DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG

Mapa de Ocupação (ROS)

Robótica Movel

30 de agosto de 2017

Sumário

Capítulo 1

Introdução

Agora que você sabe como funciona o ROS e já programou seu robô para mover pelo ambiente simulado utilizando o laser, é hora de começar a procurar outro recurso que permite uma navegação global do robô. Para fazer isso, o robô precisa saber onde está, e para onde você deseja enviar. Normalmente, isso significa que ele precisa ter um mapa do mundo e saber onde está neste mapa. Neste tutorial, vamos ver como configurar e utilizar um mapa do mundo dentro do seu pacote ROS.

Os mapas de navegação no ROS são representados por uma grade 2D, onde cada célula contém um valor que corresponde à probabilidade de ocupação. Os arquivos de mapa são armazenados como imagens em escala de cinza, com uma variedade de formatos comuns suportados (como PNG, JPG e PGM). Associado a cada mapa existe um arquivo YAML que contém informações adicionais, como a resolução (o comprimento de cada célula de grade em metros), onde é a origem do mapa e limiares para decidir se uma célula está ocupada ou desocupada.

Agora vou mostrar como criar e configurar esse arquivo utilizando o pacote exemplo.

1.1 Adicionando Mapas ao Pacote

Inicialmente, acesse o diretório do pacote onde será utilizado o mapa e crie a seguinte pasta.

```
$ roscd move_example
$ mkdir maps
$ cd maps
```

Agora vamos escolher um mapa exemplo para o nosso programa. Vamos utilizar o mapa **cave** que é o mapa (Figura 1.1) exemplo do Stage. Você pode encontrá-lo dentro do diretório do Stage. Vamos copiar essa imagem para dentro da pasta **maps** e então criar o arquivo YAML que irá permitir o ROS ler o mapa.

```
$ roscp stage cave.png .
$ touch cave.yaml
$ nano cave.yaml
```

Insira o seguinte código dentro do arquivo.

```
image: cave.png
resolution: 0.032
origin: [0.0, 0.0, 0.0]
occupied_thresh: 0.65
free_thresh: 0.196
negate: 0
```

Este arquivo irá referencia nosso mapa à imagem **cave**. A resolução indica que cada pixel da imagem serão considerados com 32 cm no mapa e a origem do mapa será $(0, 0, 0)$. Ao escolher a resolução leve em consideração a resolução utilizada no mapa do simulador **Stage** (16x16) em

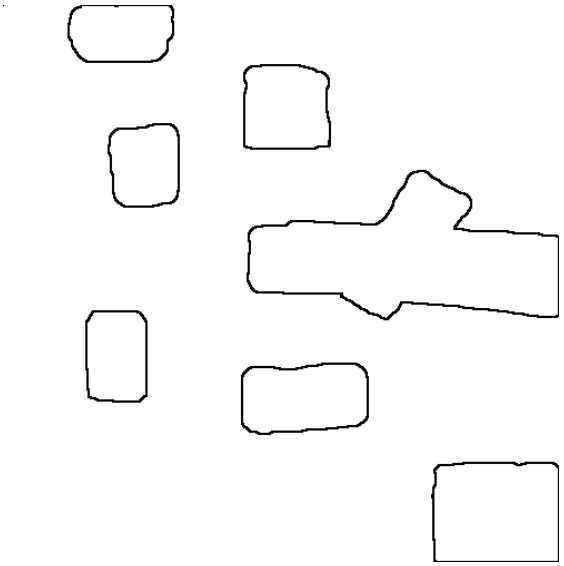

Figura 1.1: Exemplo de Mapa a ser utilizado.

relação ao tamanho real da imagem (500x500). Os outros parâmetros permitem definir limiares para células ocupadas ou desocupadas.

Após editar o arquivo salve-o e feche seu editor.

1.2 Execução

Para executar nosso mapa, vamos utilizar o node map_server. Utilize os seguintes comandos:

```
$ roscore
$ rosrun map_server map_server cave.yaml
```

Pronto! Agora temos acesso ao mapa para a figura **cave**. Vamos analisar o tipo do tópico através comando **rostopic**. Isso permite saber o tipo de mensagem que deveremos importar no nosso programa para acessar o mapa.

```
$ rostopic list
/map
/map_metadata
/rosout
/rosout_agg
$ rostopic type /map
nav_msgs/OccupancyGrid
```

Agora vamos criar um exemplo simples de programa que lê o mapa.

```
$ roscd move_example/src
$ touch map_example.py
$ nano map_example.py
```

O código abaixo lê o mapa de ocupação do tópico /**map** e imprime na saída padrão sua matriz. Na matriz, o valor indica a probabilidade da célula estar ocupada.

```
#!/usr/bin/env python
import rospy
from nav_msgs.msg import OccupancyGrid
import numpy as np

mapMsg = None

def MapCallback(msg):
```

```
  global mapMsg
  mapMsg = msg

def run():
  rospy.init_node('map_example', anonymous=False)
  rospy.Subscriber('/map', OccupancyGrid, MapCallback)
  rate = rospy.Rate(5) # 5 hz
  print "[Map Example Started]"
  global mapMsg

  while not rospy.is_shutdown():
    if mapMsg == None:
      rate.sleep()
      continue
    # transforma mapa de array para matrix
    mapa = np.asarray(mapMsg.data)
    mapa = mapa.reshape((mapMsg.info.height,mapMsg.info.width))
    print mapa
    rate.sleep()

if __name__ == '__main__':
  try:
    run()
  except rospy.ROSInterruptException:
    pass
```

Para executar o programa, utilize os seguintes comandos:

```
$ sudo chmod a+x map_example.py
$ rosrun move_example map_example.py
```

1.3 Visualização

O framework **RViz** permite visualizar o mapa e também o robô. Vamos executar o simulador **Stage** e configurar o **RViz** para que possamos acompanhar o que está acontecendo. Utilize os seguinte comandos para executar o simulador:

```
$ roscd stage/world
$ rosrun stage_ros stageros simple.world
```

Para referenciar/transformar o mapa ao robô utilizamos o nó **tf** do ROS para indicar o que mapa está deslocado em relação à posição inicial do robô. A sintaxe do comando consiste em translação mais rotação (quatérnion). Em seguida vem os tópicos dizendo que é uma transformação do frame /odom para /map e com um frequência de 10 Hz. Utilize o seguinte comando e em seguida abra o **Rviz**:

```
$ rosrun tf static_transform_publisher -8 -8 0 0 0 0 1 /odom /map 10
$ rviz
```

A figura 1.2 mostra o **Rviz** configurado com o mapa e o robô(seta vermelha). Ao mover o robô no simulador, esta interface mostra o robô se movendo na mesma proporção.

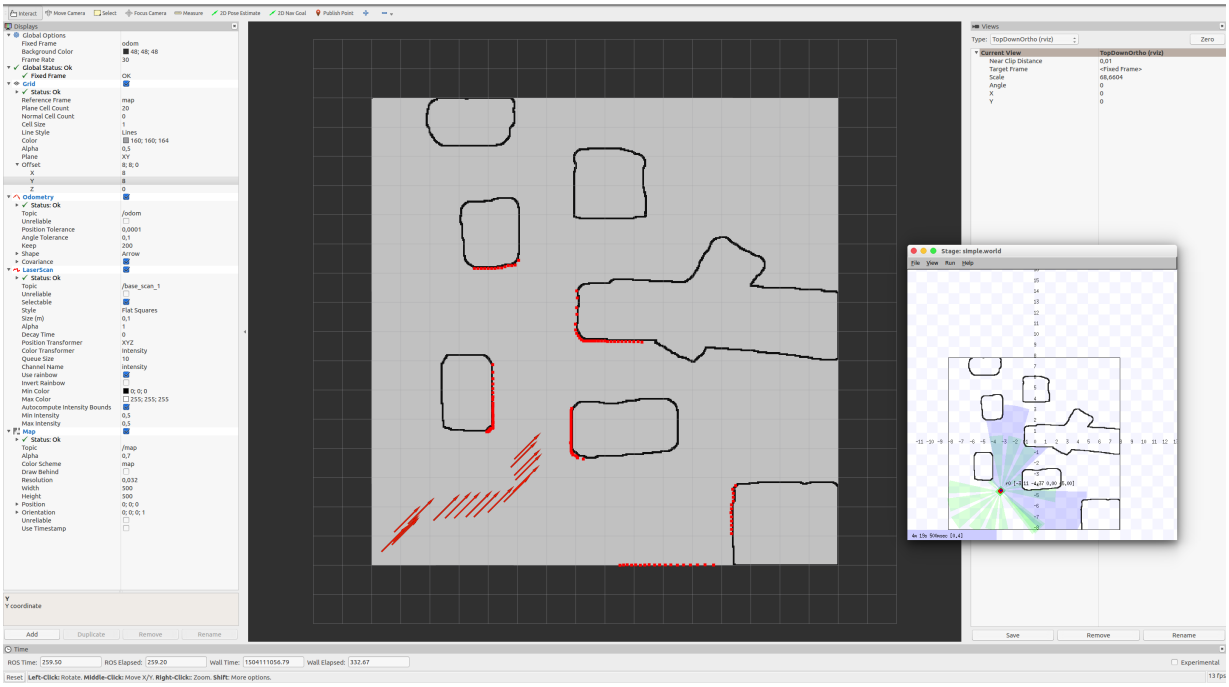

Figura 1.2: RViz configurado para mostrar o mapa de ocupação e o robô.

Referências Bibliográficas

[1] Quigley, Morgan, Brian Gerkey, and William D. Smart. Programming Robots with ROS: a practical introduction to the Robot Operating System. "O'Reilly Media, Inc.", 2015.

[2] Documentation - ROS Wiki: http://wiki.ros.org/map_server

[3] Mensagem - http://docs.ros.org/api/nav_msgs/html/msg/OccupancyGrid.html