Trabalho Final: Tópicos Avançados de Computadores

1st Eduardo Ferreira Marques Cavalcante - 202006368
Departamento de ciência da computação
Universidade de Brasília
Brasília, Brasil
202006368@aluno.unb.br

2nd Maycon Vinnicius Silva Fábio - 200059742
Departamento de ciência da computação)
Universidade de Brasília
Brasília, Brasil
200059742@aluno.unb.br

3nd Mauro Fernandes de Almeida - 211020910
Departamento de ciência da computação)
Universidade de Brasília
Brasília, Brasil
211020910@aluno.unb.br

I. Introdução

Todos os arquivos de modelagem e implementação estão disponíveis em [2].

A Missão escolhida para apresentação do trabalho foi o Keeping Clean do ROBOMAX [1], em inglês, a missão declara o seguinte:

Every room that reaches more than 1 cfu/cm^2 of *Staphylococcus aureus* must be cleaned within the next 30 minutes. The assigned robot must check if it has all the resources to fulfill the tasks.
In the case of missing equipment, the robot must go to the storage room to collect the equipment or assign the go-to-storage task to a colleague.
As the robot reaches the room, it must check if it is occupied. If the room is occupied, a message should be sent to the manager and the mission aborted. Otherwise, the robot should enter the room and mark it as occupied.
The cleaning task must be performed in order:

1) Change the furniture's covers, towels, and clothes;
2) Vacuum the floor, moving furniture when necessary;
3) Wipe the floor;
4) Sterilize all furniture and equipment in the room.

In case of failure in performing any of the steps, it immediately warns the sector manager, but does not stop executing the mission.
All personal equipment (sensors, probes) found that do not originally pertain to the room must be removed from the room. If non-identified objects are found in the room, the robot should take a photo, report it to the manager, and if collectible, take it out of the room.
In case of a low battery, recharge it and come back to finish the task.

Em resumo, a missão se trata da limpeza de quartos caso o nível de Staphylococcus alcance um nível não desejado. O robô deve, então, buscar a sala, checar sua bateria, checar seus equipamentos, ir para a sala enquanto desvia de obstáculos, checar se o quarto está ocupado(aborta a missão caso isso ocorra) e limpar e esterilizar o quarto.

II. Fundamentação Teórica

Além da implementação das *Tasks* e sua devida sequência, precisamos recriar os 3 cenários de adaptação escolhidos:

1) Verifique nível de bateria:
   Verifique que o robô tem nível de bateria acima de 15% durante a execução da missão, o robô deve navegar para a estação de recarregamento se a bateria estiver baixa.

2) Estime o ttc (time to complete) da missão e recalcule o caminho da navegação.
   Dado um caminho para um objetivo, crie uma estimativa para o TTC dessa missão e escolha o caminho mais rápido.

Dica: Pense na navegação de uma missão como uma coleção de waypoints.

3) Use o sensor LIDAR para detectar obstáculos:
   O Nav2 já faz a verificação de obstáculos durante a navegação. Mas e se for necessário termos nossa própria verificação? Por exemplo, podemos prever se algum obstáculo irá interferir com a execução da missão. Use o LIDAR para verificar se um objeto está a uma distância específica do robô durante a navegação.

A. *Cenário do nível de bateria*

No cenário em que verificamos o nível de bateria do robô, é necessário conferir por meio do tópico $battery\_state$ essa informação e caso fique abaixo de 15% mandamos o robô encaixar na estação de carregamento, isso pode ser feito por meio de uma publicação no tópico *dock*.

B. *Cenário do sensor LIDAR*

O uso do sensor LIDAR não foi feito já que usamos o tópico *navigate_to_pose* para movimentar o robô para a determinada coordenada e o próprio já lidava com o desvio de obstáculos.

C. *Cenário de estimar o time to complete (ttc)*

Para estimar o *ttc*, foi necessário o uso do *feedback* retornado pelo tópico *navigate_to_pose*, onde nele conseguimos ter uma estimativa de quanto tempo levaria para o robô chegar no atual destino. Foi feita uma estimativa para onde o robô estaria indo no momento, não uma que estimasse o tempo total da missão.

III. Modelagem e Implementação

A. *Goal Model*

A goal model é o arquivo que faz a descrição em alto nível da missão em uma perspectiva global. Assim, descrevemos ela por meio do Pistar Tool seguindo os padrões disponibilizados em [4].

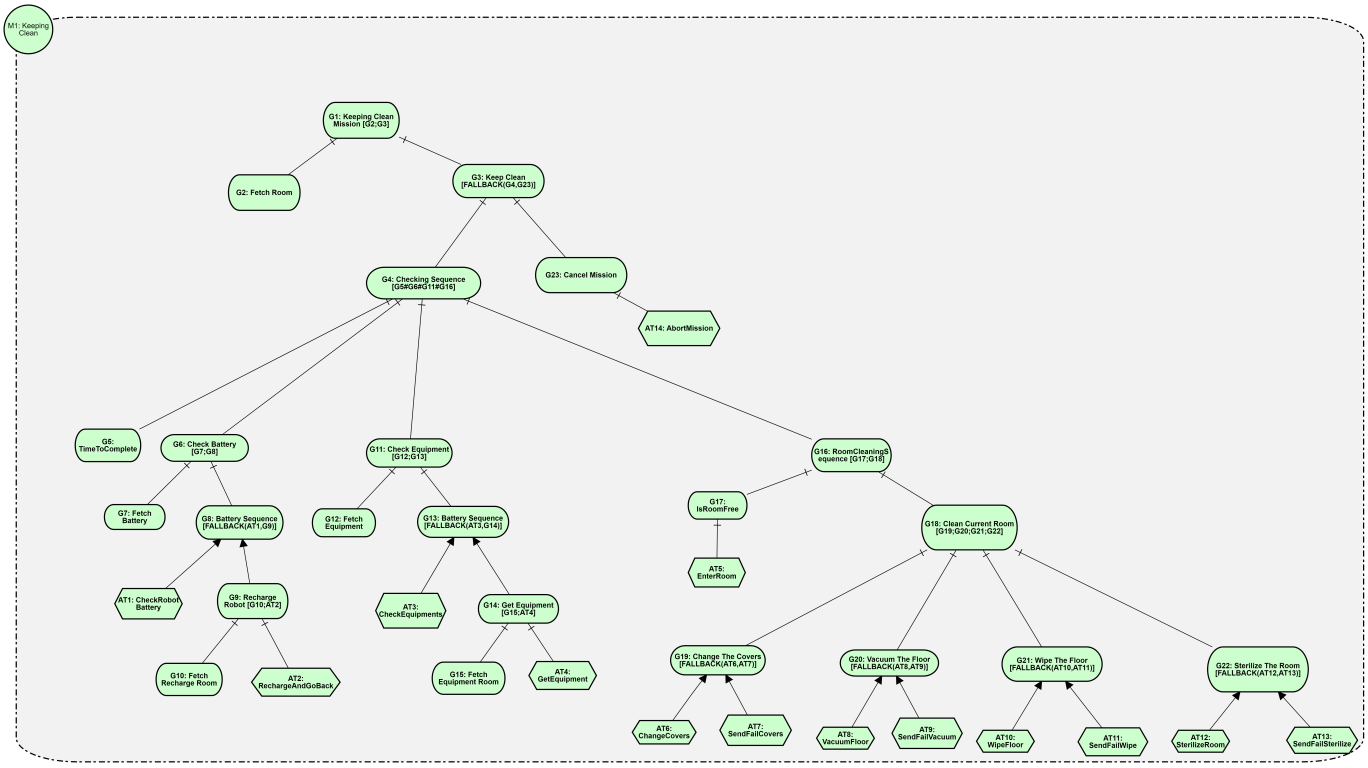

Figure 1. Goal Model final

A goal model(GM) serve como base para a Behavior Tree(BT) que será analisada posteriormente.

Nossa GM pode ser entendida em 4 partes, seguindo a ordem da esquerda para a direita de forma *depth first*:

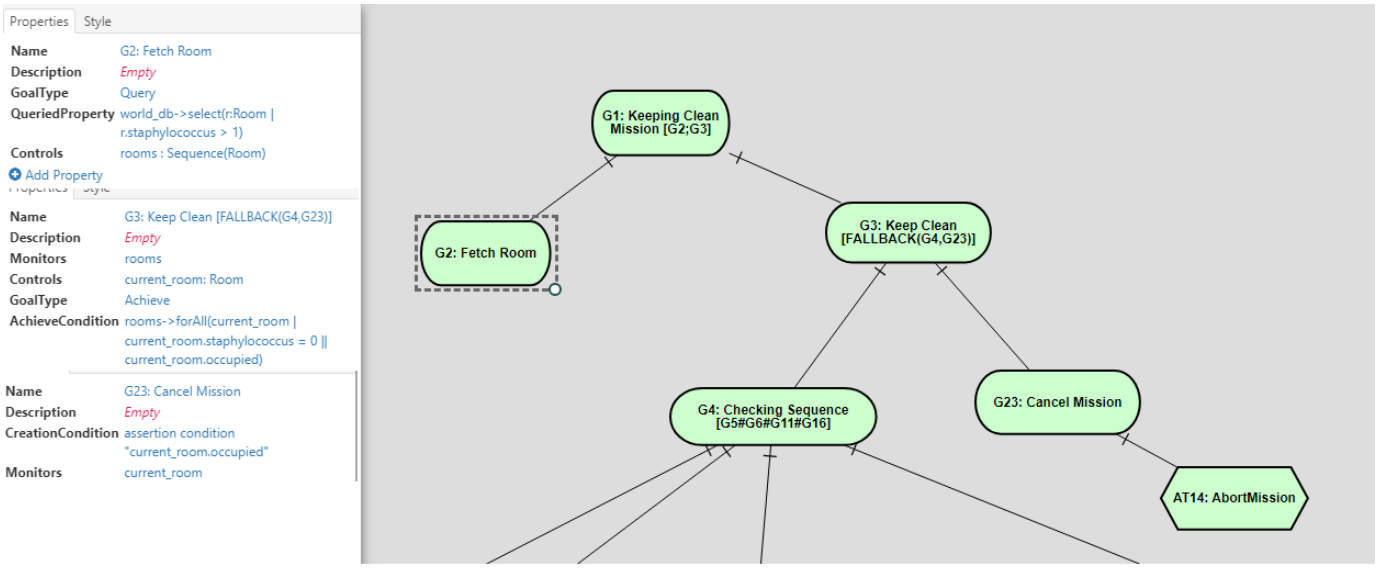

Figure 2. Parte 2: Busca das salas.

A GM começa, primeiro, buscando no $world\_db$ e colocando em uma $Sequence$ os quartos que possuem *Staphylococcus ssp.* maior que 1. Com isso, instanciamos em $G3$ um $AchieveCondition$ para declararmos a sala atual com qual o robô irá trabalhar, ao mesmo tempo que definimos que a missão(ou árvore) deve alcançar o objetivo de "limpar" todos os quartos, ou seja, executar as *tasks* de limpeza em cada um dos quartos. E como $G3$ é um $Fallback$ caso $G4$ falhe, no momento em que a sala atual($current\_room$) estiver ocupada, o robô executará $G23$, abortando a missão.

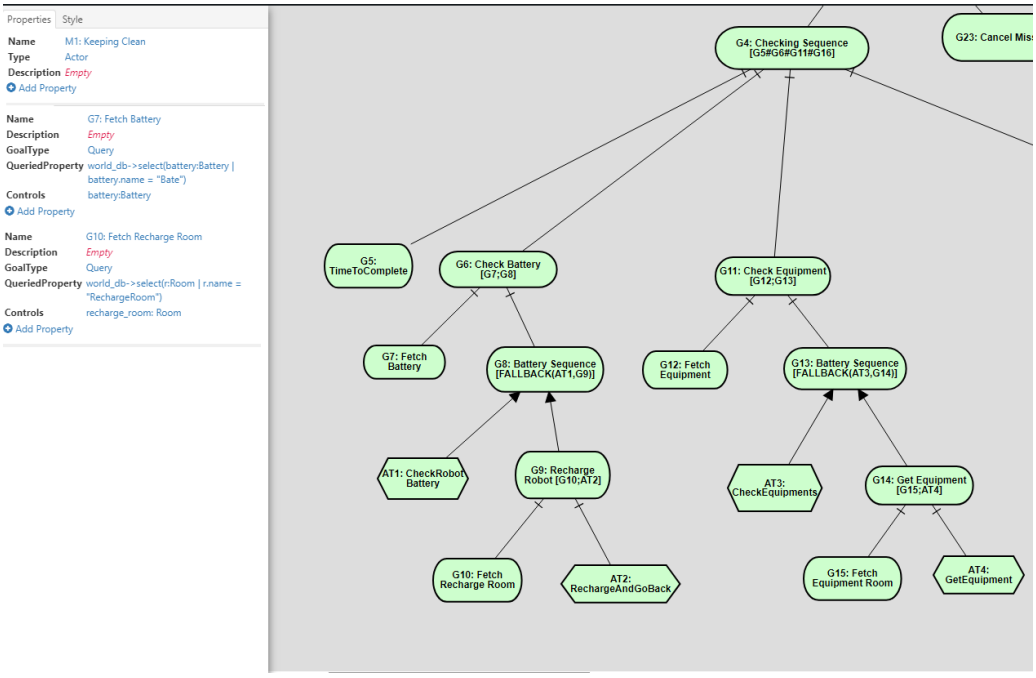

Figure 3. Parte 2: Reactive Sequence de checagem

Com as condições criadas e os quartos instanciados nos $Monitors$, iremos fazer a checagem do ttc(*time to complete*), da bateria e dos equipamento. Para o ttc, é apenas uma *Perform Goal* que irá apenas escrever no terminal por meio de *publishers e subscribers* o tempo dado pela biblioteca $nav2$. Como não criamos a coleção de *waypoints*, existe apenas o tempo estimado em que a biblioteca indica ao ir de um ponto à outro, sem "prever" os caminhos possíveis que o robô performará até concluir

seu objetivo. Para a checagem de bateria, ela ocorrerá por meio de um *Reactive Sequence* na BT, ou seja, fará a checagem se a bateria está com menos de 15% todas as vezes em que um $tick$ ocorre. Caso a bateria esteja com menos de 15%, checada em $G7$, falhando e executando a sequência de carregamento em $G9$, o qual irá recuperar o quarto de carregamento, percorrer o caminho até ela, recarregar o robô e voltar para a $current\_room$. O robô sempre volta para o último ponto e *task* em que estava antes de ir para o quarto de carregamento. Para a checagem de equipamentos, ocorre da mesma forma que. Assim, checa-se todos os equipamentos, caso não estejam nos conformes, o robô irá para o quarto de equipamentos, pegar os equipamento e voltar para o último ponto e *task* em que estava antes de buscar os equipamentos.

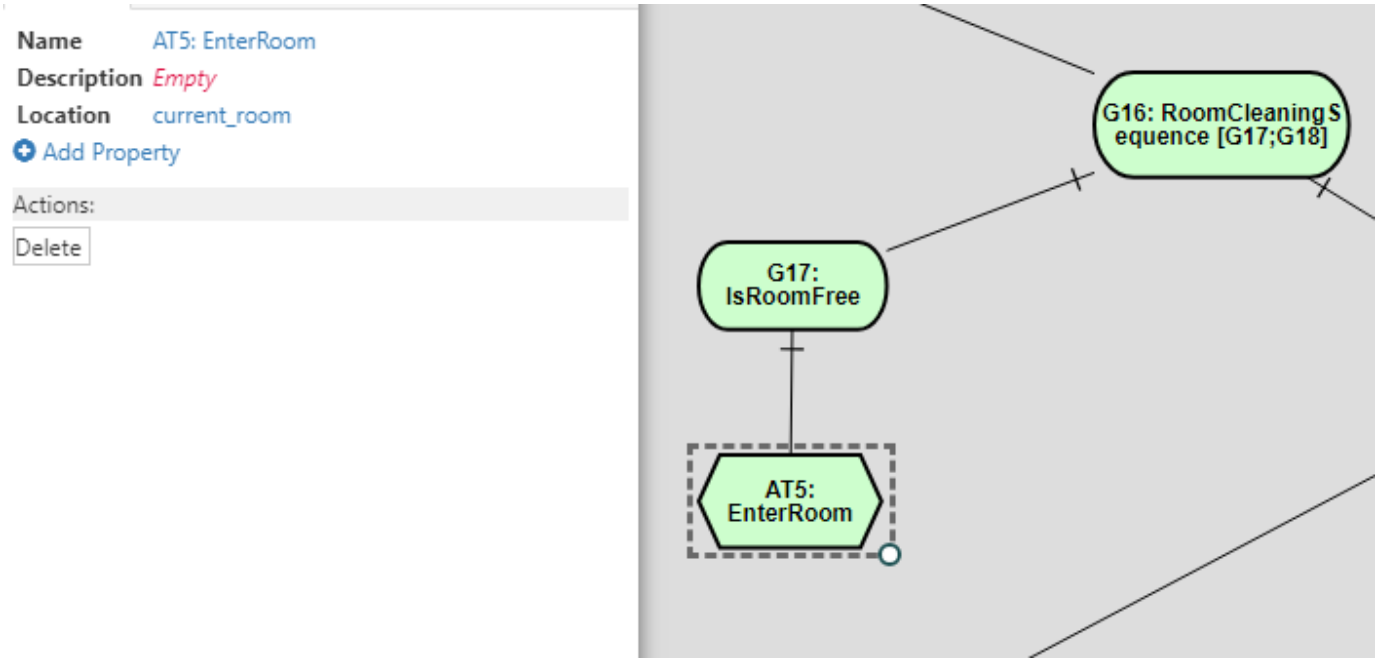

Figure 4. Parte 3: Quarto vazio ou ocupado.

A Parte 3 se trata da única checagem, ou sequência, em que fará com toda a sub árvore até agora falhe e inicializará a sequência de cancelamento da missão. A $G17$ fará a só irá realizar a *task* $AT5$ caso o $current\_room$ esteja livre, caso contrário, executará a sequência de $G23$.

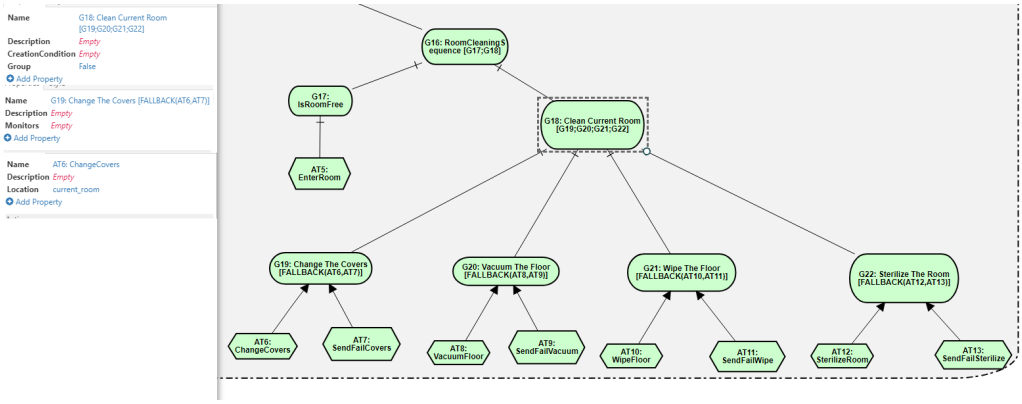

Figure 5. Parte 4: Sequência de limpeza.

A Parte 4 define as ações que o robô executará após todas as checagens anteriores, ou seja, a bateria está carregada, todos os equipamentos estão nos conformes e a sala atual está livre. Com isso, o robô fará as 4 *tasks* definidas na Introdução em sequência, mas caso ele falhe em fazer alguma ação de limpeza, uma mensagem irá ser mandada para o gerente, a ação atual cancelada e tentará executar a próxima ação descrita nos nós de $G18$.

B. HDDL

Com a GM completa, podemos implementar o HDDL, a definição de domínio. Implementado de acordo com cada um dos *goal* descritos na GM, mas, é necessário ressaltar que como foi feitas certas ações relativamente abstratas para nossa missão, como checar a bateria, checar os equipamentos, checar se $current\_room$ está ocupada, busca de equipamentos, carregamento, mandar mensagem e checar se a taxa de Staphylococcus é maior que 1 nos quartos. Assim, foi feito as seguintes implementações:

```
(define (domain hospital)      EduardoFMC, 6 days ago • feat: complete modelling?

    (:types
        room - object
        battery - object
        equipment - object
    )

    (:predicates
        (equipmentok ?r - robot)
        (occupied ?rm - room)
        (batterybelow15 ?batt - battery)
    )

    (:functions
        (recharge ?batt - battery)
        (staphylococcus ?rm - room)
    )

    (:capabilities organization equipmentsearch sndmessage recharger equipmentcheck)
```

Figure 6. Tipos, predicados e funções do HDDL.

Portanto, o robô possui poucas decomposições de missões, criando 7 ao executar a decomposição do mutrose. Uma delas a seguir:

```
------------------------------------ POSSIBLE MISSION DECOMPOSITIONS ------------------------------------
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ MISSION 1 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Task [CheckRobotBattery] With ID [AT1_1|1] with required capability [organization] at locations [] with arguments [?r,?batt=Bate] decomposed into actions:
        -> check-robot-battery ?r ?batt
AND
Task [RechargeAndGoBack] With ID [AT2_1|1] with required capabilities [organization, recharger] at locations [] with arguments [?r,?recrm=RechargeRoom,?previousrm,?batt=Bate] decomposed into actions:
        -> go-to-recharge-station ?r ?recrm
        -> wait-full-battery ?r ?batt
        -> go-back-to-room ?r ?previousrm
AND
Task [CheckEquipments] With ID [AT3_1|1] with required capability [equipmentcheck] at locations [] with arguments [?r,?eqp=EQPT] decomposed into actions:
        -> check-equipment ?r ?eqp
AND
Task [GetEquipment] With ID [AT4_1|1] with required capability [equipmentsearch] at locations [] with arguments [?r,?eqrm=EquipmentRoom,?previousrm] decomposed into actions:
        -> get-equipment ?r ?eqrm
AND
Task [EnterRoom] With ID [AT5_1|1] with required capability [sndmessage] at locations [] with arguments [?r,?rm] decomposed into actions:
        -> mark-room ?r ?rm
AND
Task [ChangeCovers] With ID [AT6_1|1] with required capability [organization] at locations [] with arguments [?r,?rm] decomposed into actions:
        -> change-covers ?r ?rm
AND
Task [SendFailCovers] With ID [AT7_1|1] with required capability [sndmessage] at locations [] with arguments [?r,?rm] decomposed into actions:
        -> send-msg-fail-covers ?r ?rm
AND
Task [VacuumFloor] With ID [AT8_1|1] with required capability [organization] at locations [] with arguments [?r,?rm] decomposed into actions:
        -> vacuum-floor ?r ?rm
AND
Task [SendFailVacuum] With ID [AT9_1|1] with required capability [sndmessage] at locations [] with arguments [?r,?rm] decomposed into actions:
        -> send-msg-fail-vacuum ?r ?rm
AND
Task [WipeFloor] With ID [AT10_1|1] with required capability [organization] at locations [] with arguments [?r,?rm] decomposed into actions:
        -> wipe-floor ?r ?rm
AND
Task [SendFailWipe] With ID [AT11_1|1] with required capability [sndmessage] at locations [] with arguments [?r,?rm] decomposed into actions:
        -> send-msg-fail-wipe ?r ?rm
AND
Task [SterilizeRoom] With ID [AT12_1|1] with required capability [organization] at locations [] with arguments [?r,?rm] decomposed into actions:
        -> sterilize-room ?r ?rm
AND
Task [SendFailSterilize] With ID [AT13_1|1] with required capability [sndmessage] at locations [] with arguments [?r,?rm] decomposed into actions:
        -> send-msg-fail-sterilize ?r ?rm
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

Figure 7. Decomposição de missão.

Grande parte das decomposições estão relacionadas com as possibilidades de falha ao checar a bateria ou os equipamentos, por exemplo, enquanto o robô executa $AT6$, ele necessita ir para o quarto de carregamento e voltar para onde havia parado.

C. Implementação

1) Behavior Tree: A *Behavior Tree* foi feita usando o software *Groot*, algumas modificações ocorreram em relação ao *Goal Model* para funcionar corretamente na execução da missão. Na figura 8 podemos observar o resultado final.

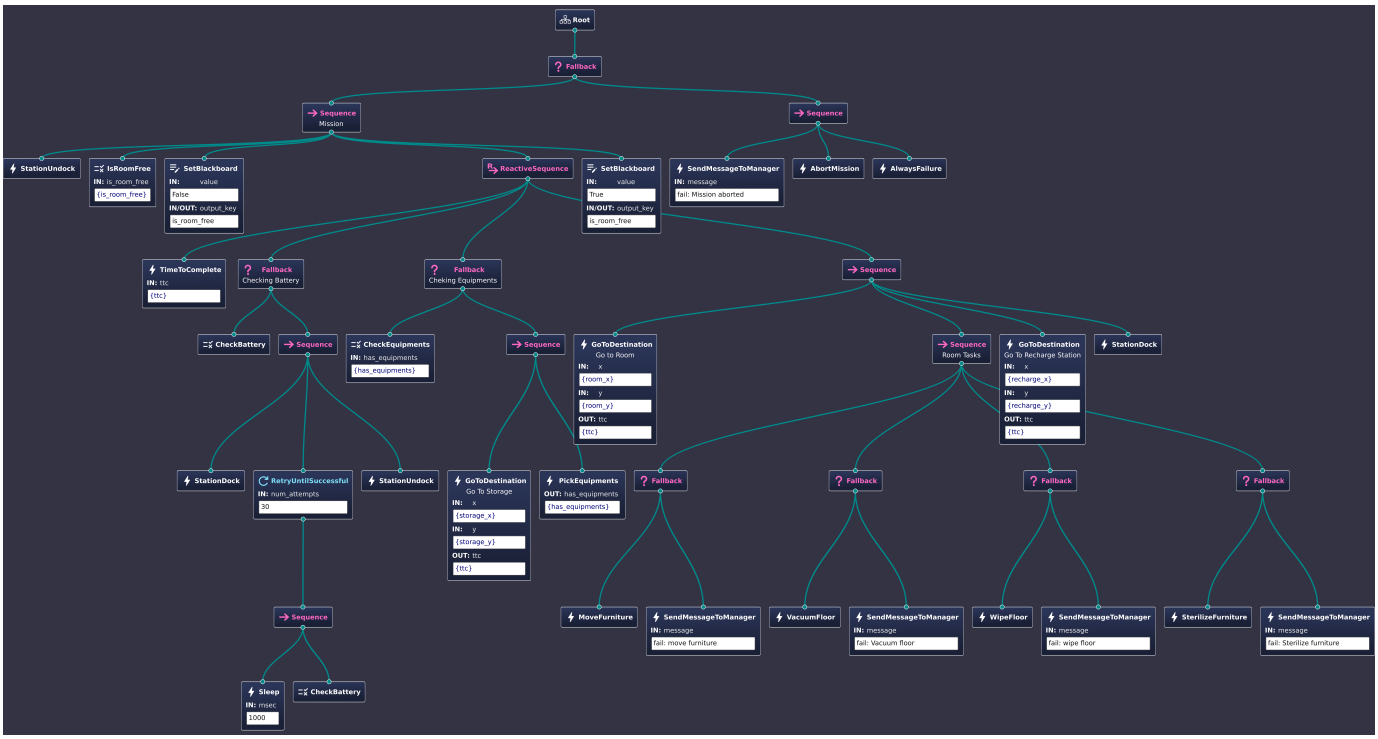

Figure 8. *Behavior Tree* da missão *keeping clean*.

Uma das adaptações foi na sub-árvore que faz a checagem da bateria, como pode ser visto na figura 9. Precisamos que, ao verificarmos que a bateria do robô está abaixo do esperado, ele se mova até a estação de carregamento (*dock*) e espere até atingir um determinado mínimo, e ao final disso desencaixe da estação e continue sua missão.

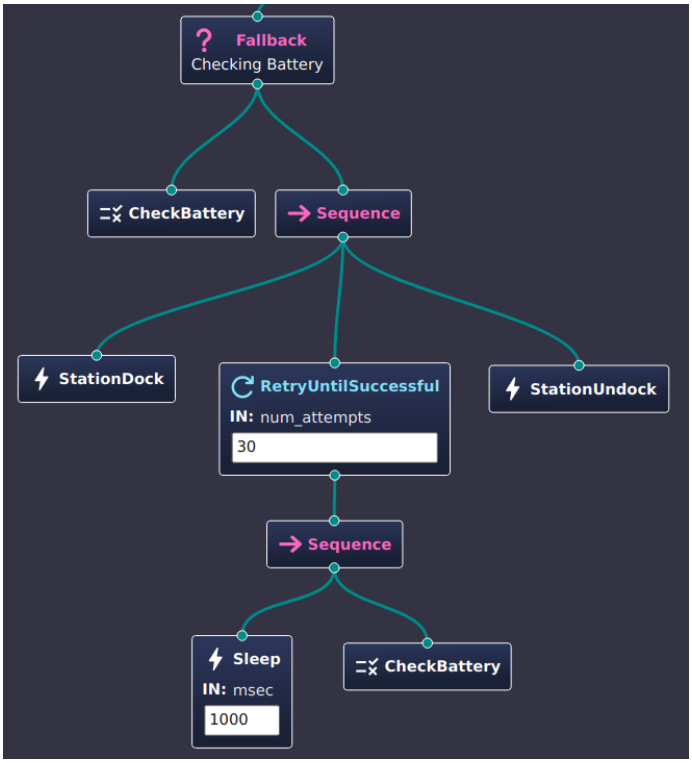

Figure 9. Sub árvore que faz a checagem da bateria do robô.

Um problema que aconteceu ao montarmos a *BT* foi de como implementamos a checagem de bateria (*CheckBattery*) e a ida ao destino (*GoToDestination*), ambos os nós eram assíncronos, ou seja, poderiam retornar o estado *RUNNING* que causava a repetição dos outros nós que fazem parte da *ReactiveSequence*. No nosso caso, ao chegar no nó *Go To Room* observado na figura 10 que retornava esse estado *RUNNING* enquanto estivesse indo ao destino, o *ReactiveSequence* repetia o ciclo de *ticks* e ao chegar no nó de checagem de bateria, começava-se uma nova função assíncrona que também retornava um *RUNNING* que por fim acabava causando um *halt* no *Go To Room*, cancelando a ida ao destino.

Para resolver esse problema, originalmente a implementação do método *onStart* do *CheckBattery* sempre retornava *RUNNING* e somente no método *onRunning* verificávamos a bateria do robô, alteramos para também verificarmos no *onStart* caso já tivéssemos a informação. Agora por que implementamos dessa forma? Ao iniciarmos a missão, demorava um tempo até alguma informação da bateria chegar ao nosso programa, por meio do *subscriber*, então enquanto essa informação não chegasse o robô não se mexia (não continuava a missão).

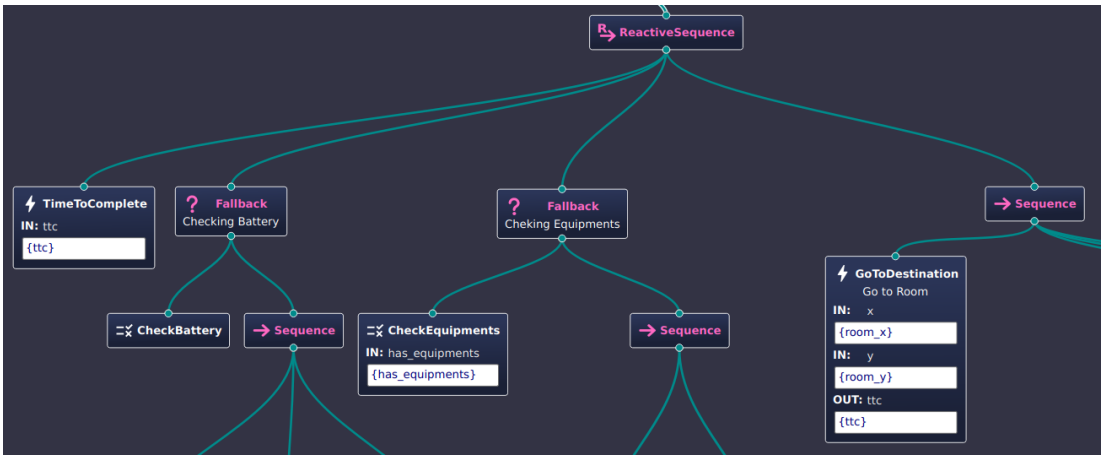

Figure 10. Seção da árvore em que ocorria problema no estado *RUNNING*.

2) Código: Sobre o código usamos o *rome_bt* como base e criamos nossos nós em cima dessa base. Discorrerei sobre eles a seguir:

Primeiro os *inputs*, é preciso passar o caminho da nossa *BT* e dos valores iniciais da *blackboard*. O exemplo abaixo funciona com oque está no repositório [2].

```
ros2 launch keeping_clean execute_bt.launch.py \
bt:="./behavior_trees/keeping_clean.xml" \
blackboard:="./behavior_trees/blackboard.json"
```

Descreverei os nós mais complexos, basicamente os que não são uma simples escrita no *console* ou verificação de algum dado na *blackboard*. Começando pelo *GoToDestination*, utilizamos a *action* "*navigate_to_pose*" passando as coordenadas de destino e ela realiza os comandos para navegar e caso precise, desviar de obstáculos. Outra peculiaridade dela é que, definimos um *output* chamado *ttc* (*time to complete*) que é igual à estimativa que essa *action* nos retorna pelo método de *feedback*, para ser usado pelo nó de estimativa de tempo, lá simplesmente consumimos esse *input* e registramos no console.

No nó *CheckBattery* utilizamos um *subscriber* no tópico "*battery_state*", sua classe foi implementada estendendo a classe *StatefulActionNode* para conseguirmos utilizar o estado *RUNNING*. Como citado anteriormente, fizemos assim porque ao iniciarmos a *BT*, a mensagem do tópico que indicava a bateria do robô demorava um tempo para iniciar, então enquanto não chegasse essa mensagem no programa, esse nó esperava com um estado *RUNNING*.

Agora um nó que percebemos a importância foi o de fazer o *dock* e *undock*, que é basicamente o robô se posicionar na base de carregamento corretamente alinhado. Eles utilizam a *action* de *dock* e *undock* vindas da biblioteca *irobot*, também adicionamos um *subscriber* para o tópico *dock_status* já que percebemos que ao fazer o pedido de *dock* ou *undock* o robô não verifica se ela já estava naquele estado e acabava acontecendo problemas na movimentação do robô.

IV. Resultados

No teste que fizemos, o robô completou a missão corretamente em um tempo de 2 minutos e 21 segundos e com a bateria restante de 98,94%. Na figura 11 podemos ver uma ilustração do caminho que o robô teve que percorrer, primeiro foi o caminho para o robô ir pegar os equipamentos (verde), após isso ele foi ao quarto realizar as tarefas (azul) e por último retornou ao ponto de origem e aportou na estação de carregamento (vermelho). Há um vídeo mostrando a execução desta missão [3].

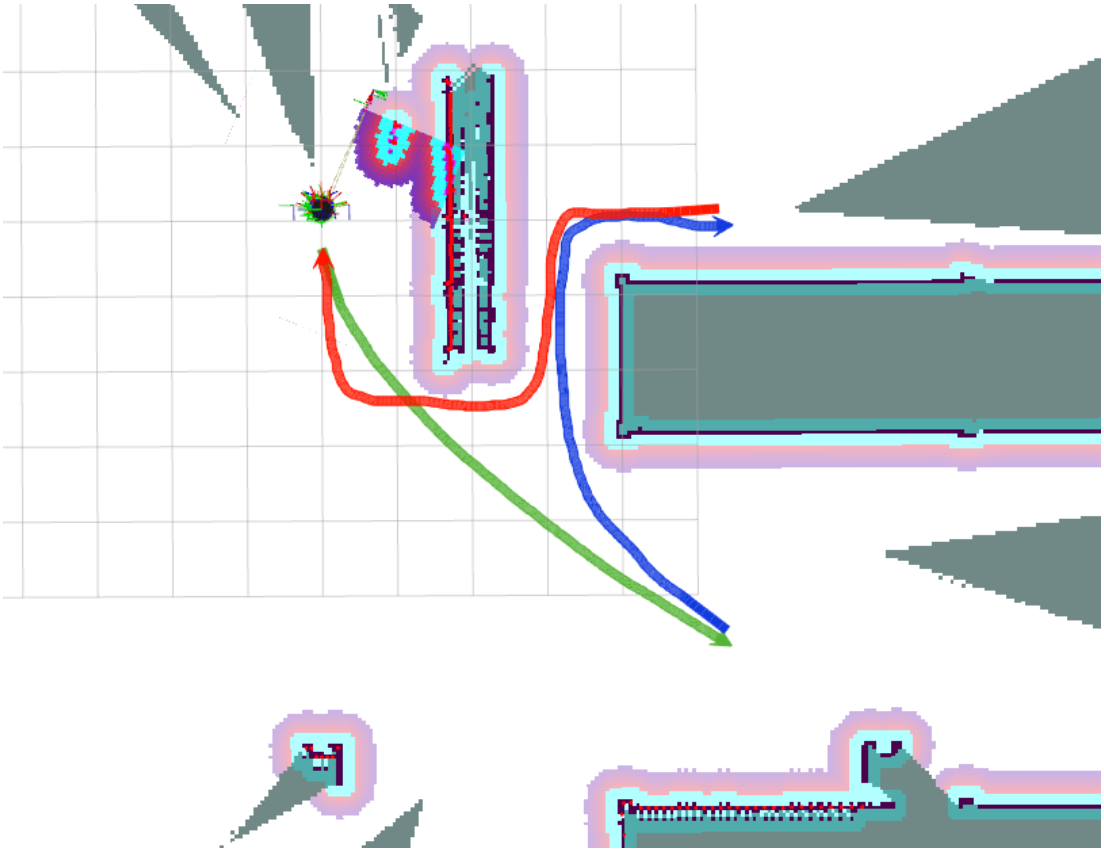

Figure 11. Caminho percorrido pelo robô no teste.

V. Conclusão e Trabalhos Futuros

Conseguimos implementar o robô para realizar a missão de forma generalizada, embora não tenhamos conseguido implementar todas as funcionalidades desejadas. Apesar de não termos conseguido integrar o LIDAR, conseguimos fazer o robô se mover para o local desejado e realizar a limpeza conforme especificado. Enfrentamos dificuldades com a modelagem HDDL e o Goal Model, principalmente devido à falta de documentação nas bibliotecas utilizadas.

Para trabalhos futuros, seria importante focar na integração completa do LIDAR para melhorar a navegação e detecção de obstáculos. Além disso, aprimorar a documentação do código e das bibliotecas utilizadas pode facilitar futuras implementações e manutenções. Continuar explorando e refinando os cenários de adaptação, como a verificação do nível de bateria e a estimativa do tempo para completar a missão, também são passos essenciais para aumentar a eficiência e a autonomia do robô.

References

[1] Mehrnoosh Askarpour, Christos Tsigkanos, Claudio Menghi, Radu Calinescu, Patrizio Pelliccione, Sergio Garcia, Ricardo Caldas, Tim J von Oertzen, Manuel Wimmer, Luca Berardinelli, Matteo Rossi, Marcello M. Bersani, and Gabriel S. Rodrigues. Robomax: Robotic mission adaptation exemplars. In *SEAMS '21: IEEE/ACM 16th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2021.

[2] Eduardo Cavalcante and Maycon Fábio. Tópicos avançados em computadores: Trabalho final. https://github.com/m4ycon/ros-tac-unb, July 2024.

[3] Eduardo Cavalcante and Maycon Fábio. Vídeo do robô executando a missão no software gazebo. https://youtu.be/yUmD4hv44kE, July 2024.

[4] Eric Gil. MutRose-Instructions. https://github.com/ericbg27/MutRose-Instructions, July 2021.