ESE-680 - Autonomous Racing	**(Due: 01/29/2020)**
Safety Node	
Instructor: Rahul Mangharam	*Name:* Student name(s), *PennID:* PennID

Course Policy: Read all the instructions below carefully before you start working on the assignment, and before you make a submission.

- All sources of material must be cited. The University Academic Code of Conduct will be strictly enforced.

Goals and Learning outcomes
The following fundamentals should be understood by the students upon completion of this lab:

- Using the LaserScan message in ROS
- Time to Collision (TTC)
- Safety critical systems

1 Overview

The goal of this lab is to develop a safety node for the race cars that will stop the car from collision when travelling at higher velocities. We will implement Time to Collision using the `LaserScan` message in the simulator.

1.1 The LaserScan Message

Recall from the previous lab that each `LaserScan` message contains several fields that will be useful to us. You'll need to subscribe to the scan topic and calculate TTC with the `LaserScan` messages.

1.2 The Odometry Message

Both the simulator node and the car itself publish `Odometry` messages (ROS documentation linked here: http://docs.ros.org/melodic/api/nav_msgs/html/msg/Odometry.html). Within its several fields, the message includes the car's position, orientation, and velocity. You'll need to explore this message type in this lab.

1.3 The AckermannDriveStamped Message

`AckermannDriveStamped` is the message type that we'll use throughout the course to send driving commands to the simulator and the car. ROS's documentation is linked here:
http://docs.ros.org/api/ackermann_msgs/html/msg/AckermannDriveStamped.html

Note that we won't be sending driving commands to the car from this node, we're only sending the brake commands. By sending an `AckermannDriveStamped` message with the velocity set to 0.0, the simulator and the car will interpret this as a brake command and hit the brakes.

1.4 The TTC calculation

Time to Collision (TTC) is the time it would take for the car to collide with an obstacle if it maintained its current heading and velocity. Between the car and its obstacle, we can calculate it as:

$$TTC = \frac{r}{[-\dot{r}]_+}$$

where r is the distance between the two objects and $\dot{r}$ is the time derivative of that distance. $\dot{r}$ is computed by projecting the relative velocity of the car onto the distance vector between the two objects. The operator $[]_+$ is defined as: $[x]_+ := \max(x, 0)$.
You'll need to calculate the TTC for each beam in the laser scan. Projecting the velocity of the car onto each distance vector is very simple if you know the angle between the car's velocity vector and the distance vector (which can be determined easily from information in the `LaserScan` message).

2 Automatic Emergency Brake with TTC

For this lab, you will make a Safety Node that should halt the car before it collides with obstacles. To do this, you will make a ROS node that subscribes to the `LaserScan` and `Odometry` messages. It should analyze the `LaserScan` data and, if necessary, publish an `AckermannDriveStamped` with the velocity field set to 0.0 m/s, and a `Bool` message set to True (http://docs.ros.org/melodic/api/std_msgs/html/msg/Bool.html). The `AckermannDriveStamped` message will be received by the Mux node and the `Bool` message will be received by the Behavior Controller, which will then tell the Mux node to select the appropriate Drive message.

Note the following topic names for your publishers and subscribers (also detailed in the skeleton code):

- LaserScan: "/scan"
- Odometry: "/odom"
- Bool message: "/brake_bool"
- Brake message: "/brake"

You can implement this node in either C++ or Python, the skeleton code for both is found in the f1tenth/f110_ros repository in the **safety** folder: https://github.com/f1tenth/f110_ros.

NOTE: Make sure to press 'B' on your keyboard in the terminal window that launched the simulator. This will activate the AEB and allow the Behavior Controller to switch the Mux to Emergency Brake when the boolean is published as True. Pressing 'B' again will toggle the AEB off.

3 Deliverables and Submission

Submit the following as {student_name}.zip on Canvas by 11:59 pm on Jan. 29th.

- Your package including the safety node, named as {student_name}_safety (make sure it compiles before you submit after changing the package name)
- Make a video of teleop (manual driving) around the Levine loop (with your safety node on) in the simulator without significant false positives.

4 Grading

We will test your code by accelerating the car down a straight towards a wall in the Levine map, and your safety node should stop the car before collision.