Sample open loop code to control a turtle bot from:
A Gentle Introduction to ROS – Indigo, R. Patrick Goebel

[Sample code from book available as timed_out_and_back.py via
">git clone https://github.com/pirobot/rbx1.git"]

This example will need to be modified to replace the Twist messages used to control a differential drive robot, like the turtlebot, with AckermannDriveStamped messages used by the RACECAR.

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

""" timed_out_and_back.py - Version 1.2 2014-12-14

    A basic demo of the using odometry data to move the robot along
    and out-and-back trajectory.

    Created for the Pi Robot Project: http://www.pirobot.org
    Copyright (c) 2012 Patrick Goebel.  All rights reserved.

    This program is free software; you can redistribute it and/or
modify
    it under the terms of the GNU General Public License as
published by
    the Free Software Foundation; either version 2 of the License,
or
    (at your option) any later version.5

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details at:

    http://www.gnu.org/licenses/gpl.html

"""

import rospy
# Modify this line to import the AckermannDriveStamped message
# (Use "rosmsg show" to figure out the correct module to use)
from geometry_msgs.msg import Twist
from math import pi

class OutAndBack():
    def __init__(self):
        # Give the node a name
        rospy.init_node('out_and_back', anonymous=False)

        # Set rospy to execute a shutdown function when exiting
        rospy.on_shutdown(self.shutdown)

        # Publisher to control the robot's speed
        # Publish correct message to the appropriate RACECAR topic
        self.cmd_vel = rospy.Publisher('/cmd_vel', Twist,
queue_size=1)
```

```
# How fast will we update the robot's movement?
rate = 50

# Set the equivalent ROS rate variable
r = rospy.Rate(rate)

# Set the forward linear speed to 0.2 meters per second
# Set the correct field in the message
linear_speed = 0.2

# Set the travel distance to 1.0 meters
goal_distance = 1.0

# How long should it take us to get there?
linear_duration = goal_distance / linear_speed

# Modify this section to drive straight forward then reverse
# (instead of turning in place at each end of the path)
# Set the rotation speed to 1.0 radians per second
angular_speed = 1.0

# Set the rotation angle to Pi radians (180 degrees)
goal_angle = pi

# How long should it take to rotate?
angular_duration = goal_angle / angular_speed

# Loop through the two legs of the trip
for i in range(2):
    # Initialize the movement command
    # Use the appropriate message and correct fields
    move_cmd = Twist()

    # Set the forward speed
    move_cmd.linear.x = linear_speed

    # Move forward for a time to go the desired distance
    ticks = int(linear_duration * rate)

    for t in range(ticks):
        self.cmd_vel.publish(move_cmd)
        r.sleep()

    # Stop the robot before the rotation
    # Use the appropriate message and correct fields
    move_cmd = Twist()
    self.cmd_vel.publish(move_cmd)
    rospy.sleep(1)

    # Now rotate left roughly 180 degrees
    # Your implementation is much simpler than this example
    # (it doesn't have turns at the ends of the path)

    # Set the angular speed
    move_cmd.angular.z = angular_speed
```

```
                # Rotate for a time to go 180 degrees
                ticks = int(goal_angle * rate)

                for t in range(ticks):
                    self.cmd_vel.publish(move_cmd)
                    r.sleep()

                # Stop the robot before the next leg
                # Use the correct message
                move_cmd = Twist()
                self.cmd_vel.publish(move_cmd)
                rospy.sleep(1)

            # Stop the robot
            # Use the correct message
            self.cmd_vel.publish(Twist())

    def shutdown(self):
        # Always stop the robot when shutting down the node.
        rospy.loginfo("Stopping the robot...")
        # Use the correct message
        self.cmd_vel.publish(Twist())
        rospy.sleep(1)

if __name__ == '__main__':
    try:
        OutAndBack()
    except:
        rospy.loginfo("Out-and-Back node terminated.")
```