#!/usr/bin/env python
import rospy
import signal
import sys
import numpy as np
import math

from nav_msgs.msg import OccupancyGrid
from sensor_msgs.msg import LaserScan

allow user to terminate script with CTRL+C

def signal_handler(signal, frame):
sys.exit(0)
signal.signal(signal.SIGINT, signal_handler)

UNKNOWN = -1
FREE = 0
OBST = 100 #obst ist gesund

#x,y in meters
def setCell(x,y,val):
global grid
x = -x # mirror
res = grid.info.resolution

```
x_scaled = x * 1.0 / res + grid.info.width / 2.0
y_scaled = y * 1.0 / res + grid.info.height / 2.0

if x_scaled >= grid.info.width or x_scaled < 0 or y_scaled >= grid.info.height or y_
    return
offset = (int(round(x_scaled)) - 1) * grid.info.height
if (grid.data[int(offset) + int(round(y_scaled) - 1)] == OBST):
    return
grid.data[int(offset) + int(round(y_scaled) - 1)] = val
```

def init():
global grid

```
global pub_grid
pub_grid = rospy.Publisher("scan_grid", OccupancyGrid, queue_size=1)
grid = OccupancyGrid()
grid.info.resolution = 1/10.0 # 10 cells per meters
grid.info.width = 120 #12m
grid.info.height = 120 #12m
grid.info.origin.orientation.x = 0
grid.info.origin.orientation.y = 0
grid.info.origin.orientation.z = 0
grid.info.origin.orientation.w = 1
grid.info.origin.position.x = 0
grid.info.origin.position.y = 0
grid.info.origin.position.z = 0.1
#set the whole grid to unknown:
grid.data = [UNKNOWN for i in range(grid.info.width*grid.info.height)]
```

```
rospy.init_node('foobar', anonymous=True)
rospy.Subscriber("/scan", LaserScan, scanCallback, queue_size=1)
```

```
#https://scipython.com/book/chapter-6-numpy/examples/creating-a-rotation-matrix-in-numpy/
def rotate(v,a):
rad = (a * np.pi / 180.0) + np.pi / 2
c, s = np.cos(rad), np.sin(rad)
R = np.matrix('{} {}; {} {}'.format(c, -s, s, c))
return np.dot(R,v)
```

```
#http://www.roguebasin.com/index.php?title=Bresenham%27s_Line_Algorithm
def get_line(start, end):
"""Bresenham's Line Algorithm
Produces a list of tuples from start and end
```

```
>>> points1 = get_line((0, 0), (3, 4))
>>> points2 = get_line((3, 4), (0, 0))
>>> assert(set(points1) == set(points2))
>>> print points1
[(0, 0), (1, 1), (1, 2), (2, 3), (3, 4)]
>>> print points2
[(3, 4), (2, 3), (1, 2), (1, 1), (0, 0)]
"""
# Setup initial conditions
x1, y1 = start
x2, y2 = end
dx = x2 - x1
```

```
dy = y2 - y1

# Determine how steep the line is
is_steep = abs(dy) > abs(dx)

# Rotate line
if is_steep:
    x1, y1 = y1, x1
    x2, y2 = y2, x2

# Swap start and end points if necessary and store swap state
swapped = False
if x1 > x2:
    x1, x2 = x2, x1
    y1, y2 = y2, y1
    swapped = True

# Recalculate differentials
dx = x2 - x1
dy = y2 - y1

# Calculate error
error = int(dx / 2.0)
ystep = 1 if y1 < y2 else -1

# Iterate over bounding box generating points between start and end
y = y1
points = []
for x in range(x1, x2 + 1):
    coord = (y, x) if is_steep else (x, y)
    points.append(coord)
    error -= abs(dy)
    if error < 0:
        y += ystep
        error += dx

# Reverse the list if the coordinates were swapped
if swapped:
    points.reverse()
return points
```

def addValidLidar(a,l):
global grid
#make angle and length to vector:
e1 = np.array([[1.0],[0.0]])
vec = rotate(e1*l, a)

```
#dirty way, but get_line needs integers and our vector is float:
l = get_line((0,0),(int(vec[0] * 100),int(vec[1] * 100)))
u = [(x / 100.0, y / 100.0 ) for (x,y) in l]
for (x,y) in u:
    setCell(x,y,FREE)
setCell((vec[0]),(vec[1]),OBST)
```

```
def setInfWhite(a):
global grid
#make angle and length to vector:
e1 = np.array([[1.0],[0.0]])
inf_length = min(grid.info.height, grid.info.width)
vec = rotate(e1*inf_length, a)
```

```
#dirty way, but get_line needs integers and our vector is float:
l = get_line((0,0),(int(vec[0] * 100),int(vec[1] * 100)))
u = [(x / 100.0, y / 100.0 ) for (x,y) in l]

for (x,y) in u:
    setCell(x,y,FREE)
```

```
def pubGrid():
global grid
global pub_grid
pub_grid.publish(grid)
resetGrid()

def resetGrid():
global grid
grid.data = [UNKNOWN for i in range(grid.info.width*grid.info.height)]

def scanCallback(data):
global grid
rs = data.ranges
for a in range(len(rs)): # angle = [0:359]
r = rs[a] #radius / lidar_len / dist to obstacle
if not math.isinf(r) and data.intensities[a] != 0:
addValidLidar(a,r) #add obstacle and free cells from origin
else:
setInfWhite(a) #add free cells for all angles with no obstacle
pubGrid()

if name__ == '__main':
try:
init()
rospy.spin()
```

```
except rospy.ROSInterruptException:
pass
```