Análise do robô bombeiro para o desvio de mobília

Firefighter robot analysis for furniture diversion

Felipe Arisi
Escola da Tecnólogia e Gestão
Instituto Politécnico da Guarda - IPG
Guarda, Portugal
arisi@alunos.utfpr.edu.br

Resumo — **Projeto de uma simulação de um robô bombeiro respeitando as regras do *Polytechnic of Guarda Fire-Fighting Robot Contest.* O robô deve ser capaz de navegar pelo circuito com quarto quartos e atuar nos modos com mobilia e realizar a viagem de retorno.**

Palavras Chave – robô bombeiro, solução genérica, mobília.

Abstract — **Project of a simulation of a firefighter robot respecting the rules of the Polytechnic of Guard Fire-Fighting Robot Contest. The robot must be able to navigate the circuit with four rooms and operate in furnished modes and make the return trip.**

Keywords - firefighter robot, generic solution, furniture..

I. INTRODUÇÃO

O artigo remete-se ao desenvolvimento de uma simulação de um robô bombeiro atendendo as regradas do do *Polytechnic of Guarda Fire-Fighting Robot Contest.* O objetivo básico do robô é conseguir navegadar por quatro quartos diferentes e apagar uma chama na qual encontra-se dentro de um deles.

A prova também apresenta alguns modos de funcionamento diferentes, dentre eles pode-se citar:

Return Trip: O robô deve apagar a chama e rotorno ao local do ínicio.

Furtinure: Movéis são postos nos quartos com o objetivo de dificultar a visão e apagar a chama.

Para o desenvolvimento do robô, foi proposta ou solução genérica, ou seja, mesmo com a arena pré-definida o robô conseguiria se adaptar a circustâncias diferentes, todavia existe algumas particularidades que por esse metódo não foi possível realizar em uma totalidade.

Neste artigo será comentado a respeito do ambiente de simulação escolhido, o desenvolvimento do robô bombeiro e também alguns testes e resultados.

II. AMBIETNE DE SIMULAÇÃO

O ambiente de simulação escolhido foi o *Robot Basic* o qual foi desenvolvido pelo professor John Blankenship em conjunto com Samuel Mishal. É um simulador gratuito e poderoso no qual apresenta mais de 800 funções e comandos no qual podem ser utilizados para simulações de robôs, do mundo real, de jogos e de funções para engenharia.

Para o desenvolvimento do robô bombeiro, foram escolhidas algumas funções nas quais simulem o máximo o robô físico.

A. *Sensores:*

- rRange (Sensor Ultrassônico): Mede a distância entre o robô até um obstáculo. Usando como parâmetro o ângulo do robô onde o sensor estará posicionado. Como exemplo na Fig. 1 o robô detecta a distância de um objeto a sua frente.

```
1. rRange (0)
```

Figure 1. Exemplo de sensor ultrassônico.

- rBumber (Sensor de obstáculo): Caso o robô colida com algum objeto pode-se utilizar esse sensor para verificar a colisão. Seu uso é descrito na Fig. 2. Para o funcionamento dele é preciso verificar bit a bit.

```
1. bmp = rBumper()

2. if(bmp & 4
3.    //code
4. elseif(bmp & 8)
5.    // code
6. elseif(bmp & 2)
7.    // code
8. endif
```

Figure 2. Exemplo de funcionamento de um Bumper

- rSense (Sensor de linha): Caso o robô passe por uma linha é possível verificar com este sensor. Seu funcionamento é descrito na Fig. 3, onde é

chamada a função e passado como parâmetro a cor que ele deve verificar

```
1. if(rSense(WHITE) & 2)
```

Figure 3. Exemplo de funcionamento de um Sensor de linha

- rLook (Sensor de chama): Como o Simulador não apresenta uma chama propriamente dita, foi utilizado um objeto na cor amarela. E este sensor detecta se existe uma cor a frente. Exemplo de uso na fig. 4, ele percorre uma distância a sua escolha e é passado como parâmetro o ângulo, similar ao rRange.

```
1. if(rLook(0) == YELLOW)
```

Figure 4. Exemplo de funcionamento de um Sensor de chama.

B. *Atuadores:*

- rForward (movimentação): Utilizado para o robô se movimentar, basta passar o quando ele deve se movimentar. Caso o valor positivo ele se movimenta para frente e caso contrário ele vai em marcha ré. Exemplo na fig. 5.

```
1. rForward 1
```

Figure 5. Movimentar o robô para frente.

- rTurn (girar): Utilizado para o robô fazer curvar, passado como parâmetro o ângulo da curva. Valores positivos vira para a esquerda e negativos para a direta. Exemplo na fig. 6.

```
1. rTurn 90
```

Figure 6. Girar o robô em 90º.

III. SISTEMA DESENVOLVIDO

Para o desenvolvimento da lógica do robô, foi escolhida a arquitetura baseada em comportamentos implementada com uma máquina de estados finito.

Quatro estados foram necessários para o desenvolvimento: *wait*, *navigate*, *approach* e *putout*. As fases de transição entre os estados, estão descritas na fig. 7

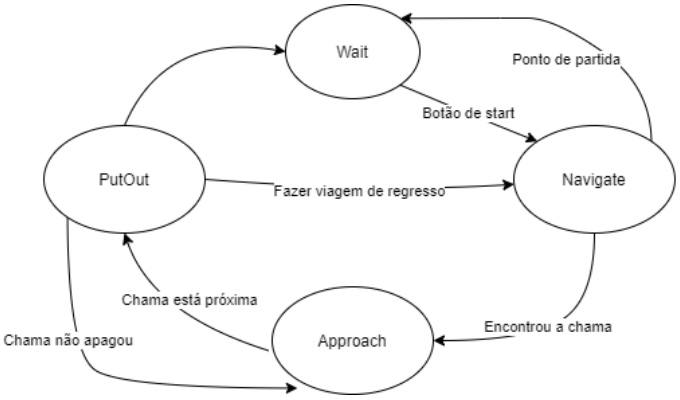

Figure 7. Maquina de estados finitos.

A. *WAIT:*

O robô deverá manter-se parado até que o usuário aperte o botão de *start*. Na fig. 8, ilustra como o robô deverá ficar.

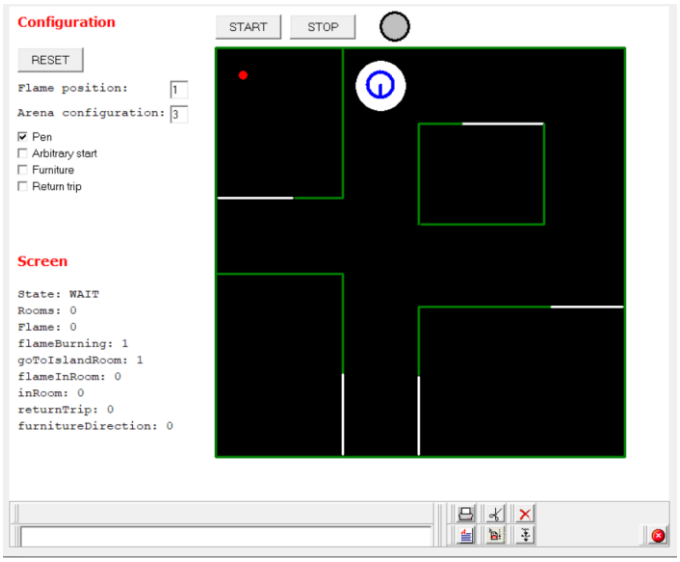

Figure 8. Posição do estado *WAIT*.

Nesse estado também ocorrem as funções *RESET* da simulação, onde permitira o robô começar novamente sua trajeto e também a manobra para ele sair do ciruclo branco inicial.

Na fig. 9 pode-se perceber o funcionamento do estado WAIT a nível de programação.

```
1. waitState:
2.     goSub setLedOFF
3.     enableButton "START", true
4.     repeat
5.       getButton btn
6.       if(btn == "RESET")
7.         goSub initializeVariables
8.         goSub drawArena
9.         goSub initializeRobot
10.         endif
11.       until btn == "START"
12.
13.       enableButton "START", false
14.       enableCheckBox "CBAStart", false
15.
16.       goSub maneuverToExitWhiteCircle
17.       state = NAVIGATE
18.     return
```

Figure 9. Função do estado WAIT.

B. NAVIGATE:

O estado mais importe entre todos, ou aquele que acomula mais funções. O robô aqui é resposável por se locomover seguindo a parade da direita, também precisa entrar nos quartos e localizar a chama.

Na fig. 10 mostra como o robô se deseloca pela arena.

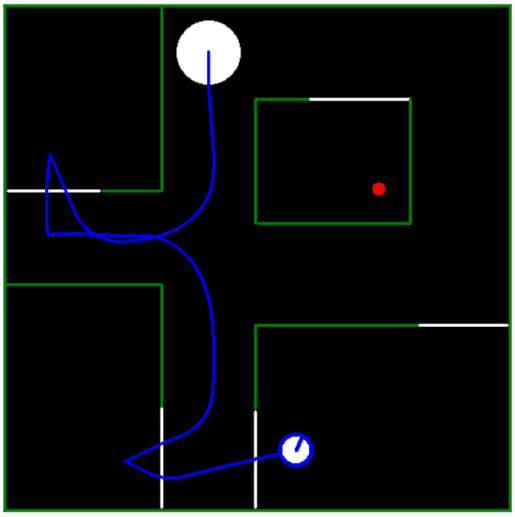

Figure 10. Locomoção do robô pela arena.

A primeira coisa que o robô faz neste estado é verificar se existe alguma parade na frente, caso tenha, ele gira 90º para a esquerda. Em seguida verifica se houve uma colisão nos bumpers.

O próximo passo, é fazer a navegação em si. Faz a comparação entre duas distancias e comparando a distancia da parade da direta ele aproxima ou afasta.

Também existe uma varáivel chamada *goToIslandRoom* que caso ele complete o percuso e não ache a chama ele vai para o quarto ilha. E para isso existe esta variavél que uma vez setada como -1, começa a seguir a parade da esquerda. Na fig. 11 a exemplificaçao desse código.

```
1.    if(rRange(0) < DIST_FRONT_LIMIT)
then rTurn -90
2.    goSub checkBumpers

4.    dist = rRange(90*goToIslandRoom)
5.    if(dist >= DIST1 and dist <=
DIST2)
6.       rForward 1
7.    elseif(dist < DIST1)
8.      rTurn -1*goToIslandRoom
9.      rForward 1
10.   elseif(dist > DIST2)
11.     rTurn 1*goToIslandRoom
12.     rForward 1
13.   endif
```

Figure 11. Função de navegação.

O restante da função é direcionada para o robô entrar no quarto e conseguir detectar a chama.

A primeira coisa que ele faz é detectar o tipo de chão. Caso for uma linha e ainda não tenha apagado a chama ele faz uma varredura. Caso for uma linha e ele já tenha apagado a chama ele só continua o caminho, se for um ciruclo e ele ainda não apagou, muda a forma de navegação para ir para o quarto ilha. Como podemos ver na fig. 12.

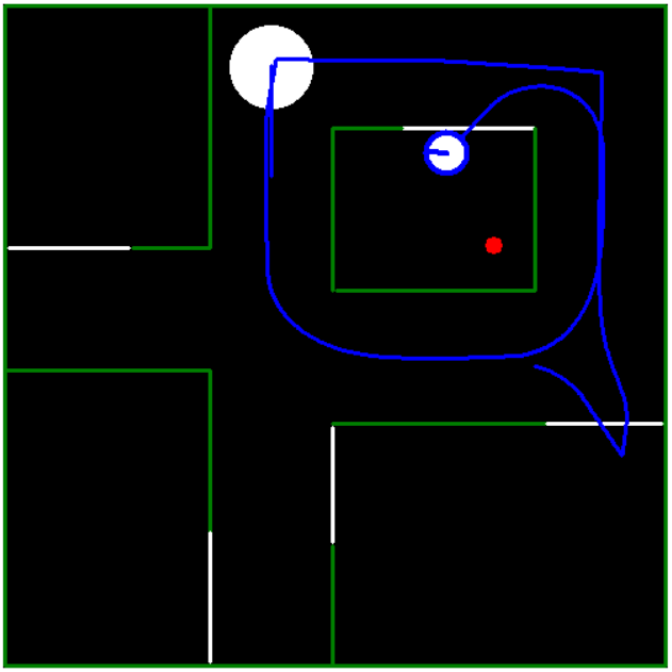

Figure 12. Robo realizando o caminho para o quarto ilha.

E a última função que ele realiza no estado navegação é realizar a manobra para a paragem.

Na fig. 13 ele mostra o código para estas funções. No quarto 2 ele faz uma manobra de virar o robô para que seja possível verificar todas as velas.

```
1. if(tag == LINE_TAG and inRoom ==
true)
2.         inRoom = false
3.     elseif(tag == LINE_TAG and
inRoom == false)
4.         inRoom = true
5.     endif
6.
7.     if(tag == LINE_TAG and
flameBurning == true)
8.       rooms = rooms + 1
9.       call moveForward(50)
10.       if(rooms == 2)
11.         rTurn 45
12.         goSub scanRoom
13.       else
14.         goSub scanRoom
15.       endif
16.       if(flameInRoom == true)
17.         goSub getFlameDirection
18.         if(flameDirection != 0)
19.           goSub setLedON
20.           state = APPROACH
21.           return
22.         endif
23.       else
24.          if(rooms == 2) then rTurn
-45
25.          goSub
maneuverToGoToNextRoom
26.       endif
27.     elseif(tag == LINE_TAG and
flameBurning == false and inRoom ==
true)
28.          goSub maneuverToGoToTheEnd
29.     elseif(tag == CIRCLE_TAG and
flameBurning == false)
30.         goSub maneuverToStop
31.         state = WAIT
32.     elseif(tag == CIRCLE_TAG and
flameBurning == true)
33.         goSub
meneuverToGoToIslandRoom
34.         goToIslandRoom = -1
35.     endif
```

Figure 13. Segunda parte do estado *Navigate*.

C. *APPROACH:*

Estado responasavel por o robô se aproximar da vela e conseguir estingui-la. Ele vai verificando a posição na qual a vela se encontra e ajustando o robô para a direção correta. Pode-se visualizar essa aproximação na fig. 14.

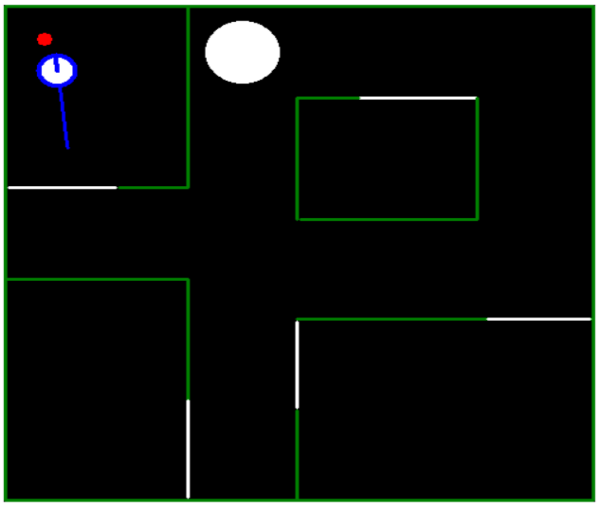

Figure 14. Robô realizando o modo de aproximação.

Este estado também é responsável por desviar dos móveis caso o modo seja escolhido.

Outra funcionalidade que este modulo apresenta, é que se ele perder a chama por algum momento, o robô faz um movimento especial dependendo o quarto que ele se encontra no momento.

Na fig. 15 tem o exemplo da codificação deste estado.

```
1.    approachState:
2.         goSub checkBumpers
3.         goSub getFurnitureDirection
4.         if(furnitureDirection != 0)
then goSub maneuverToDodgeTheFurniture
5.         goSub getFlameDirection
6.
7.         if(flameDirection == 0)
8.           goSub meneuverToScanRoom
9.           goSub scanRoom
10.        elseif(flameDirection == 1)
11.          rTurn 2
12.        elseif(flameDirection == 2)
13.          rTurn 1
14.        elseif(flameDirection == 3)
15.          if(rRange(0) < 15)
16.            state = PUT_OUT
17.            return
18.          endif
19.          rForward 1
20.        elseif(flameDirection == 4)
21.          rTurn -1
22.        elseif(flameDirection == 5)
23.          rTurn -2
24.        endif
25.     return
```

Figure 15. Código do estado Approach.

Como descrito anteriormente, este modulo também é responsável por desviar dos móveis. A solução proposta aqui é ir desviando conforme a distância das paredes laterais. Todavia, o resultado não foi muito positivo, conforme será abordado no próximo tópico.

A codificação para esta alternativa, está descrita na fig. 16.

```
1. maneuverToDodgeTheFurniture:
2.       if(rRange(90) < rRange(-90))
3.         rTurn -90
4.         call moveForward(20)
5.         rTurn 90
6.         call moveForward(1)
7.       else
8.         rTurn 90
9.         call moveForward(20)
10.         rTurn -90
11.         call moveForward(1)
12.       endif
13.   return
```

Figure 16. Código para desviar da mobilia.

D. PutOut:

O último estado remete-se a ação do robô conseguir apagar a vela. Como não existe uma simulação de uma vela em si, então foi realizado algo com um fator aleatório, ou seja, quando o robô estiver perto o suficiente ele tenta 'apagar' a vela e caso obtenha sucesso ele ou entra para o estado *wait* ou o *navigate*, caso não obtenha sucesso ele tenta apagar a vela novamente.

Na fig. 17, mostra como fica a simulação após apagar a vela.

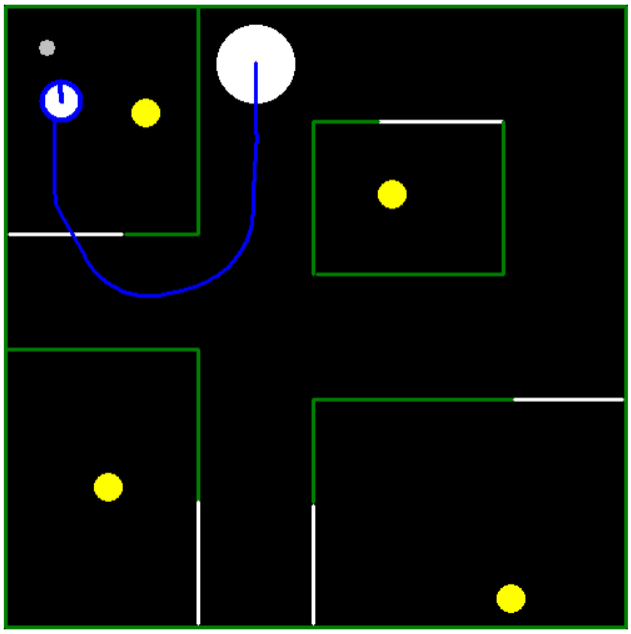

Figure 17. Simulação após o robô extinguir a vela.

Já a nível de codificação, pode-se verificar na fig. 18 o desenvolvimento deste estado.

```
1. putOutStateV2:
2.      x = random(99) + 1
3.      if(x < 75)
4.        circle candleBaseOX + candleX
- candleR, candleBaseOY + candleY -
candleR, candleBaseOX + candleX +
candleR, candleBaseOY + candleY +
candleR, GRAY, GRAY
5.      endif
6.
7.      rForward -10
8.      goSub getFlameDirection
9.      if(flameDirection != 0)
10.        goSub setLedON
11.        state = APPROACH
12.        return
13.      endif
14.      goSub setLedOFF
15.      flameBurning = false
16.      if(returnTrip == true)
17.        state = NAVIGATE
18.      else
19.        state = WAIT
20.      endif
21.   return
```

Figure 18. Codificação do estado putOut.

Como relato até agora, em poucos casos o robô faz uma manobra específica em detrimento ao quarto. Podemos notar isso também na Fig. 19 onde mostra funções de manobra.

```
maneuverToExitWhiteCircle:
   repeat
     rForward 1
     goSub getFloorTag
   until tag == NO_TAG
  Return

  maneuverToStop:
    call moveForward(50)
    rTurn 270
  Return

  maneuverToGoToNextRoom:
     rTurn 210
     call moveForward(30)
     inRoom = false
   Return

  meneuverToGoToIslandRoom:
    call moveForward(50)
    rTurn 270
    call moveForward(50)
  Return

  maneuverToGoToTheEnd:
     inRoom = false
     rForward -10
     rTurn 270
  return
```

Figure 19. Código para desviar da mobilia.

A única manora que utiliza a localização dos quartos é a para scanear o quarto. Como descrita na figura 20.

```
meneuverToScanRoom:
    if (rooms == 1)
      rTurn 90
      rForward 5
      rTurn -90
    elseif(rooms == 2)
      rForward 5
    elseif(rooms == 3)
      rTurn 10
      rForward 10
    elseif(rooms == 4)
      rTurn -90
      rForward 10
      rTurn 90
    elseif(rooms == 5)
      rForward 2
    endif
  return
```

Figure 20. Scam Room.

Provando então que a escolha da solução foi a mais genéria possível, ou seja, o robô tentar se adaptar a situação independente do local que ele se encontra.

IV. Testes e Resultados

Para o teste, foi realizada a seguinte metodologia: Um cronometro começava a contar a partir do momento que o robô saísse da base e parava quando retornava. É considerado como um sucesso quando o robô consegue retornar ao destino e conseguiu apagar a chama.

As falhas foram agrupadas conforme os problemas que o robô apresentou durante o teste.

Em questão de amostra, em cada tipo de modo testado foram realizados 24 itens diferentes, ou seja, três para cada posição que a vela pode se encontrar.

A. Furniture:

Os resultados apresentados no modo mobília, não foram satisfatórios. Como apresentado na Tabela 1.

TABLE I. Teste com mobília

	Resultados Mobília	
	Taxa de Sucesso	***Tempo médio***
Resultados	66,67%	47,45s

Dentre os problemas encontrados, pode-se verificar na Tabela 2.

TABLE II. Erros do modo com mobília

	Taxa de erro
Colidiu ao desviar da mobília	62,5 %
Não achou a chama	25%
Outros	12,5 %

Duas cois chamam a atenção nos resultados. A primeira que no quarto ilha o robô colidiu quase todas as vezes para desviar da mobília.

Outro ponto a ser analisado é que em alguns casos a vela se contrapoem ao robô e ele não consegue indintifica-la.

A. Modo Normal:

Neste caso foi considerado o robô conseguir apagar a chama e realizar a viagem de regresso.

Os resultados estão descritos na Tabela 3.

TABLE III. Erros do modo com mobília

	Resultados Sem Mobília	
	Taxa de Sucesso	***Tempo médio***
Resultados	91,66%	41,18s

Aqui os resultados foram muito melhores. Em apenas dois casos o robô colidiu ao realizar alguma manobra.

V. CONCLUSÕES

Dado em vista os resultados apresentados acima, o robô em uma operação sem mobília apresenta ótimos resultados e consegue ainda fazer uma viagem de regresso em um tempo relativamente bom.

Todavia, quando os móveis são adicionados, embora o tempo de regresso tenha aumentado mas não de uma maneira siginificativa, a taxa de sucesso ficou muito baixa se levarmos em conta que o robô precisaria estinguir um incêndio num mundo real.

Esse problema pode-se levar em considereção por dois fatores. O primeiro, o robô poderia levar em consideração o quarto no qual ele se encontrava para conseguir desviar do objeto, assim alterando um pouco a arquitetura proposta aqui.

A segunda opção, o robô ainda poderia se mover independente do quarto que se encontra, ou seja, a função *maneuverToDodgeTheFurniture* é desenvolvida de forma insuficiente.

Outra possibilidade para corrigir esse problema, é o robô tentar desviar do objeto no estado *navigate* e não no *approach*, assim não aconteceria o problema de não reconhcer a chama e também facilitaria a aproximação do robô.

REFERÊNCIAS BIBLIOGRÁFICA

[1] Blankenship, J. and Mishal, S. (2011), “Robot Programmer's Bonanza”, McGraw-Hill, ISBN 978-0-07-154797-0..

[2] Carreto, C. (2019). Apontamentos da unidade curricular de Robótica do curso de Engenharia Informática, Escola Superior de Tecnologia e Gestão, Instituto Politécnico da Guarda.