CS3027/5095 Robotics

Practical IV

2016

Introduction

In this practical, we'll write a wall following node in ROS, and learn about rviz, a powerful visualisation tool for ROS.

Tasks

1. Building on the theory described in class, write a program which will allow a robot to follow a wall. You'll need to decide whetehr it'll follow a wall to its left or right, and might need additional parameters such as the distance to remain from the wall. A sample solution is shown at the end of this practical, but don't just copy it, rather use bits of it if you get stuck, and understand how it works.

2. Rviz is a powerful visualisation package for ROS. Full documentation can be found at `http://wiki.ros.org/rviz`, but most of the instructions are for C++. Here, we'll look at how to draw some markers in rviz. Before starting, fire up stage.

   (a) Rviz can be started with the `rviz` command.

   (b) Like other ROS components, rviz listens to topics, and visualises them. If you click on the "add" button, you'll see that there are many things you can add, but ignore them for now (don't forget to play around with them later though!).

   (c) You'll notice on the left that under global options, there's a "Fixed Frame" entry, which probably says "map". This is the frame of reference with regards to which everything else will get displayed. If you double click on "map", it'll allow you to edit it. Change it to "base_link", meaning that everything will be displayed with respect to the robot's frame of reference.

   (d) Now click on "Add", "By topic" and select "base_scan". You'll see some red dots appear, representing the points that the robot detects. We're now going to write a program that does something similar.

(e) Create a python file that — in addition to the usual imports — imports

```
from visualization_msgs.msg import Marker
from visualization_msgs.msg import MarkerArray
```

(f) Create a publisher that publishes objects of type `MarkerArray`, and add the following method to your program. Then invoke the method with some x and y coordinates, and r,g,b values between 0 and 1. Set frame to "/base_link". The program assumes your publisher is called `pub`.

```
      def add(x,y,r,g,b,frame):
                mr=Marker()
                mr.header.frame_id=frame
                mr.ns="basic"
                mr.id=self.i
                mr.type=mr.SPHERE
                mr.action=mr.ADD
                mr.pose.position.x=x
                mr.pose.position.y=y
                mr.pose.orientation.w=1
                mr.scale.x=1.5
                mr.scale.y=1.5
                mr.scale.z=1.5
                mr.color.r=r
                mr.color.g=g
                mr.color.b=b
                mr.color.a=1.0
ma=MarkerArray()
ma.markers.append(mr)
pub.publish(ma)
```

(g) Going through "Add" in rviz, you should be able to add the topic and have spheres displayed on your screen when the program is run.

(h) You should see a small sphere appear on the screen at the appropriate X and Y coordinates. Note that if you add more markers to the marker array, they will all appear, *assuming that each has a unique id*.

(i) Now extend the wall following program to display the robot's centre and the direction in which the robot will move. The sample solution at the end of the practical shows how to do this. Try change it so that the direction appears as an arrow rather than a sphere.

(j) Challenge task: extend the wall following code to implement a Bug algorithm.

(k) Set up the map server, and see if you can get the map to display in rviz.

```
#!/usr/bin/env python

import Marker
import roslib
import math
import rospy
import tf
from geometry_msgs.msg import Twist
from geometry_msgs.msg import PointStamped
from geometry_msgs.msg import Point
from std_msgs.msg import Header
from sensor_msgs.msg import LaserScan

LEFT=1
RIGHT=0

class Wallfollower:
        def __init__(self):
                rospy.init_node('wallfollower')
                self.safeDistance=2
                self.following=LEFT #following wall on LEFT or RIGHT

                self.listener=tf.TransformListener()
                self.publisher=rospy.Publisher('/cmd_vel',Twist,queue_size=10)
                self.subscriber=rospy.Subscriber('/base_scan',LaserScan,

                self.m=Marker.Markers() #used for drawing in rviz

        def scanReceived(self,ls):
                a=ls.angle_min
                i=ls.angle_increment

                #initialise angles for scanning to left or right of the robot
                #depending on wall following direction.
                if self.following==RIGHT:
                        start=-999
                        end=0
                else:
                        start=0
                        end=999

                #set ranges to arbitrary numbers initially
                rmin=9999
                amin=999
                for r in ls.ranges: #find closest point
                        if (a<=end and a>=start and r<rmin):
```

```
                amin=a
                rmin=r

        a+=i


#compute point in laser frame of reference
#and transform to robot frame
x=rmin*math.cos(amin)
y=rmin*math.sin(amin)

ps=PointStamped(header=Header(stamp=rospy.Time(0),
                                        frame_id="/base_laser_link")
p=self.listener.transformPoint("/base_link",ps)
x=p.point.x
y=p.point.y
mag=math.sqrt(x*x+y*y) #useful for later, magnitude of vector

#tangent vector
tx=0
ty=0

if (self.following==RIGHT):
        tx=-y
        ty=x
else:
        tx=y
        ty=-x

tx/=mag
ty/=mag

#where we should be going
dx=x+tx-self.safeDistance*x/mag
dy=y+ty-self.safeDistance*y/mag
self.m.add(dx,dy,0,0,1.0,"base_link") #draw heading

theta=0

if (dx>0 and rmin<ls.range_max): #tangent is in front of us
        theta=math.atan(dy/dx)
elif (self.following==RIGHT):
        theta=-1
else:
        theta=1
```

```
                twist=Twist()
                if (dx>0):
                        twist.linear.x=dx
                twist.angular.z=theta
                self.publisher.publish(twist)
                self.m.add(0,0,1.0,0,0,"base_link") #draw origin
                self.m.draw()

robot=Wallfollower()
rospy.spin()
```

```
import rospy
import roslib
from visualization_msgs.msg import Marker
from visualization_msgs.msg import MarkerArray

class Markers:
        def __init__(self):
                self.i=0
                self.markers=[]
                self.pub=rospy.Publisher("/scanMarkers",MarkerArray,
                                                                queue_size=200)
        def add(self,x,y,r,g,b,frame):
                mr=Marker()
                mr.header.frame_id=frame
                mr.ns="basic"
                mr.id=self.i
                mr.type=mr.SPHERE
                mr.action=mr.ADD
                mr.pose.position.x=x
                mr.pose.position.y=y
                mr.pose.orientation.w=1
                mr.scale.x=1.5
                mr.scale.y=1.5
                mr.scale.z=1.5
                mr.color.r=r
                mr.color.g=g
                mr.color.b=b
                mr.color.a=1.0
                self.markers.append(mr)
                self.i+=1

        def draw(self):
                markerArray=MarkerArray()
                for m in self.markers:
```

```
                markerArray.markers.append(m)
        self.pub.publish(markerArray)
        self.i=0

def clear(self):
        markerArray=MarkerArray()
        for m in self.markers:
                m.action=m.DELETE
                markerArray.markers.append(m)
        self.pub.publish(markerArray)
```