5.Lidar guard

This chapter needs to be used with a car chassis and other sensors to operate. This is just an explanation of the implementation method. In fact, it cannot be run. It needs to be used with the company's rosmaster-X3 car to achieve this function.

If you need to transplant it to your own motherboard, you need to install other dependency files.

1. Function Description

After the program is started, the car will track the nearest point target.

When the target point moves, it will move with the target point.

When the target point is close to the car and is less than the set distance, the buzzer will sound until the target point is far away from the set distance of the car.

2. Code path

```
~/ydlidar_ws/src/yahboomcar_laser/yahboomcar_laser/laser_warning.py
```

3. Start up

Input following command:

```
ros2 launch yahboomcar_laser laser_warning_launch.py
```

4. Core code analysis

laser_warning_launch.py

```
from launch import LaunchDescription
from launch_ros.actions import Node

import os
from ament_index_python.packages import get_package_share_directory
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource

def generate_launch_description():
    lidar_node = IncludeLaunchDescription(
        PythonLaunchDescriptionSource([os.path.join(
         get_package_share_directory('rplidar_ros'), 'launch'),
         '/lidar.launch.py'])
      )

    laser_warning_node = Node(
        package='yahboomcar_laser',
        executable='laser_warning',
    )
    bringup_node = IncludeLaunchDescription(
        PythonLaunchDescriptionSource([os.path.join(
        get_package_share_directory('yahboomcar_bringup'), 'launch'),
        '/yahboomcar_bringup_X3_launch.py'])
    )

    launch_description = LaunchDescription([
        lidar_node,
        laser_warning_node,
        bringup_node
        ])
    return launch_description
```

Start the lidar lidar_node, chassis bringup_node and tracking function node laser_warning_node.

laser_warning.py

Lidar callback function, here explains how to obtain the obstacle distance information at each angle, then obtain the ID of the minimum distance, calculate the angular velocity, and then compare the minimum distance with the set distance. If it is less than the set distance, Then let the buzzer sound, and finally publish the speed topic data.

```
  def registerScan(self, scan_data):
        if not isinstance(scan_data, LaserScan): return
        ranges = np.array(scan_data.ranges)
        minDistList = []
        minDistIDList = []

        for i in range(len(ranges)):
            angle = (scan_data.angle_min + scan_data.angle_increment * i) *
RAD2DEG
            if abs(angle) < self.laserAngle and ranges[i] != 0.0:
                minDistList.append(ranges[i])
                minDistIDList.append(angle)
        if len(minDistList) == 0: return
        minDist = min(minDistList)
        minDistID = minDistIDList[minDistList.index(minDist)]
        if self.Joy_active or self.Switch == True:
            if self.Moving == True:
                self.pub_vel.publish(Twist())
                self.Moving = not self.Moving
            return
        self.Moving = True
        if minDist <= self.ResponseDist:
            if self.Buzzer_state == False:
                b = Bool()
                b.data = True
                self.pub_Buzzer.publish(b)
                self.Buzzer_state = True
        else:
            if self.Buzzer_state == True:
                self.pub_Buzzer.publish(Bool())
                self.Buzzer_state = False
        velocity = Twist()
        ang_pid_compute = self.ang_pid.pid_compute((180 - abs(minDistID)) / 36,
0)
        if minDistID > 0: velocity.angular.z = -ang_pid_compute
        else: velocity.angular.z = ang_pid_compute
        if ang_pid_compute < 0.02: velocity.angular.z = 0.0
        self.pub_vel.publish(velocity)
```