Laboratório 04

Robôs Autônomos - 2022-2
Departamento de Engenharia Elétrica
Prof. Ricardo Mello

1 Introdução

Neste laboratório, iremos finalizar a atividade proposta na aula teórica de número 3, Introdução ao ROS, e iremos implementar um controlador para a tartaruga do *turtlesim*.

Dada uma posição objetivo $(x_d, y_d)^T$, queremos levar o robô desde uma posição arbitrária $(x, y)^T$ até o objetivo. Para isso, precisamos eliminar o erro de posição $(\tilde{x}, \tilde{y})^T = (x_d - x, y_d - y)^T$. Também podemos definir o problema usando coordenadas polares; temos o erro de posição:

$$\begin{pmatrix} \rho \\ \alpha \end{pmatrix} = \begin{pmatrix} \sqrt{\tilde{x}^2 + \tilde{y}^2} \\ \arctan \frac{\tilde{y}}{\tilde{x}} - \Psi \end{pmatrix} \tag{1}$$

Onde ρ é a distância euclidiana até o ponto desejado, Ψ é a orientação do robô e α é o ângulo entre a orientação atual do robô e o vetor ρ.

Vamos implementar uma lei de controle simples para verificar se conseguimos controlar o robô:

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} v_{max} \tanh \rho \\ k_\omega \alpha \end{pmatrix} \tag{2}$$

onde v_{max} é a velocidade máxima permitida do robô e k_ω é uma constante positiva. Siga as instruções abaixo para implementar seu controlador.

2 Atividades

2.1 - Comece tendo por base o laboratório passado. Baixe a imagem Docker do ROS 2 Humble:

```
$ docker pull osrf/ros:humble-desktop-full
```

Lembre-se de verificar se seu usuário tem permissão para executar o Docker. Habilite o uso de interface gráfica:

```
$ xhost + local:docker
$ export DISPLAY=:1
```

Rode o *container*:

```
$ docker run --name ros -it --net=host --device /dev/dri/ \
  -e DISPLAY=$DISPLAY -v $HOME/.Xauthority:/root/.Xauthority:ro \
  osrf/ros:humble-desktop-full
```

E não se esqueça de usar sempre o *namespace* quando for inicializar um nó. Por exemplo:

```
$ ros2 run turtlesim turtlesim_node --ros-args -r __ns:=/IniciaisDoSeuNome
```

Como lembrete, o comando abaixo abre um terminal dentro do *container*:

```
$ docker exec -it ros bash
```

E o *script* de inicialização deve ser executado ao abrir um novo terminal:

```
$ . ros_entrypoint.sh
```

Por fim, baixe o editor de texto *nano*:

```
$ apt -y update && apt -y install nano
```

2.2 - Dentro do *container*, inicialize um *workspace* e crie um pacote chamado *turtle_control_IniciaisDoSeuNome*. Caso não se lembre de como fazer, o tutorial sobre *workspace* está aqui, o tutorial sobre criação de pacotes está aqui. Por simplicidade, os comandos a seguir resumem o processo:

```
$ mkdir -p ~/ros2_ws/src && cd ~/ros2_ws && colcon build && cd src/
$ ros2 pkg create --build-type ament_python --node-name turtle_control turtle_control_Nome
```

O último comando cria o pacote com um *script turtle_control* dentro da pasta de mesmo nome do pacote. O tutorial sobre criação de nós está aqui.

2.3 - Para salvar seu pacote fora do *container* e tornar seu código acessível à comunidade, iremos utilizar o Github. Siga os passos neste link para criar um repositório no Github com seu pacote. O repositório deve ser aberto (público). Antes do fim da aula, lembre-se de sincronizar novamente seu pacote com o repositório no Github (comandos básicos neste link).

2.4 - No seu pacote,modifique o *script* chamado *turtle_control.py* para conter o nó principal do sistema de controle. Programe seu nó como uma classe, tal qual ensinado nos tutoriais que vimos anteriormente. Este nó deve (i) subscrever ao tópico de *pose* do *turtlesim*, (ii) subscrever ao tópico */goal*[1], onde são publicadas mensagens do tipo *geometry_msgs/msg/Pose2D*, e (iii) publicar no tópico de comando de velocidade do *turtlesim*. A classe que define seu nó de controle deve conter os seguintes métodos:

- __init__: utilize a rotina de inicialização para chamar as rotinas de inicialização compartimentadas a seguir;
- init_publisher: inicialize o *publisher*. Siga o tutorial básico do ROS 2 para criar um *timer* e um *callback* para o *publisher*;
- init_subscribers: inicialize seus *subscribers*, ligando-os com os *callbacks* adequados;
- init_variables: inicialize aqui as variáveis que serão utilizadas. Ex: $self.x$, $self.x\_error$, $self.x\_goal$, $self.k\_omega$, etc.;

[1] Ao rodar seu nó, você irá utilizar o *namespace* das suas iniciais. Porém, não deve haver menção ao *namespace* no seu código.

- pose_callback: *callback* para receber a pose do *turtlesim*;
- goal_callback: *callback* para receber a posição objetivo;
- pub_callback: método principal do nó, implementado como *callback* do *publisher*. Aqui você deverá computar o erro, implementar o controle e publicar a mensagem de velocidade para o *turtlesim*.

Dica: primeiro crie a estrutura do código, com todos os métodos, porém sem o conteúdo completo. Você pode, por exemplo, utilizar um *print* como *placeholder* (e.g., *print("Aqui será o inicializado o publisher")*). Implemente e teste cada parte, ao invés de testar o código por completo. Ao testar o *subscriber* da pose da tartaruga, lembre-se de verificar a posição inicial dela.

Para verificar a estrutura de mensagens do tipo *geometry_ msgs/msg/Pose2D*, utilize o seguinte comando:

```
$ ros2 interface show geometry_msgs/msg/Pose2D
```

Você deverá extrair x_d e y_d a partir dos campos x e y dessa mensagem.

Para testar, lembre-se de fazer o *build* e *source* do *workspace* antes de executar o nó.

```
$ cd ros2_ws && colcon build && source install/setup.bash
```

Mais uma dica: você pode definir um limiar de erro a partir do qual as velocidades serão sempre zero. Isso evita da tartaruga ficar eternamente se movimentando tentando convergir para um erro igual a zero.

2.5 - Demonstre o funcionamento do seu pacote abrindo, primeiro, o *turtlesim*, depois o seu nó de controle e, por fim, em outro terminal, utilize o comando abaixo para configurar a posição desejada da tartaruga:

```
$ ros2 topic pub /IniciaisDoSeuNome/goal geometry_msgs/msg/Pose2D \
    "{x: 7.0, y: 7.0, theta: 0.0}"
```

2.6 - Após verificar o funcionamento do controle, atualize o repositório remoto com seu pacote e envie o link do seu repositório para o professor.

2.7 - Ao finalizar o laboratório, lembre-se de encerrar o *container* e remover a imagem. Certifique-se de que a versão mais atual do seu pacote está disponível no repositório no Github.

```
$ docker stop ros
$ docker rm ros
$ docker rmi osrf/ros:humble-desktop-full
```