Lab 2: Automatic Emergency Braking

Instructor: INSTRUCTOR *Name:* STUDENT NAME, *StudentID:* ID

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.

1   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

2   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 (TTC) using the `LaserScan` message in the simulator.

2.1   The `LaserScan` Message

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

2.2   The `Odometry` Message

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

2.3 The `AckermannDriveStamped` Message

`AckermannDriveStamped` is the message type that well use throughout the course to send driving commands to the simulator and the car. ROSs documentation is linked here: http://docs.ros.org/api/ackermannmsgs/html/msg/AckermannDriveStamped.html Note that we wont be sending driving commands to the car from this node, were 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.

2.4 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 cars velocity vector and the distance vector (which can be determined easily from information in the `LaserScan` message).

3 Automatic Emergency Braking 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/stdmsgs/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_labs/lab2` repository in the safety folder: https://github.com/f1tenth/f1tenth_labs/tree/master/lab2.

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.

4   Deliverables and Submission

Submit the following as `studentname_lab2.zip` (replace studentname with your name):

1. Your package including the safety node, named as `studentname_safety`. **Make sure it compiles before you submit after changing the package name.**

2. Make a youtube video of teleop (manual driving) around the Levine loop (with your safety node on) in the simulator without significant false positives. Add this link to a text file named `studentname_lab2_video.txt`

5   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.

5.1   Rubric

Topics	Points
Compilation	30
Provided Video	20
Correctly stops before collision	30
Correctly calculates TTC	10
Able to navigate through the hallway	10
Total	100

6   FAQ

Q: How should the TTC threshold be determined?
A: We recommend trial and error on a variety of situations (e.g. driving down the hall vs straight at a wall) to minimize false positives while preventing crashes. You can get a good initial estimate using the cars deceleration (8.26 $\frac{m}{s^2}$) and its velocity for keyboard driving (1.8 m $\frac{m}{s}$)

Q: Should false be published on the `brake bool` topic when the AEB is not in effect?
A: Yes. Publishing false will make sure that no Drive messages on the brake topic are sent to the Simulator, and will allow other controllers (e.g. keyboard) to take back control.

Q: If the AEB is not engaged, should a `AckermannDriveStamped` message be published on the brake topic?
A: No, the `brake` topic is meant to only have messages that bring the car to a stop, so there is no need to send anything in this case. Moreover, if the AEB is not engaged, messages on the brake topic will not be sent to the Simulator.

Q: Does the order of messages published matter? That is, is there a difference between:
`brake_publisher.publish(brake_msg);`
`brake_bool_publisher.publish(brake_bool_msg);`
and
`brake_bool_publisher.publish(brake_bool_msg);`
`brake_publisher.publish(brake_msg);`
A: No. Due to latency between sending and receiving messages, this realistically wont matter.

Q: How are the coordinate frames of the `LaserScan` and `Odometry` data oriented with respect to the car?
A: The `Odometry` data is in the frame centered on the rear axle, seen below. The message gives the magnitude and sign of the cars velocity. It is always directed in the `+x` direction (red axis). The `LaserScan` data is in the LiDars frame, seen below towards the front of the car.
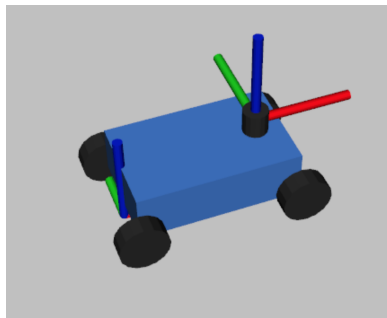
Q: When testing driving forwards and in reverse, the cars distance from the wall after braking seems different between these two cases. Am I doing something wrong, or should I try and use different TTC thresholds?
A: The ranges in the `LaserScan` message are from the LiDars frame (seen above) which is not in the center of the car. An ideal AEB system would subtract the distance from the LiDar to the edge of the car for each beam, and use that distance in the TTC calculation. A two-threshold system could work too, but for this assignment, as long as it doesnt crash during keyboard teleop,

a single threshold value is good enough. More sophisticated solutions are always welcome!

Q: After the car is stopped by the AEB, if I drive the car towards the wall again, it will be stopped by the brake again. But if I repeat this, the car will move closer and closer to the wall until a collision occurs before the AEB can stop the car. Is this be acceptable or should the AEB prevent collisions in all circumstances?

A: A collision in this unique situation is fine. When the car is extremely close to the wall and begins to move, it is impossible to stop it in time.