ROS-Explorer

Docker

- Funciona como uma máquina virtual, só que sem a queda de performance. O chroot seria uma comparação mais apropriada.
- Utiliza o mesmo kernel do host, criando processos que enxergam uma árvore de diretórios própria.
- Permite rodar um Ubuntu 16.04 com ROS-kinetic em qualquer distribuição Linux.
- Documenta os passos necessários para recriar uma instalação.

Docker 1/3

```
FROM osrf/ros:kinetic-desktop-full

# Prefer ROS repository over Ubuntu
# a priority above 1000 will allow even downgrades
RUN echo '\
Package: * \n\
Pin: release o=ROS \n\
Pin-Priority: 1001 \n\
' > /etc/apt/preferences
                                ...
```

Docker 2/3

```
                                ...
# ros-kinetic-librealsense.postinst fails to detect docker
RUN apt-get update \
&&  apt-get install -y \
     ros-kinetic-librealsense \
||  rm /var/lib/dpkg/info/ros-kinetic-librealsense.postinst
-f \
&&  apt-get -f install \
&&  rm -rf /var/lib/apt/lists/*
                                ...
```

Docker 3/3

```
                                ...
RUN apt-get update \
&&  apt-get install -y \
     tmux \
     git \
     vim \
     ros-kinetic-turtlebot-gazebo \
     ros-kinetic-turtlebot-stage \
     python-scipy \
&&  apt-get upgrade -y \
&&  rm -rf /var/lib/apt/lists/*
```

Instalação

- Instalação do projeto:

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
git clone https://github.com/atilaromero/ros-explorer.git
cd ~/catkin_ws/
catkin_make
source devel/setup.bash
```

- Inicialização do simulador:

```
roslaunch turtlebot_stage turtlebot_in_stage.launch
```

Movimentação simples (alternativa ao teleop)

- Movimento por coordenadas sem desvio de obstáculos

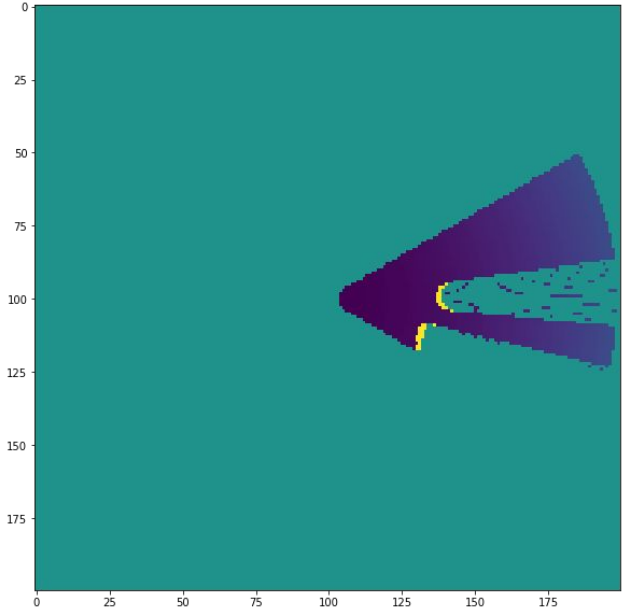
```
rosrun ros-explorer move.py 1 0
```

Exploração

```
rosrun ros-explorer explore.py
```

- Entrada:
  - `/scan` - leitura do laser
- Saídas:
  - `/cmd_vel_mux/input/navi` - comandos de movimento
  - `/mymap` - mapa construído com a leitura do laser
  - `/myfield` - campo potencial para exploração
- Salva mapa no diretório atual no formato png

Exploração - laser scan -> mapa



Exploração - laser scan -> mapa

```
def ranges2cart(ranges, range_min, range_max, angle_min, angle_increment):
   imgsize = 100
   rangestep = range_max/imgsize
   ranges = np.array(ranges)
   r, phi = np.mgrid[0:range_max:rangestep,-np.pi:np.pi:angle_increment]
   phimin_index = int((angle_min+np.pi)/angle_increment)
   ranges2d = np.zeros(r.shape)
   ranges2d[:,phimin_index:phimin_index+ranges.shape[0]]=ranges
   ranges2d[np.isnan(ranges2d)] = range_max*2
   v = np.zeros(r.shape)
   v[r>ranges2d] = 0
   v[r<ranges2d] = -1
   v[r<range_min] = 0
   v[(ranges2d!=0) & (ranges2d!=range_max) & (np.abs(r-ranges2d)<0.1)] =1
   v *= np.cos(r/range_max) #v é a primeira imagem do slide anterior
   dst = cv2.linearPolar(v.T, (imgsize,imgsize),imgsize,cv2.WARP_FILL_OUTLIERS | cv2.WARP_INVERSE_MAP)
   dst2 = np.zeros((imgsize*2,imgsize*2))
   dst2[:,imgsize:] = dst[:imgsize*2,:][:,::-1]
   return dst2 #dst2 é a segunda imagem do slide anterior
```

Exploração - harmonic potential fields

```
kernel = np.array([
           [0,1,0],
           [1,0,1],
           [0,1,0]
           ])/4.0

def mkBVPMap(worldmap, steps=100, walls=None):
    if walls is None:
        walls = np.ones(worldmap.shape) *  0.9
    for x in range(steps):
        walls[worldmap> 0.3]=1 #walls
        walls[worldmap== 0]=-1 #unknown area
        walls = cv2.filter2D(walls,- 1,kernel)
    return walls
```

Exploração - publicação do mapa

```
def publishMap(worldmap, topicName, publisher, x,y, rot):
   msg = OccupancyGrid()
   msg.header.stamp = rospy.Time.now()
   msg.header.frame_id = topicName
   msg.info.resolution = 0.05
   msg.info.width = worldmap.shape[0]
   msg.info.height = worldmap.shape[1]
   msg.info.origin.orientation = Quaternion(0,0,0,1)
   msg.info.origin.position.x = -x*msg.info.resolution  #mapa segue o robo e ajusta posicao
   msg.info.origin.position.y = -y*msg.info.resolution
   msg.data = 100/(1+np.exp(-worldmap))
   msg.data[worldmap == 0]=-1
   msg.data = msg.data.T.astype(np.int8).ravel()
   publisher.publish(msg)
   br = tf.TransformBroadcaster()
   br.sendTransform((0,0,0),
                    tf.transformations.quaternion_from_euler(0, 0, -rot-np.pi/2),
                    rospy.Time.now(),
                    topicName,
                    "base_link") # base do robo
```

Exploração