YAM (Yet Another Mouse) – Um Robot Virtual com Planeamento de Caminho a Longo Prazo

Pedro Ribeiro[1]

1 - Finalista da Licenciatura em Ciência de Computadores da F.C.U.P

***Resumo* – Este artigo tem uma descrição resumida do funcionamento interno do robot virtual YAM (Yet Another Mouse), que ganhou a edição 2002 do Concurso Ciber-Rato. O robot caracteriza-se por ter uma forte componente de planeamento a longo prazo, visto que usa a informação do sensor de farol para descobrir a posição global da saída do labirinto. Desse modo pode procurar o melhor caminho para a saída tendo em conta as zonas já visitadas, evitando caminhos "bons" localmente, mas "maus" na globalidade.**

***Abstract* – This article has a brief description of the internal functioning of the virtual robot YAM (Yet Another Mouse), which won the 2002 edition of the "Ciber-Rato" contest. The robot is caracterized by having a strong component of long term planning, since it uses the info given by the beacon sensor to discover the global position of the exit of the maze. In that way it can search the best path to the exit having in account the visited zones, avoiding paths that are locally "good", but globally "bad".**

I. INTRODUÇÃO

O concurso Ciber-Rato consiste basicamente na simulação de um ambiente virtual (um labirinto), no qual agentes virtuais completamente autónomos (os "ratos") competem concorrentemente por um objectivo (chegar à saída ou "queijo"). À partida os agentes nada sabem sobre a disposição do labirinto, e toda a informação que recebem é dada por sensores (obstáculos, farol, colisão, chão e bússola). Para actuarem, os robots dispõem de dois motores e de um sinalizador de final de prova.

O YAM foi desenvolvido no âmbito do estágio da Licenciatura em Ciência de Computadores da Faculdade de Ciências da Universidade do Porto e foi decidido usar a linguagem de programação C (usando o compilador GCC no Linux) para a implementação do robot virtual, por ser esta a linguagem que o autor melhor conhece, e de modo a usar a API de programação fornecida pela organização do concurso, que já tem implementadas primitivas suficientes para efectuar toda a comunicação com o simulador do Ciber-Rato. Seria muito difícil detectar erros no programa só com informação textual e por isso foi criado um visualizador gráfico (o "YAM-viewer") usando a XLib, que permite representar de uma forma gráfica todos os processos internos do YAM, que muito ajudou na detecção de erros no programa e na confirmação das ideias teóricas sobre o comportamento do robot.

II. ARQUITECTURA DO YAM

A simulação processa-se em tempo discreto, e não de um modo contínuo, onde cada ciclo de simulação dura aproximadamente 80ms (no caso da edição deste ano). No início de cada ciclo são enviadas os dados dos sensores e depois o agente tem até ao final desse mesmo ciclo para decidir quais as acções a tomar, pois as posições dos robots só são actualizadas no final do ciclo. Desse modo, e como 80ms provaram ser tempo suficiente para os processos internos de decisão do YAM, foi decidido usar uma arquitectura tradicional, com uma única thread de execução, onde o fluxo de dados se processa como indicado na fig. 1.

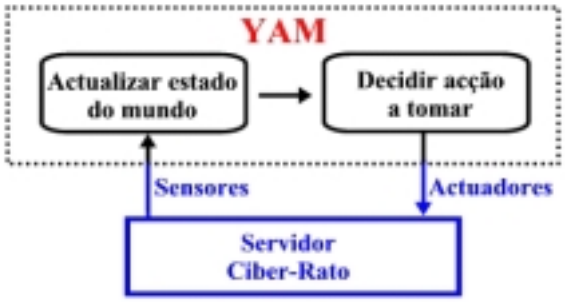

Fig. 1 - Arquitectura do YAM

III. ESTADO INTERNO DO MUNDO

De modo a usar de forma eficiente a informação recebida pelos sensores, foi decidido construir uma representação interna do ambiente exterior, que consiste basicamente em três componentes:

- Estado actual do agente (posição e orientação)
- Mapa de paredes do labirinto
- Mapa da saída (farol)

Inicialmente é considerado que o robot está na posição (0,0), ou seja, a origem das coordenadas é o ponto de partida do YAM. De seguida apresenta-se uma descrição resumida do funcionamento de cada uma das componentes.

A. Estado actual do agente

A orientação do robot é obtida directamente em cada ciclo usando a bússola, que embora tenha ruído, fornece

sempre uma boa aproximação do valor real. Tendo em conta a orientação e as potências dadas ao motor no ciclo anterior, é também actualizada a posição do robot no mundo, supondo que não existe nenhum ruído. Se considerarmos (em relação ao ciclo *n*) que *X[n]* é a coordenada do eixo horizontal do robot ,*Y[n]* a coordenada vertical, *D[n]* a potência dada ao motor direito, *E[n]* a potência dada ao motor esquerdo e $\theta[n]$ a sua orientação, então:

$$Lin[n] = (D[n] + E[n]) / 2.0$$
$$X[n+1] = X[n] + \cos(\theta[n]).Lin[n]$$
$$Y[n+1] = X[n] + \text{sen}(\theta[n]).Lin[n]$$

Foi verificado experimentalmente que o ruído não era suficiente para provocar desvios importantes na posição real do robot.

B. Mapa de paredes do labirinto

O labirinto tem a dimensão máxima de 17,5x17,5, e para facilitar a sua análise foi tornado discreto, sendo representado por uma matriz de 175x175. Como a referência das coordenadas é o ponto de partida do robot, e este pode estar em qualquer posição do labirinto, os mapas do labirinto têm dimensão 350x350 (o quádruplo da dimensão de um labirinto), de modo a permitir prever o movimento do robot em qualquer direcção. Em relação às paredes, é guardado um mapa com a probabilidade (positiva ou negativa) de um determinado ponto do mapa ter um obstáculo/parede. Inicialmente, todos os pontos são neutros (têm probabilidade zero). Depois, em cada ciclo, os pontos ocupados pelo rato são marcados com probabilidade mínima (se o robot está nessa posição, de certeza que não existem obstáculos). A fig. 2 ilustra a utilização que o YAM faz dos valores recebidos pelos sensores. Se o valor recebido por um sensor indica *V*,

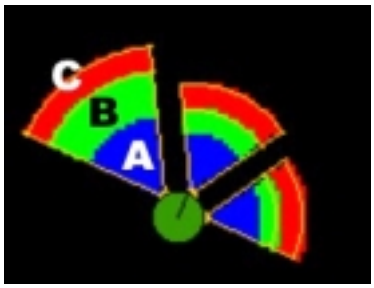

Fig. 2 – Utilização dos sensores de obstáculos (YAM-viewer)

então é porque existe um obstáculo sensivelmente à distância de *1/V*, algures no alcance do sensor. No entanto o valor *V* vem com um ruído máximo de *noise_obstacle*, logo, no pior dos casos (em termos de proximidade) o obstáculo está a *1/(V+noise_obstacle)*. Sendo assim, toda a zona mais próxima do que essa distância no alcance do sensor é marcada com probabilidade mínima de ter parede (corresponde à zona marcada com a letra A). Depois, a zona seguinte (zona B), até à distância *1/V* é reduzida a sua probabilidade de ter parede (é subtraído ao valor armazenado um valor proporcional à distância ao sensor). Sabe-se que uma parede tem uma espessura mínima de *min_wall_width*, logo a zona desde *1/V* até *(1/V)+min_wall_width* (zona C) vê aumentada a sua probabilidade de ter parede (é somado um valor proporcional à distância ao sensor). No entanto, por exemplo, um obstáculo pode estar situado apenas nos limites da detecção do sensor, e poderia erradamente marcar todos os 60 graus de abertura como tendo parede. Para compensar esse facto, a zona C aumenta 4 vezes menos a probabilidade de ter parede em relação ao decréscimo provocado pela zona B. Desse modo, é fácil "desmarcar" paredes falsas (provocadas pela situação referida anteriormente ou por dados gerados pela proximidade de outro robot), mas é mais complicado uma zona ficar definida como sendo parede. Depois de algumas pequenas afinações, este sistema revelou-se muito fiável, criando desenhos quase perfeitos das zonas percorridas pelo robot, como se pode ver no exemplo dado na fig. 3.

Fig. 3 – Exemplo de mapa de paredes. A azul estão as zonas com pouca probabilidade de ter parede e a amarelo as que têm forte probabilidade de serem paredes. O percurso do robot está marcado a verde . (YAM-viewer)

C. Mapa da saída

Do mesmo modo que para as paredes, existe uma matriz que guarda a probabilidade de cada ponto ser a saída. Em cada ciclo, o valor recebido pelo sensor de farol, *a(farol)*,

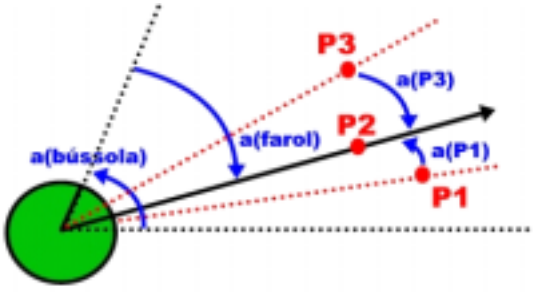

Fig. 4 – Utilização do sensor de farol

juntamente com o valor da bússola, *a(bússola)*, permite actualizar os valores da matriz. Seja *a(P)* o ângulo relativo a um ponto, tal como está exemplificado na fig. 4. Se *a(P)*=0, como acontece com o ponto P2, o ponto é marcado com grande probabilidade de ter farol. À medida que *a(P)* vai aumentando, os pontos vão sendo marcados proporcionalmente cada vez com menos probabilidade. Desse modo, ao fim de alguns ciclos, a saída real vai ficando cada vez com mais probabilidade, pois o seu valor *a* será sempre pequeno. Veja-se a título de exemplo o que acontece na fig. 5, onde claramente o YAM já descobriu onde fica a saída, na zona marcada com branco.

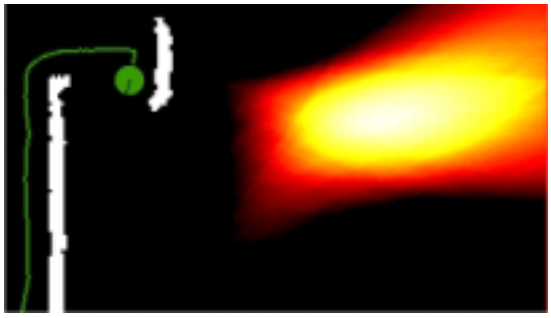

Fig. 5 –Mapa da saída. As zonas mais claras têm mais probabilidade de ter a saída. (YAM-viewer)

D. Pré-Cálculos

Como todas as tarefas atrás mencionadas demoram algum tempo, foram pré-calculadas, para todos os ângulos possíveis, as alterações a fazer nos mapas, criando uma espécie de "máscaras". Quando é recebido um valor, as máscaras são colocadas por cima da posição actual do robot, e os valores são actualizados em consonância.

IV. DETERMINAÇÃO DA ACÇÃO A TOMAR

Tendo em conta o mapa da saída, o YAM escolhe o ponto com maior probabilidade (com coordenadas *Sx* e *Sy*) como o objectivo a atingir. Todos os pontos do mapa dos obstáculos com um valor superior a um dado parâmetro são considerados parede, e todos os outros pontos são considerados como não tendo parede. Depois, o robot procura descobrir o melhor caminho para a saída, efectuando uma espécie de pesquisa em largura a partir da *P[Sx][Sy]*, tendo por base o seguinte algoritmo (onde *Ax* e *Ay* são as coordenadas da posição actual do robot e *V(P)* os pontos vizinhos de distância *Dist* do ponto P, que não são parede, explicados na fig. 6):

```
INICIALIZAR lista L com P[Sx][Sy]
Paux = RETIRA_PRIMEIRO(L)
ENQUANTO distância(Paux, P[Ax][Ay]) >= Dist:
    ADICIONAR V(Paux) a L
    Paux = RETIRA_PRIMEIRO(L)
```

Depois, se a cada ponto acrescentado a L for associado o ponto que o fez entrar para a lista (o "pai" do ponto), basta

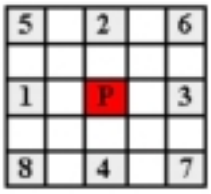

Fig. 6 – Vizinhança de distância 2 de um ponto P . Os vizinhos são visitados na ordem descrita, desde que não estejam marcados como sendo parede.

seguir os pontos pela ordem inversa para descobrir o caminho até à saída. Depois é criado um farol artificial no ponto visível (não "escondido" por paredes) do caminho descoberto mais distante da posição actual do YAM, e o rato segue esse farol de uma forma reactiva, segundo um algoritmo muito semelhante ao robot exemplo fornecido pela organização do concurso, apenas modificado para aumentar a velocidade de ponta do rato e fornecer-lhe a capacidade de andar paralelamente às paredes. Como é óbvio, quanto mais pequeno for o valor do parâmetro *Dist*, mais tempo demora este algoritmo, e por isso mesmo no concurso foi usado com o valor três (determinado experimentalmente), que revelou uma boa relação qualidade/tempo. Todo este processo atrás descrito é exemplificado na fig. 7.

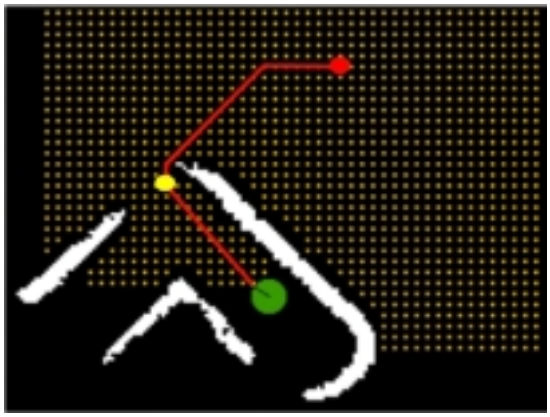

Fig. 7 – Caminho a seguir pelo robot. O círculo vermelho é a saída, a amarelo está o farol artificial, e os pontos que foram explorados são marcados a laranja. A branco estão marcadas as paredes. (YAM-viewer)

V. CONCLUSÃO

O comportamento do YAM nos testes realizados foi extremamente regular. Obviamente que o factor sorte (a posição de partida, o ruído aleatório, o comportamento dos outros robots, etc.) também conta, mas o YAM mostrou ser um robot equilibrado, ou seja, não dependente do tipo de labirinto para ter uma boa prestação. No concurso real, o Ciber-Rato 2002, esse facto foi comprovado, tendo o YAM vencido quase todas as provas em que participou (excepto na 2ª manga), incluindo a final, o que o tornou no novo campeão do Ciber-Rato. De notar que os labirintos usados na competição eram muito "simples" (com excepção do da final), o que fez com que a diferença do YAM para robots mais "primitivos" se tornasse mais pequena.