

Sistema FIEB



PELO FUTURO DA INOVAÇÃO

CENTRO UNIVERSITÁRIO SENAI CIMATEC

Relatório dos desafios

Desafios: Webots, Turtlesim, Husky, CPP e Python

Apresentada por: Marcella Giovanna Silva dos Santos

Orientador: Prof. Marco Reis, M.Eng.

novembro 2021

Marcella Giovanna Silva dos Santos

Desafios: Webots, Turtlesim, Husky, CPP e Python

Salvador
Centro Universitário SENAI CIMATEC
2020

Resumo

Com o intuito de aperfeiçoar e aprender alguns conceitos aplicados na robótica, foi realizados cinco desafios além de todos os tutoriais de ros, os quais utilizam diferentes aplicações como: simulação de robôs, ROS, Webots e linguagens de programação (python e c++). Para que dessa forma seja possível a atuação no laboratório.

Palavras-chave: Conceitos, robótica, simulação, ROS, programação.

Abstract

In order to improve and learn some basic concepts in robotics, five challenges were carried out in addition to all the tutorials of ros, they use different applications such as: training simulation, ROS, Webots and programming languages(python and c ++). For what in this way it is possible to work in the laboratory

Keywords: Concepts, robotics, simulation, ROS, programming.

Introdução

1.0.1 Webots

Webots é um aplicativo de desktop de código aberto e multiplataforma usado para simular robôs. Ele fornece um ambiente de desenvolvimento completo para modelar, programar e simular robôs. Ele foi projetado para um uso profissional e é amplamente utilizado na indústria, educação e pesquisa. Cyberbotics Ltd. mantém Webots como seu produto principal continuamente desde 1998.

1.0.2 Turtlesim SETPOINT POSITION

Uma maneira simples de aprender o básico de ROS é usando o simulador turtlesim. Que consiste na simulação de uma janela grafica que mostra um robô com formato de tartaruga. Esta tartaruga pode ser movida por toda a janela utilizando comandos de ROS como `roscore`, `roscum...` ou utilizando o teclado/joystick.

1.0.3 Husky

O Husky é um veículo terrestre não tripulado (UGV) robusto, pronto para uso ao ar livre, adequado para pesquisas e aplicações de prototipagem rápida e oferece suporte total a ROS.

1.0.4 CPP workbook

O C++ é uma linguagem de programação de nível médio, baseada na linguagem C. Tem uma enorme variedade de códigos, pois além de seus códigos, pode contar com vários da linguagem C. Esta variedade possibilita a programação em alto e baixo níveis.

1.0.5 *Python workbook1 e 2*

Python é uma linguagem de programação de alto nível, ou seja, com sintaxe mais simplificada e próxima da linguagem humana, utilizada nas mais diversas aplicações, como desktop, web, servidores e ciência de dados.

1.1 *Objetivos*

- Webots: Desenvolver um sistema de navegação autônoma, de forma que o robô consiga chegar á região iluminada do mapa pré-definido, evitando todos os obstáculos do percurso em 2 minutos.
- Turtlesim: Fazer um controlador de posição para a tartaruga que da um ponto (X, Y) a tartaruga deve atingir o ponto com certo erro associado.
- Husky: Simular a navegação do robô Husky como está descrito no tutorial do ROS.
- CPP workbook: Demonstrar os conhecimentos adquiridos nos tutoriais de c++.
- Python workbook1 e 2: Fornecer uma oportunidade para que todos evoluam no conceito de codificação.

1.1.1 *Objetivos Específicos*

1.1.2 *Webots*

Os objetivos específicos deste desafio são:

- Fazer navegação do PIONEER;
- Utilizar o repositório <https://github.com/Brazilian-Institute-of-Robotics/desafiorobotica.git>;
- Utilizar sensor de luminosidade;
- Simular no webots;

1.1.3 *Turtlesim SETPOINT POSITION*

Os objetivos específicos deste desafio são:

- A posição do ponto de ajuste deve ser definida por meio de linha de comando;
- A escolha da linguagem de programação é Python;
- O código deve estar disponível em um repositório GitHub;
- O erro aceito é em torno de 0,1;

1.1.4 *Husky*

Os objetivos específicos deste desafio são:

- Fazer o tutorial Husky;
- Utilizar os pacotes do Husky;
- Simular a navegação nos 4 modos;
- Simular usando o gazebo;

1.1.5 *CPP workbook*

Os objetivos específicos deste desafio são:

- Fazer os desafios propostos;
- Utilizar C++ para a resolução;
- O código deve estar disponível em um repositório GitHub;

1.1.6 *Python workbook1 e 2*

Os objetivos específicos deste desafio são:

- Fazer os desafios propostos;
- Utilizar Python para a resolução;
- Cada exercício deve ser preparado na menor quantidade de linhas possível;
- Os comentários são necessários para você explicar cada etapa do código;

1.2 Justificativa

Demonstrar os conhecimentos adquiridos dos topicos de Webots,ROS,CPP e python durante o processo de todos os desafios, a fim de ter um ótimo desempenho nas atividades do Laboratório.

1.3 Organização do documento

Este documento apresenta 5 capítulos e está estruturado da seguinte forma:

- **Capítulo 1 - Introdução:** Contextualiza o âmbito, no qual os desafios estão inseridos. Apresenta, portanto, a definição dos desafios, objetivos e justificativas do mesmo, além de como este relatório dos desafios está estruturado;
- **Capítulo 2 - Fundamentação Teórica:** As teorias usadas em cada um dos desafios;
- **Capítulo 3 - Materiais e Métodos:** De que forma os desafios foram feitos;
- **Capítulo 4 - Resultados:** Os resultados obtidos em cada um dos desafios propostos;
- **Capítulo 5 - Conclusão:** Apresenta as conclusões.

Conceito do projeto

2.1 *Webots*

O software Projeta facilmente simulações robóticas completas usando a biblioteca de ativos Webots que inclui robôs, sensores, atuadores, objetos e materiais.

2.1.1 *PIONEER*

Veículo terrestre não tripulado (VTNT) da empresa Adept MobileRobots. São equipados com sonares e encoders, possibilitando, por exemplo, o mapeamento de um ambiente desconhecido.

Figura 2.1: PIONEER

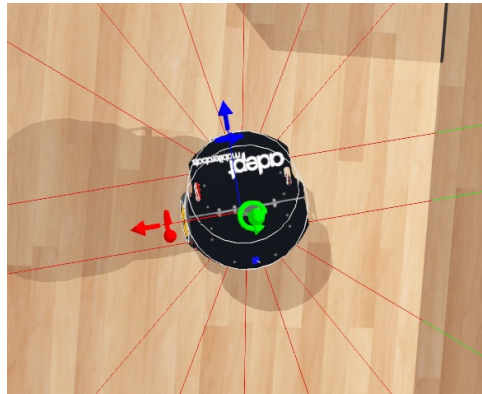


Fonte: robotica.ufv.br/laboratorios/.

2.1.2 *Sensor de distância*

O sensor de distância, é um tipo de sensor que mede a distância entre o robô e um objeto desejado. Abaixo pode ser visto 16 feixes que representam as posições dos sensores de distância do Pioneer no Webots.

Figura 2.2: Representação do sensor de distância

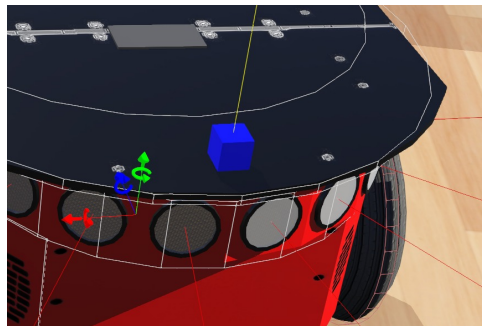


Fonte: Própria.

2.1.3 Sensor de Luminosidade

sensor é medir a intensidade de luz do ambiente ao seu redor, variando o estado de sua saída digital caso detectado um determinado nível de luminosidade. Abaixo pode ser visto um cubo amarelo que representa o sensor luminosidade adicionado no Pioneer.

Figura 2.3: Representação do sensor de luminosidade



Fonte: Própria.

2.2 Turtlesim

2.2.1 Nós ROS

Um nó é apenas um arquivo executável dentro de um pacote do ROS. Nós no ROS usam bibliotecas clientes para se comunicar com outros nós. Nós podem publicar e/ou subscrever em tópicos, além de poderem prover serviços também.

2.2.2 tópicos ROS

Nós podem publicar mensagens em tópicos assim como podem subescrever mensagens de tópicos. Como é o exemplo dos nós turtlesim e turtle-teleop-key que se comunicam entre si através do tópico turtle1command-velocity turtle-teleop-key está publicando a tecla digitada no tópico, enquanto o turtlesim subescreve o mesmo tópico para receber a tecla digitada. A imagem abaixo mostra os nós turtlesim e teleop, e o tópico command-velocity que faz a comunicação entre os dois nós.

Figura 2.4: nós e tópicos ROS

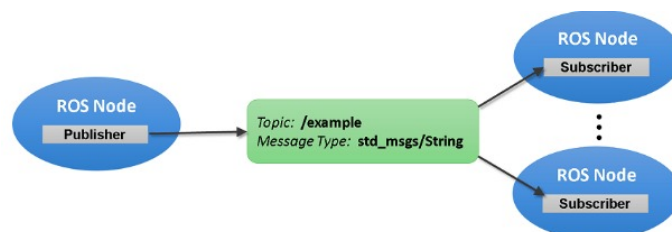


Fonte: Tutorials-UnderstandingTopics

2.2.3 Publisher e Subscribers

Publisher publica as informações ou mensagens no tópico e o subscriber subescreve o tópico e recebe as informações publicadas. As mensagens são transmitidas em um tópico e cada tópico possui um nome exclusivo na rede ROS. Se um nó deseja compartilhar informações, ele usa publisher para enviar dados a um tópico. Um nó que deseja receber essas informações usa subscriber desse mesmo tópico. Além de seu nome exclusivo, cada tópico também possui um tipo de mensagem, que determina os tipos de mensagens que podem ser transmitidas naquele tópico. O conceito de publisher e subscriber pode ser melhor visto na figura abaixo.

Figura 2.5: publisher e subscriber ROS

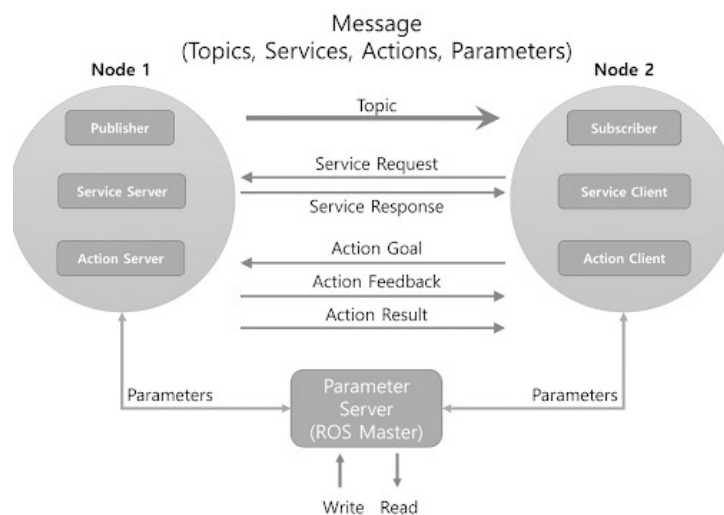


Fonte: la.mathworks.com/help//ros/ug/exchange-data-with-ros-publishers-and-subscribers.html

2.2.4 Services e messages

Os nós comunicam-se entre si publicando mensagens em tópicos . Uma mensagem é uma estrutura de dados simples, composta por campos digitados. Os nós também podem trocar uma mensagem de solicitação e resposta como parte de uma chamada de serviço ROS . O modelo publisher e subscriber é um paradigma de comunicação muito flexível, mas seu transporte unilateral para muitos não é apropriado para interações de solicitação e resposta RPC, que geralmente são necessárias em um sistema distribuído. A solicitação e resposta é feita por meio de um Serviço, que é definido por um par de mensagens : uma para a solicitação e outra para a resposta.

Figura 2.6: services e messages ROS



Fonte: robinrobotic.blogspot.com/2019/06/ros-terminology.html

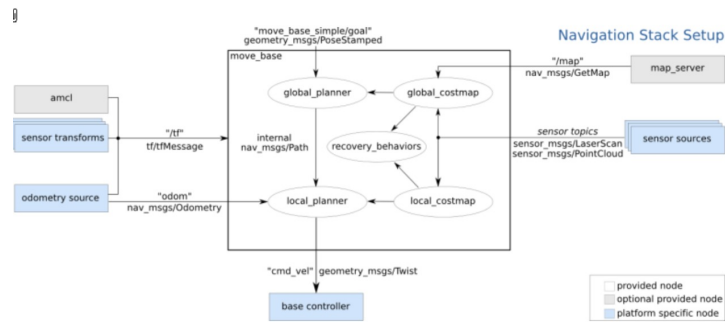
2.3 Husky

2.3.1 Move Base Demo

Realiza o planejamento autônomo básico e movimento no Husky com um scanner a laser publicando no tópico de digitalização. O nó move-base fornece uma interface ROS para configurar, executar e interagir com a pilha de navegação em um robô.

A pilha de navegação recebe informações de odometria e fluxos de sensores e emite comandos de velocidade para enviar ao robô.

Figura 2.7: move base



Fonte:

Por padrão, o nó move-base executará as seguintes ações para tentar limpar o espaço:

Primeiro, os obstáculos fora de uma região especificada pelo usuário serão removidos do mapa do robô. Em seguida, se possível, o robô executará uma rotação no local para liberar espaço. Se isso também falhar, o robô limpará seu mapa de forma mais agressiva, removendo todos os obstáculos fora da região retangular na qual ele pode girar no lugar. Isso será seguido por outra rotação no local. Se tudo isso falhar, o robô considerará seu objetivo inviável e notificará o usuário de que ele foi abortado

2.3.2 AMCL Demo

Realiza o planejamento autônomo e movimento com localização em um Husky simulado com um scanner a laser publicando no tópico de digitalização. amcl é um sistema de localização probabilística para um robô se movendo em 2D. Ele implementa a abordagem de localização adaptativa que usa um filtro de partículas para rastrear a posição de um robô em relação a um mapa conhecido.

2.3.3 Gmapping

realizar planejamento autônomo e movimento com localização e mapeamento simultâneo (SLAM), em um Husky simulado com um scanner a laser publicando no tópico de digitalização.

O pacote gmapping fornece SLAM baseado em laser, como um nó ROS chamado slam-gmapping. Usando slam-gmapping, você pode criar um mapa de grade de ocupação 2-D a partir de dados de laser e pose coletados por um robô móvel.

2.3.4 *Frontier Exploration*

Planejamento de exploração e gmapping para mapeamento e localização (SLAM).

Implementação de exploração de fronteira para ROS, estendendo-se na pilha de navegação existente. Ele aceita objetivos de exploração via actionlib, envia comandos de movimento para move-base.

2.4 *CPP workbook*

2.5 *Python workbook1 e 2*

Desenvolvimento do projeto

Nesta seção será descrito o procedimento utilizado para construção de cada um dos desafios.

3.1 *Webots*

O robô utilizado foi o Pioneer o qual usa seus 16 sensores de distância pré-instalados para obter informações do mundo em todas as direções e um sensor adicional: o sensor de luminosidade que rastreia a irradiância local e envia um sinal para o robô quando ele lê mais de $750 \text{ W} / \text{m}^2$, o que significa que está perto o suficiente da luminária de chão para disparar o STOP.

3.1.1 *controle*

A navegação do robô pelo mapa é baseada em uma máquina de 4 estados que determina se ele deve se mover para frente, virar à esquerda, direita ou parar quando atingir seu objetivo final.

Assim, a divisão foi feita em quatro casos, são eles:

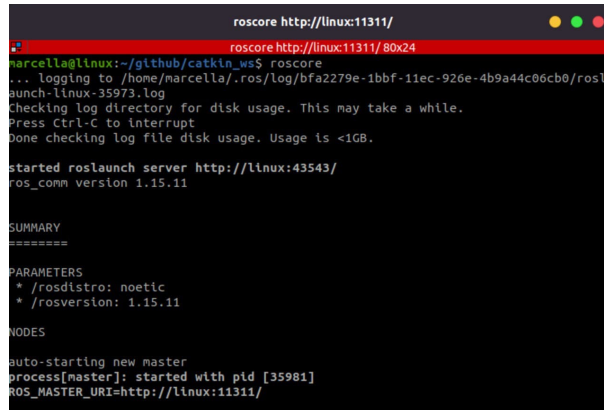
- FORWARD: Anda para frente e se houver algum obstáculo à frente ele começa a tomar a decisão de virar em qualquer direção para evitá-lo.
- ESQUERDA: Vire à esquerda até que a detecção de objetos não seja mais possível;
- DIREITA: Vire à direita até que a detecção de objetos não seja mais possível;
- STOP: Quando o sensor de luz detecta a quantidade de luminosidade configurada (neste caso $750 \text{ W} / \text{m}^2$), o robô deve parar.

Todos os sensores de distância coletam dados do ambiente que determinam se o robô deve se mover para frente ou para os lados.

3.2 Turtlesim

Para que o desafio seja cumprido é necessário rodar o roscore e o nó para que a janela da turtle apareça na tela.

Figura 3.1: inicializando o ros



```

roscore http://linux:11311/
marcella@linux:~/github/catkin_ws$ roscore
... logging to /home/marcella/.ros/log/bfa2279e-1bbf-11ec-926e-4b9a44c06cb0/ros1
launch-linux-35973.log
checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://linux:43543/
ros_comm version 1.15.11

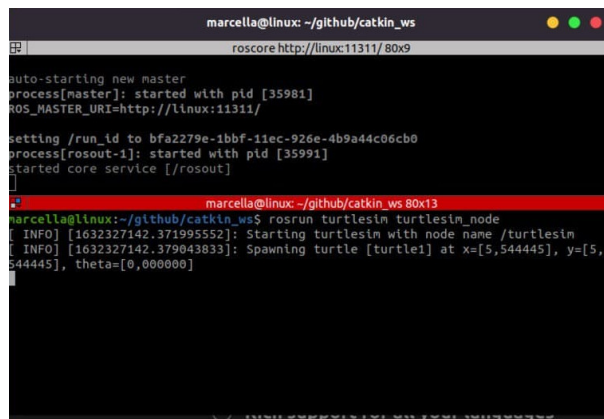
SUMMARY
=====
PARAMETERS
 * /roscore: noetic
 * /rosversion: 1.15.11

NODES
auto-starting new master
process[master]: started with pid [35981]
ROS_MASTER_URI=http://linux:11311/

```

Fonte: Autoria própria.

Figura 3.2: Rodando o nó do turtlesim



```

marcella@linux:~/github/catkin_ws
roscore http://linux:11311/ 80x9
auto-starting new master
process[master]: started with pid [35981]
ROS_MASTER_URI=http://linux:11311/

setting /run_id to bfa2279e-1bbf-11ec-926e-4b9a44c06cb0
process[rosout-1]: started with pid [35991]
started core service [/rosout]

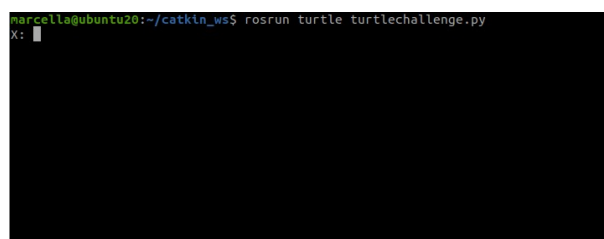
marcella@linux:~/github/catkin_ws 80x13
marcella@linux:~/github/catkin_ws$ roslaunch turtlesim turtlesim_node
[INFO] [1632327142.371995552]: Starting turtlesim with node name /turtlesim
[INFO] [1632327142.379043833]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]

```

Fonte: Autoria própria.

por fim executa-se o arquivo .py que foi a linguagem escolhida para completar este desafio, como mostra a figura abaixo. É posto as coordenadas x e y no caso do exemplo do desafio

Figura 3.3: Executando arquivo



```

marcella@ubuntu20:~/catkin_ws$ roslaunch turtlesim turtlesim_node
X:

```

Fonte: Autoria própria.

é 1 e 1 e a tartaruga deve ir até essa coordenada quando o erro for no máximo 0.1.

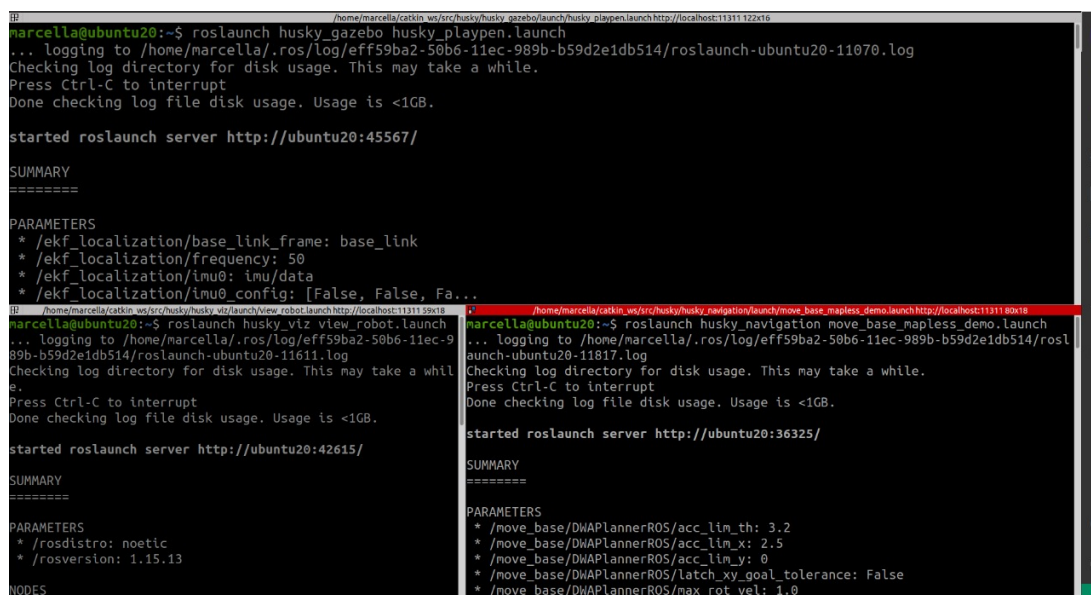
3.3 Husky

3.3.1 Move Base

Após toda a configuração e instalação dos pacotes do Husky encontrado no repositório <https://github.com/husky/husky>, foi executado o nó move-base no qual será enviado um comando para o robô Husky que tentará atingir a posição enviada desviando dos obstáculos e caso entre em alguma posição em que se esteja travado executa comportamentos de recuperação para continuar o trajeto enviado.

Para a execução do nó move-base é necessário três comandos o primeiro inicia o ambiente de simulação o gazebo, o segundo o visualizador rviz e o terceiro a demonstração move-base.

Figura 3.4: Comandos move base



```

marcella@ubuntu20:~$ roslaunch husky_gazebo husky_playpen.launch
... logging to /home/marcella/.ros/log/eff59ba2-50b6-11ec-989b-b59d2e1db514/roslaunch-ubuntu20-11070.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu20:45567/

SUMMARY
=====
PARAMETERS
 * /ekf_localization/base_link_frame: base_link
 * /ekf_localization/frequency: 50
 * /ekf_localization/imu0: imu/data
 * /ekf_localization/imu0_config: [False, False, Fa...

marcella@ubuntu20:~$ roslaunch husky_viz view_robot.launch
... logging to /home/marcella/.ros/log/eff59ba2-50b6-11ec-989b-b59d2e1db514/roslaunch-ubuntu20-11611.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu20:42615/

SUMMARY
=====
PARAMETERS
 * /roscdistro: noetic
 * /rosversion: 1.15.13
NODES

marcella@ubuntu20:~$ roslaunch husky_navigation move_base_mapless_demo.launch
... logging to /home/marcella/.ros/log/eff59ba2-50b6-11ec-989b-b59d2e1db514/roslaunch-ubuntu20-11817.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu20:36325/

SUMMARY
=====
PARAMETERS
 * /move_base/DWAPlannerROS/acc_lim_th: 3.2
 * /move_base/DWAPlannerROS/acc_lim_x: 2.5
 * /move_base/DWAPlannerROS/acc_lim_y: 0
 * /move_base/DWAPlannerROS/latch_xy_goal_tolerance: False
 * /move_base/DWAPlannerROS/max_rot_vel: 1.0

```

Fonte: Autoria própria.

3.3.2 AMCL

O Tutorial amcl do Husky mostra como é usado o move-base com o sendo assim, amcl obtém um mapa baseado em laser que precisa ser ativado no description do robô encontrado

no próprio repositório após a ativação o laser faz varreduras e transforma em mensagens que geram estimativas de posição.

Figura 3.5: Comandos amcl

```

marcella@ubuntu20:~$ roslaunch husky_gazebo husky_playpen.launch
... logging to /home/marcella/.ros/log/8610be8e-50ba-11ec-989b-b59d2e1db514/roslaunch-ubuntu20-21318.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu20:37399/

SUMMARY
=====

PARAMETERS
 * /ekf_localization/base_link_frame: base_link
 * /ekf_localization/frequency: 50
 * /ekf_localization/imu0: imu/data

marcella@ubuntu20:~$ roslaunch husky_viz view_robot.launch
... logging to /home/marcella/.ros/log/8610be8e-50ba-11ec-989b-b59d2e1db514/roslaunch-ubuntu20-21765.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu20:38437/

SUMMARY
=====

PARAMETERS
 * /amcl/gui_publish_rate: 10.0
 * /amcl/kld_err: 0.85

```

Fonte: Autoria própria.

3.3.3 gmapping

Após a execução do nó slam-gmapping será levado para o tópico sensor-msgs/LaserScan mensagens e constrói um mapa a partir dos dados de laser e posições coletados pelo husky.

Figura 3.6: Comandos gmapping

```

marcella@ubuntu20:~$ roslaunch husky_gazebo husky_playpen.launch
... logging to /home/marcella/.ros/log/13843f46-50bd-11ec-989b-b59d2e1db514/roslaunch-ubuntu20-28560.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu20:44729/

SUMMARY
=====

PARAMETERS
 * /ekf_localization/base_link_frame: base_link
 * /ekf_localization/frequency: 50
 * /ekf_localization/imu0: imu/data
 * /ekf_localization/imu0_config: [False, False, Fa...
 * /ekf_localization/imu0_differential: True

marcella@ubuntu20:~$ roslaunch husky_viz view_robot.launch
... logging to /home/marcella/.ros/log/13843f46-50bd-11ec-989b-b59d2e1db514/roslaunch-ubuntu20-29041.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu20:40995/

SUMMARY
=====

PARAMETERS
 * /roslistro: noetic
 * /rosversion: 1.15.13

NODES

```

```

marcella@ubuntu20:~$ roslaunch husky_navigation gmapping_demo.launch
... logging to /home/marcella/.ros/log/13843f46-50bd-11ec-989b-b59d2e1db514/roslaunch-ubuntu20-29262.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu20:37641/

SUMMARY
=====

PARAMETERS
 * /move_base/DWAPlannerROS/acc_lin_th: 3.2
 * /move_base/DWAPlannerROS/acc_lin_x: 2.5
 * /move_base/DWAPlannerROS/acc_lin_y: 0

```

Fonte: Autoria própria.

3.3.4 Frontier-exploration

O Frontier-exploration pacote fornece um costmap-2d que fornece uma implementação de um mapa de custo 2D que leva em dados de sensor do mundo, constrói uma grade de ocupação 2D ou 3D dos dados, e actionlib cliente que fornece uma interface padronizada com tarefas preemptivas

Figura 3.7: Comandos frontier-exploration

The figure consists of three terminal screenshots. The top screenshot shows the command `roslaunch husky_gazebo husky_playpen.launch` being executed, with output indicating logging to a specific directory and checking disk usage. The bottom-left screenshot shows `roslaunch husky_viz view_robot.launch` being executed, with similar logging output. The bottom-right screenshot shows `roslaunch husky_navigation exploration_demo.launch` being executed, with output showing parameters for the move_base/DWAPlannerROS node, including `acc_lim_th: 3.2`, `acc_lim_x: 2.5`, and `acc_lim_y: 0`.

Fonte: Autoria própria.

3.4 CPP

3.5 Python

Resultados

Os resultados obtidos nos desafios foram:

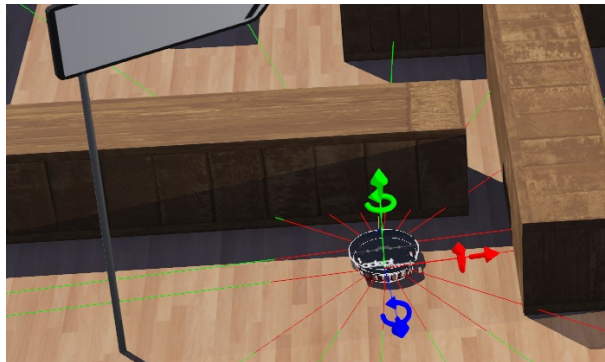
4.1 *Webots*

Cada um dos 16 sensores de distância incluídos no Pioneer são encontrados no arquivo de controle do robô `challengecontroller.c` e contêm valores de peso para cada roda, que são baseados na posição do sensor no robô e mostram como sua leitura influencia o comportamento do duas rodas. Além disso, o valor do peso auxilia na tomada de decisão, pois, se a leitura de algum sensor cair abaixo do valor mínimo definido, significa que o robô está próximo a um obstáculo e deve mudar de direção usando um dos casos acima. No final, ele foi usado para determinar o comportamento do robô em todos os valores de peso em cada roda vezes um fator de velocidade, que é dependente tanto da relação de leitura do sensor quanto do valor de distância mínima.

Também faz parte do controle o sensor de luz que foi adicionado para completar o desafio, este sensor ativa o estado STOP quando detecta um brilho maior que 750 W / m^2 e assim o desafio é concluído.

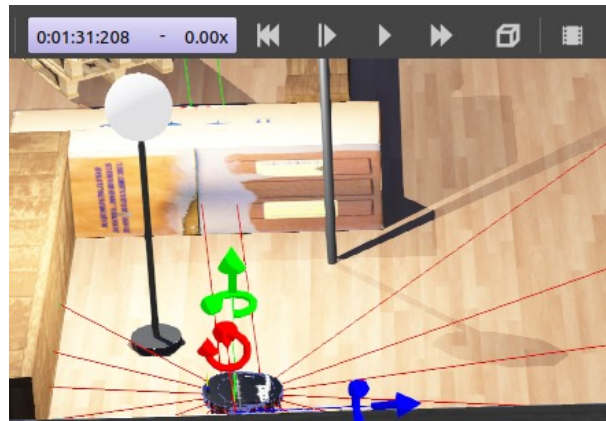
As fotos mostram o pioneer iniciando, chegando na área luminosa e parando no tempo correto, completando assim o desafio proposto, para uma melhor visualização da solução tem um link do video logo abaixo e o controle pode ser encontrado no repositório <https://github.com/marcellabecker/desafiorobotica>

Figura 4.1: Pioneer iniciando o percurso



Fonte:Própria.

Figura 4.2: Pioneer completando o percurso



Fonte:Própria.

Link para assistir ao vídeo:<https://www.youtube.com/watch?v=RftRevxZUOE>

4.2 Turtlesim SETPOINT

Inicialmente no código foi feita uma função de callback que é chamada quando o subscriber recebe um dado do tópico. Logo depois o nó é iniciado, para que possa ser feito os publishers e subscribers no tópicos, sendo assim vel-publisher publica mensagens no tópico cmd-vel e posecb recebe a msg de coordenada da posição.

Goal-x e goal-y recebem os inputs das coordenadas x e y para onde a tartaruga deve ir.

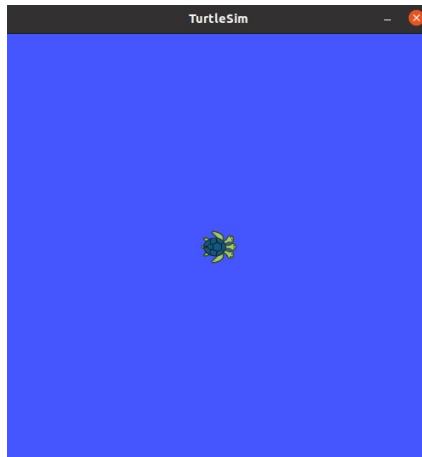
kp-lin e kp-ang são os controles de velocidade linear e angular definidos para que a tartaruga não ande muito rápido nem muito devagar e controla o seu ângulo de curvatura durante o movimento.

A FUNÇÃO is-shutdown() verifica se seu programa deve sair, portanto enquanto não entrar no if que está dentro do while significa que o erro ainda é maior que 0.1 e a turtle ainda não chegou ao seu destino sendo assim o programa deve continuar rodando, até que o erro seja 0.1 como manda o desafio e ela deve parar.

As figuras abaixo mostram a janela da turtle antes das coordenadas e depois dela andar para a coordenada (1,1).

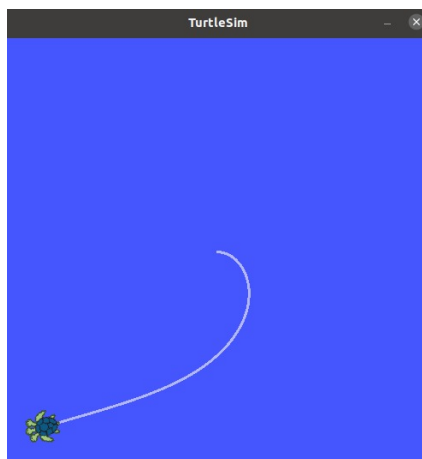
Para uma melhor visualização da solução tem um link do vídeo logo abaixo e o controle pode ser encontrado no repositório <https://github.com/marcellabecker/turtle>

Figura 4.3: janela da turtle



Fonte:Própria.

Figura 4.4: Turtle chegou ao destino



Fonte:Própria.

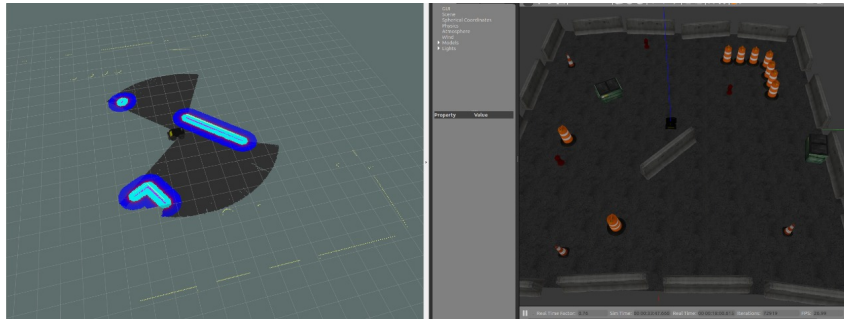
Link para assistir ao vídeo:<https://youtu.be/NNkwSgDSuY>

4.3 Husky

4.3.1 Move-base

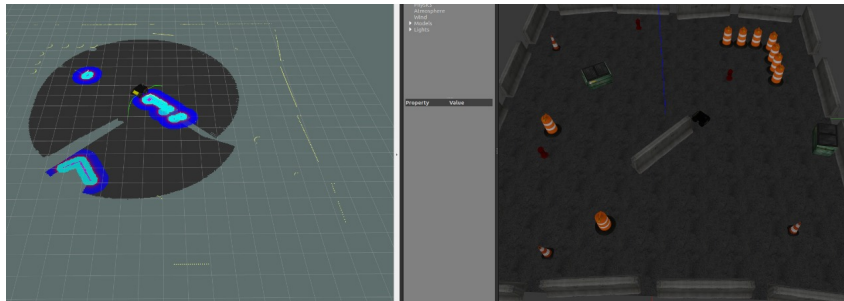
Com todos os comando de movimentação já configurados e as janelas rviz e gazebo ja abertas a movimentação é feita pelo mouse podendo também usar o teclado pelo comando teleop-twist-keyboard ou por controle remoto. As imagens abaixo mostram antes, durante e depois que a movimentação for concluida.

Figura 4.5: Husky antes do movimento



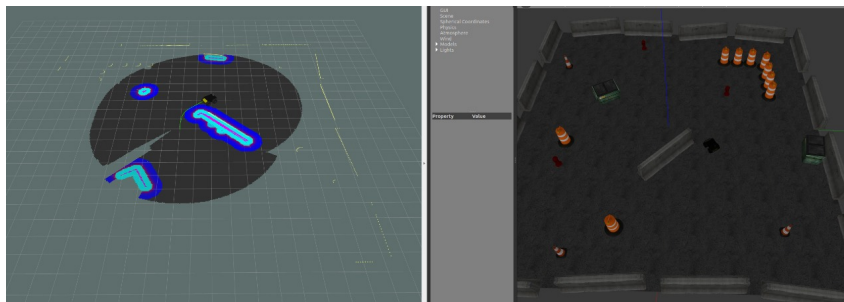
Fonte:Própria.

Figura 4.6: Husky durante o movimento



Fonte:Própria.

Figura 4.7: Husky depois o movimento

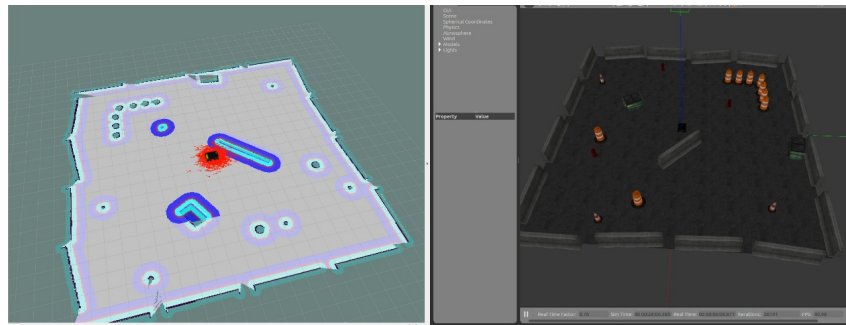


Fonte:Própria.

4.3.2 AMCL

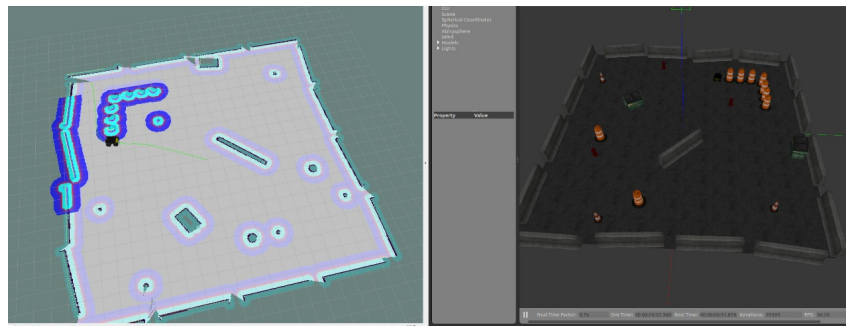
Com todos os comando de movimentação já configurados e as janelas rviz e gazebo ja abertas a movimentação é feita pelo mouse podendo também usar o teclado pelo comando teleop-twist-keyboard ou por controle remoto. As imagens abaixo mostram antes, durante e depois que a movimentação for concluída. É possível ver que diferente do move-base o Husky consegue ter um alcance bem maior de visualização, o todos formados pelo mapa e o laiser destaca os que estão mais perto.

Figura 4.8: Husky antes do movimento



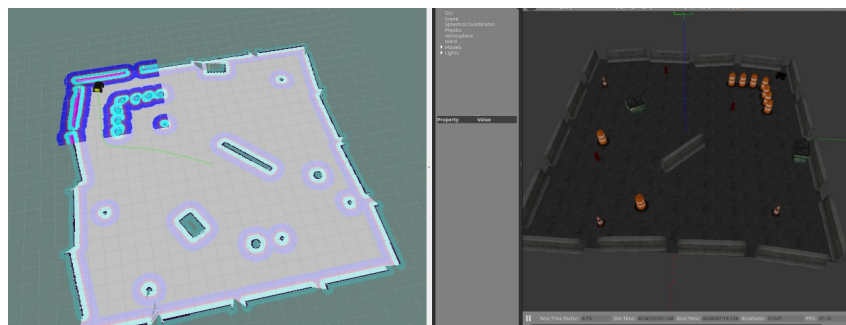
Fonte:Própria.

Figura 4.9: Husky durante o movimento



Fonte:Própria.

Figura 4.10: Husky depois o movimento



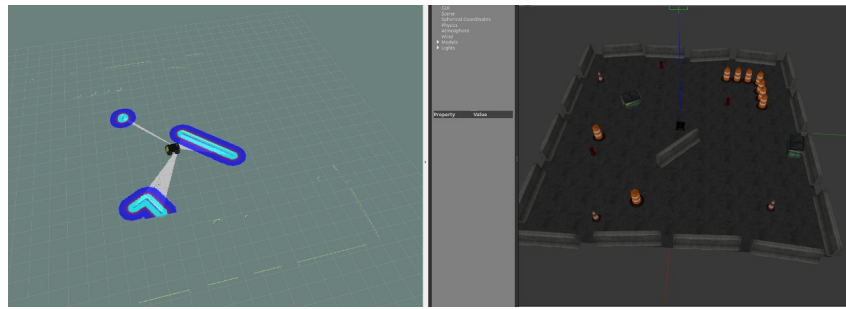
Fonte:Própria.

4.3.3 Gmapping

Com todos os comando de movimentação já configurados e as janelas rviz e gazebo ja abertas a movimentação é feita pelo mouse podendo também usar o teclado pelo comando teleop-twist-keyboard ou por controle remoto.

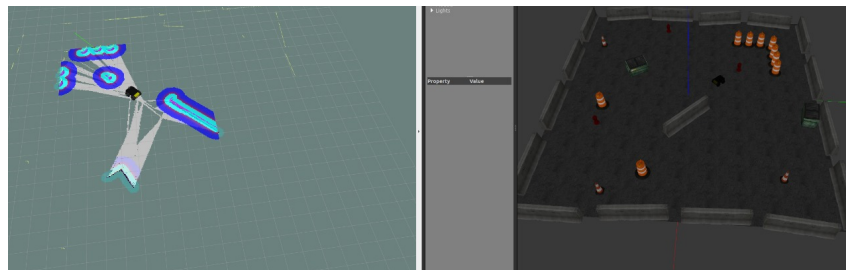
As imagens abaixo mostram antes, durante e depois que a movimentação for concluída.

Figura 4.11: Husky antes do movimento



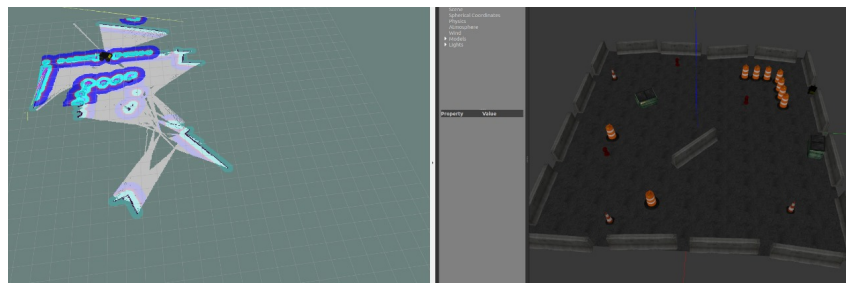
Fonte:Própria.

Figura 4.12: Husky durante o movimento



Fonte:Própria.

Figura 4.13: Husky depois o movimento



Fonte:Própria.

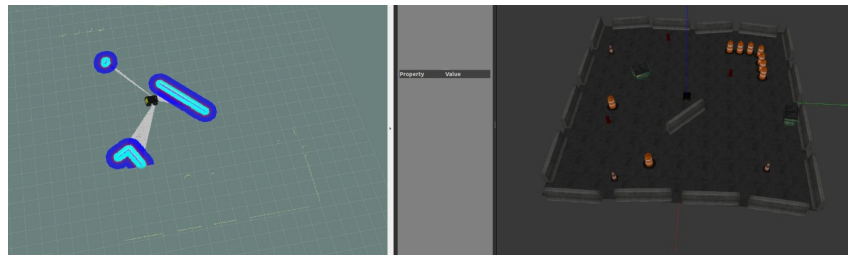
A diferença do Gmapping é o SLAM que localiza e mapeia o ambiente simultaneamente, como pode ser visto o Husky enquanto está se localizando para andar no mapa, também constrói o mapa.

4.3.4 *Frontier-exploration*

Com todos os comando de movimentação já configurados e as janelas rviz e gazebo já abertas a movimentação é feita pelo mouse podendo também usar o teclado pelo comando teleop-twist-keyboard ou por controle remoto.

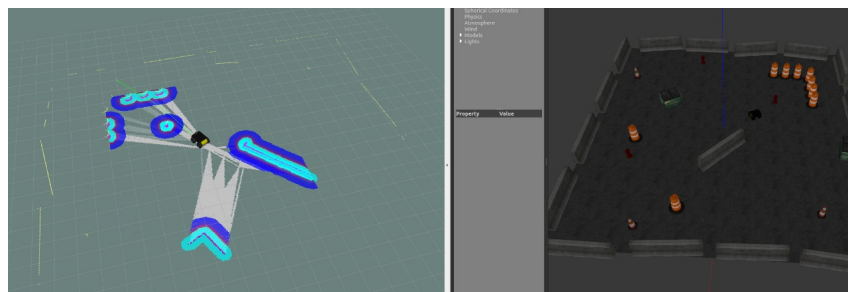
As imagens abaixo mostram antes, durante e depois que a movimentação for concluída.

Figura 4.14: Husky antes do movimento



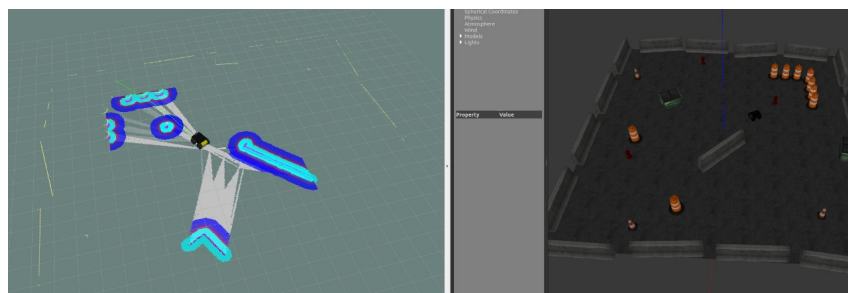
Fonte:Própria.

Figura 4.15: Husky durante o movimento



Fonte:Própria.

Figura 4.16: Husky depois o movimento



Fonte:Própria.

Conforme o robô se move, você deve ver o mapa estático cinza crescer. Ocasionalmente, o algoritmo de gmapping irá realocar o robô, causando um salto discreto no mapa. Quando a meta de exploração for concluída, você verá uma mensagem escrito DONE na janela do terminal.

4.4 *cpp*

4.5 *python*

Conclusão

Chegou a hora de apresentar o apanhado geral sobre o trabalho de pesquisa feito, no qual são sintetizadas uma série de reflexões sobre a metodologia usada, sobre os achados e resultados obtidos, sobre a confirmação ou rechaço da hipótese estabelecida e sobre outros aspectos da pesquisa que são importantes para validar o trabalho. Recomenda-se não citar outros autores, pois a conclusão é do pesquisador. Porém, caso necessário, convém citá-lo(s) nesta parte e não na seção seguinte chamada **Conclusões**.

5.1 Considerações finais

Brevemente comentada no texto acima, nesta seção o pesquisador (i.e. autor principal do trabalho científico) deve apresentar sua opinião com respeito à pesquisa e suas implicações. Descrever os impactos (i.e. tecnológicos, sociais, econômicos, culturais, ambientais, políticos, etc.) que a pesquisa causa. Não se recomenda citar outros autores.

Referências

Desafios: Webots, Turtlesim, Husky, CPP e Python

Marcella Giovanna Silva dos Santos

Salvador, novembro 2021.