PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS
ESCOLA POLITÉCNICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

RELATÓRIO TÉCNICO: DESENHO DA LETRA “F” UTILIZANDO ROS 2 E TURTLESIM

Felipe Grolla Freitas — RA: 24004846

Campinas — Sp 2024

1. Introdução

Este relatório visa descrever, de forma técnica e detalhada, o processo de utilização do ROS 2 (Robot Operating System) e do nó turtlesim para desenhar a letra “F” em um ambiente simulado. Através deste experimento, foi possível compreender o funcionamento de nós, tópicos e mensagens no ROS 2, bem como a interação entre esses elementos para controlar a movimentação de um robô (tartaruga) em um plano 2D.

2. Configuração do Ambiente

O ambiente de simulação foi configurado utilizando o pacote turtlesim, sendo uma ferramenta básica do ROS 2 para simulação de robótica. Para realizar o experimento, os seguintes passos foram executados:

2.1. Iniciar o ROS 2: Certifique-se de que o ambiente do ROS 2 está corretamente configurado.

— *Comando para verificar a versão do ROS 2*:

```
ros2 --version
```

2.2. Iniciar o nó turtlesim: O nó principal utilizado é o turtlesim_node, responsável por inicializar a simulação da tartaruga.

— *Comando para iniciar o nó*:

```
ros2 run turtlesim turtlesim_node
```

A partir desse comando, a janela do turtlesim é aberta, mostrando a tartaruga no centro do plano 2D.

2.3. Controlar a tartaruga: para controlar a tartaruga e desenhar a letra “F”, utilizamos o serviço de publicação de mensagens no tópico /turtle1/cmd_vel. Esse tópico recebe mensagens do tipo geometry_msgs/msg/Twist, que definem a velocidade linear e angular da tartaruga.

3. Criação de Nós e Tópicos

Nó `turtlesim_node`:

— Responsável por controlar a movimentação da tartaruga na tela.

— Assina o tópico `/turtle1/cmd_vel` para receber comandos de velocidade.

Nó `teleop` (ou equivalente):

— Captura os comandos do teclado e os converte em comandos de movimento.

— Publica esses comandos no tópico `/turtle1/cmd_vel`.

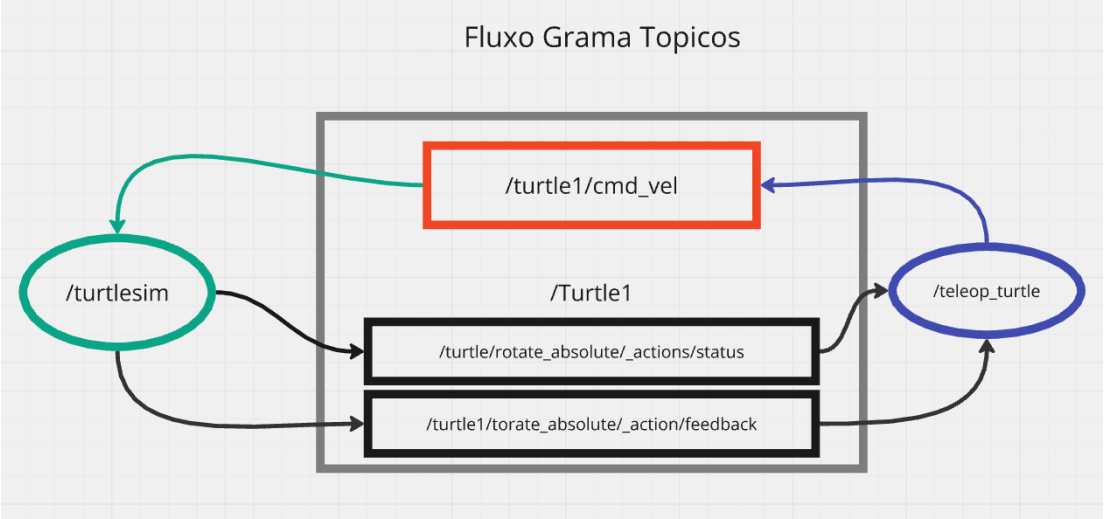
Tópico `/turtle1/cmd_vel`:

— O nó `teleop` publica comandos de velocidade neste tópico, enquanto o `turtlesim_node` assina para receber as mensagens e movimentar a tartaruga.

Interação entre Nós e Tópicos:

— Nó `teleop`: publica comandos de velocidade linear e angular no tópico `/turtle1/cmd_vel` com base nas teclas pressionadas pelo usuário.

— Nó `turtlesim_node`: recebe as mensagens publicadas no tópico `/turtle1/cmd_vel`, processa os comandos e movimenta a tartaruga na tela conforme as velocidades recebidas.



4. Serviços

Funcionamento dos Serviços no ROS 2

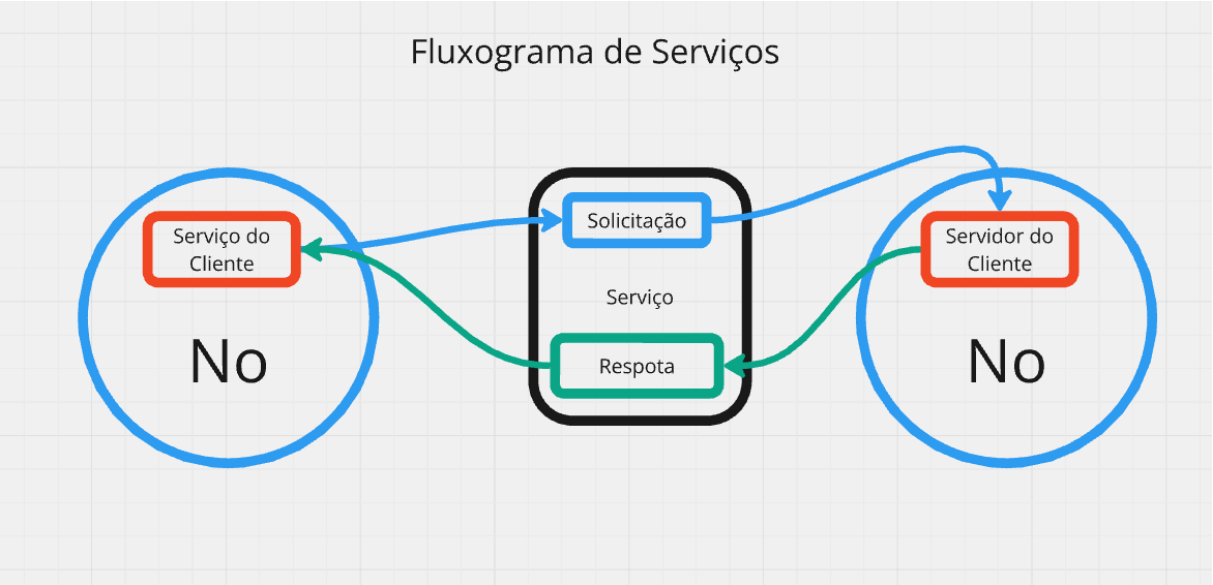
Os serviços no ROS 2 permitem a comunicação síncrona entre nós, semelhante a uma chamada de função remota. Um nó atua como *cliente*, enviando uma solicitação, enquanto outro nó, o *servidor*, processa a solicitação e envia uma resposta de volta.

Componentes dos Serviços

— Serviço: Cada serviço é definido por uma interface que especifica o tipo de mensagem enviada na solicitação e na resposta. No `turtlesim`, por exemplo, o serviço `reset` reinicia a posição da tartaruga.

— Cliente: O nó que faz a chamada de serviço é o cliente. Ele envia a solicitação e aguarda pela resposta do servidor.

— Servidor: O nó que oferece o serviço é o servidor. Ele recebe a solicitação do cliente, executa a operação solicitada e retorna a resposta ao cliente.



5. Parâmetros

Alterando a Cor da Tartaruga

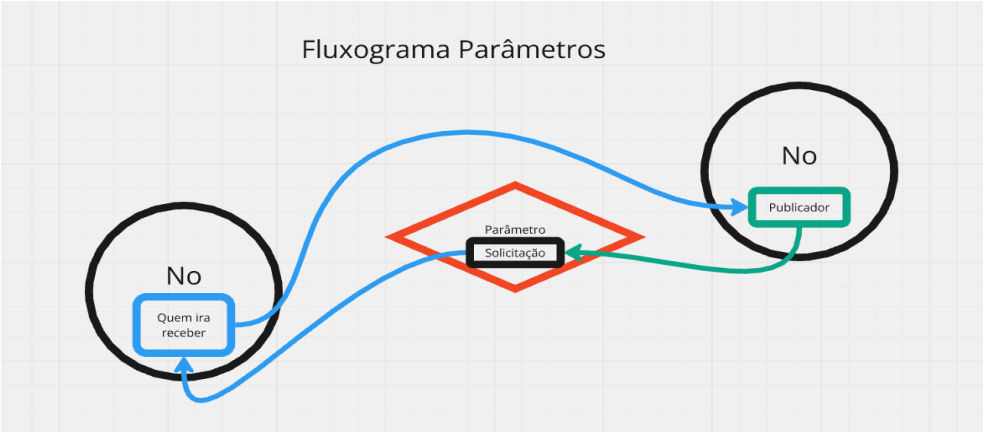
No `turtlesim`, é possível mudar a cor da tartaruga ajustando os parâmetros do nó. Os parâmetros principais são: `turtle1/background_r`, `turtle1/background_g` e `turtle1/background_b`, que controlam a cor de fundo, além de `turtle1/color`, que define a cor da própria tartaruga. No experimento, foi alterado apenas o parâmetro `turtle1/background_r` para o valor 155, modificando o tom de vermelho do fundo.

Diferença entre Parâmetros e Tópicos

— *Parâmetros*: São variáveis que configuram o comportamento de um nó e podem ser ajustadas durante a execução, permitindo que você personalize o comportamento do sistema sem recompilar. Eles podem ser definidos na execução do nó ou por meio de comandos, como o `ros2 param set`. No caso do `turtlesim`, a cor de fundo foi ajustada dinamicamente por meio de parâmetros.

— *Tópicos*: Diferente dos parâmetros, os tópicos permitem a comunicação assíncrona entre nós, onde um nó publica mensagens e outros podemos assiná-las para recebê-las. Enquanto os parâmetros possibilitam configurar nós, os tópicos são utilizados para a troca contínua de dados, como comandos de movimento no simulador.

Esse processo de ajuste dinâmico do parâmetro reflete a flexibilidade essencial que se busca ao configurar sistemas mais complexos na engenharia de computação, permitindo explorar as configurações sem interromper a execução.



8. Acoes

Diferença Entre Ações, Tópicos e Serviços

— *Ações*:

Natureza: Assíncrona, com feedback contínuo.

Uso: Permite que um cliente envie uma solicitação a um servidor e receba atualizações sobre o progresso, além de uma resposta final ao término da ação.

Exemplo: Mover um robô para uma posição específica, com atualizações constantes sobre o progresso do movimento.

— *Tópicos*:

Natureza: Assíncrona, sem garantia de resposta.

Uso: Facilitam a comunicação contínua entre nós, onde um nó publica dados e outros nós, que assinam o tópico, consomem essas mensagens.

Exemplo: Envio contínuo de dados de sensores, como leituras de temperatura ou distância.

— *Serviços*:

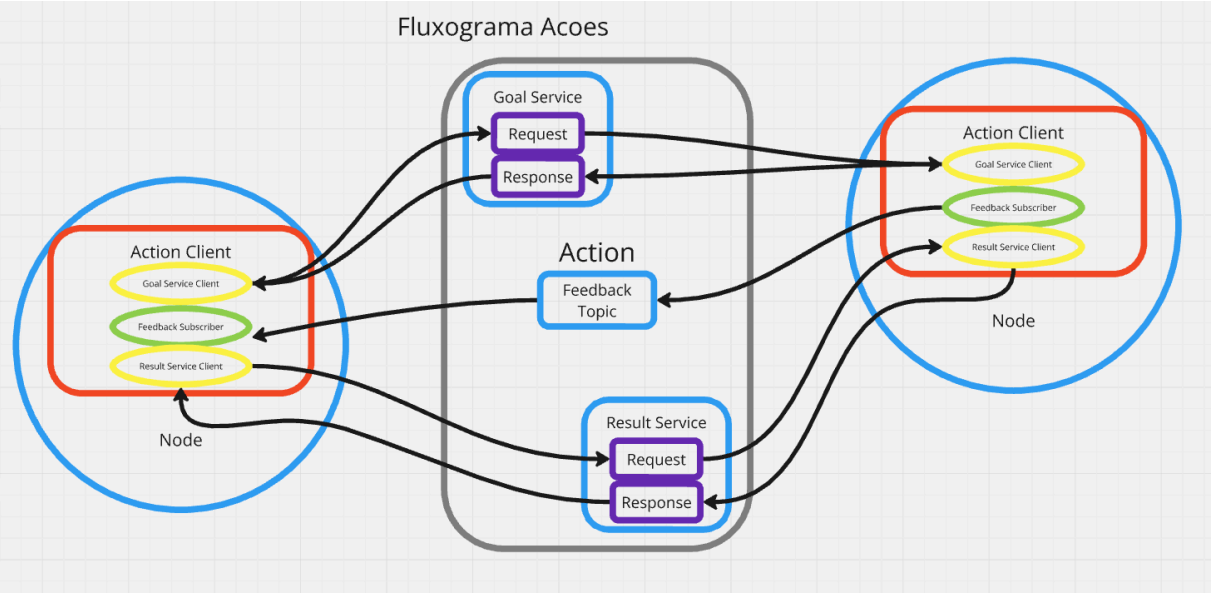
Natureza: Síncrona.

Uso: Um nó solicita algo a outro nó e aguarda uma resposta imediata.

Exemplo: Consultar a posição atual de um robô ou reiniciar o sistema de controle.

Uso de Ações em um Contexto Prático

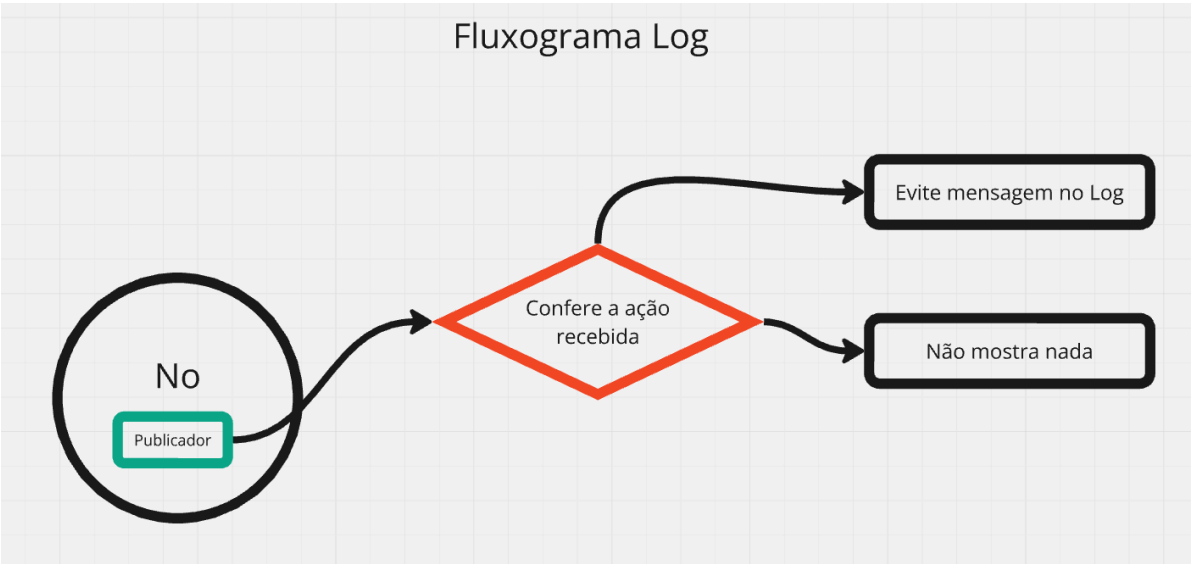
Ações são especialmente úteis em robótica quando uma tarefa pode demorar e é importante monitorar o progresso. Por exemplo, em tarefas de manipulação como pegar um objeto, o cliente envia a tarefa ao servidor, que executa a ação. Durante o processo, o cliente recebe feedback contínuo sobre o status, como se o objeto foi agarrado corretamente ou se houve um erro na execução. Isso garante que o cliente possa acompanhar e ajustar a execução conforme necessário.



7. Logging

Importância dos Logs

Habilitar e configurar o nível de log de um nó, como o `turtlesim_node`, é uma prática essencial para o monitoramento e a manutenção eficiente de sistemas. A capacidade de registrar e analisar mensagens de log desempenha um papel crucial no desenvolvimento e operação de sistemas robustos em ambientes de robótica, permitindo identificar problemas, monitorar o desempenho e garantir o bom funcionamento dos nós.



8. Inicialização de Nós

Explicação sobre o Sistema de Inicialização (Launching) no ROS 2

O sistema de lançamento (launching) no ROS 2 oferece uma maneira estruturada e eficiente de iniciar múltiplos nós, além de configurar seus parâmetros, facilitando a execução de sistemas complexos. Aqui estão alguns benefícios-chave desse sistema:

— *Automação*: Usando arquivos de lançamento (`launch`), os desenvolvedores podem automatizar a inicialização de diversos nós com um único comando, economizando tempo e eliminando a necessidade de iniciar cada nó manualmente.

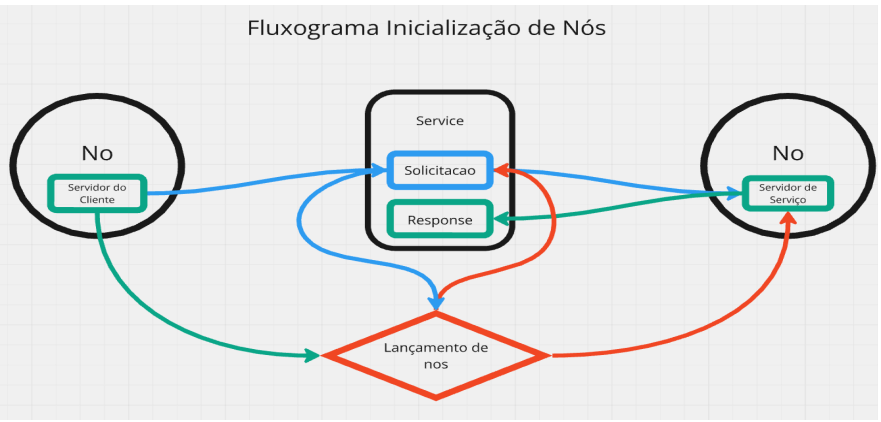
— *Configuração Centralizada*: Os parâmetros e as configurações de todos os nós podem ser centralizados em um único arquivo `launch`, permitindo uma gestão mais organizada e consistente de todas as configurações necessárias para o sistema.

— *Gerenciamento de Dependências*: O sistema de `launch` pode gerenciar as dependências entre os nós, garantindo que eles sejam iniciados na ordem correta. Isso é essencial em situações em que um nó depende de dados ou serviços de outro para funcionar adequadamente.

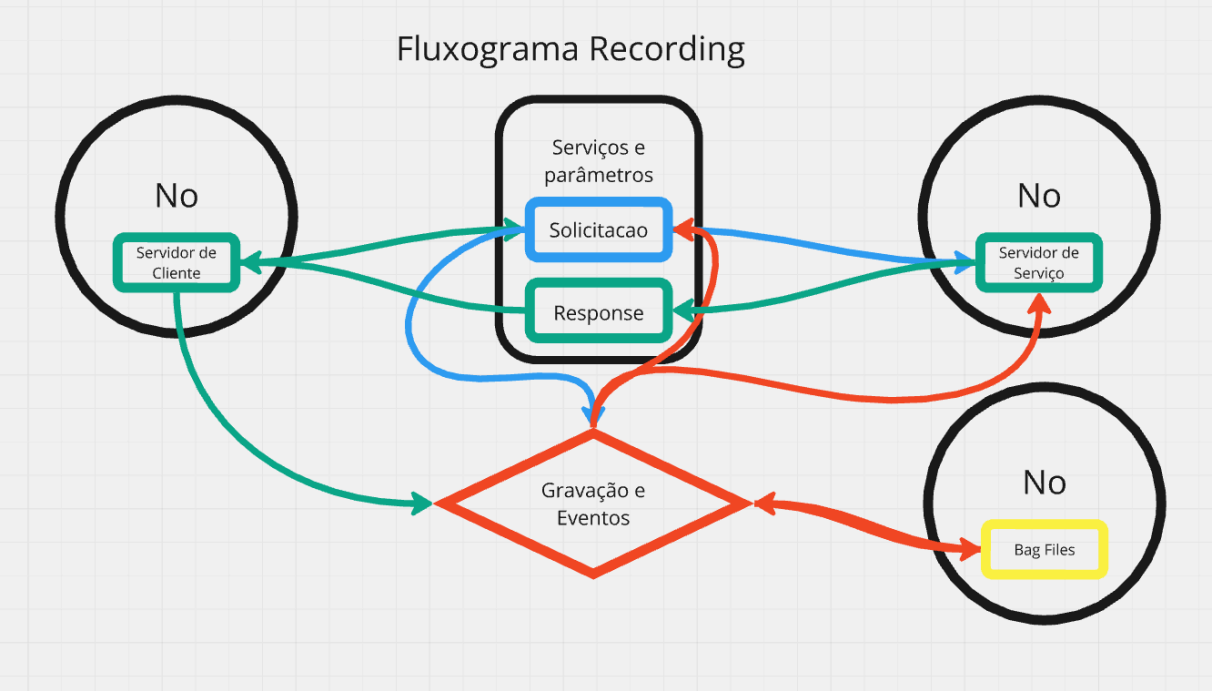
— *Flexibilidade*: Arquivos de `launch` permitem testar rapidamente diferentes combinações de nós e configurações, acelerando o desenvolvimento e experimentação de novas funcionalidades.

— *Escalabilidade*: Para sistemas maiores, o uso de arquivos de lançamento organiza o gerenciamento de muitos nós eficazmente, tornando o sistema escalável à medida que novos nós ou funcionalidades são adicionados.

Em resumo, o sistema de `launch` do ROS 2 é uma ferramenta poderosa que otimiza a inicialização, configuração e gerenciamento de nós, oferecendo flexibilidade e escalabilidade para sistemas de robótica complexos.



9. Gravação



Explicação do Comando de Gravação

O comando `ros2 bag record` é utilizado para capturar e armazenar mensagens de tópicos específicos em um arquivo de bag (bag file), sendo o formato padrão do ROS para registro de dados. No experimento, foi utilizado o comando:

ros2 bag record -o “FELIPE_2404846” /cmd_vel

Este comando grava todas as mensagens publicadas no tópico `/cmd_vel` em um arquivo de bag chamado “FELIPE_2404846”.

Estrutura do Bag File

O arquivo de bag é organizado para armazenar eficientemente dados e metadados relevantes, como:

— Informações sobre os tópicos gravados.

— Taxa de publicação das mensagens.

— Horários exatos em que as mensagens foram recebidas.

Utilidade para Análise Posterior

Os arquivos de bag são valiosos para uma análise posterior do sistema, permitindo revisitar execuções sem a necessidade de reproduzir o sistema em tempo real. Isso pode ser utilizado para:

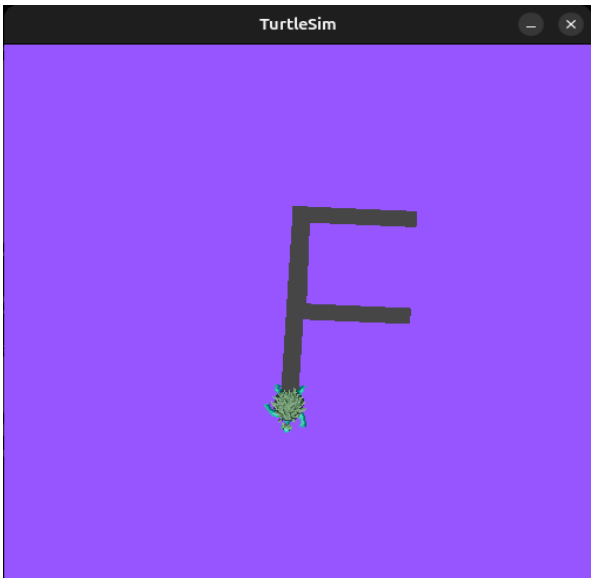
— *Debugging*: Identificar e corrigir problemas durante a execução dos nós.

— *Simulações*: Reproduzir o comportamento do sistema em cenários simulados para testar desenvolvimentos futuros.

— *Análise de Desempenho*: Avaliar a eficiência e comportamento de algoritmos de controle e tomada de decisão.

Esses dados gravados são essenciais para refinar o desenvolvimento e garantir que o sistema funcione como esperado.

Após colocar os comandos, a primeira letra do meu nome fica assim:



O comando ros2 bag record /cmd_vel inicia a gravação das mensagens publicadas no tópico cmd_vel e as armazena em um arquivo bag. O arquivo bag é útil para análise posterior, ao permitir reproduzir e estudar o comportamento do robô, refazer simulações ou identificar falhas, utilizando os dados gravados.

Primeiro comando: ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist “{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.57}}” --once

Segundo comando: ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist “{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}” --once

Terceiro comando: ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist “{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: -1.57}}” --once

Quarta comando: ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist “{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}” --once

Quinto comando: ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist “{linear: {x: -2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}” --once

Sexto comando: ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist “{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}” --once

Sétimo comando: ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist “{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: -1.57}}” --once

Oitavo comando: ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist “{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.57}}" --once

Nono comando: ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist “{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}” --once

Décimo Comando: ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist “{linear: {x: -2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}” --once

Décimo primeiro comando: ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist “{linear: {x: 0.0, y: -1.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}” --once

Décimo segundo comando: ros2 topic pub /turtle1/cmd_vel geometry_msgs/msg/Twist “{linear: {x: 1.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}” --once