Desenvolvimento de Veículos Autônomos de Inspeção e Monitoramento em Unidades Flutuantes de Armazenamento e Transferência de Petróleo e Gás Natural (FPSO's)

Elizabeth Carvalho
Instituto Infnet
Rio de Janeiro, Brasil
elizabeth.carvalho@al.infnet.edu.br

Giovanna Anacleto
Instituto Infnet
Rio de Janeiro, Brasil
giovana.anacleto@al.infnet.edu.br

***Resumo*—Acidentes na indústria brasileira causam prejuízos em diversas esferas, como por exemplo econômica e ambiental. Com o aumento da perfuração e exploração de novos poços para extração de petróleo e gás natural, é imprescindível que os ambientes offshore estejam munidos de informações para realizar ações preventivas a fim de evitar tragédias. Sendo assim, é de suma importância trazer elementos da tecnologia, tal como robótica, para apoiar na realização dessas tarefas, tais como a utilização de robôs autônomos que monitorem continuamente o ambiente no qual esteja inserido por meio de câmeras e sensores. Neste cenário, os principais objetivos da solução proposta nesse artigo são a navegação autônoma do robô e a inspeção/monitoramento do local por meio extração de imagens coletadas pela câmera.**

Palavras chaves*—*fpso, inspeção, monitoramento, robô, visão computacional, navegação autônoma, navigation stack, qrcode

I. Introdução

O ambiente de trabalho em áreas industriais no Brasil foi classificado como um dos mais perigosos do mundo [1]. Com foco no quesito segurança, é possível avaliar a grande importância de sistemas de apoio que auxiliem nas inspeções e monitoramento desses ambientes.

Sob essa análise e dando foco à indústria de Óleo e Gás e sua crescente demanda [2], a proposta desse artigo tem como objetivo desenvolver um sistema capaz de deslocar-se em um *FPSO*[1] e apoiar na realização do processo de inspeção e monitoramento da mesma.

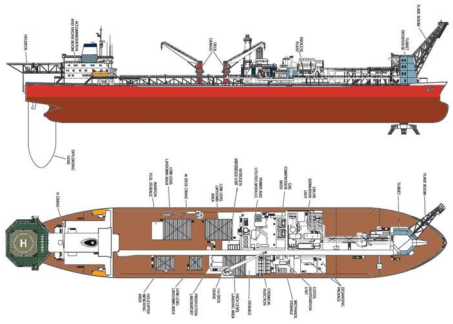

Figura 1: Exemplo de uma estrutura de FPSO[1] [3]

Atualmente, risco de explosão por vazamento de gases é um dos quatro maiores riscos nos quais os colaboradores offshore estão expostos [4] e, na maioria dos casos, as inspeções são feitas de forma manual. A solução tem como objetivo a utilização de um robô com navegação de forma autônoma (também podendo ser operado remotamente se necessários for), coletar dados relacionados ao controle de qualidade do ar utilizando sensores acoplados para realizar a inspeção do local, bem como câmeras térmicas a fim de avaliar riscos de explosão devido a altas temperaturas e, também, realizar inspeção de equipamentos por meio de imagens extraídas de câmeras, utilizando algoritmos de visão computacional. Com isso, a utilização da inspeção autônoma torna possível a prevenção de acidentes ao ser humano e catástrofes financeiras e ambientais, a exemplo do acidente ocorrido na plataforma *Piper Alpha* [5] em 1988 que foi considerado o pior desastre petrolífero marítimo.

Neste artigo usaremos como exemplo um *FPSO*[1] baseado no arranjo tipo "A"[6] de uma UEP[5] [7] casco de *VLCC*[6] [8] convertido com ancoragem do tipo *Spread Mooring*[7][9].

Figura 2: Arranjo tipo "A"

As cores na figura acima classificam o grau de risco de vazamento de gases ou explosão dos módulos.

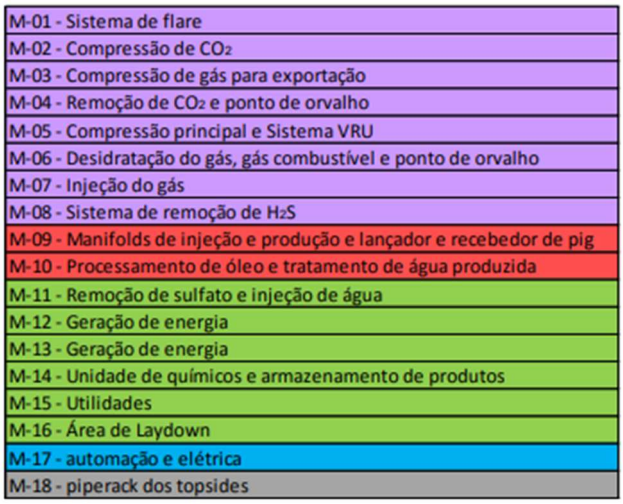

Figura 3: Identificação dos módulos em um FPSO [6]

Módulo	Risco
Automação e Elétrica	
Área de Laydown	
Laboratório	
Utilidades	
Unidade de químicos e armazenamento de produtos	
Geração de energia	
Injeção de água e remoção de sulfato	
Processamento de óleo e tratamento de água	
Manifolds e pigs	
Remoção de H_2S	
Remoção de CO_2	
Injeção de gás	
Desidratação de gás, gás combustível e HCDP	
Compressão principal e VRU	
Compressão de exportação	
Compressão de CO_2	
Sistema de Flare	

Figura 4: Classificação dos módulos quanto ao risco de vazamento de gases ou explosão [6]

A solução proposta neste artigo foi realizada em ambiente simulado e os objetivos principais são a navegação autônoma do robô e a inspeção/monitoramento do local por meio extração de imagens coletadas pela câmera através da leitura de tags *QRCode*.

O restante deste artigo está organizado da seguinte forma: seção 2 apresenta a solução proposta para o problema, seção 3 apresenta a explicação dos algoritmos implementados, na seção 4 apresenta os testes realizados e resultados obtidos e finalmente a seção 5 conta com a conclusão no qual chegamos.

II. Apresentação da Solução

O sistema ideal seria um robô quadrúpede devido a diversidade de dimensões e obstáculos em diferentes níveis espalhados pelos módulos que compõe um *FPSO*[1] e equipado com: sensor *LiDAR*[2] para exploração e navegação autônoma por meio da técnica *SLAM*[3]; sensores de gases, como por exemplo MQ-136 utilizado para detecção de H2S (Gás Sulfídrico) no ar; câmera térmica para monitorar a temperatura do local e assim gerar informações para apoiar na detecção de possíveis cenários de anomalias; câmera estéreo a fim de realizar as inspeções por meio de técnicas de visão computacional; *GPS*[4] para captura do movimento/posicionamento.

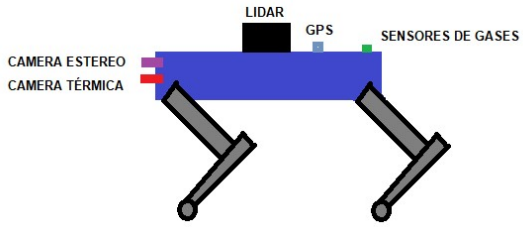

Figura 5: Sistema de robô quadrúpede ideal

A fim de testar a viabilidade técnica do sistema nesse artigo, a modelagem da solução será realizada utilizando o software *Gazebo* [10] e será utilizado o modelo de base móvel "*TurtleBot3 Waffle Pi*" criada pela da empresa *Robotics* [11].

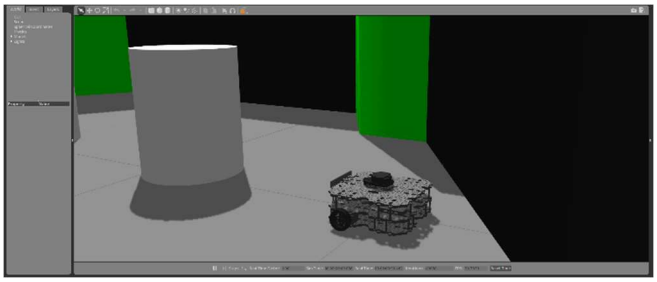

Figura 6: Ferramenta de simulação Gazebo [12]

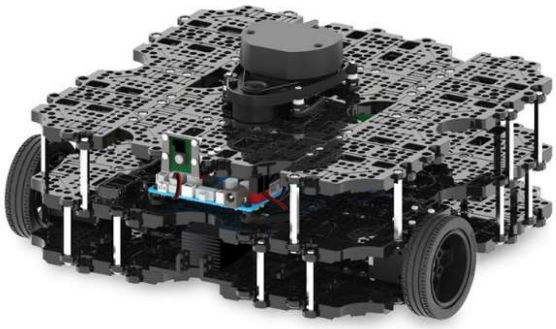

Figura 7: Modelo "Turtlebot3 Waffle Pi" da empresa Robotics

O modelo possui dimensões 281 x 306 x 141 (L x C x A, em mm) e peso de 1.8Kg. Conta com câmera, *LiDAR*[2] 360º para *SLAM*[3] e Navegação, como computador de bordo utiliza a placa de prototipação *RaspberryPi* [13] que dispõe de módulo *Wi-fi* integrado permitindo a comunicação via rede, além de ser compatível com sistema *ROS1* [14] e com execução de scripts na linguagem de programação *Python,* que foi a escolhida para ser utilizada na implementação da solução proposta. Também dispõe simulações de navegação e *SLAM*[3] prontos para serem utilizados, tornando o modelo apropriado para validação da solução devido a sua praticidade.

III. MATERIAIS E MÉTODOS

O cenário fictício para esta solução "*FPSO World*", foi criado com uso do software *Gazebo*, utilizando o recurso "*Building Editor*" [15], com objetivo de simular o processo de navegação e inspeção em um ambiente virtual.

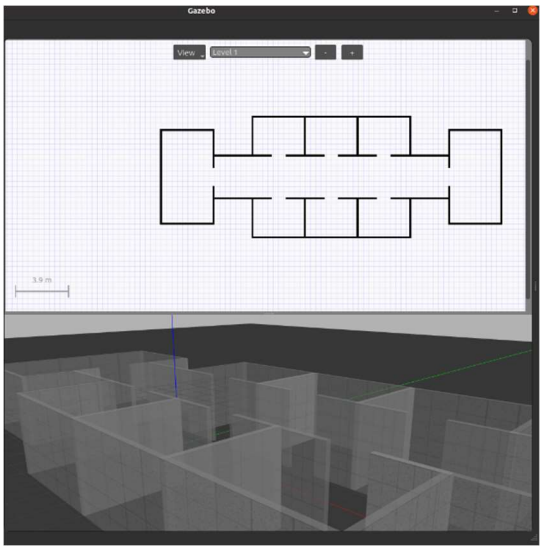

Figura 8: Construção do cenário no Gazebo

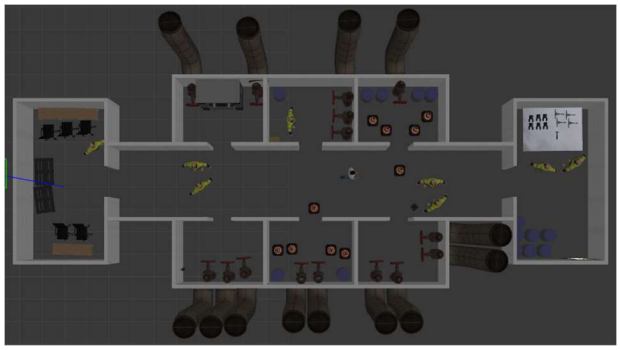

Figura 9: "FPSO World" criado no Gazebo

Foi incluído no cenário, tags *QRCode* que identificam o objeto a ser inspecionado/monitorado. As tags foram criadas utilizando o aplicativo *QRScanner* [16] e convertidas para modelos (objetos) a fim de serem utilizados na modelagem de um cenário no software *Gazebo* através da biblioteca "*AR_Tag Models for Gazebo*" [17].

Figura 10: A esquerda, QRCode gerado pelo aplicativo. A direita, tag QRCode inserida no Gazebo.

Para realizar as tarefas relacionadas à visão computacional foi escolhida a biblioteca *OpenCV*[8] [18], sendo esta multiplataforma, tornando-a compatível com a placa de prototipação *RaspberryPi* utilizada no modelo de robô escolhido, de código livre e documentação abrangente. Para decodificação do *QRCode* foi utilizada a biblioteca *Pyzbar* [19].

Para realização da navegação autônoma, foi utilizada a biblioteca *ROS 2D Navigation Stack* [20] que obtém informações da posição do atual robô, fluxos de sensores, definição de pose de objetivo e emite comandos de velocidade seguros que são enviados para a base móvel. Dessa forma, o deslocamento do robô será realizado através da leitura das coordenadas previamente definidas e ordenadas, contidas em um arquivo de texto no formato *YAML*[9] [21] (para leitura deste, foi utilizada a biblioteca *PyYAML* [22], fazendo com que ao final todos os pontos de interesse tenham sido inspecionados e monitorados.

O hardware utilizado criação/execução das simulações, bem como implementação e execução dos scripts de navegação e coleta de imagens foi de um notebook rodando nativamente o sistema operacional *Ubuntu* 20.04.2*LTS* com *CPU Intel Core i7*-5500U 64*bits* 1685*MHz*, 8*GB* de memória *RAM*, 240*GB* de armazenamento em *SSD* e placa de vídeo integrada.

O cenário "*FPSO World*" foi mapeado utilizando a técnica de *SLAM*[3] por meio da ferramenta *RViz [23],* que permite com que robô seja visualizado pela ferramenta por meio de leitura dos sensores através dos tópicos.

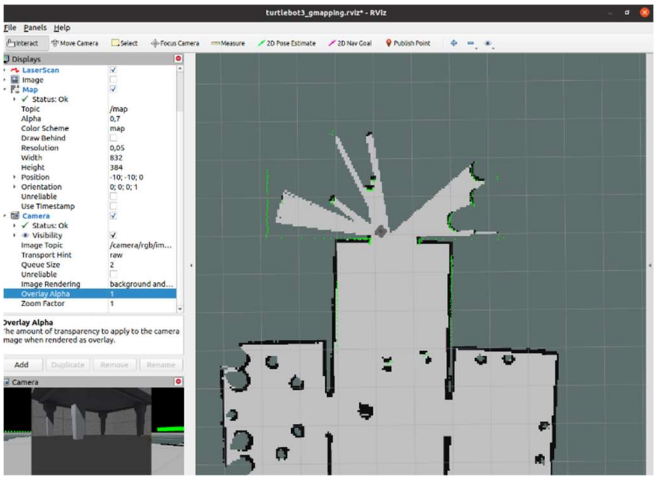

Figura 11: Mapeamento por meio de SLAM utilizando a ferramenta Rviz

Após o mapeamento foi realizado a definição das coordenadas de destino no arquivo de texto de forma a resultar em uma navegação cíclica a fim de monitorar o ambiente de maneira mais eficiente.

Figura 12: Mapa criado a partir do processo de SLAM

Para realizar o deslocamento do robô de forma autônoma, foi implementada uma solução utilizando como base a

biblioteca 2D Navigation Stack que usa um planejador global e local para cumprir a meta de navegação e gerenciar a comunicação dentro da pilha de navegação de forma a alcançar os objetivos definidos.

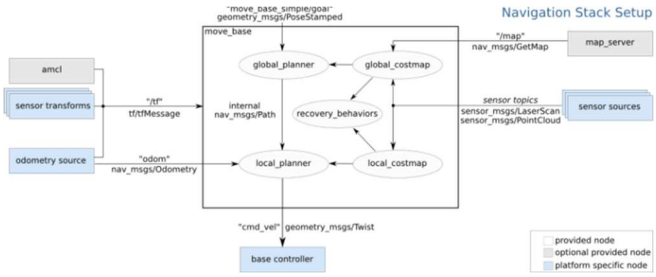

Figura 13: Tópicos do core "move_base" utilizados pela biblioteca 2D Navigation Stack.

As informações dos sensores do robô são coletadas, transformadas e combinadas com uma estimativa da posição do robô com base na sua posição inicial que é definida através de algoritmo. Esta informação é publicada para que tópico *core* "*move_base*" possa ajustar o posicionamento do robô no contexto do mapa, fazendo com que os cálculos de trajetória baseados nas transformações sejam condizentes com o ponto de origem no mapa.

As coordenadas dos objetivos de destino, chamadas de "*Pose*", possuem a posição e orientação finais do objeto, possibilitando que o robô se deslocar para o módulo do *FPSO* especificado na rota previamente definida em arquivo de texto.

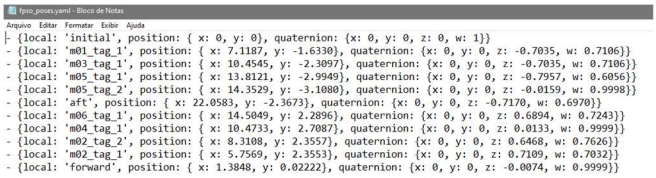

Figura 14: Coordenadas dos pontos de interesse lidas pelo algoritmo

O primeiro passo consiste em atualizar a posição do robô em relação ao ponto de origem no mapa.

Algoritmo 1 Configurando a Pose Inicial

Input: Leitura de arquivo de texto contendo a coordenada do ponto inicial do robô em relação ao cenário/mapa.
Output: Atualização da posição inicial do robô em relação ao ponto de origem do mapa.

1 Leitura da coordenada

2 Leitura do arquivo de texto no formato YAML contendo a pose
3 Resgatar a primeira pose da lista, previamente acordada que é a pose inicial

4 Atualização da pose

5 Setar a posição inicial através do tópico "/initialpose"

Figura 15: Algoritmo estruturado de atualização da pose inicial

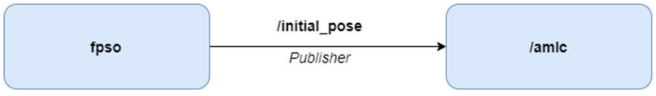

Figura 16: Estrutura de comunicação (escrita) para atualização de pose inicial lida pelo sistema de localização probabilística para um robô se movendo em 2D. Ele implementa a abordagem de localização de Monte Carlo adaptativa (ou amostragem KLD) que usa um filtro de partículas para rastrear a pose de um robô em um mapa conhecido [24].

Após lido, um objetivo de destino, é realizada uma publicação contendo a pose (posição e orientação) que é calculada uma trajetória e que seja enviado por meio de tópico os comandos de velocidade que farão com que o robô se desloque evitando obstáculos mapeados na etapa de *SLAM* onde foram estabelecidos os mapas de custo local e global, até alcançar o local e o objeto a inspecionado, sendo este processo monitorado constantemente através da leitura da posição em que se encontra o robô de forma que a sua trajetória seja sempre ótima.

Ao chegar na coordenada de destino, o robô aguarda um tempo pré-determinado a fim de que os sensores coletem as informações necessárias e após, segue para a coordenada seguinte da lista.

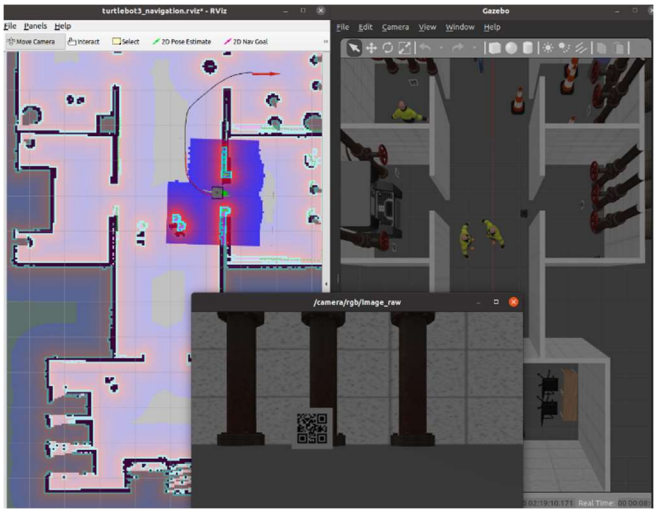

Figura 17: A esquerda, vista do Rviz com mapa de custo em azul e vermelho; A direita, o robô no cenário simulado no Gazebo; Ao Centro, visão da câmera do fixada no robô.

Para navegação, é iniciado um nó chamado "*fpso*", que escreve a nova pose de destino no tópico de "*/move_base_simple/goal*" disponibilizado pela biblioteca *2D Navigation Stack* que de forma abstraída e resumida fará a leitura das informações de odométricas e publicação no tópico de controle de velocidade. O nó também lê o tópico "*move_base/result*" que informa se a pose já foi alcançada e com isso permite que os processos de inspeção com a câmera e demais sensores sejam iniciados e depois tenha um novo destino setado.

Algoritmo 2 Navegação autônoma

Input: Leitura de arquivo de texto contendo as coordenadas dos pontos de interesse a serem inspecionados.
Output: Definição de velocidade linear e angular a fim de alcançar a(s) coordenada(s) desejada(s).

1 Leitura da coordenada

2 Leitura do arquivo de texto no formato YAML contendo as poses
3 Armazenar a lista em um array a ser consultado de forma iterativa .

4 Publicação da pose de destino

5 Resgatar a pose de destino a ser enviada no passo atual da iteração
6 Publicação no tópico "/move_base_simple/goal" da pose de destino.

7 Avaliação se objetivo alcançado

8 Leitura do tópico "/move_base/result", a fim de avaliar se a pose definida foi alcançada (status = 3).
8.1 Se alcançada, aguardar por 3 segundos e seguir para nova pose (item 4)
8.2 Caso contrário, prosseguir com item 8, mantendo atual deslocamento.

Figura 18: Algoritmo estruturado da navegação autônoma por meio de rota

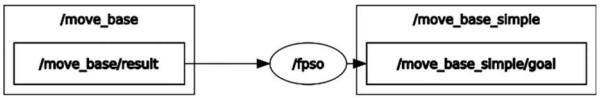

Figura 19: Estrutura de comunicação (leitura/escrita) de tópicos, bem como o processamento a ser realizado no processo de navegação autônoma.

Contemplando as etapas de coleta de informações durante o tempo pré-determinado, foi implementada, também, uma solução que abrange o reconhecimento das tags por meio da biblioteca *OpenCV*, utilizando em conjunto a biblioteca *CVBridge* [25], que faz uma ponte de comunicação para publicação das imagens coletadas e tratadas em um tópico no *ROS*. Para a parte inicial de captura da visão do robô, foi realizada a leitura do tópico "*/camera/rgb/image_raw*", que foi tratada em uma função utilizando o *OpenCV*, retornando um objeto de imagem que abre uma janela com o resultado desse tratamento:

Figura 20: Janela de leitura do tópico de câmera do robô

Este tratamento, contempla a verificação constante da imagem atual capturada, na tentativa de identificação e decodificação de polígonos que formam um *QRCode*, utilizando a biblioteca de decodificação *Pyzbar*. Quando um objeto *QRCode* é encontrado, o algoritmo realiza a decodificação, identifica o objeto utilizando um contorno e insere um texto resultante da decodificação referente ao tipo de módulo e válvula previamente mencionados:

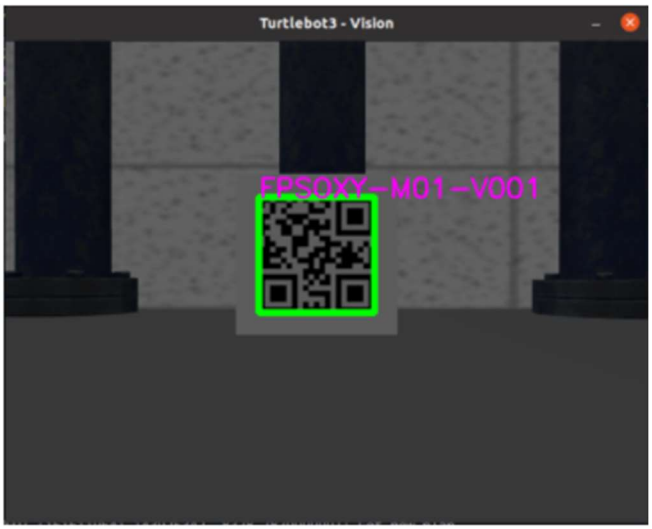

Figura 21: Identificação da tag

Após identificada a tag de *QRCode*, é realizada a leitura do tópico "*/move_base/result*" a fim de checar se robô chegou ao seu objetivo de destino. A resultante para identificação da chegada do robô até o objetivo de destino é o código de status 3.

```
[ INFO] [1616119667.289661251, 8333.387000000]: Got new plan
[ INFO] [1616119668.744106988, 8333.587000000]: Got new plan
[ INFO] [1616119669.771850565, 8333.787000000]: Got new plan
[ INFO] [1616119669.772598336, 8333.787000000]: Goal reached
```

Figura 22: Momento da troca de status do tópico /move_base/result

Com essa resultante, o robô identifica o status de chegada e se já existiu uma captura de foto do ambiente, assim realizando a captura da imagem da câmera, salvando em um diretório e realizando a publicação das informações no tópico criado "*/image_info_tag*" capturadas no momento da imagem como: nome da imagem, posição do robô e horário:

```
starwars@starwars-Inspiron-7348:~$ rostopic echo /image_info_tag
image_info_taWARNING: no messages received and simulated time is active.
Is /clock being published?
data: "tag: FPSOXY-M01-V001, pose: [x: 7.1187, y: -1.6330], timestamp: 20210318231920"
---
```

Figura 23: Leitura do tópico om as informações da imagem e coordenadas do robô

Exemplo do algoritmo estruturado:

Algoritmo 3 Identificação e Extração de Imagem

Input: Leitura da imagem capturada em tempo real pelo robô.
Output: Envio da captura e coordenadas da imagem.

1 Identificação do QRCODE

2 Leitura do tópico da câmera acoplada ao robô.
3 Identificação do QRCODE aplicando OpenCV.

4 Avaliar se o robô encontra-se em posição

5 Leitura do tópico "*/move_base/result*" para avaliar se a câmera já se encontra em frente ao objeto de interesse (Status = 3)
5.1 Se sim, avançar ao item 6.
5.2 Caso contrário, aguardar.

6 Capturando imagem

7 Extração da imagem atual através da leitura do tópico "*/camera/rgb/image_raw*"
8 Envio da imagem para um diretório definido.

9 Publicação das coordenadas da imagem

10 Publicação em tópico "*/image_info_tag*" passando as coordenadas da imagem atual baseado na posição do robô, timestamp e nome da tag

Figura 24: Algoritmo estruturado do algoritmo de extração de imagem.

Com isso, a estrutura de tópicos e nós ficam descritas desta forma:

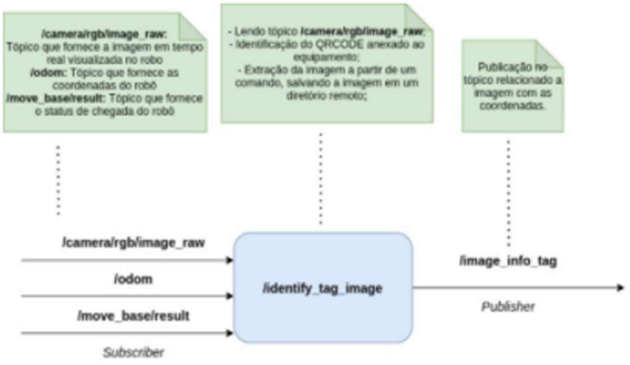

Figura 25: Estrutura de comunicação (leitura/escrita) de tópicos, bem como o processamento a ser realizado no processo de extração de imagem por meio de QRCode.

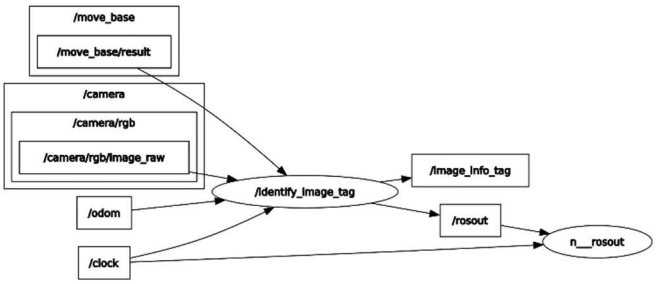

Figura 26: Estrutura de tópicos e nós sendo visualizados na ferramenta RQT Graph [26]

Os algoritmos e demais arquivos necessários para implementação dessa solução se encontram em um repositório de códigos [27] disponível no *GitHub* [28].

IV. Resultados Obtidos

Para avaliar se o processo de navegação ocorreu de forma satisfatória, foi criada uma rota passando por todos as tags *QRCode* inseridas no cenário, sendo ao todo 10 pontos de interesse tal como pode ser observado na Figura 12.

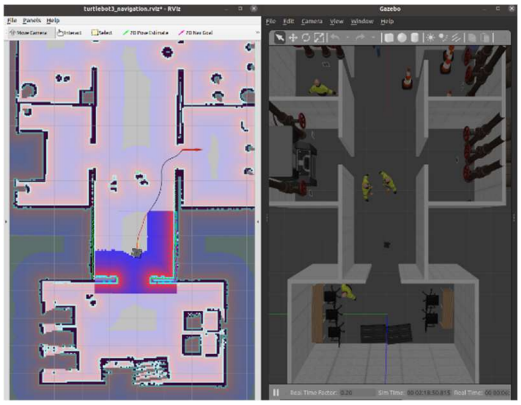

Figura 27: Robô iniciando o deslocamento ao ponto de interesse 1

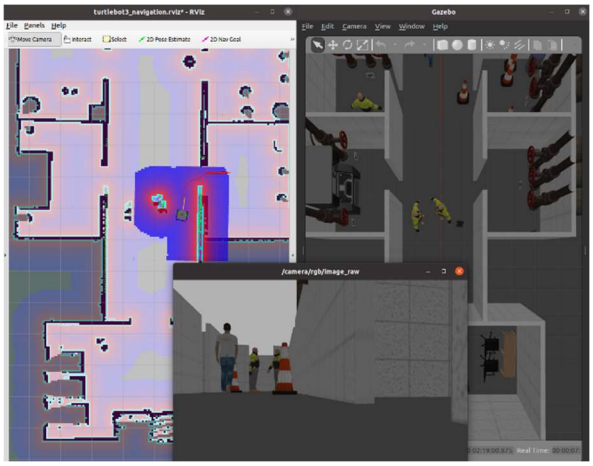

Figura 28: Robô em deslocamento

A fim de monitorar a evolução dos testes foi adicionado um log de exibição no *prompt* para verificar a pose de destino corrente, tempo de deslocamento entre as poses, permitindo com isso que seja mensurado também a duração de uma volta completa.

```
[INFO] [1616106668.141761, 0.000000]: Deslocando para pose: 1 - m01_tag_1 em 18/03/2021 19:31:08
[INFO] [1616106811.831140, 8455.349000]: Deslocando para pose: 2 - m03_tag_1 em 18/03/2021 19:33:31
[INFO] [1616106937.480850, 8485.957000]: Deslocando para pose: 3 - m05_tag_1 em 18/03/2021 19:35:37
[INFO] [1616107082.315259, 8523.068000]: Deslocando para pose: 4 - m05_tag_2 em 18/03/2021 19:38:02
[INFO] [1616107418.142159, 8604.875000]: Deslocando para pose: 5 - aft em 18/03/2021 19:43:38
[INFO] [1616107689.718981, 8672.679000]: Deslocando para pose: 6 - m06_tag_1 em 18/03/2021 19:48:09
[INFO] [1616107869.453187, 8717.683000]: Deslocando para pose: 7 - m04_tag_1 em 18/03/2021 19:51:09
[INFO] [1616108053.964185, 8761.994000]: Deslocando para pose: 8 - m02_tag_2 em 18/03/2021 19:54:13
[INFO] [1616108219.081081, 8803.991000]: Deslocando para pose: 9 - m02_tag_1 em 18/03/2021 19:56:59
[INFO] [1616108294.390906, 8823.392000]: Deslocando para pose: 10 - forward em 18/03/2021 19:58:14
[INFO] [1616108458.771914, 8864.495000]: Deslocando para pose: 1 - m01_tag_1 em 18/03/2021 20:00:58
[INFO] [1616108580.878855, 8894.501000]: Deslocando para pose: 2 - m03_tag_1 em 18/03/2021 20:03:00
[INFO] [1616108708.408617, 8926.998000]: Deslocando para pose: 3 - m05_tag_1 em 18/03/2021 20:05:08
```

Figura 29: Log da evolução da rota

Como pode ser observado, as poses de destino foram alcançadas em sua totalidade e em um tempo razoavelmente bom (em torno de 30 a 35 minutos), levando em consideração o tamanho do cenário da simulação que possui em torno de 44m de largura por 80m de comprimento. Contudo em alguns momentos observamos que o robô executou alterações de planejamento de rota a fim de realizar regulação de postura que resultaram em um aumento no tempo entre os deslocamentos. Outro fator observado foi que quanto maior o número de processos sendo executados em segundo plano no

notebook onde a simulação foi realizada, maior foi o tempo entre os deslocamentos.

Para a realização da avaliação sobre a leitura das tags, identificação e captura das imagens, foram feitos os testes em tempo de execução na navegação autônoma, em ambiente simulado. Desta forma, como pode ser observado na Figura 30, realizamos a leitura das tags em um dos pontos de interesse de navegação do robô.

Figura 30: Leitura e identificação das tags durante o processo de navegação

Na Figura 31, podemos observar que após a identificação, foi salva a imagem correspondente a tag lida no ponto de interesse do robô.

Figura 31: Diretório com as imagens capturadas salvas

E, na Figura 32, podemos observar também, a figura final salva, mostrando todas as características do ambiente.

Figura 32: Imagem salva com as características capturadas no mundo

V. Conclusão

Por meio deste artigo observamos a viabilidade da solução proposta, tendo em vista os resultados dos testes podemos verificar que é totalmente possível a realizar, com apoio de um robô, o processo de monitoramento e inspeção de um *FPSO* de maneira mais eficaz, obtendo com isso um modelo de atuação mais proativo, tendo em vista que as conformidades poderão ser avaliadas em cenários diversos e com muito mais frequência, gerando economia no processo.

Os processos de navegação e extração de imagens em ambiente simulado poderiam ser transferidos para o robô quadrúpede já que seu nível de controle possibilita com que possa andar pelos ambientes encontrados em *FPSO*, com degraus, chão gradeado e desníveis diversos, sendo também apropriada para ambientes insalubre e explosivos [29].

Os robôs de atmosfera explosiva que possuem componentes eletrônicos, motores de acionamento, transdutores para ultrassom são encapsulados de forma a ficar selados e preenchidos de gás inerte para que o oxigênio possa ser eliminado. Esses robôs possuem um sistema de detecção de pressão interna, pois quando ocorre uma variação da pressão, o sistema de energia é cortado, impedindo contato com atmosfera explosiva [30].

No relacionado a trabalhos futuros, os próximos passos seriam, ainda em nível de prototipação, realizar a navegação de forma a desviar de obstáculos dinâmicos, permitindo assim com que sejam inspecionados locais com trânsito de pessoas e a utilização de reconhecimento de imagem para identificação de anormalidade em peças/componentes e aferição de mostradores com identificação de leitura.

Glossário

1. ***FPSO***: do inglês *"Floating Production Storage and Offloading"* – em português "Unidade Flutuante de Armazenamento e Transferência de Petróleo e Gás Natural".
2. ***LiDAR:*** do inglês *"Light Detection And Ranging"* – em português "Detecção de luz e alcance".
3. ***SLAM:*** do inglês "*Simultaneous Localization And. Mapping"* – em português "Localização e Mapeamento Simultâneos".
4. ***GPS***: do inglês "*Global Positioning System*" – em português "Sistema de Posicionamento Global".
5. **UEP:** Unidade de Esforço de Produção.
6. **VLCC**: do inglês *"Very Large Crude Carrier"*.
7. **Spreed Mooring**: Amarração com quadro de ancoragem.
8. **OpenCV**: do inglês *"Open Source Computer Vision Library"* – em português "Biblioteca de código aberto para visão computacional".
9. **YAML**: do inglês *"YAML Ain't Markup Language"* – em português "YAML não é linguagem de marcação".

Rerências

[1] "Operadoras estimam aportes de R$ 5,4 bi em exploração para 2021" [Online]. Disponível: https://epbr.com.br/operadoras-estimam-aportes-de-r-54-bi-em-exploracao-para-2021/

[2] "Conheça Os Riscos Ocupacionais E Perigos Nas Plataformas Petrolíferas" [Online]. Disponível: https://blog.volkdobrasil.com.br/conheca-os-riscos-ocupacionais-e-perigos-nas-plataformas-petroliferas/

[3] "FPSO - Floating Production, Storage and Offloading" [Online]. Disponível: https://www.globalsecurity.org/military/systems/ship/platform-fpso-comp.htm

[4] "Conheça Os Riscos Ocupacionais E Perigos Nas Plataformas Petrolíferas" [Online]. Disponível:

https://blog.volkdobrasil.com.br/conheca-os-riscos-ocupacionais-e-perigos-nas-plataformas-petroliferas/

[5] "Piper Alpha" [Online]. Disponível: https://pt.wikipedia.org/wiki/Piper_Alpha

[6] "Arranjos de convés FPSOs para operação no pré-sal – Gabriel Campagnac Wollner" [Online] Disponível: http://monografias.poli.ufrj.br/monografias/monopoli10017063.pdf

[7] "ERConsultoria – Esforço de Produção" [Online] Disponível: https://www.ecrconsultoria.com.br/pt-br/insights/artigo/controladoria-estrategica/o-que-e-uep-unidade-de-esforco-de-producao

[8] "Wikipedia – VLCC' [Online] Disponível: https://pt.wikipedia.org/wiki/Very_Large_Crude_Carrier

[9] "Maxwell Puc-Rio – 2.6. Sistemas de Ancoragem" [Online] Disponível: https://www.maxwell.vrac.puc-rio.br/8242/8242_3.PDF

[10] "Gazebo.Sim.Org" [Online] Disponível: http://gazebosim.org/

[11] "TurtleBot3 Waffle Pi" [Online]. Disponível: https://www.robotis.us/turtlebot-3-waffle-pi

[12] "Robotics" [Online]. Disponível: https://emanual.robotis.com/assets/images/platform/turtlebot3/simulation/turtlebot3_world_waffle.png

[13] "RasperryPi.org" [Online]. Disponível: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[14] "WikiRos.org" [Online]. Disponível: http://wiki.ros.org/ROS/Introduction

[15] "GazeboSim.org Tutorials – Building Editor" [Online]. Disponível: http://gazebosim.org/tutorials?cat=build_world&tut=building_editor

[16] "QR Code Reader – BachaSoft" [Online] Disponível: https://play.google.com/store/apps/details?id=com.apple.qrcode.reader&hl=en_US&gl=US

[17] "GitHub – Mikael Arguedas" [Online] Disponível: https://github.com/mikaelarguedas/gazebo_models

[18] "OpenCV – Python Tutorials" [Online] Disponível: https://opencv-python-tutroals.readthedocs.io/en/latest/index.html

[19] "Pypi.org – Pyzbar" [Online] Disponível: https://pypi.org/project/pyzbar/

[20] "Ros.org – Navigation" [Online] Disponível: http://wiki.ros.org/navigation

[21] "Wikipedia – YAML" [Online] Disponível: https://pt.wikipedia.org/wiki/YAML

[22] "Pypi.org – PyYAML" [Online] Disponível: https://pypi.org/project/PyYAML/

[23] "WikiRos.org – Rviz" [Online] Disponível: http://wiki.ros.org/rviz

[24] "WikiRos.org - AMCL" [Online] Disponível: https://wiki.ros.org/amcl

[25] "WikiRos.org - CVBridge" [Online] Disponível: http://wiki.ros.org/cv_bridge

[26] "WikiRos.org – RQT Graph" [Online] Disponível: http://wiki.ros.org/rqt_graph

[27] "Github – FPSO Autonomous Vehicle" [Online] Disponível: https://github.com/anacletogiovanna/fpso-autonomous-vehicle

[28] "GitHub" [Online] Disponível: https://docs.github.com/pt/github/getting-started-with-github

[29] "Uma arquitetura de controle para um robô quadrúpede com comportamento reflexivo de estabilidade" [Online] Disponível: http://repositorio.ufes.br/handle/10/4100

[30] "ABNT" [Onine] Disponível: https://www.abntcatalogo.com.br/norma.aspx?ID=456844