

Relatório do Laboratório 1:

Máquina de Estados Finita e Behavior Tree

Isabelle Ferreira de Oliveira
CT-213 - Engenharia da Computação 2020
Instituto Tecnológico de Aeronáutica (ITA)
São José dos Campos, Brasil
isabelle.ferreira3000@gmail.com

Resumo—Esse relatório documenta a implementação do comportamento de um robô Roomba, um robô de limpeza desenvolvido pela empresa iRobot. O comportamento foi idealizado e simplificado em um simulador, cujo código base, com parte da implementação, já havia sido fornecido pelo professor. Para o restante da codificação, implementou-se máquina de estados finita e behavior tree. Por fim, os resultados das duas implementações foram comparados.

Index Terms—Roomba, máquina de estados, behavior tree

I. INTRODUÇÃO

Comportamentos autônomos de agentes podem ser implementados por diversos métodos atualmente. Entre esses métodos, pode-se organizar os possíveis comportamentos em Máquinas de Estados e Árvores de Comportamento (Behavior Trees).

Máquina de Estados é um modelo matemático para descrever um sistema. Nesse modelo, apenas um estado está sendo executado a cada momento e, para transitar entre um estado e outro, são necessários acontecimentos em específicos [1]. É possível representar o comportamento de um sistema descrito por Máquina de Estados por meio de um grafo direcionado, como o apresentado na Figura 1.

Behavior Tree é outro modelo matemático também com a finalidade de descrever um sistema. Já nesse modelo, as folhas da árvore guardam os comportamentos mais básicos e os nós mais internos podem definir se os comportamentos básicos acontecerão em sequência, em ordem randômica, em paralelo, etc., acrescentando grau de complexidade aos comportamentos do agente. Após a execução de cada nó, é retornado Success (quando tarefa terminou com sucesso), Failure (quando tarefa falhou) ou Running (quando a tarefa deve continuar execução na próxima iteração) [1]. A representação de uma Behavior Tree pode ser vista na Figura 2.

É possível modelar o comportamento de um robô de limpeza autônomo do tipo Roomba (desenvolvido pela empresa iRobot) a partir desses métodos descritos anteriormente, conforme foi descrito nesse relatório.

II. IMPLEMENTAÇÃO DE ESTADOS

Na parte relativa a implementação da máquina de estados, era necessário preencher os códigos das funções *check_transition* e *execute*, além do construtor das classes

MoveForwardState, *MoveInSpiralState*, *GoBackState* e *RotateState*.

Para calcular quanto tempo já está sendo executado um determinado estado, foi adicionado nos respectivos construtores um contador que era acrescido cada vez que era executado o comportamento desse estado. Assim, o tempo seria esse contador vezes o tempo gasto em cada execução, conforme sugerido no roteiro do laboratório [1].

As transições em cada estado foram implementadas nas respectivas funções *check_transition* de cada classe de estado, conforme o apresentado na Figura 1, retirado do roteiro do laboratório [1].

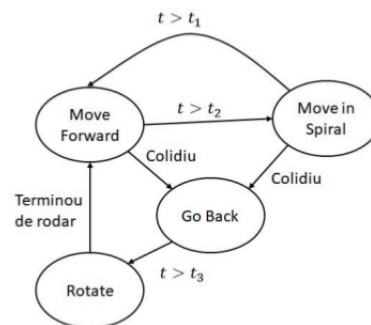


Figura 1. Máquina de estados finita do comportamento do Roomba.

Já as funções *execute*, além de realizarem o acréscimo do contador de execuções, também eram responsáveis por atualizar as velocidades do Roomba, a fim de que ele realizasse os movimentos esperados para cada estado. A ideia de cada uma das implementações também foi apresentada nas subseções abaixo, escritas em pseudo-Python.

Uma breve descrição em alto nível das implementações foi apresentada nas subseções a seguir, com implementações escritas em pseudo-Python.

A. Estado Move Forward

Verificação das transições da máquina de estados na função *check_transition* para o estado *Move Forward*:

```
if COLIDIU COM A PAREDE:
    MUDAR PARA ESTADO "GO BACK"
elif TEMPO_NESSE_ESTADO >
    TEMPO_NO_MOVE_FORWARD:
```

MUDAR PARA ESTADO "MOVE IN SPIRAL"

Atualização das velocidades do Roomba na função *execute* para o estado *Move Forward*:

```
ROOMBA.set_velocity(VELOCIDADE_LINEAR =  
    FORWARD_SPEED, VELOCIDADE_ANGULAR = 0)
```

B. Estado Move In Spiral

Verificação das transições da máquina de estados na função *check_transition* para o estado *Move in Spiral*:

```
if COLIDIU COM A PAREDE:  
    MUDAR PARA ESTADO "GO BACK"  
elif TEMPO_NESSE_ESTADO >  
    TEMPO_NO_MOVE_IN_SPIRAL:  
    MUDAR PARA ESTADO "MOVE FORWARD"
```

Atualização das velocidades do Roomba na função *execute* para o estado *Move In Spiral*:

```
ROOMBA.set_velocity(VELOCIDADE_LINEAR =  
    FORWARD_SPEED, VELOCIDADE_ANGULAR =  
    FORWARD_SPEED / (INITIAL_RADIUS_SPIRAL +  
    SPIRAL_FACTOR * TEMPO)
```

O cálculo dessa velocidade angular foi feito tendo em vista que o raio da espiral varia conforme a equação $r(t) = r_0 + b \cdot t$, onde r_0 é o raio inicial do espiral e b é o fator do espiral, e que velocidade angular é velocidade linear dividido pelo raio da curva no instante.

C. Estado Go Back

Verificação das transições da máquina de estados na função *check_transition* para o estado *Go Back*:

```
if TEMPO_NESSE_ESTADO > TEMPO_NO_GO_BACK:  
    MUDAR PARA ESTADO "ROTATE"
```

Atualização das velocidades do Roomba na função *execute* para o estado *Go Back*:

```
ROOMBA.set_velocity(VELOCIDADE_LINEAR =  
    BACKWARD_SPEED, VELOCIDADE_ANGULAR = 0)
```

D. Estado Rotate

Verificação das transições da máquina de estados na função *check_transition* para o estado *Rotate*:

```
if TEMPO_NESSE_ESTADO > TEMPO_NO_ROTATE:  
    MUDAR PARA ESTADO "MOVE FORWARD"
```

O tempo que o Roomba passa no estado *Rotate* é calculado da seguinte maneira: um ângulo aleatório é escolhido entre $-\pi$ e π para que ele faça a rotação e, dado a velocidade angular pré-estabelecida empregada nesse movimento, o tempo será o módulo desse ângulo dividido por essa velocidade.

Atualização das velocidades do Roomba na função *execute* para o estado *Go Back*:

```
if ANGULO_EH_NEGATIVO:  
    ROOMBA.set_velocity(VELOCIDADE_LINEAR = 0,  
        VELOCIDADE_ANGULAR = -ANGULAR_SPEED)  
else:  
    ROOMBA.set_velocity(VELOCIDADE_LINEAR = 0,  
        VELOCIDADE_ANGULAR = ANGULAR_SPEED)
```

III. IMPLEMENTAÇÃO DE BEHAVIORS

Já na parte relativa a implementação da Behavior Tree, era necessário preencher os códigos das funções *enter* e *execute*, além do construtor das classes *RoombaBehaviorTree*, *MoveForwardNode*, *MoveInSpiralNode*, *GoBackNode* e *RotateNode*.

Para calcular quanto tempo já está sendo executado um determinado estado, foi feito de forma análoga ao feito na máquina de estados. Assim, foi adicionado nos construtores das classes folhas o contador que era acrescido cada vez que era executado o comportamento desse estado. Assim, o tempo também seria esse contador vezes o tempo gasto em cada execução.

Ao entrar em um Behavior, ou seja, na execução da função *enter*, eram setadas as velocidades que permaneceriam constantes durante aquele behavior, além de serem zerados os contadores. No caso específico do behavior *Move In Spiral*, como sua velocidade era alterada em cada instante, essa atualização da velocidade foi feita na função *execute*.

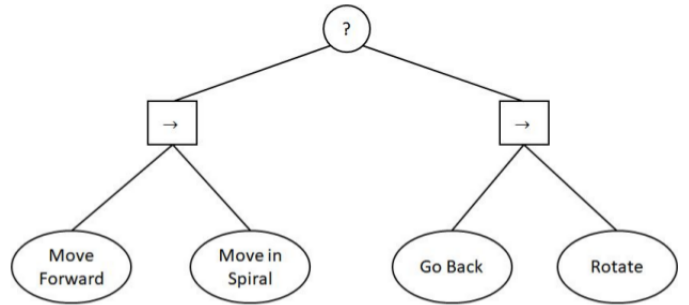


Figura 2. Behavior tree do comportamento do Roomba.

Já as funções *execute*, além de realizarem o acréscimo do contador de execuções, também retornavam *Success*, *Failure* ou *Running*, dependendo da situação que se encontrava o Roomba.

Uma breve descrição em alto nível das implementações foi apresentada nas subseções a seguir, com implementações escritas em pseudo-Python.

A. Roomba Behavior Tree

A Behavior Tree precisou ser inicializada no construtor da classe *RoombaBehaviorTree*. Sua implementação aconteceu fazendo de raiz o nó do tipo Selector, e adicionando como filhos da raiz dois nós do tipo Sequence. Por fim, foram adicionados nós folhas de cada behavior, conforme apresentado na Figura 2. Dessa forma, a ideia de implementação, apresentada em pseudo-Python, foi apresentada abaixo.

```

raiz = SelectorNode("root")

sequence_node_left = SequenceNode("left")
sequence_node_left.add_child(MoveForwardNode())
sequence_node_left.add_child(MoveInSpiralNode())

sequence_node_right = SequenceNode("right")
sequence_node_right.add_child(GoBackNode())
sequence_node_right.add_child(RotateNode())

raiz.add_child(sequence_node_left)
raiz.root.add_child(sequence_node_right)

```

B. Behavior Move Forward

Execução do behavior na função *execute* para o estado *Move Forward*:

```

if TEMPO > TEMPO_NO_MOVE_FORWARD:
    return SUCCESS
if COLIDIU COM A PAREDE:
    return FAILURE
else:
    return RUNNING

```

Atualização das velocidades do Roomba na função *enter* para o estado *Move Forward*:

```

ROOMBA.set_velocity(VELOCIDADE_LINEAR =
    FORWARD_SPEED, VELOCIDADE_ANGULAR = 0)

```

C. Behavior Move In Spiral

Execução do behavior na função *execute* para o estado *Move In Spiral*:

```

if TEMPO > TEMPO_NO_MOVE_IN_SPIRAL:
    return SUCCESS
if COLIDIU COM A PAREDE:
    return FAILURE
else:
    return RUNNING

```

Atualização das velocidades do Roomba na função *enter* para o estado *Move In Spiral*:

```

ROOMBA.set_velocity(VELOCIDADE_LINEAR =
    FORWARD_SPEED, VELOCIDADE_ANGULAR =
    FORWARD_SPEED / (INITIAL_RADIUS_SPIRAL +
    SPIRAL_FACTOR * TEMPO)

```

D. Behavior Go Back

Execução do behavior na função *execute* para o estado *Go Back*:

```

if TEMPO > TEMPO_NO_GO_BACK:
    return SUCCESS
else:
    return RUNNING

```

Atualização das velocidades do Roomba na função *enter* para o estado *Go Back*:

```

ROOMBA.set_velocity(VELOCIDADE_LINEAR =
    BACKWARD_SPEED, VELOCIDADE_ANGULAR = 0)

```

E. Behavior Rotate

Execução do behavior na função *execute* para o estado *Rotate*:

```

if TEMPO > TEMPO_NO_ROTATE:
    return SUCCESS
else:
    return RUNNING

```

O tempo que o Roomba passa no estado *Rotate* e o ângulo que é rotacionado foram calculados de forma análoga ao que foi feito na implementação por máquina de estados.

Atualização das velocidades do Roomba na função *enter* para o estado *Rotate*:

```

if ANGULO EH NEGATIVO:
    ROOMBA.set_velocity(VELOCIDADE_LINEAR = 0,
        VELOCIDADE_ANGULAR = -ANGULAR_SPEED)
else:
    ROOMBA.set_velocity(VELOCIDADE_LINEAR = 0,
        VELOCIDADE_ANGULAR = ANGULAR_SPEED)

```

IV. RESULTADOS E CONCLUSÕES

Os resultados obtidos após a implementação do comportamento do Roomba por meio de *Máquina de Estados* foram apresentados nas Figuras 3 e 4; e por meio de *Behavior Tree* nas Figuras 5 e 6.

Foi possível notar que o comportamento do Roomba pode ser independentemente implementados por *Máquina de Estados* e por *Behavior Tree*. Suas simulações são inclusive idênticas até certo ponto, como pode-se observar a partir da comparação entre as Figuras 3 e Figura 5.

Isso era esperado, uma vez que, até o instante da imagem, a simulação tinha poucos segundos de execução, somente os estados/nós *Move Forward* e *Move In Spiral* tinham sido executados e nesses comportamentos não há nenhum fator que modifique os resultados independente das quantidade de vezes executadas.

Já para as Figuras 4 e 6, o Roomba já tinha sofrido pelo menos 1 colisão com a parede. Nesse instante, o robô executa o comportamento *Go Back* e passa para o *Rotate*, no qual um ângulo aleatório entre $-\pi$ e π foi escolhido e rotacionado.

Observou-se conforme o esperado que não há mais garantia que os movimentos sejam idênticos, dada a aleatoriedade dos ângulos. Para o caso da Figura 4, por exemplo, foram necessárias algumas colisões com a parede até que o Roomba consiga andar tempo suficiente para entrar no comportamento *Move In Spiral*. Já no caso da Figura 6, apenas uma colisão foi necessária.

REFERÊNCIAS

- [1] M. Maximo, "Roteiro: Laboratório 1 - Máquina de Estados Finita e Behavior Tree". Instituto Tecnológico de Aeronáutica, Departamento de Computação. CT-213, 2019.

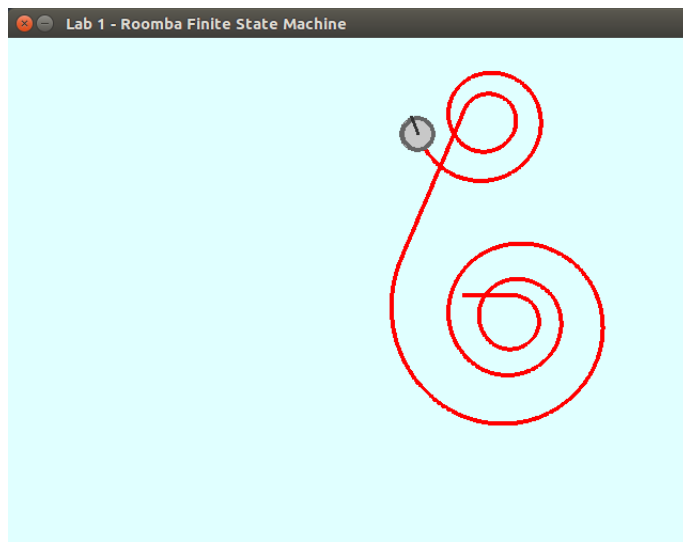


Figura 3. Simulação do Roomba para implementação por *Máquina de Estados*. Na figura, é possível notar os estados Move Forward e Move In Spiral.

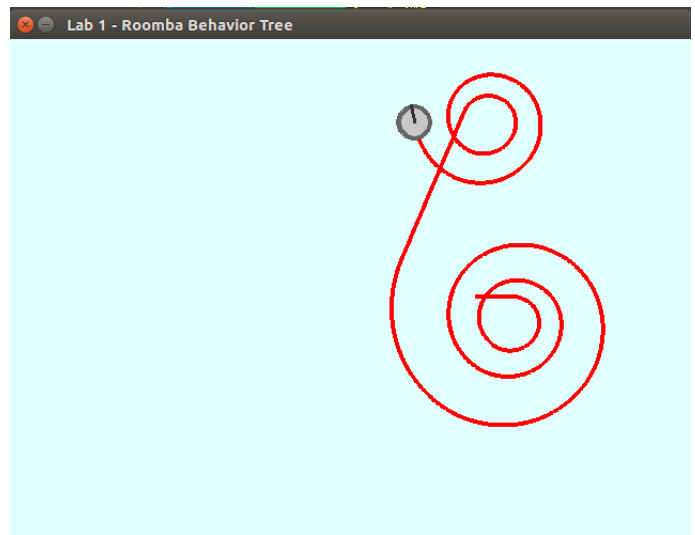


Figura 5. Simulação do Roomba para implementação por *Behavior tree*. Na figura, é possível notar os estados Move Forward e Move In Spiral.

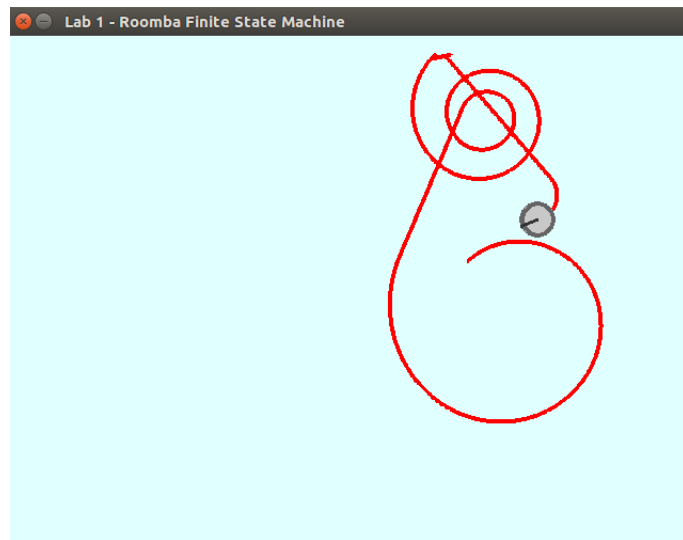


Figura 4. Simulação do Roomba para implementação por *Máquina de Estados* segundos após a Figura 3. Na figura, é possível notar os estados Move Forward, Move In Spiral, Go Back e Rotate.



Figura 6. Simulação do Roomba para implementação por *Behavior tree* segundos após a Figura 5. Na figura, é possível notar os estados Move Forward, Move In Spiral, Go Back e Rotate.