Desenvolvimento de um protótipo de veículo autônomo terrestre controlado por agente BDI*

Vitor Luis Babireski Furio[1], **Maiquel de Brito**[1]

[1]Departamento de Engenharia de Controle, Automação e Computação
Universidade Federal do Santa Catarina (UFSC)
Blumenau – SC – Brazil

vitorluis.babireskifurio@gmail.com, maiquel.b@ufsc.br

Abstract. *This paper describes the implementation of an autonomous ground vehicle controlled by a BDI agent. This approach provides to the vehicle features such as autonomy, social skills, proactivity, and reactivity. Besides implementing the behaviour of the vehicle in an agent-oriented programming language, this paper describes the integration between hardware and software.*

Resumo. *Este artigo descreve a implementação de um veículo autônomo terrestre controlado por um agente BDI. Esta abordagem confere ao veículo características como autonomia, capacidade de colaborar com outros veículos, proatividade e reatividade. Além de implementar o comportamento do veículo em uma linguagem de programação orientada a agentes, descreve-se aspectos da integração entre* hardware *e* software*.*

1. Introdução

Veículos autônomos podem ser aplicados não apenas no tráfego urbano em substituição aos automóveis guiados por humanos, mas também em cenários como a Indústria 4.0, por exemplo. Em diversas aplicações, é desejável que estes veículos tenham autonomia, capacidade de colaborar com outros veículos, bem como sejam capazes de conciliar comportamento proativo e reativo. Estas também são características de agentes BDI que, assim, se mostram uma metáfora computacional adequada para este tipo de sistema. Este artigo descreve a implementação de um veículo autônomo terrestre controlado por um agente BDI. O foco, neste trabalho, está na implementação do comportamento de um veículo autônomo através de uma linguagem de programação orientada a agentes BDI, bem como na integração entre *hardware* e *software* em uma entidade única.

2. Propósito da aplicação

A aplicação desenvolvida tem o propósito de avaliar o funcionamento do *framework* `embedded-mas`[1] para o desenvolvimento de agentes integrados a dispositivos físicos. Este é um *framework* de propósito geral para o desenvolvimento deste tipo de agente. Ele fornece uma extensão a agentes padrão Jason [Bordini et al. 2007].

Agentes interagem com o ambiente através dos processos de percepção e ação. A extensão fornecida pelo *framework* acrescenta funcionalidades a estes dois processos. No

*This short paper describes a demonstration submitted to WESAAC.

[1]`http://github.com/embedded-mas/embedded-mas`

processo de percepção, os valores adquiridos pelos sensores são tratados como percepções do agente, influenciando na atualização de suas crenças e, consequentemente, em seu processo de tomada de decisão. Este é um processo passivo: o agente não atua (e nem é programado) para fazer leitura de sensores. Ao contrário, os mudanças no estado do ambiente percebidas pelos sensores são automaticamente convertidas em percepções.

No processo de ação, as ações habilitadas pelos atuadores são incorporadas ao repertório de ações do agente. Assim, do ponto de vista do agente (e de seu desenvolvimento), estas ações são ações internas. A manipulação de atuadores é tratada como parte do funcionamento do próprio agente e não como uma atuação sobre um elemento pertencente ao ambiente. Assim, a manipulação dos atuadores torna-se parte integrante do ciclo de raciocínio do agente.

3. Demonstração

O desenvolvimento da solução inclui decisões a respeito da integração entre hardware e software e da codificação do comportmento do veículo em uma linguagem de programação orientada a agentes. Estes aspectos são descritos a seguir.

3.1. Arquitetura da solução

O veículo é implementado através de um *kit de robótica*. Trata-se de um veículo de três rodas (Figura 1). Motores ligados a duas dessas rodas movimentam o veículo. Estes são os atuadores incorporados ao agente. Além disso, o veículo conta com um sensor ultrassônico para detectar a distância de obstáculos. Essa distância é continuamente tratada como percepção do agente, sendo então convertida em crenças.

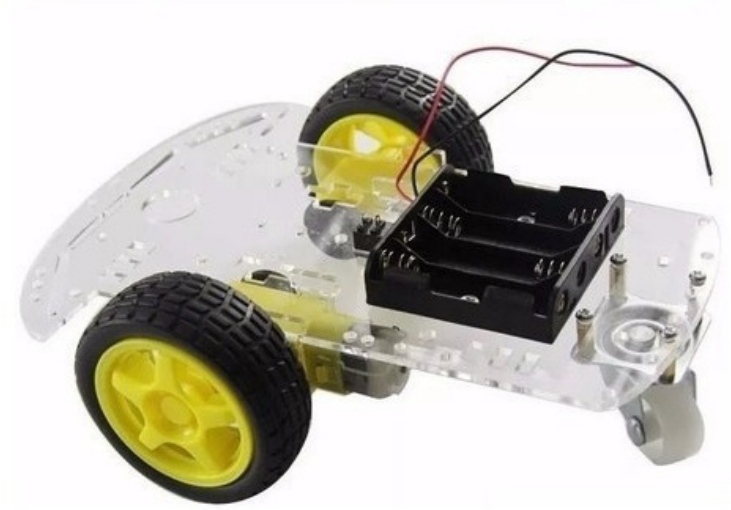

Figura 1. Kit de robótica usado para montar o veículo

Os motores e o sensor são ligados a portas digitais de uma placa *Arduino UNO*, responsável tanto por iniciar e parar o funcionamento dos motores quanto por coletar os valores lidos pelo sensor. O Arduino é ligado, através de conexão serial, a um computador de placa única *Raspberry Pi* em que o agente Jason é executado. O programa embarcado no Arduino continuamente (i) envia dados de sensores ao Raspberry e (ii) recebe instruções do agente executado no Raspberry a respeito da manipulação dos motores.

3.2. Comunicação serial

O correto funcionamento do sistema requer um tratamento apropriado dos dados enviados pelo Arduino, que efetivamente controla o *hardware* do sistema, para o agente executado no Raspberry Pi. Este tratamento inclui (i) a garantia de integridade das informações trocadas entre as duas partes e (ii) a representação adequada das informações para que sejam processadas de forma correta. Para isso, definiu-se um formato para as informações trocadas entre Arduino e Raspberry. A cada ciclo de execução, o programa embarcado no Arduino envia informações a respeito dos valores dos sensores. Esta informação é inserida em um pacote composto pelos seguintes campos: (i) preâmbulo, com 2 bytes, indicando o início de uma nova mensagem; (ii) indicador de tamanho de mensagem, que é um número inteiro de tamanho variável, usado pelo receptor para verificar a integridade da mensagem enviada; (iii) indicador de início de conteúdo, com 2 bytes, indicando que, na sequência, serão enviados os valores dos sensores; (iv) valores dos sensores, com tamanho variável; e (v) indicador de fim de mensagem, com 2 bytes, indicando o final da mensagem.

3.3. Integração de sensores ao sistema de percepção e atualização de crenças

O sistema de percepções do agente lê continuamente a porta serial e, seguindo o protocolo descrito, trata os bytes recebidos, transformando as informações em percepções que serão levadas em conta na atualização de crenças do agente. É importante notar que o agente (executado no Raspberry Pi) e o programa embarcado no Arduino são (sub)sistemas assíncronos. Assim, é possível que mudanças nos valores dos sensores sejam informadas ao sistema de percepções do agente mais rápido do que este possa processá-las. Isto requer algum tipo de tratamento. Algumas estratégias possíveis são: (i) selecionar o valor mais atual de cada sensor e descartar os demais, mantendo uma visão atualizada do ambiente sem preocupação com o histórico de mudanças; (ii) selecionar o valor mais antigo ainda não processado de cada sensor, permitindo que o agente considere a história das mudanças do ambiente, porém correndo o risco de ter uma visão desatualizada do mesmo; (iii) adotar uma abordagem baseada em amostragem, selecionando aleatoriamente valores entre aqueles ainda não processados de cada sensor. Estes são alguns exemplos entre outros possíveis. A escolha da estratégia deve levar em conta as características do cenário e os requisitos da aplicação.

3.4. Integração de atuadores ao repertório de ações do agente

No desenvolvimento desta aplicação, as ações habilitadas pelos atuadores fazem parte do repertório de ações do agente. Estas ações são incorporadas à própria implementação do agente e, quando invocadas, são executadas atomicamente como parte do seu ciclo de raciocínio. Para que isso seja possível, tais ações são tratadas como *internal actions* características do agentes Jason. No código ilustrado a Figura 2, as linhas 4 e 8 são exemplo destas ações.

Nesta aplicação, faz-se uma diferença entre *ação* e *atuação*. O primeiro conceito refere-se às ações especificadas no código do agente, tratando do resultado esperado a partir de sua execução. Por exemplo, na Figura 2 – linhas 4 e 8 – especifica-se que o agente vai executar ações que o levem para a frente e para trás. A *atuação*, por sua vez, refere-se à execução dos atuadores físicos que leva a termo a *ação* especificada no código do agente. Por exemplo, nesta aplicação, a ação de andar para a frente requer que os dois motores do veículo rodem em sentido horário. A ação de andar para trás, por sua

```
1. !move.                                  // desejo inicial de mover-se
2.
3. +!move : distanciaFrente(X) & X<10      // se há um obstáculo próximo
4.    <-  tras;                            // executa a ação de mover-se para trás
5.        !move.                           // continua a mover-se
6.
7. +!move : distanciaFrente(X) & X>10      // se não há um obstáculo próximo
8.    <-  frente;                          // executa a ação de mover-se para a frente
9.        !move.                           // continua a mover-se
```

Figura 2. Fragmento de código do agente

vez, requer que os dois motores rodem em sentido anti-horário. A implementação destas atuações é ilustrada, com algumas simplificações, na Figura 3. A aplicação aqui descrita contém outras ações e atuações. Por exemplo, a ação de virar à direita requer comportamentos diferentes dos motores. Estes detalhes podem ser verificados na implementação completa da solução.[2] É importante destacar que um mesmo agente (com um mesmo conjunto de ações) pode ter um diferente conjunto de atuações correspondentes em diferentes *hardwares*. Por exemplo, um veículo com quatro rodas e motores possivelmente terá diferentes acionamentos de motores para as mesmas ações do agente.

```
1.   public boolean frente() {     // implementação de movimento à frente
2.      doMotor1Horario();         // motor 1 deve rodar em sentido horário
3.      doMotor2Horario();         // motor 2 deve rodar em sentido horário
4.   return true;}
5.
6.   public boolean tras() {       // implementação de movimento para trás
7.      doMotor1Antihorario();     // motor 1 deve rodar em sentido anti-horário
8.      doMotor2Antihorario();     // motor 2 deve rodar em sentido anti-horário
9       return true; }
```

Figura 3. Fragmento de código de atuações do agente

4. Conclusão

Com o desenvolvimento da aplicação aqui descrita, tem-se um veículo autônomo controlado por agente BDI em que *hardware* e *software* são integrados em uma entidade única. Os valores dos sensores são incorporados às percepções do agente enquanto as ações habilitadas pelos atuadores são incorporadas ao repertório de ações do agente. Esta versão da aplicação explora esta integração bem como a utilização de programação orientada a agentes para implementar o comportamento do veículo. Em trabalhos futuros, pretende-se explorar aplicações multiagentes usando esta abordagem, de forma que múltiplos veículos desenvolvidos da forma aqui descrita possam atuar de forma coordenada em cenários mais complexos.

Agradecimentos. Vitor Luis Babireski Furio é Bolsista do CNPq - Brasil (PIBIC – Propesq/UFSC).

Referências

Bordini, R. H., Hübner, J. F., and Wooldrige, M. (2007). *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley Series in Agent Technology. John Wiley & Sons.

[2]Disponível em `http://github.com/embedded-mas/embedded-mas/tree/master/examples/CarExample`