zalidis, A. Thallas, stdr_simulator, available at http://wiki.ros.org/stdr_simulator