# UMA ARQUITETURA DISTRIBUÍDA DE HARDWARE E SOFTWARE PARA CONTROLE DE UM ROBÔ MÓVEL AUTÔNOMO

Ricardo de Sousa Britto,* Adelardo Adelino Dantas de Medeiros,* Pablo Javier Alsina*

*Universidade Federal do Rio Grande do Norte
Departamento de Engenharia de Computação e Automação
Natal, Rio Grande do Norte, Brasil

Email: [rbritto, adelardo, pablo]@dca.ufrn.br

**Abstract**— In this paper, we present a hardware-software architecture for controlling the autonomous mobile robot Kapeck. The Kapeck robot is composed of a set of sensors and actuators organized in a CAN bus. Two embedded computers and eigth microcontroller-based boards are used in the system. One of the computers hosts the vision system, due to the significant processing needs of this kind of system. The other computer is used to coordinate and access the CAN bus and to accomplish the other activities of the robot. The microcontroller-based boards are used with the sensors and actuators. The robot has this distributed configuration in order to exhibit a good real-time behavior, where the response time and the temporal predictability of the system is important. We adopted the hybrid deliberative-reactive paradigm in the proposed architecture to conciliate the reactive behavior of the sensors-actuators net and the deliberative activities required to accomplish more complex tasks.

**Keywords**— Control Architecture, CAN Bus, Hybrid Deliberative-Reactive Paradigm.

**Resumo**— Neste artigo, apresenta-se uma arquitetura de hardware e software para controle do robô móvel autônomo Kapeck. O robô Kapeck é composto por um conjunto de sensores e atuadores organizados em um barramento CAN. Dois computadores embarcados e oito placas microcontroladas foram utilizadas no sistema. Um dos computadores foi utilizado para o sistema de visão, devido à grande necessidade de processamento deste tipo de sistema. O outro computador foi utilizado para coordenar e acessar o barramento CAN e realizar as outras atividades do robô. Placas microcontroladas foram utilizadas nos sensores e atuadores. O robô possue esta configuração distribuída para um bom desempenho em tempo-real, onde os tempos de resposta e a previsibilidade temporal do sistema são importantes. Foi seguido o paradigma híbrido deliberativo-reativo para desenvolver a arquitetura proposta, devido à necessidade de aliar o comportamento reativo da rede de sensores-atuadores com as atividades deliberativas necessárias para realizar tarefas mais complexas.

**Palavras-chave**— Arquitetura de Controle, Barramento CAN, Paradigma Híbrido Deliberativo-Reativo.

## 1 Introdução

O projeto de uma arquitetura de controle visa habilitar um robô móvel autônomo a operar em seu ambiente utilizando-se de seus recursos físicos e computacionais. Seu sistema de controle deve assegurar o cumprimento de suas tarefas de maneira estável e robusta. Muitas são as arquiteturas propostas, porém não existe um paradigma definitivo, que atenda a todas as funcionalidades requeridas (Medeiros, 1998).

Os paradigmas mais representivos são os paradigmas hierárquico, reativo e híbrido (deliberativo-reativo). Algums exemplos de arquiteturas de controle existentes na literatura (Murphy, 2000) são as arquiteturas NASREM, Subsunção , AuRa, SFX, Saphira, TCA, PyramidNet (Roisenberg et al., 2004), SHA (Kim et al., 2003), arquitetura híbrida proposta por Adouane (Adouane and Le Fort-Piat, 2004), COHBRA (Heinen, 2002), arquitetura de Ly (Ly et al., 2004) e arquitetura LAAS (Alami et al., 1993).

Uma arquitetura de controle bem definida é essencial para a implementação de robôs móveis complexos. Ela engloba diferentes módulos de hardware e software e determina como esses devem ser implementados, assim como o relacionamento entre eles.

Este trabalho possue como principais objetivos a especificação de uma arquitetura de hardware e software para o robô Kapeck, por meio da qual será implementada uma organização de controle, visando prover mobilidade e autonomia para o robô. Uma aplicação exemplo foi implementada para demonstração da arquitetura aqui proposta.

## 2 Arquitetura de Hardware

O Kapeck é um robô multi-tarefa não-holonômico que se locomove através de rodas com acionamento diferencial. Ele possue dois manipuladores com 5 graus de liberdade, uma cabeça estéreo com duas câmeras e um colar com 8 sonares (Os sonares possuem alcance máximo de oito metros). O robô pode ser visto na Figura 1.

O sistema de locomoção diferencial é composto por duas rodas independentes com um motor CC cada e duas rodas soltas. Cada roda motorizada possue um encoder acoplado, que gera 1024 pulsos por revolução.

Os manipuladores possuem cinco juntas e uma garra, com um motor CC acoplado a cada uma delas. Em cada manipulador, quatro juntas pos-

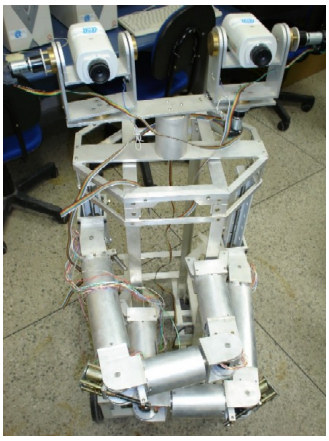

Figura 1: Robô Kapeck

suem potenciômetros acoplados, uma junta (junta do ombro) possue um encoder, que gera 1024 pulsos por revolução, e um fim de curso. A garra não possue realimentação para acionamento do motor. No estágio atual do projeto, os módulos referentes aos manipuladores do robô Kapeck ainda não foram incluídos na arquitetura, mas a característica modular da arquitetura aqui proposta facilita esta inclusão no futuro.

A cabeça estéreo possue cinco motores CC. Cada motor possue um encoder, que gera 1024 pulsos por revolução, e um fim de curso acoplado. A visão é efetuada por duas câmeras CCD Gradiente Color SC-60. Essa câmera possue resolução máxima de 640X480 pixels e captura 30 quadros por segundo.

O período de amostragem empiricamente escolhido para medição dos encoders, potenciômetros e acionamento dos motores foi de 100 ms. O período de amostragem empiricamente escolhido para acionamento dos sonares foi de 200 ms.

Todos os sensores e atuadores do robô, com execeção das câmeras, foram organizados em um barramento CAN. O protocolo CAN (Controler Area Network) (Bosch, 1991) foi desenvolvido nos laboratórios da multinacional Bosch para ser utilizado na indústria automobilística, porém ganhou popularidade em outras áreas. Esse protocolo é bastante adequado para sistemas que utilizam muitos sensores e atuadores e possuem restrições temporais. O CAN pode funcionar com múltiplos mestres ou em modo mestre-escravo. Esse protocolo pode atingir velocidades de 1 Mbit/s em distâncias de até 30 metros. A distância máxima permitida de um nó a um mestre é de 5 kilômetros, onde a velocidade máxima alcançada é de 10 Kbits/s.

Devido ao baixo período de amostragem e ao curto período de tempo necessário para acionar sensores e atuadores, a existência de paralelismo real na execução dos processos do robô se torna imprescindível. Sendo assim, a arquitetura de hardware do Kapeck ( Figura 2 ) foi desenvolvida de forma a prover processamento distribuído, através dos 10 processadores existentes no robô. Essa arquitetura é modular, o que significa que a inclusão de novos módulos de hardware, bem com a de módulos de software, é possível e de fácil execução.

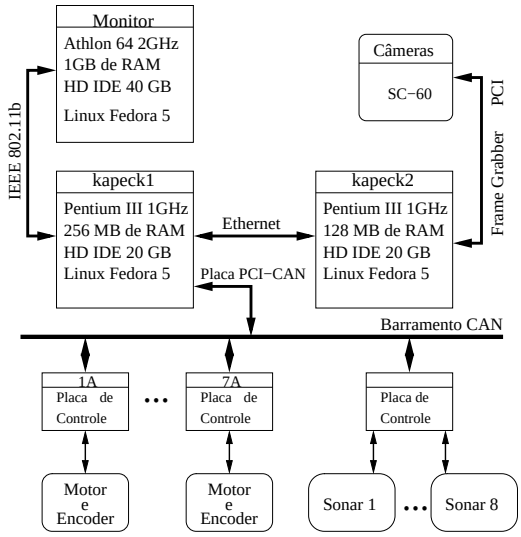


Figura 2: Arquitetura de Hardware

A arquitetura de hardware é composta por um computador comum, dois computadores embarcados e oito placas microcontroladas. Os computadores embarcados e as placas microcontroladas localizam-se no robô. Eles concentram funções distintas, fornecendo um ambiente propício à execução em paralelo dos módulos de software.

O computador `monitor` (Figura 2) localiza-se fora do robô, comunicando-se com esse através de uma rede wireless do tipo IEEE 802.11b. Ele serve para monitorar e interagir com o robô, contendo o módulo de interação com o usuário.

O computador `kapeck1` (Figura 2) abriga todo o software responsável pela deliberação da arquitetura de controle, assim como o controle de alto nível dos sensores e atuadores. Ele é responsável por fazer a interface com o `monitor`, passando a ele informações do sistema, recebendo e executando possíveis comandos.

O computador `kapeck2` (Figura 2) abriga o sistema de visão (módulo `visionSis`). O sistema de visão foi colocado em um computador a parte devido a sua grande exigência computacional. Esse computador é conectado ao `kapeck1` através de uma rede ethernet tcp/ip ponto-a-ponto (cabo crossover).

As placas microcontroladas (Figura 2) são reponsáveis pelo controle de baixo nível dos sensores e atuadores da arquitetura. Dessa forma, existe um ganho de desempenho do sistema devido ao processamento dedicado dessas placas, liberando em parte os computadores embarcados para ou-

tras tarefas. Essas placas são conectadas ao computador `kapeck1` por meio de uma rede CAN funcionando em modo mestre-escravo. Isso significa que as placas só utilizam o barramento se ordenadas pelo mestre da rede, contido no `kapeck1`. Todas as placas microcontroladas possuem o microprocessador de 8 bits PIC 18F258. Este microprocessador foi escolhido devido ao seu suporte ao protocolo CAN, além de dispor de gerador de sinais PWM (Modulação em Largura de Pulso), temporizadores e portas de entrada e saída digitais, necessários às aplicações. Dois tipos de placas foram desenvolvidas, chamadas de placa de controle de motor e placa de controle de sonar.

A placa de controle de motor é responsável por gerar o sinal PWM para um determinado motor através de um algoritmo de controle PID (Proporcional Integral Diferencial), recebendo do controlador de alto nível as referências de posição ou velocidade. Esta placa também possuirá a função de adquirir as medições de encoder. Caso a placa seja acoplada a algum motor da cabeça estéreo, também receberá o sinal de uma chave de fim de curso. Os dados medidos são retornados para o módulo de percepção, localizado no `kapeck1`, apenas quando esse módulo os requisita assincronamente. As medições de encoder também são utilizadas localmente pelo algoritmo de controle PID.

A placa de controle de sonar é responsável pelo funcionamento dos sonares do Kapeck. Ela coleta assincronamente informações dos sonares quando o módulo de percepção requisita tais informações. A placa retorna para o módulo de percepção uma medida da distância ao obstáculo mais próximo.

## 3 Arquitetura de Software

De modo a aproveitar o hardware distribuído do robô Kapeck, a organização de controle desenvolvida busca paralelizar ao máximo os módulos de software. A interação entre esses módulos é feita utilizando áreas de memória compartilhada chamadas de *blackboards* (Hayes-Roth, 1985). O acesso aos sensores e atuadores localizados no barramento CAN do robô é feito por meio de uma classe `C++` chamada `procCAN`.

### *3.1 Blackboards*

Os *blackboards* são implementados através de três classes `C++` chamadas de `bBoard`, `bBoardAtt`, `bBoardRe` e do módulo de software `servBBoard`.

A classe `bBoard` é utilizada para instanciar objetos que criam e destroem áreas de memória compartilhada e gerenciam seu acesso em exclusão mútua via semáforos (Tanenbaum, 2003). Ela é utilizada pelos módulos de inicialização do sistema, responsáveis por iniciar todos os outros módulos de software e criar todos os *blackboards* e semáforos a serem utilizados para troca de informações dentro do sistema. Todo *blackboard*, quando é criado, recebe uma chave de identificação (ID) única.

A classe `bBoardAtt` é utilizada para se acoplar a *blackboards* já criados. Fornece as primitivas de leitura e escrita no *blackboard*. Ao se instanciar um objeto dessa classe passa-se como parâmetro o ID do *blackboard* ao qual se deseja acoplar.

A classe `bBoardRe` tem as mesmas funções e funciona de forma semelhante à classe `bBoardAtt`, porém se acoplando a um *blackboard* que foi criado em outra máquina. Isso é feito através de uma conexão por *socket tcp* com o `servBBoard` do outro computador. Ao se instanciar um objeto dessa classe, passam-se como parâmetros o IP do outro computador e a ID do *blackboard* ao qual se deseja acoplar. A utilização de *blackboards* locais ou remotos é transparente para o programa cliente, pois ambas as classes `bBoardAtt` e `bBoardRe` oferecem as mesmas primitivas de leitura e escrita.

O `servBBoard` é um módulo de software implementado em ambos os computadores embarcados. Ele efetua operações de leitura e escrita em *blackboards* locais em nome de clientes localizados em computadores remotos. Uma instância da classe `bBoardAtt` comunica com o `servBBoard` do computador onde reside o *blackboard* a ela associado.

Um exemplo de arquitetura que utiliza *blackboard* pode ser visto na arquitetura COHBRA (Heinen, 2002). Na arquitetura COHBRA é utilizado apenas um *blackboard* central onde os módulos de software compartilham informações, diferente da abordagem distribuída apresentada neste artigo.

### *3.2 procCAN*

Essa classe torna transparente aos módulos de software características do hardware do robô. A existência desta classe facilita alterações do hardware do robô, pois torna a implementação dos módulos de software desacoplada diretamente dos módulos de hardware, uma vez que os módulos de software operam sobre um barramento CAN lógico, que encapsula o barramento CAN real.

O protocolo CAN possue nativamente um mecanismo que controla o acesso simultâneo ao barramento, através do protocolo CSMA/CA (*Carrier Sense - Multiple Access with Collision Avoidance*) (Bosch, 1991). Porém, na organização de controle aqui apresentada, diferentemente da maioria das aplicações que utilizam barramento CAN, existe um nó da rede CAN (computador `kapeck1`) onde mais de um processo concorre pelo acesso ao barramento. Nesta situação, a existência de um mecanismo que garanta a exclusão mútua no

acesso ao barramento é necessária, visto que a inexistência de tal mecanismo pode ocasionar colisão de mensagens antes mesmo destas chegarem ao barramento CAN.

A exclusão mútua do barramento é garantida utilizando a rede CAN em modo mestre-escravo e um árbitro de barramento. Todos os métodos implementados pela classe `procCAN` internamente requisitam o acesso ao barramento CAN real. Por meio dos métodos da classe, os módulos que necessitarem enviar ou receber informações de algum sensor ou atuador do barramento devem mandar uma requisição ao árbitro. Se algum módulo já estiver utilizando o barramento, o módulo requisitante será bloqueado até que o árbitro conceda o acesso ao barramento para o mesmo. Uma vez que o acesso foi concedido, o módulo requisitante deve executar as ações desejadas e em seguida sinalizar ao árbitro que o barramento está livre para ser utilizado por outros módulos.

Um exemplo de arquitetura que utiliza o barramento CAN é a arquitetura presente no trabalho de Coronel (Coronel et al., 2005). Nesse trabalho é utilizado um protocolo de comunicação de alto nível, a cima do protocolo CAN, chamado de SCo-CAN (*Shared Channel on CAN*) para controlar o acesso ao barramento.

## 4 Organização de Controle

A organização de controle desenvolvida para o robô Kapeck (Figura 3) baseia-se no paradigma híbrido deliberativo-reativo (Murphy, 2000) devido a necessidade de aliar o comportamento reativo da rede de sensores-atuadores com as atividades deliberativas necessárias para realizar tarefas mais complexas. Desta forma, a organização de controle é composta pelos módulos de software `Perception`, `Localizer`, `Cartographer`, `actionPl`, `pathPl`, `refRoGen`, `avoiderObs`, `posCon`, `headCon`, `refHeGen` e `visionSis`.

O módulo `Perception` possue a função de requisitar às placas microcontroladas leituras dos encoders e dos sonares em uma taxa de amostragem fixa. Esse módulo escreve em *blackboards* os valores recebidos, de forma a permitir aos outros módulos de software utilizarem esses valores. Esse módulo escreve em três *blackboards*: um para os deslocamentos angulares das rodas, outro para os deslocamentos angulares das partes da cabeça estéreo e um terceiro para as medições dos sonares.

O módulo `Localizer` lê os dados dos *blackboards* onde o módulo `Perception` escreve, além dos dados gerados pelo `visionSis`, através de um *blackboard* remoto. Esse módulo calcula a posição do robô através de um processo de fusão sensorial, utilizando a posição calculada com os dados dos encoders, os dados retornados pelos sonares e os dados retornados pelo `visionSis`. A melhor estimativa da posição do robô é exportada em outro *blackboard*. Devido à maior velocidade de aquisição das informações dos encoders, a estimativa é atualizada mais freqüentemente utilizando essas informações. Tal posição é posteriormente ajustada fazendo uso das informações de sonares e do `visionSis`, de mais lenta aquisição, porém mais precisas.

O `Cartographer` mapea o ambiente utilizando os dados exportados pelo módulo `Perception` nos *blackboards*. Esse módulo pode também conter um mapa conhecido *a priori*. O mapa gerado pelo `Cartographer` ou conhecido *a priori* é exportado em um *blackboard*. No exemplo apresentado neste artigo, o mapa é topológico, implementado na forma de um grafo.

O `actionPl` lê os dados dos *blackboards* onde os módulos `Localizer` e `Cartographer` exportam dados. Com esses dados, e conhecendo a tarefa global do robô, ele determina a ação a ser executada.

No exemplo apresentado neste trabalho, a tarefa global do robô é explorar um prédio com vários ambientes até encontrar uma marca conhecida. Desta forma, existem quatro ações possíveis:

1. Ir para centro de ambiente não-percorrido;
2. Vasculhar ambiente não-percorrido com cabeça estéreo;
3. Ir para a marca quando ela for encontrada;
4. Ir para um outro ambiente ainda não-percorrido.

O mapa topológico do prédio é conhecido *a priori*. A exploração é feita através de uma busca em profundidade no grafo que representa a topologia de conexões dos ambientes no prédio.

No caso da ação 1, uma ação corresponde a ir ao centro do ambiente onde o robô se encontra. A ação 2 corresponde a executar movimentos pré-determinados com a cabeça estéreo. Estes movimentos são executados quando o robô atinge o centro do ambiente. Tais ações possuem como objetivo fornecer uma melhor posição para o `visionSis`, que funciona em paralelo, capturar e processar imagens.

A ação 3 corresponde a ir para posição da marca encontrada, fornecida pelo `visionSis`. Quando esta ação é disparada, significa que o robô atingiu seu objetivo global, que é encontrar a marca, e a exploração do prédio é finalizada.

No caso da ação 4, uma ação corresponde a ir para um ambiente ainda não-percorrido pelo robô. O caminho que vai do ambiente atual até o próximo ambiente não-percorrido é gerado sobre o grafo resultante da busca em profundidade aplicada ao mapa do prédio. O caminho no grafo é determinado pelo algoritmo de Dijkstra (Cormen et al., 2002). Como o mapa é conhecido *a priori* e tem dimensões reduzidas, todos os caminhos são

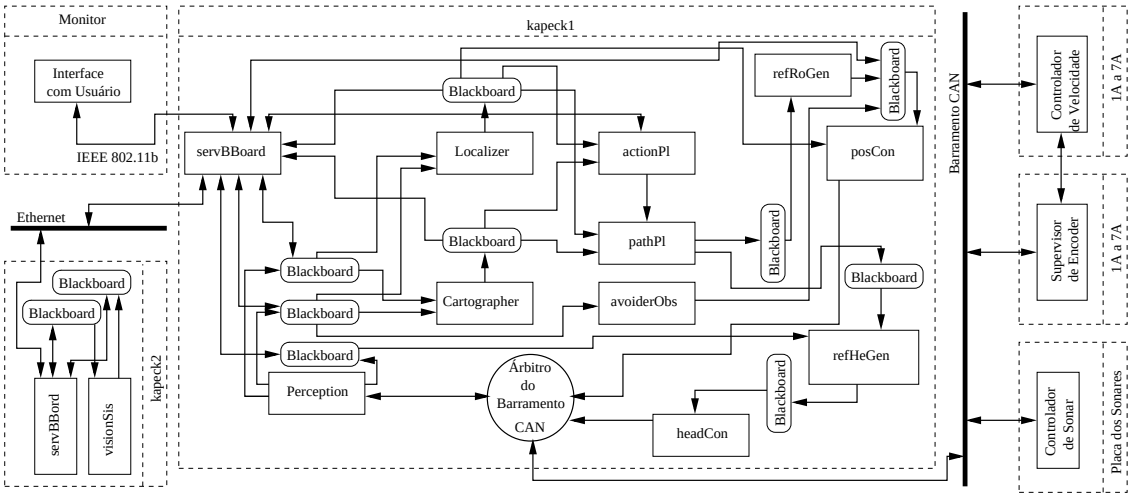


Figura 3: Organização de Controle

pré-calculados na inicialização do robô. Assim, o `actionPl` é responsável pela execução da busca em profundidade e pela escolha da ação.

As ações são passadas síncronamente ao módulo `pathPl`. Caso o `actionPl` dispare uma ação do tipo 1, o `pathPl` exporta a posição do centro do ambiente onde o robô se encontra. No caso do disparo de uma ação do tipo 2, o `pathPl` exporta posições a serem atingidas pelas partes da cabeça estéreo. No caso de uma ação do tipo 3, o `pathPl` exporta a posição da marca encontrada. No caso de uma ação do tipo 4, é gerado um caminho geométrico que leva ao ambiente final desejado, passando pelos ambientes intermediários que integram o caminho no grafo. Dois *blackboards* são utilizados pelo `pathPl`: Um para exportar os dados necessários para levar o robô de um local a outro; um segundo para exportar os dados necessários para movimentar a cabeça estéreo.

O `refRoGen` lê os dados exportados pelo `pathPl` e gera posições de referência a cada período de amostragem, exportando essas posições em um *blackboard*.

O `posCon` é um módulo reativo que possue a função de gerar os percentuais de velocidade para as rodas do robô, por meio de um controlador PID, de modo a levar o robô para uma posição lida do *blackboard* no qual o `refRoGen` exporta dados. Esses percentuais são enviados via CAN para as placas de controle de motor das rodas, fornecendo a referência para o controle embarcado nessas placas. Esse módulo lê também os dados exportados pelo `Localizer`, necessários ao controlador PID.

O `avoiderObs` lê os dados gerados pelo módulo `Perception` e monitora esses dados de forma a identificar obstáculos no caminho do robô. Uma vez que um obstáculo é encontrado, uma posição segura é gerada. Esta posição é exportada no mesmo *blackboard* onde o `refRoGen` exporta dados.

O `headCon` funciona de forma semelhante ao `posCon`, porém ele lê suas referências a partir do *blackboard* onde o módulo `refHeGen` escreve.

O `refHeGen` gera posições aleatórias as quais são possíveis de serem alcançadas pelas partes da cabeça estéreo do robô e as exporta em um *blackboard* que é lido `headCon`. Porém, quando o `actionPl` dispara uma ação do tipo 2, o `refHeGen` deixa de funcionar de forma aleatória, lendo as posições gerados pelo `pathPl` até o término da ação e exportando as posições lidas. Após o término da ação, o `refHeGen` volta a gerar posições aleatórias.

O `visionSis` possue a função de processar as imagens capturadas pelas câmeras em busca da posição do robô e, no exemplo deste trabalho, de uma marca existente no mapa. Quando uma marca é encontrada, o `visionSis` sinaliza para o `actionPl`, que dispara uma ação do tipo 2. Juntamente com a sinalização, a posição da marca é exportada em um *blackboard*. A posição do robô estimada é exportada em outro *blackboard* distinto.

## 5 Resultados Experimentais

O exemplo desenvolvido para demonstrar a arquitetura aqui proposta consiste em um robô que explora um ambiente conhecido *a priori* em busca de uma marca. Uma vez que a marca é encontrada, o robô se dirige a ela. Neste exemplo foram utilizados os seguintes módulos de software da organização de controle: `Perception`, `Localizer`, `Cartographer`, `actionPl`, `pathPl`, `refRoGen`, `posCon`, `headCon`, `refHeGen` e `visionSis`.

Na Figura 4 pode ser visto o resultado do experimento. Nos momentos T0, T3, T6 e T10 o `actionPl` dispara uma ação do tipo 1. Nos momentos T1, T4, T7 e T11 o `actionPl` dispara uma ação do tipo 2. Nos momentos T2, T5, T8 e T12 o `actionPl` dispara uma ação do tipo 4. Porém, no momento T13 o `visionSis` identifica a marca, fazendo com que o `actionPl` dispare uma ação do tipo 3. Neste momento, o robô despreza o resto

do caminho gerado no momento T12 e dirige-se à marca, utilizando as coordenadas fornecidas pelo `visionSis`. No momento T14, o robô alcança a posição da marca, completando sua tarefa global.

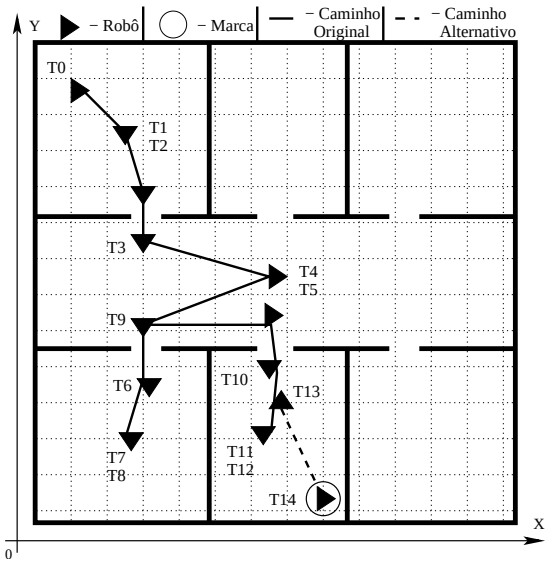


Figura 4: Resultado do Experimento

## 6 Conclusões

Foram apresentadas neste trabalho as idéias principais de uma arquitetura de hardware e software para um robô móvel autônomo. Em particular, nós apresentamos o sistema de interação entre módulos de software via *blackboards*, bem como o acesso ao barramento CAN por meio de uma representação lógica do mesmo. A arquitetura aqui proposta busca apresentar não apenas aspectos da organização de software, mas também a organização de hardware, além de apresentar a ligação entre eles e indicar como eles devem ser utilizados de modo a se obter um melhor desempenho do sistema robótico. Os conceitos aqui apresentados foram aplicados no robô Kapeck. Pretende-se nos próximos trabalhos o aperfeiçoamento dos módulos de software da organização de controle bem como o desenvolvimento de módulos de hardware e software para os manipuladores do robô. Na atual fase do trabalho, utiliza-se um mecanismo muito simples para evitar colisões no acesso ao barramento CAN. Uma evolução do trabalho será a utilização de um protocolo de comunicação em cima do protocolo CAN, otimizando o uso dos recursos nativos oferecidos por esta tecnologia.

## Referências


Adouane, L. and Le Fort-Piat, N. (2004). Hybrid behavioral control architecture for the cooperation of minimalist mobile robots, *Proceedings of ICRA'04*, New Orleans,USA, pp. 3735–3740.

Alami, R., Chantila, R. and Espiau, B. (1993). Designing an inteligent control archtecture for autonomous mobile robots, *Proceedings of ICAR'93*, Tokyo, Japão, pp. 435–440.

Bosch, R. (1991). Bosch can specification, *Technical report*, Bosch Gmbh.

Cormen, T. H., Leiserson, C. E., Stein, C. and Rivest, R. L. (2002). *Algoritmos: Teoria e Prática*, 1 edn, Campus.

Coronel, J. O., Benet, G., Simó, J. E., Pérez, P. and Albero, M. (2005). Can-based control architecture using de scocan communiation protocol, pp. 381–384.

Hayes-Roth, B. (1985). A blackboard architecture for control, *Artificial Inteligence 26*.

Heinen, F. J. (2002). *Sistema de controle híbrido para robôs móveis autônomos*, Master's thesis, Centro de Ciências Exatas e Tecnológicas, UNISINOS, São Leopoldo, RS.

Kim, J.-O., Im, C.-J., Shin, H.-J., Yi, K. Y. and Lee, H. G. (2003). A new task-based control architecture for personal robots, *Proceedings of IROS'03*, pp. 1481–1486.

Ly, D. N., Asfour, T. and Dillmann, R. (2004). A modular and embedded control architecture for humanoid robots, *Proceedings of IROS'04*, Sendai, Japão, pp. 2775–2780.

Medeiros, A. A. D. (1998). A survey of control architectures for autonomous mobile robots, *Journal of Brazilian Computer Society* .

Murphy, R. R. (2000). *Introduction to AI robotics*, 2 edn, MIT Press, Cambridge, Massachusetts 02142.

Roisenberg, M., Barreto, J. M., Silva, F. d. A., Vieira, R. C. and Coelho, D. K. (2004). Pyramidnet: A modular and hierarchical neural network architecture for behavior-based robotics, *Proceedings of ISRA'04*, Querétero, México, pp. 32–37.

Tanenbaum, A. S. (2003). *Sistemas Operacionais*, 2 edn, Prentice-Hall.