第二十七届中国机器人及人工智能大赛

智能驾驶比赛项目

技 术 报 告

学　　校：广东工业大学

队伍名称：光阴之外

参赛队员：廖卓远、孙梓灏、李运涛

带队教师：苏畅、龙超

关于技术报告和研究论文使用授权的说明

本人完全了解第二十七届中国机器人及人工智能大赛有关保留、使用技术报告和研究论文的规定，即：参赛作品著作权归参赛者本人，比赛组委会和赞助公司可以在相关主页上收录并公开参赛作品的设计方案、技术报告以及参赛模型车的视频、图像资料，并将相关内容编纂收录在组委会出版论文集中。

参赛队员签名：

带队教师签名：

日　　期：2025.06.08

目　录

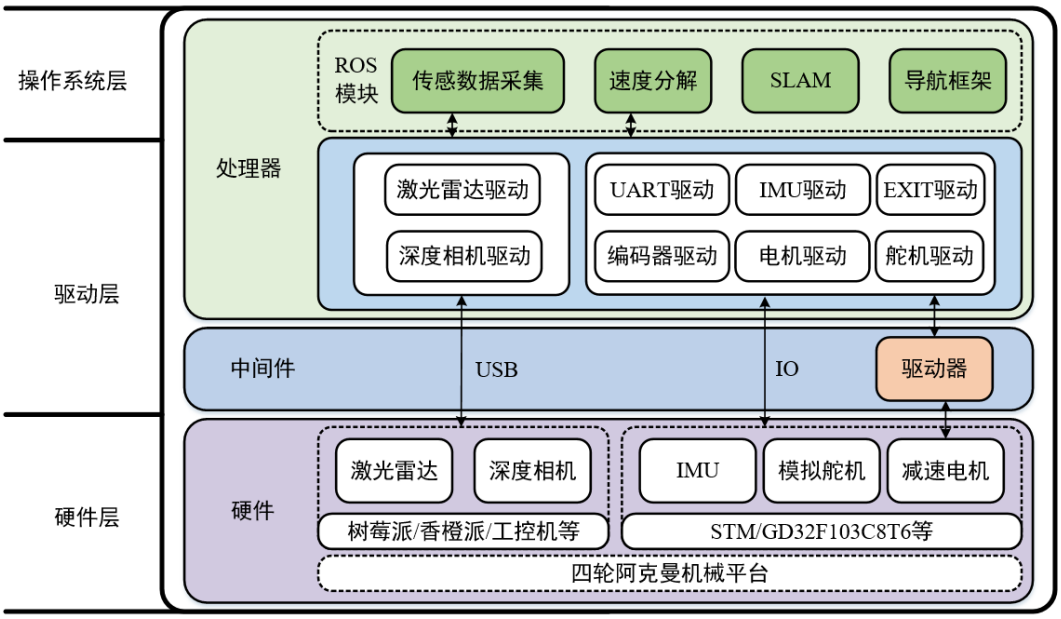
第1章 方案概述

一、技术实现路线

在硬件搭建上，小车选用了经典的基于阿克曼（Ackermann）原理的底盘结构，该结构作为常见的底盘结构设计，常用于多种类型的移动机器人、汽车设计和机械研究当中。其基本观点是： 汽车在行驶过程中，每个车轮的运动轨迹，都必须完全符合它的自然运动轨迹，从而保证轮胎与地面间处于纯滚动而无滑移现象。[1]其底盘主要由驱动轮、转向轮和底盘框架构成，通过电机驱动和转向控制实现小车的灵活移动与转向。[2]

电路链接图？

小车的总体架构图如下。



小车安装了多种传感器，包括 2D 激光雷达、里程计、IMU（惯性测量单元）和摄像头等，它们都与 ROS 控制板进行了连接，并且可以通过 ROS 系统中对传感器驱动和参数的配置去控制它们。例如，摄像头通过 USB 接口连接，配置分辨率和帧数这两个数据，并启动 ROS 节点用于捕获图片；激光雷达通过 USB 或者网口连接，配置了扫描频率和角度分辨率，用于提供周围环境的详细信息，帮助车辆准确判断方位；IMU 则通过串口或者 12C 接口连接，配置采样频率和坐标系转换，用于实时提供车辆的加速度和线速度等信息；里程计数据源于编码器，需配置相关参数用于获取小车位置和速度信息。

多传感器数据融合方法是小车实现自主导航和避开路障的核心方法，本系统采用基于概率的数据融合方法，将激光雷达、摄像头、IMU、里程计和 car 芯片等传感器的数据融合，提升环境感知的准确性和鲁棒性。在具体实现上，利用卡尔曼滤波、粒子滤波等算法对传感器数据进行融合处理。[3]

在建图算法方面，通过 Gmapping 技术与 RBPF 算法的结合，利用激光雷达所提供的环境信息、小车的运动模型和观测模型，结合粒子滤波算法估计小车的大概位置和地图信息，成功构建地图。

在导航算法方面，基于 A*算法的全局路径规划和基于 DWA（Dynamic Window

Approach）算法的局部路径规划，实现小车点到点寻路局部最优和起点到终点的全局最优的最优解，并且还能实时调整小车的速度和方向，确保小车自主导航、避开路障和跟随全局路径。

对于小车的移动和操作控制，提供了两种方式。一种是键盘控制，一种是通过自主移动模式下的单点导航和多点导航。键盘控制是在终端中输入 ros2 run racecar racecar_teleop.py 脚本，利用键盘控制小车的前进、后退、左转、右转和强制停止等操作，例如空格和 K 键强制停止，W 和 X 键将小车角速度向左调整了 5pwm，这些指令通过 ROS 话题（如 cmd_vel）发送给小车。另一种是通过自主移动模式下，打开传感器，选择目标点，小车会根据导航算法和传感器数据自主规划路径并移动，无需通过键盘发送指令，而是由 ROS 系统内部的节点和算法自动完成。本次任务就是主要由第二种移动方式完成。

二、创新点

1. GMapping 算法学习及地图构建:

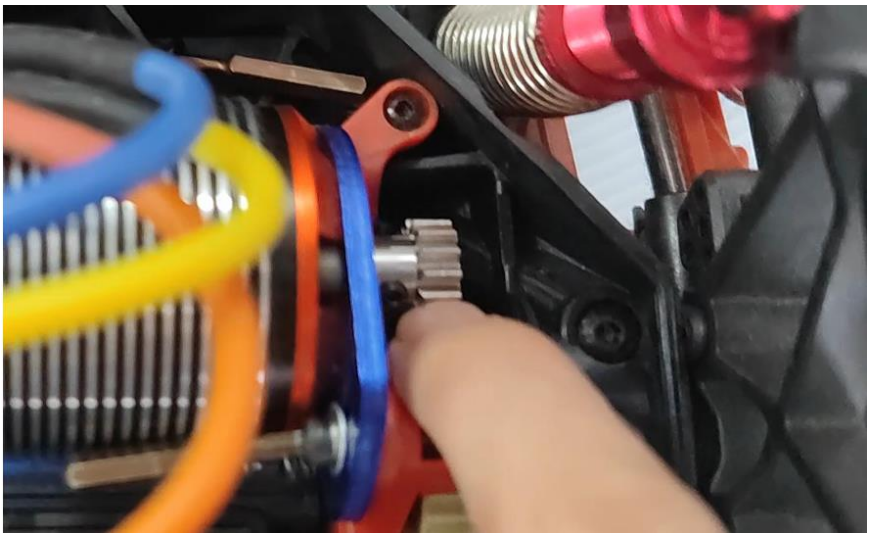
为实现同步定位与地图构建 (SLAM)，小车应用了 Gmapping 技术。该技术依托于 Rao-Blackwellized 粒子滤波 (RBPF) 框架，可实时生成并维护基于概率的二维栅格地图。Gmapping 在提升地图精度和保障运动过程中实时更新能力方面表现突出，有效支撑了后续的自主导航与路径规划。该算法流程具体为：依赖里程计信息初始化粒子位姿（先验分布），随后利用激光雷达扫描数据与候选地图的匹配程度计算粒子权重（似然函数），进而通过高权重粒子集近似表征机器人的真实状态(后验估计)。基于其鲁棒性和高效性,Gmapping 被广泛认为是 ROS 中 navigation 导航包集中推荐的二维建图算法包之一。

2. 导航算法的优化:

在传统机器人导航算法中，地图栅格化后，机器人轨迹规划多依赖于 A* 算法或 Dijkstra 搜索算法。这些方案大多基于差速驱动机器人模型。但是显而易见，对于受非完整约束（即采用阿克曼转向机构的车辆）的传统四轮车辆而言，此类规划出的路径通常难以有效跟踪实现。因此，我们尝试着优化导航算法，使用了适用于 ROS 的混合 A* (Hybrid A*) 路径规划器插件。

3.阿克曼底盘结构的应用与优化:

原小车因避震弹簧刚度不足及阻尼器阻力偏小的问题，导致转弯工况下侧偏角过大，进而影响了激光雷达与 IMU（惯性测量单元）的环境感知与位姿估计精度。由此构建的局部地图存在偏差，致使导航系统失效，表现为小车定位失准、易偏离预定路径，并存在碰撞锥桶或挡板的风险。为解决这些问题，小车在机械结构上进行了针对性改进，特别是在底盘采用了阿克曼转向结构。该结构设计不仅显著提升了车辆行驶稳定性，更通过精确的电机驱动与转向协同控制，实现了灵活高效的移动与转向能力，赋予小车更好的机动性，使其能够适应复杂多变的地形与环境，从而有效提升了整体导航性能与可靠性。如下图调整了底盘-齿间距和舵机中值。



4. 通过激光雷达数据分析确定锥桶位置：

激光雷达系统通过发射激光束来探测目标的位置、速度等特性。其运作机制是首先发射探测激光束探测到目标物体，然后将反射信号与原始发射信号比较，经过处理后可获取目标的相关参数，这些参数包括但不限于距离、方位、高度、速度、姿态乃至形状。相比于传统微波雷达，相较于传统微波雷达，激光雷达以波长更短、方向性更优的激光束作为探测媒介，工作频率显著提高。因此，激光雷达具备高分辨率、强隐蔽性及卓越的抗有源干扰能力等核心优势。

5. 代码的优化和一键启动脚本

小车在行驶过程中速度过快，容易撞上场地和障碍物，而且难以用键盘控制小车急停，并且有时在以此编写了一个控制小车暂停的代码，用于急停小车。同时由于实验过程中需要反复多次启动小车以测试效果，为了方便启动流程和实验的进行，我们编写了一个脚本用于一键启动小车的开启。

6. 使用深度学习 Yolov5n 进行视觉识别

小车在行驶过程中需对交通标志（如斑马线、红绿灯）进行识别和响应。系统引入了基于深度学习的 YOLOv5n 目标检测模型。该模型以其优异的实时性和轻量级特性著称，特别适合部署于移动平台资源受限的计算单元。通过车载摄像头实时采集图像流，YOLOv5n 模型能够高效地完成斑马线区域的语义分割及红绿灯状态（红灯/绿灯）的分类识别，输出高置信度的检测框与类别信息。识别结果被无缝集成至小车的决策控制系统中，触发精确的制动指令，确保小车能在检测到斑马线或红灯时及时、稳定地停止，完成了所给的任务。

三、测试效果

经实验场地测试验证，雷达系统在广域探测性能方面表现卓越，尤其在高速工况下仍保持优秀的检测能力。在此过程中，卡尔曼滤波技术的应用显著提升了系统鲁棒性，其通过概率模型有效抑制了误判风险，并显著降低了噪声对检测结果的干扰。进一步融合 IMU 与编码器数据后，系统在高速状态下实现了厘米级导航定位精度，且基本消除了误判现象，传感器综合性能达到预期设计目标。

测试表明，系统在大转角、直线路径、S 型弯道及大曲率圆弧等复杂路况下均保持稳定可靠的检测准确性，全程未出现误判情况。在 Gmapping 建图方面，虽整体效果良好，但由于测试环境中存在的一些特定因素，如挡板密封不完全导致激光束穿过缝隙并返回数值，以及测试场地地面不平整，存在凸起或凹坑，这些因素可能会对 IMU 和编码器的性能产生微弱影响,进而对建图效果造成一定影响。

第2章 问题描述

一、 应用问题：

在观察小车的点位的时候，我们不难发现，小车的点位具有四个数值。

0.532031	-0.20347	-0.09096	0.995855
2.373739	-1.29312	0.120228	0.992746
4.867598	0.188687	0.455626	0.890171

但是在正常的理解中，坐标系应该只有 X、Y、Z 三个轴，因此对这个问题进行了研究。

由于欧拉角在表示姿态时会遇到万向锁的问题，这就导致同一种空间状态欧拉角的表示方式不唯一，当出现万向锁现象时，同一种旋转有无数种欧拉角表示形式，从而导致了欧拉角差值时出现问题。[4]所以为了解决这些问题，数学上 想出了用四元数的形式来表征姿态的方法。 四元数是由 1 个实数加上 3 个复数组合而成，通常可以表示成 w+xi+yj+zk 或者（w，(x，y，z)），其中 w、x、y、z 都是实数，而 i^2=j^2=k^2=-1，i^0=j^0=k^0=1。而对于四元数的运算法则，其大体运算法则与拥有三个坐标系的空间坐标点类似，详见如下。[5]

四元数的和：$q_1+q_2=\left(w_1+w_2,\left(v_1+v_2\right)\right)$

四元数乘法：

$$q_1q_2=w_1w_2-v_1\cdot v_2+w_1v_2+w_2v_1+v_1\times v_2=\left(w_1w_2-v_1\cdot v_2,\left(w_1v_2+w_2v_1+v_1\times v_2\right)\right)$$

四元数的模：$\left\|q_1\right\|=\sqrt{w_1^2+x_1^2+y_1^2+z_1^2}$

四元数的共轭：$q_1^*=\left(w_1,-v_1\right)$

四元数的逆：$q_1^{-1}=\frac{q_1^*}{\left\|q_1\right\|}$

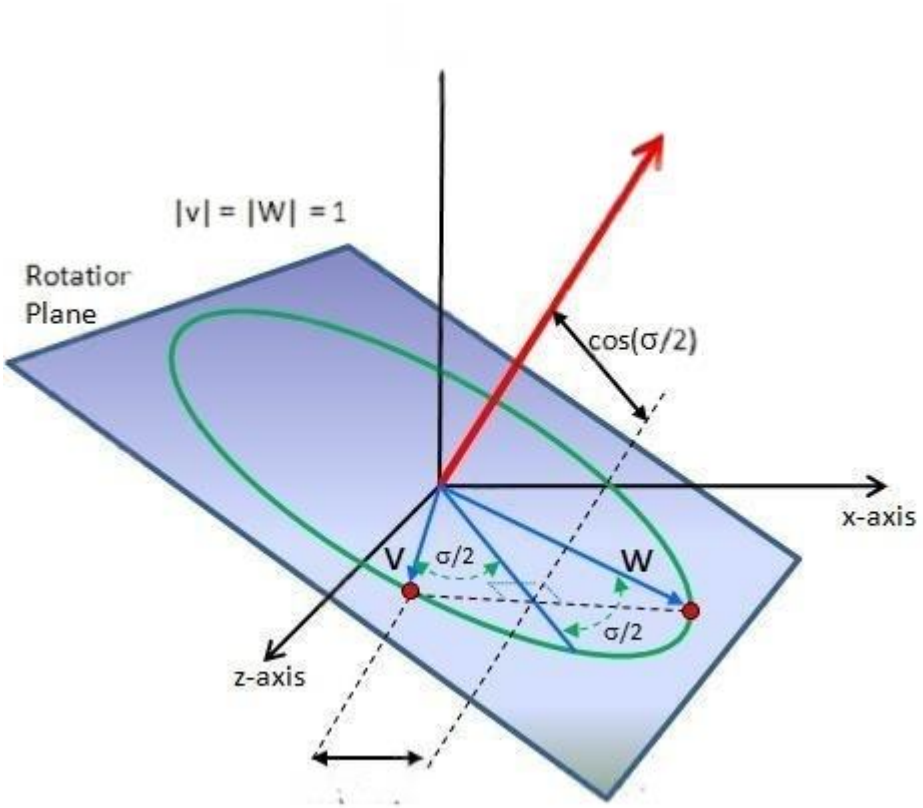
为了防止万象锁现象和其导致的数学问题，采用了表示简单的四元数，只有四个元素，且计算量小，但是不直观。

在使用自主导航的时候，首先要在传感器内获取导航的目标点，在 Rviz 运行界面标记目标点后运行 get.py，可以将标记的坐标点写入 csv 文件中，详细代码见下。

```
class GoalPoseSubscriber(Node):
    def __init__(self):
        super().__init__('get_point')
        self.subscription = self.create_subscription(
            PoseStamped,
            '/goal_pose',
            self.pose_stamped_cb,
            10)
        self.subscription  # prevent unused variable warning
        self.points = []
        self.settings = termios.tcgetattr(sys.stdin)

    def pose_stamped_cb(self, msg):
        gx = msg.pose.position.x
        gy = msg.pose.position.y
        gz = msg.pose.orientation.z
        gw = msg.pose.orientation.w
        self.points.append([gx, gy, gz, gw])

    def get_key(self):
        tty.setraw(sys.stdin.fileno())
        rlist, _, _ = select.select([sys.stdin], [], [], 0.1)
        if rlist:
            key = sys.stdin.read(1)
        else:
            key = ''
        termios.tcsetattr(sys.stdin, termios.TCSADRAIN, self.settings)
        return key


def data_write_csv(self, file_name, datas):
    with codecs.open(file_name, 'w+', 'utf-8') as file_csv:
        writer = csv.writer(file_csv, delimiter=',', quoting=csv.QUOTE_MINIMAL)
        for data in datas:
            writer.writerow(data)
    self.get_logger().info('Data written to CSV successfully!')
```

为了更好的理解自主导航运行以及四元数和欧拉角之间的关系，通过查阅资料和总结，有以下关系。

假设存在一根旋转轴 u，有一个绕 u 轴旋转 σ 角度的这么一个旋转存在，其中 u 是旋转轴的单位向量，q 是一个单位四元数。



它对任何向量施加以下算子运算后可以得到该向量绕 u 轴旋转 σ 角度后的向量：

$$w = qvq^*$$

并且它们两者可以通过以下公式转换。

$$q0 = \cos(\alpha/2)\cos(\beta/2)\cos(\gamma/2) + \sin(\alpha/2)\sin(\beta/2)\sin(\gamma/2)$$

$$q1 = \sin(\alpha/2)\cos(\beta/2)\cos(\gamma/2) - \cos(\alpha/2)\sin(\beta/2)\sin(\gamma/2)$$

$$q2 = \cos(\alpha/2)\sin(\beta/2)\cos(\gamma/2) + \sin(\alpha/2)\cos(\beta/2)\sin(\gamma/2)$$

$$q3 = \cos(\alpha/2)\cos(\beta/2)\sin(\gamma/2) - \sin(\alpha/2)\sin(\beta/2)\cos(\gamma/2)$$

$$\alpha = \text{atan2}(2(q0q1 + q2q3), 1 - 2(q1^2 + q2^2))$$

$$\beta = \text{asin}(2(q0q2 - q3q1))$$

$$\gamma = \text{atan2}(2(q0q3 + q1q2), 1 - 2(q2^2 + q3^2))$$

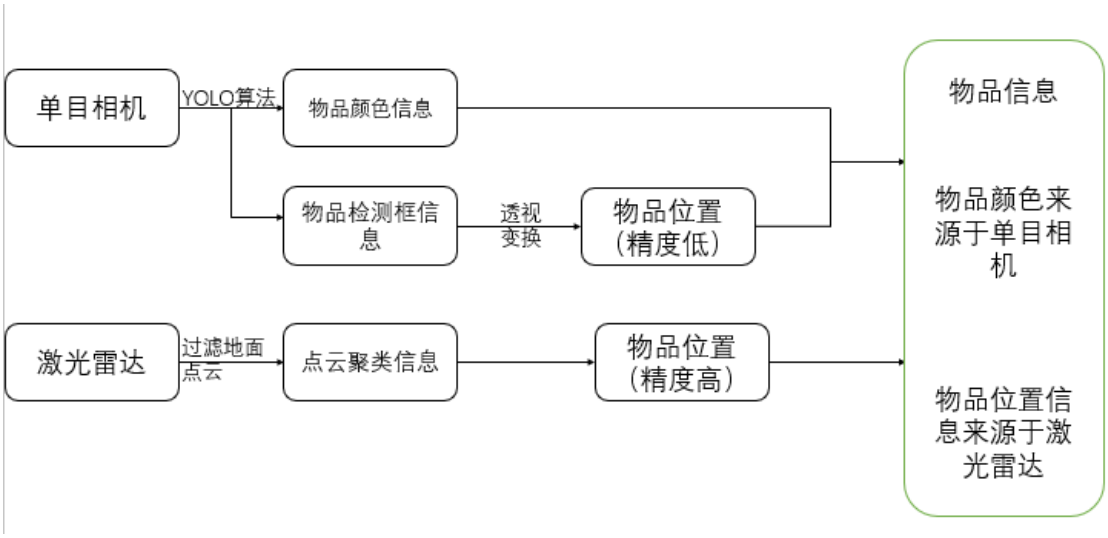
二、视觉问题：

1. 红绿灯和斑马线的视觉识别：

以红绿灯为例，在小车的红绿灯识别与定位流程中，我们融合了激光雷达

的高精度测量能力和基于 Python 的 OpenCV 强大的图像处理能力。首先，通过激光雷达向周围环境发射激光束并接收反射信号，精确测量物体的距离和角度信息，利用这些数据，我们可以在车辆坐标系中初步确定红绿灯的三维空间位置，为后续视觉识别提供关键的搜索区域。随后，我们利用 OpenCV 库加载并处理激光雷达获取的区域图像，通过颜色空间转换、灰度化、边缘检测和分割等图像预处理技术突出红绿灯特征，并结合图像处理算法如 Canny 边缘检测、霍夫变换等，实现红绿灯的轮廓识别和定位。这种方案不仅确保了定位的精确性，还通过 OpenCV 的辅助识别功能增强了识别的准确性和鲁棒性。同时，由于激光雷达和 OpenCV 都具有实时性高、灵活性强以及多平台支持的特点，该方案在导航小车面对复杂环境时能够快速、准确地响应红绿灯区域，并具有良好的可定制性和广泛的适用。

虽然单相机图像具有丰富的语义信息，可以直接得到红绿灯的颜色。但是，相机测距存在环境适应性差、精度低等问题，其物品输出结果也会和真实信息存在较大差别。单相机视觉方案通常依赖于图像处理技术来识别和定位物体，这包括边缘检测、特征点匹配等方法。这些方法在精度上通常不如激光雷达，因为激光雷达能够提供直接的物理距离测量。激光雷达通过测量激光束从发射到接收的时间差来确定距离，这种测量方式具有较高的精度和稳定性。以下是检测物品的总流程。



激光雷达测距的原理：LS01X 激光雷达采用了激光三角测距原理，开发的高频图像采集处理系统，默认工作的测量频率为每秒 14400 点， 每次测距过程中，LS01D 的脉冲调制激光器发射红外激光信号，该激光信号照射到目标物体后产生反射光斑，该反射光斑经过一组光学透镜由 LS01X 的图像采集 处理系统接收。经过 LS01X 的内嵌信号处理模块实时解算，目标物体与 LS01X 雷达的距离值以及相对方位角度值将从通讯接口中输出。 在机械旋转模块的带动下，LS01X 的高频核心测距模块将进行顺时钟旋转，从而实现对周围环境 360 度扫描测距。[6]

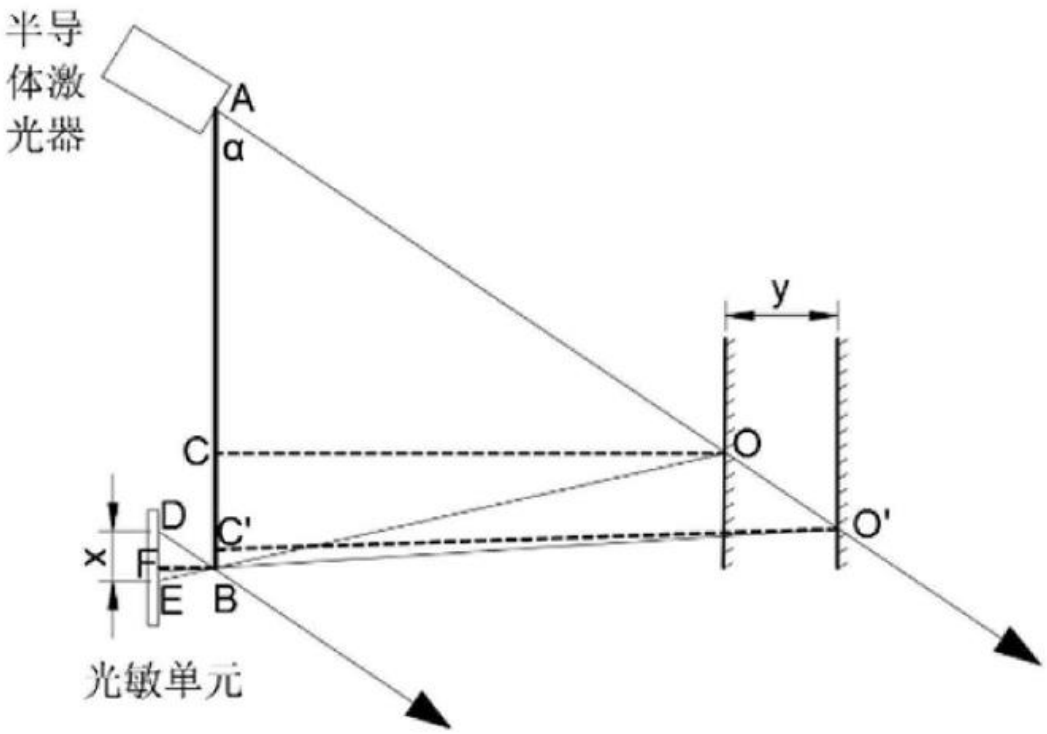


图 1：激光三角法斜射式光路图

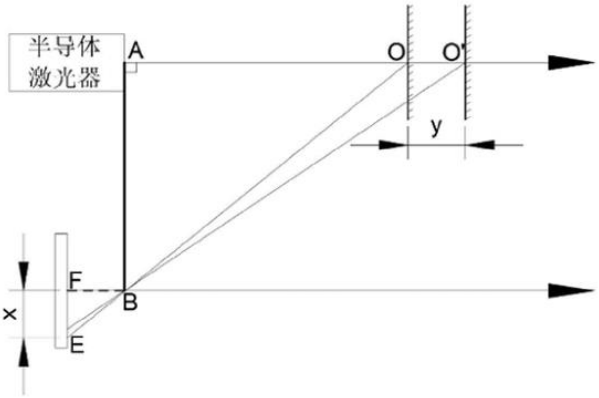


图 2：激光三角法直射式光路图

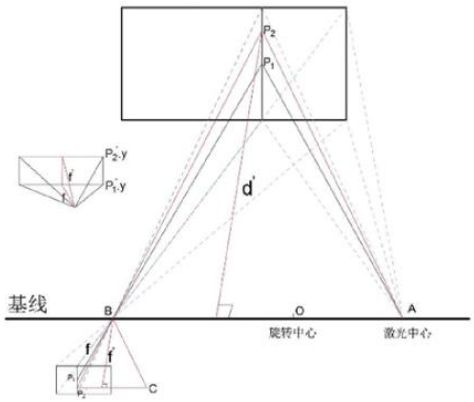
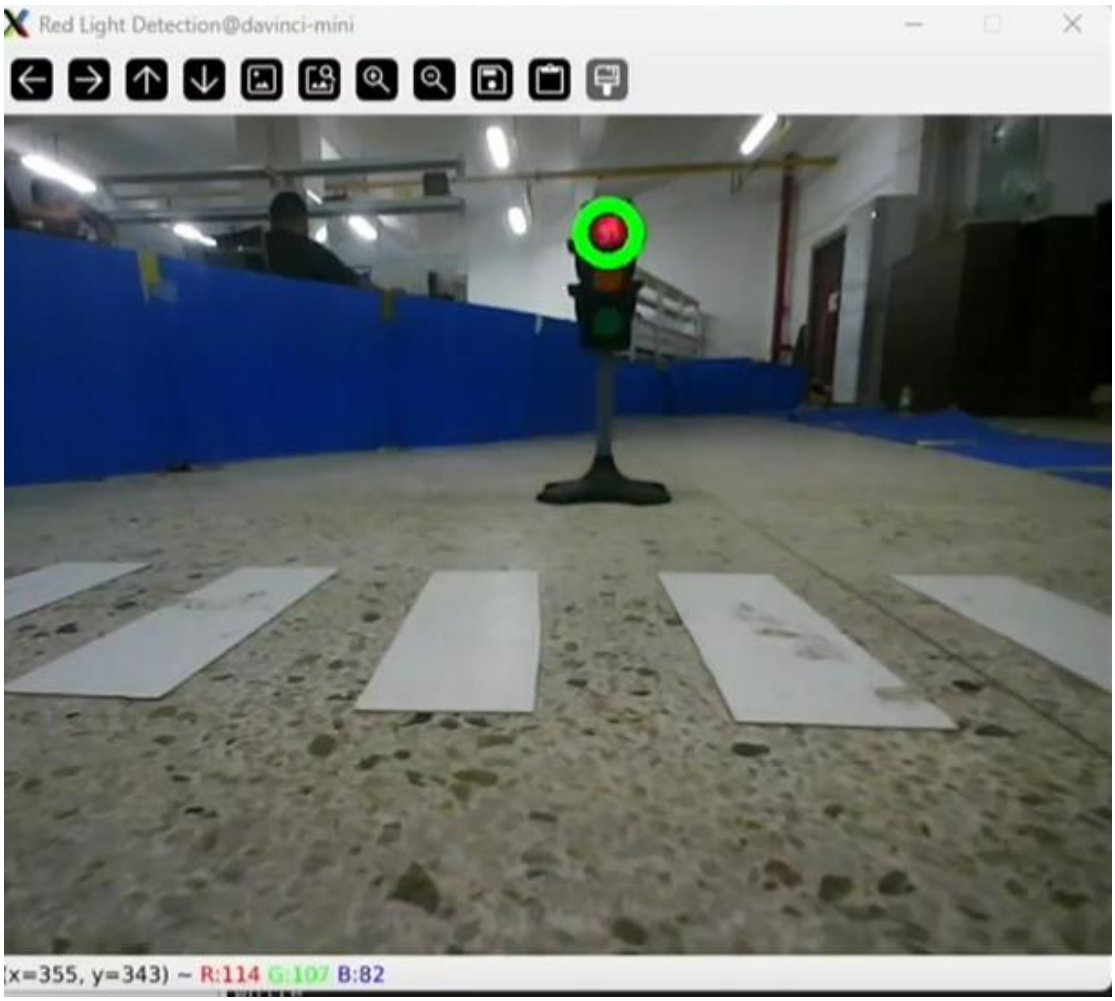
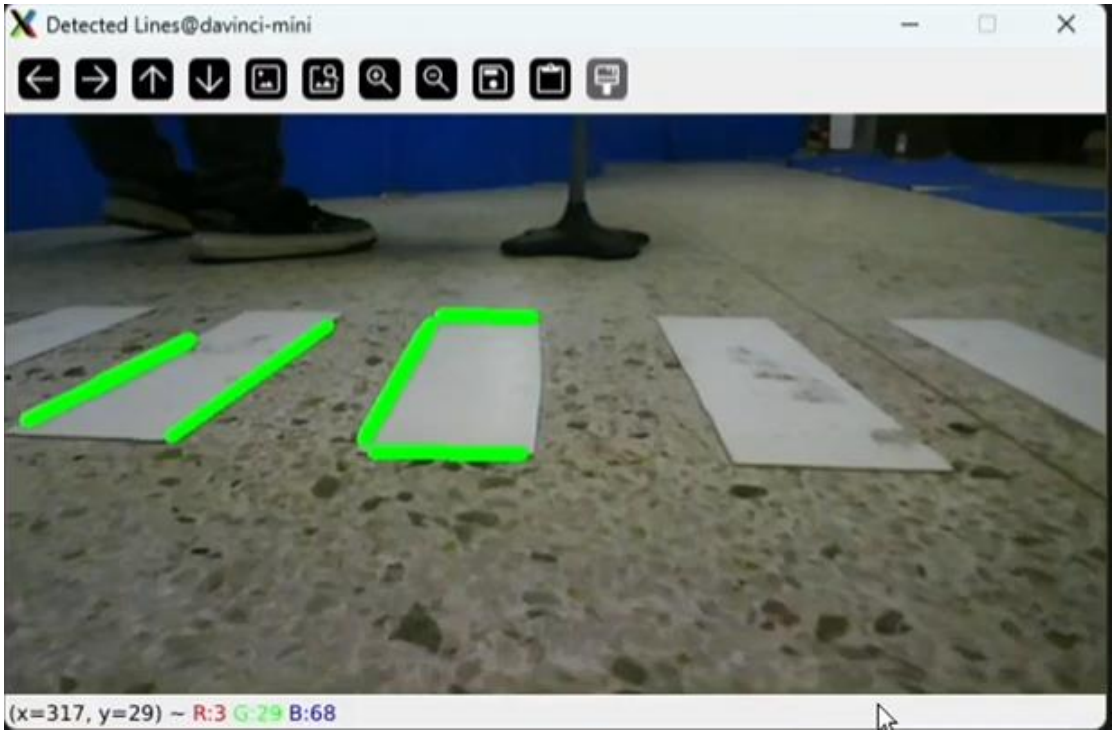


图 3：3D 扫描测距原理示意图

实际操作：通过进入小车路径，直接运行已经配置好的代码文件，出现两个窗口，一个对应的是红绿灯检测，另外一个对应的是斑马线检测。





经过测试和调整，我们摇晃红绿灯和调整位置之后也能成功识别。下面介绍

关键代码。

在红绿灯检测函数中，首先对输入图像进行预处理，应用中值滤波以抑制噪声，并采用高斯模糊技术进一步平滑图像。随后，将预处理后的图像从 BGR 色彩空间转换至 HSV 色彩空间，便于进行基于色相的精确颜色分割。针对红灯识别，算法根据红色在 HSV 空间中定义的色相、饱和度及明度范围阈值，从原始图像中分割并提取出潜在的红色区域。为提高分割区域的质量，对提取的红色区域进行形态学操作，包括侵蚀和膨胀。最后，算法通过轮廓检测识别处理后的区域轮廓，并依据轮廓面积等几何特征进行筛选与评估。若检测到的轮廓符合预设的大小及形状阈值，则判定为有效红灯信号。检测结果实时发布至名为 detect 的通信话题中，供下游模块订阅使用。该话题可以用于控制停车。

```
def red_light_detection(frame):
    # 设置默认参数值
    smooth_kernel_size = 21
    gaussian_blur_val = 15
    morph_kernel_size = 5
    min_area = 100
    max_area = 3000

    smooth = cv2.medianBlur(frame, smooth_kernel_size)
    gaussian_blurred = cv2.GaussianBlur(smooth, (gaussian_blur_val, gaussian_blur_val), 0)
    hsv = cv2.cvtColor(gaussian_blurred, cv2.COLOR_BGR2HSV)

    lower_red1 = np.array([0, 100, 100])
    upper_red1 = np.array([10, 255, 255])
    lower_red2 = np.array([170, 100, 100])
    upper_red2 = np.array([180, 255, 255])


    mask1 = cv2.inRange(hsv, lower_red1, upper_red1)
    mask2 = cv2.inRange(hsv, lower_red2, upper_red2)
    mask = mask1 + mask2

    kernel = np.ones((morph_kernel_size, morph_kernel_size), np.uint8)
    morph = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)

    contours, _ = cv2.findContours(morph, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

```
    detected = False
    output_image = frame.copy()
    for contour in contours:
        area = cv2.contourArea(contour)
        if min_area < area < max_area:
            (x, y), radius = cv2.minEnclosingCircle(contour)
            center = (int(x), int(y))
            radius = int(radius)
            circle_area = np.pi * (radius ** 2)

            if min_area < circle_area < max_area:
                cv2.circle(output_image, center, radius, (0, 255, 0), 8)
                label ="red"
                msg = String()
                msg.data = label
                publisher.publish(msg)
                print(label)
                detected = True

    cv2.imshow('Red Light Detection', output_image)

    if cv2.waitKey(1) & 0xFF == 27:
        return False

    return detected
```

在斑马线检测函数中，由于斑马线通常在下方，所以只保留图像的下半部分。首先将 RBG 彩图转化为灰度图，然后应用中值滤波或高斯模糊，减少图像噪声。之后使用 Canny 算法检测图像的边缘，通过使用形态学闭运算来连接相邻的边缘，形成连线的线条结构。最后，查找闭运算后图像的轮廓，创建感兴趣的区域 ROI，再在图像上画出线段表示斑马线。检测结果实时发布至名为 detect 的通信话题中，供下游模块订阅使用。该话题可以用于控制停车。

```python
def zebra_crossing_detection(frame):
    # 设置默认参数值
    crop_percent = 0.3
    median_blur_val = 15
    gaussian_blur_val = 0
    canny_threshold1 = 50
    canny_threshold2 = 134
    morph_kernel_size = 1
    area_threshold = 3000
    hough_threshold = 16
    line_threshold = 10

    # 处理图像
    height = frame.shape[0]
    cropped_image = frame[int(crop_percent * height):, :]

    gray = cv2.cvtColor(cropped_image, cv2.COLOR_BGR2GRAY)

    # 应用中值滤波
    if median_blur_val > 1:
        blurred = cv2.medianBlur(gray, median_blur_val)
    else:
        blurred = gray

    # 应用高斯模糊
    if gaussian_blur_val > 1:
        gaussian_blurred = cv2.GaussianBlur(blurred, (gaussian_blur_val, gaussian_blur_val), 0)
    else:
        gaussian_blurred = blurred

    edges = cv2.Canny(gaussian_blurred, canny_threshold1, canny_threshold2)

    kernel = np.ones((morph_kernel_size, morph_kernel_size), np.uint8)
    closed_edges = cv2.morphologyEx(edges, cv2.MORPH_CLOSE, kernel)
```

```python
    contours, _ = cv2.findContours(closed_edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    mask = np.zeros_like(closed_edges)
    for contour in contours:
        if cv2.contourArea(contour) > area_threshold:
            cv2.drawContours(mask, [contour], -1, (255), thickness=cv2.FILLED)

    roi = cv2.bitwise_and(cropped_image, cropped_image, mask=mask)

    roi_gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    roi_edges = cv2.Canny(roi_gray, canny_threshold1, canny_threshold2)

    lines = cv2.HoughLinesP(roi_edges, 1, np.pi / 180, threshold=hough_threshold, minLineLength=50, maxLineGap=1

    line_image = np.copy(cropped_image)
    if lines is not None:
        for line in lines:
            for x1, y1, x2, y2 in line:
                cv2.line(line_image, (x1, y1), (x2, y2), (0, 255, 0), 8)
                label ="white"
                msg = String()
                msg.data = label
                publisher.publish(msg)  #发布消息
                print(label)

    detected = lines is not None and len(lines) > line_threshold

    cv2.imshow('Detected Lines', line_image)

    if cv2.waitKey(1) & 0xFF == 27:  # 按下 'Esc' 键退出
        return False

    return detected
```

第3章 技术方案

一、整体方案：

我们的小车是一款基于阿克曼底盘结构的经典小车模型。为确保在各种环境下都能提供稳定且精确的数据支持，小车集成了里程计、IMU 惯导和激光雷达等高精度传感器。我们采用了卡尔曼滤波算法，有效地结合了里程计和 IMU 惯导的数据，显著提升了小车的定位精度，实现了多源数据的融合与高效利用。

在实现小车自主定位巡航的过程上，我们采用了动态窗口规划法（Dynamic Window Approach，DWA）作为核心路径规划算法。该算法基于构建好的栅格地图和预设规划好的坐标点位，能够规划出从起点到终点的最优路径。同时实现动态调整小车的行进速度和方向的策略，我们能够确保小车在移动过程中避开障碍物，并准确地到达目的地。

在地图构建方面，我们采用的是 Rviz 软件，借助了 odometry 和 gmapping 功能包，来达成实时处理来自传感器的数据的目的，并据此构建和优化环境地图。我们同时标记了关键坐标点，为地图的构建和导航任务提供了有力的支撑作用。

为了进一步增强了小车的定位准确性，我们结合了里程计的实时测量数据，实现了同步定位与地图构建（SLAM）技术。实时测量的左右轮转速信息被用来推演小车的轮式里程计信息，这一信息在 SLAM 过程中起到了关键的辅助作用。这样，我们不仅实现了小车的精准定位，也实现了其自主导航功能。[7]

在整个技术方案中，我们统筹机器人反馈的深度信息、IMU 信息和里程计信息，进行必要的参数调整规划，确保各个模块能够协同工作，实现小车的自主导航功能。

二、分模块详细介绍：

1、GMapping 算法及地图构建：

Gmapping 是基于粒子滤波算法实现的 SLAM。在 GMapping 算法中，里程计数据获取粒子群的先验位姿，预测机器人可能会存在的位置，再通过雷达数据与地图的匹配程度对所有粒子进行打分，并计算每个粒子在地图上的权重，选择权重较高的例子作为新的样本集，并根据它们拟合地图，最终构建出真实环境，是 ROS 中 navigation 导航包集中推荐的二维建图算法包。[8]

2、激光雷达数据接受分析与滤波处理

激光雷达是一种基于光学飞行测距的主动传感系统。其工作原理是向目标发射近红外波段的调制激光束，然后通过 APD/SPAD 探测器接受反射信号，将接受到的反射信号与发射信号进行比较，经过数据的处理，获得目标相关信息，如目标距离、方位、高度、速度、姿态、甚至形状等参数。与普通微波雷达相比，激光雷达使用的是激光束，工作频率较微波更高，具有亚厘米级的空间分辨率、微秒级的时间分辨率、隐蔽性好、抗有源干扰能力强等优点。 Laser_filters 包是 ROS 系统下的滤波功能包，主要内容是许多用于处理 sensor_msgs/LaserScan 消息的通用过滤器。这些过滤器导出为旨在与过滤器包配合使用的插件，且可直接在 sensor_msgs/LaserScan 上运行。[9]

利用 Ubuntu 操作系统控制 ROS 智能小车上的激光雷达，利用 roslaunch ls01b ls01b_v2.launch 开启激光雷达节点，读取激光雷达返回数据并发布，启动 laser_filters 功能包节点并且订阅激光雷达返回数据，对数据进行处理，得到经过滤波的数据。

3.数据融合：对 imu 和 odom 进行融合：

robot_pose_ekf 是 ROS Navigation stack 中的核心状态估计模块,通过扩展，卡尔曼滤波器对 imu、里程计 odom、视觉里程计 vo 等多个传感器的数据进行融合,有效得出平面移动机器人的真实位置姿态估计。robot_pose_ekf 专门为平面上的轮式移动机器人优化设计,因此 odom 信息中的 z,pitch 和 roll 分量可以被忽略。IMU 可以提供车体坐标系基于地球重力为参考系的绝对姿态(RPY 角),其中 Roll 和 Pitch 是绝对角度,因为有重力方向作为参考,而偏航角 Yaw 则是一个相对角度(如果 IMU 中没有集成电子罗盘测量地球磁场角作为参考)。IMU 姿态的协方差矩阵代表了姿态测量的不确定度。robot_pose_ekf 节点默认会从 odom、 imu_data、 vo 这三个 topic 上订阅消息,通过 remap 将其映射到新名称的 topic 上。重映射是基于替换的思想,每个重映射包含一个原始名称和一个新名称。每当节点使用重映射中的原始名称时,ROS 客户端库就会将它默默地替换成其对应的新名称。imu 信息的协方差矩阵中代表机器人航向角的分量方差为 1e-6,而里程计信息的协方差矩阵中机器人姿态分量的协方差为 1e3,两个值相差很大。在进行姿态融合策略时,对于 IMU 的权重分配会更重,里程计姿态信息被有效抑制。

◦ 协方差矩阵表征测量不确定性:

$$R_{imu} = \begin{bmatrix} \sigma_{\phi}^2 & 0 & 0 \\ 0 & \sigma_{\theta}^2 & 0 \\ 0 & 0 & 1 \times 10^{-6} \end{bmatrix}$$

位置协方差设置反映累积误差特性:

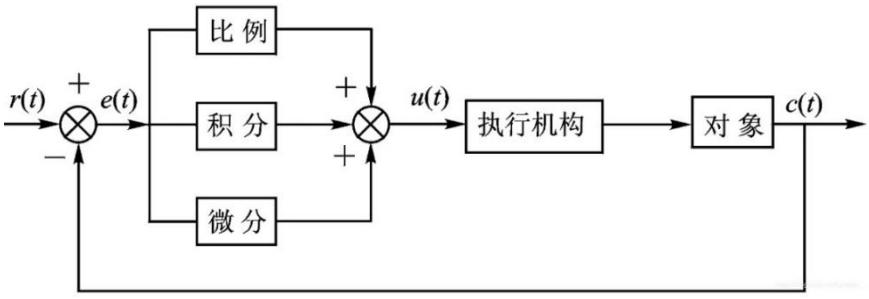
$$Q_{odom} = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & 1 \times 10^{3} \end{bmatrix}$$

4、基于 PID 控制下的舵机调节避障：

PID 是 Proportional（比例）、Integral（积分）、Differential（微分）的缩写，该控制算法是结合比例、积分和微分三种环节于一体的控制算法，也是连续系统中技术最为成熟、应用最为广泛的一种控制算法。[9]

舵机的控制通常需要一个时基脉冲，该信号由控制板发出。舵机的控制信号周期一般为 20ms，高电平控制舵机的转角，其脉冲宽度从 0.5-2.5 对应舵盘位置的 0-180 度，呈线性变化趋势。舵机的频率决定响应速度和扭矩，在高速过弯时，频率越高，响应速度越快，扭矩越弱，抓地力越小，可能导致行驶不稳，其输出角度只与提供脉宽有关，只有当改变脉冲宽度时，才会改变输出角度。

为了让智能小车能够顺利依照规定路径行进且不碰到障碍物（锥桶），我们通过激光雷达获取锥桶的准确位置，设置一定的范围距离参数，小车在距离参数以内时， PID 算法输出不断修正的脉宽信号，控制舵机的角度，实现避开障碍物，在规定轨迹上行进。[10]



$$\mathrm{U}(t) = kp(err(t) + \frac{1}{T_I}\int err(t)dt + \frac{T_D derr(t)}{dt})$$

第4章 方案实现

一. GMapping 算法及地图构建

（1）.回调函数，对雷达数据进行处理。

```
// 激光数据回调函数（核心入口）
void SlamGmapping::laserCallback(sensor_msgs::msg::LaserScan::ConstSharedPtr scan) {
    //扫描计数和节流处理
    laser_count_++;
    if ((laser_count_ % throttle_scans_) != 0) return;

    static tf2::TimePoint last_map_update = tf2::TimePointZero;

    //首次扫描初始化
    if(!got_first_scan_) {
        if(!initMapper(scan)) return;
        got_first_scan_ = true;
    }
//处理当前扫描
GMapping::OrientedPoint odom_pose;
if(addScan(scan, odom_pose)) {
    //获取最佳粒子位姿
    GMapping::OrientedPoint mpose = gsp_->getParticles()[gsp_->getBestParticleIndex()].pose;

    //计算坐标系变换
    tf2::Quaternion q;
    q.setRPY(0, 0, mpose.theta);
    tf2::Transform laser_to_map = tf2::Transform(q, tf2::Vector3(mpose.x, mpose.y, 0.0)).inverse();

    q.setRPY(0, 0, odom_pose.theta);
    tf2::Transform odom_to_laser = tf2::Transform(q, tf2::Vector3(odom_pose.x, odom_pose.y, 0.0));

    //更新map->odom变换（线程安全）
    std::lock_guard<std::mutex> lock(map_to_odom_mutex_);
    map_to_odom_ = (odom_to_laser * laser_to_map).inverse();

    //条件更新地图（0.5秒间隔）
    tf2::TimePoint timestamp = tf2_ros::fromMsg(scan->header.stamp);
    if(!got_map_ || (timestamp - last_map_update) > map_update_interval_) {
        updateMap(scan);
        last_map_update = tf2_ros::fromMsg(scan->header.stamp);
    }
}
```

（2）. 更新栅格地图

该函数把将栅格数据类型定义为 ScanMatcher,先创建一个该类型的对象然后获取最佳粒子储存进 particle best 中，之后发布位姿熵，创建好临时地图后，将 Gmapping 地图转为 ros 占用栅格。

```
// 更新并发布地图（核心建图函数）
void SlamGmapping::updateMap(const sensor_msgs::msg::LaserScan::ConstSharedPtr scan) {
    RCLCPP_DEBUG(this->get_logger(), "开始地图更新");
    std::lock_guard<std::mutex> map_lock(map_mutex_);

    //配置扫描匹配器
    GMapping::ScanMatcher matcher;
    matcher.setLaserParameters(scan->ranges.size(), &(laser_angles_[0]), gsp_laser_->getPose());
    matcher.setlaserMaxRange(maxRange_);
    matcher.setusableRange(maxUrange_);
    matcher.setgenerateMap(true);

    //获取最佳粒子
    GMapping::GridSlamProcessor::Particle best = gsp_->getParticles()[gsp_->getBestParticleIndex()];

    //发布位姿熵
    std_msgs::msg::Float64 entropy;
    entropy.data = computePoseEntropy();
    if(entropy.data > 0.0) entropy_publisher_->publish(entropy);

    //初始化地图（如果是第一次）
    if(!got_map_) {
        map_.info.resolution = delta_;
        map_.info.origin = geometry_msgs::msg::Pose();  // 初始化为零位姿
    }

    //计算地图中心
    GMapping::Point center;
    center.x = (xmin_ + xmax_) / 2.0;
    center.y = (ymin_ + ymax_) / 2.0;

    //创建临时地图
    GMapping::ScanMatcherMap smap(center, xmin_, ymin_, xmax_, ymax_, delta_);
```

```
//反向遍历轨迹树并注册扫描
for(GMapping::GridSlamProcessor::TNode* n = best.node; n; n = n->parent) {
    if(!n->reading) continue;

    // 计算有效区域并注册扫描
    matcher.invalidateActiveArea();
    matcher.computeActiveArea(smap, n->pose, &((*n->reading)[0]));
    matcher.registerScan(smap, n->pose, &((*n->reading)[0]));
}

//调整地图尺寸（如果地图已扩展）
if(map_.info.width != (unsigned int)smap.getMapSizeX() ||
   map_.info.height != (unsigned int)smap.getMapSizeY())
{
    // 计算新边界
    GMapping::Point wmin = smap.map2world(GMapping::IntPoint(0, 0));
    GMapping::Point wmax = smap.map2world(GMapping::IntPoint(smap.getMapSizeX(), smap.getMapSizeY()));
    xmin_ = wmin.x; ymin_ = wmin.y;
    xmax_ = wmax.x; ymax_ = wmax.y;

    // 调整ROS地图参数
    map_.info.width = smap.getMapSizeX();
    map_.info.height = smap.getMapSizeY();
    map_.info.origin.position.x = xmin_;
    map_.info.origin.position.y = ymin_;
    map_.data.resize(map_.info.width * map_.info.height);
}
```

```
//将GMapping地图转换为ROS占用栅格
for(int x=0; x < smap.getMapSizeX(); x++) {
    for(int y=0; y < smap.getMapSizeY(); y++) {
        GMapping::IntPoint p(x, y);
        double occ = smap.cell(p);  // 获取单元格占据概率

        // 转换为ROS占用值 (-1:未知, 0:空闲, 100:占据)
        if(occ < 0) {
            map_.data[MAP_IDX(map_.info.width, x, y)] = -1;
        } else if(occ > occ_thresh_) {
            map_.data[MAP_IDX(map_.info.width, x, y)] = 100;
        } else {
            map_.data[MAP_IDX(map_.info.width, x, y)] = 0;
        }
    }
}

// 设置地图头信息
got_map_ = true;
map_.header.stamp = get_clock()->now();
map_.header.frame_id = map_frame_;

//发布地图和元数据
sst_->publish(map_);
sstm_->publish(map_.info);
```

二. 对比赛场地进行构建。

1. 运行车

```
$ ros2 launch racecar Run_car.launch.py
```

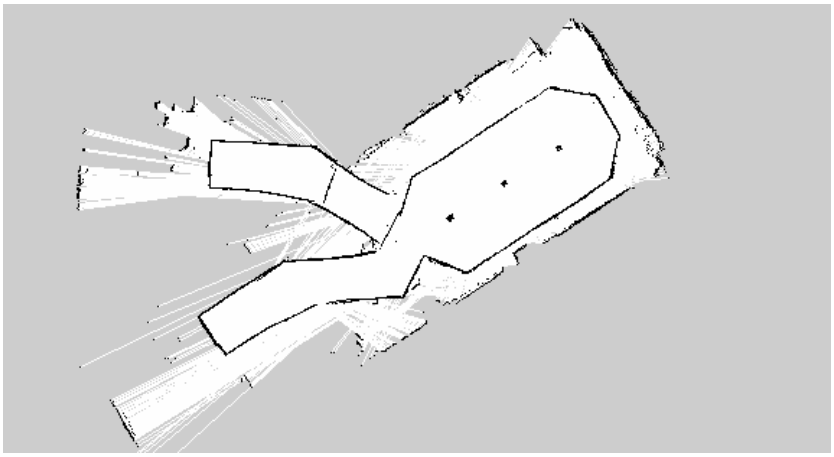
2. 运行 gmapping

```
$ ros2 launch slam_gmapping slam_gmapping.launch.py
```

3. 运行小车使小车边运行边建图

```
$ ros2 run racecar racecar_teleop.py
```

如下为我们通过 GMapping 算法完成的建图：



三. 基于 opencv 辅助识别功能调节避障

（1）在小车正前方将红色和蓝色的锥桶间距 1.5m 并排放，用来模拟比赛过程中的路况情况。

（2）打开终端，依次输入以下命令行；

找到视觉识别代码存放的路径

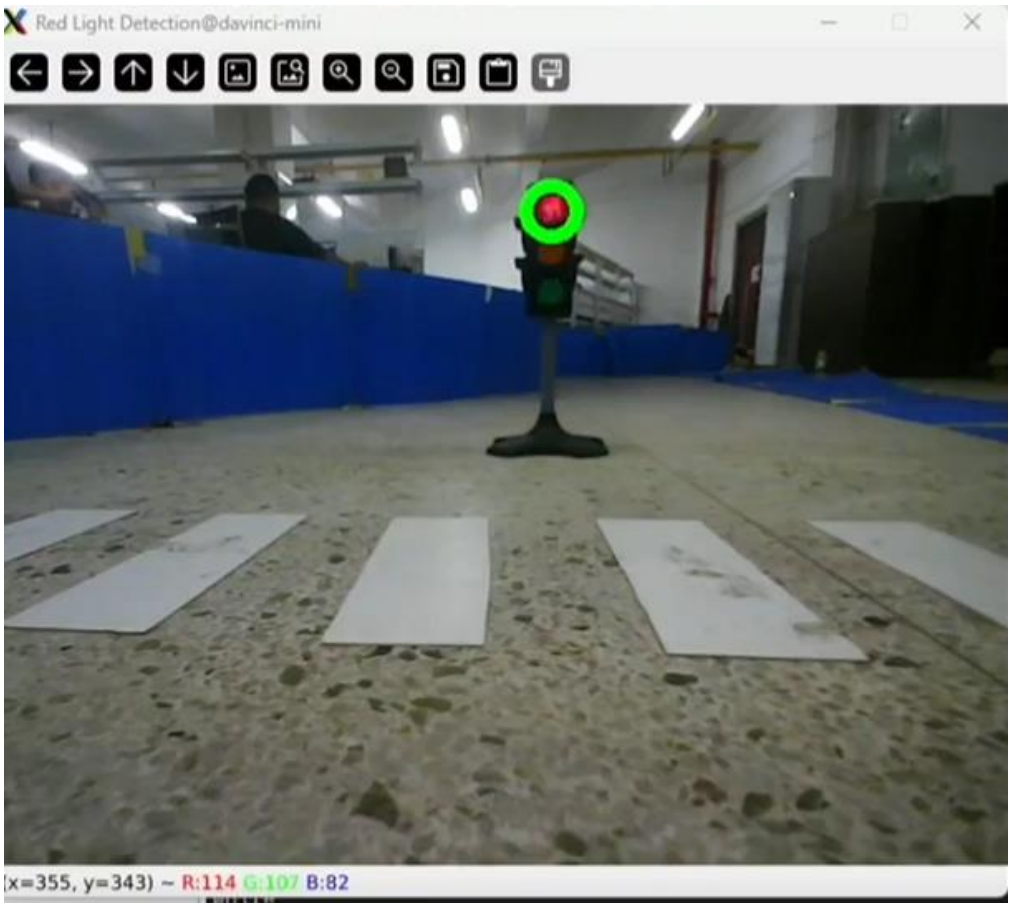
```
$ cd /home/davinci-mini/racecar/src/red/opencv_test
```

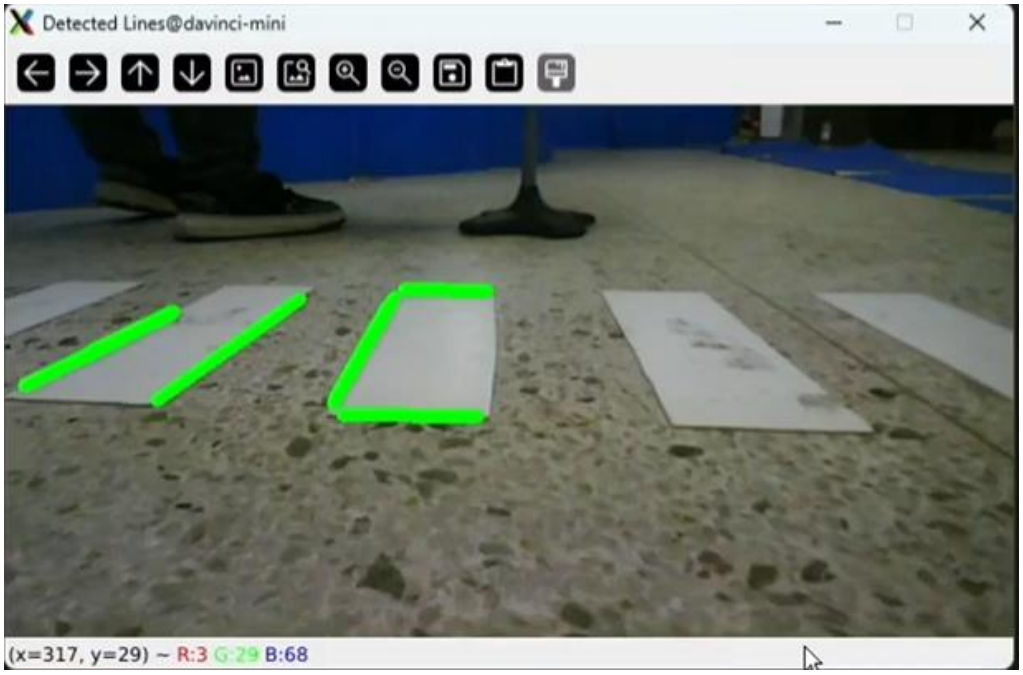
给小车摄像头的权限

```
$ sudo chmod 666 /dev/video0
```

运行视觉识别

```
$ python3 test.py
```





小车识别到前方有红绿灯和斑马线，具体实现过程中，我们采用中值滤波和高斯模糊的方法使得图新识别更加精准，并且运用形态学操作，侵蚀膨胀让红绿灯的轮廓闭合，再在摄像头窗口中查找轮廓，发布消息。

第5章 测试分析

【填写说明：通过测试与对比，论证系统的有效性，包括数据来源、数据规模、环境配置、测试过程、分析与结论等等。各参赛队务必重视数据测试，所有对自己作品准确性、有效性、稳定性，甚至作品受欢迎的程度的宣称，都应该得到数据结果或对比实验的支持，否则评审人有理由怀疑其真实性】

一、里程计调整分析：

通过调整小车的里程计公式，校准小车的里程计，使小车的初始点位和建图点位识别得更加清晰，运行过程的途径更加精确，行驶的稳定性也上升了。

打开终端输入以下命令行：

```
$ cd racecar
bash gmapping
ros2 topic echo /odom_combined | grep -A8 pose
```

即可打印小车的里程计信息，

```
pose:
  pose:
    position:
      x: 0.0
      y: 0.0
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 1.3250734547665935e-05
```

通过移动小车，小车的 x 和 y 轴陆续增加，我们将小车移动了 3.0m，但是里程计上打印的和 3.0m 有些许偏差。

```
pose:
  pose:
    position:
      x: 3.2153242019789268
      y: 0.9999702797887137
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 0.015458035728985109
```

K = 实际里程 / 打印里程

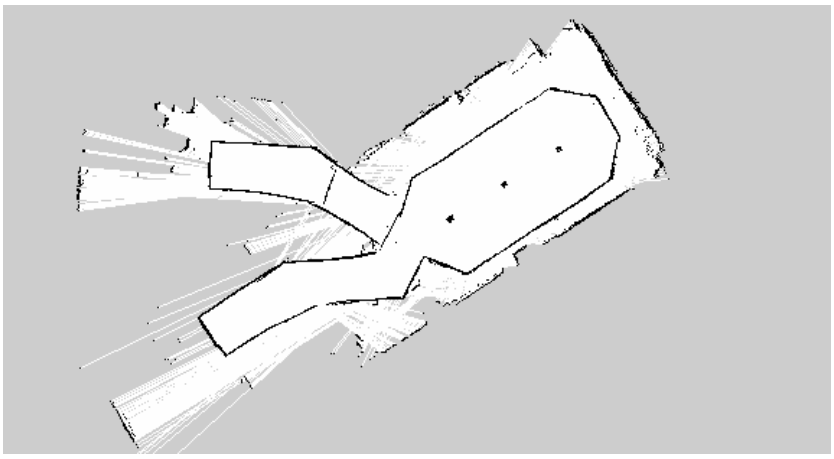
$Vel = K * v_{angle} * C / 1024.0 / 0.02 * k_{self} * 0.25 * 1.03$

通过上述公式，我们修正了小车的里程计不准确的问题。

二、建图精度分析：

通过 GMapping 算法实现定位建图，使小车运行稳定性大幅提升，建立的地

图也更加精准，改进结构后的建图如下：

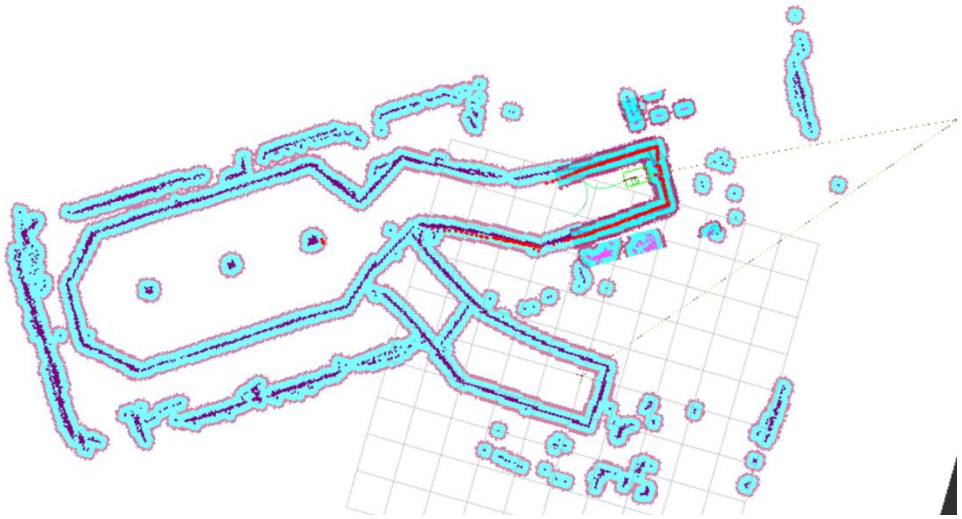


三、膨胀系数的调整分析：

在小车的运行过程中，如果路径规划出现偏差，导致目标点和小车距离过近，若小于膨胀距离，便会导致路径规划失败，所以在全局和局部规划器[9]的配置文件里重新设置小车的膨胀距离，小车的导航效果大大提升。

```
cost_scaling_factor: 10.0                #成本缩放因子，决定障碍物成本随距离的递减速度。
inflation_radius: 0.25                  #膨胀半径
```


```
cost_scaling_factor: 10.0                #成本缩放因子，决定障碍物成本随距离的递减速度。
inflation_radius: 0.25                  #膨胀半径
```

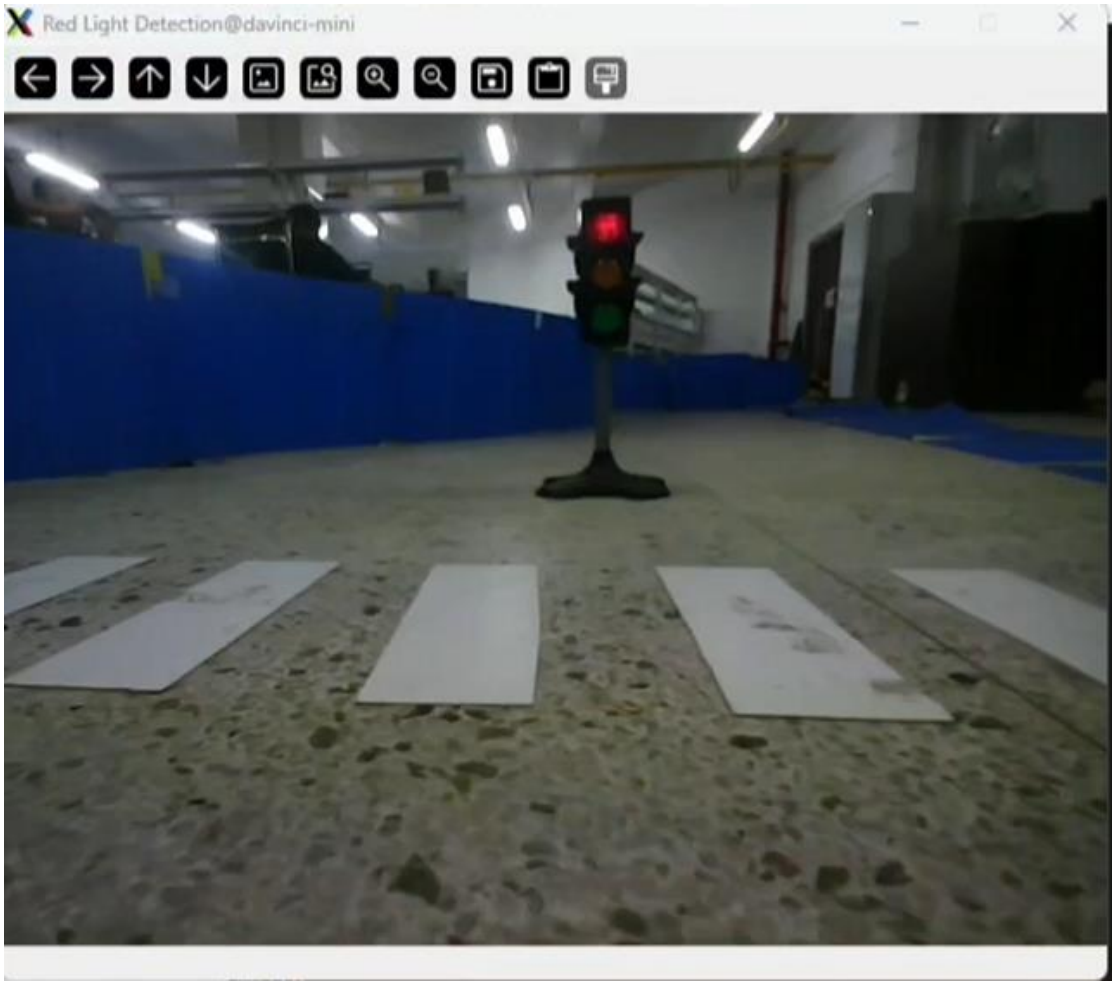
四、小车行为参数的修改分析：

即使小车构建地图的精度大大提高，但是经过测试发现，在小车转弯时构建地图还是会出现略微漂移，这可能是由于地图精度、小车传感器感应或者人为观测图不精准等原因。但是此种情况无法再在机械结构上加以改进，故在在软件层面上加以改进。以下是改进后的版本。改进后，减少了离散化导致的路径抖动，提升了转角精度，增强了最终定位精度补偿建图漂移，小车的物理运动更加稳定。

```
expected_planner_frequency: 5.0     #预期的规划器频率。如果当前频率小于预期频率，则显示警告消息。
GridBased:
  plugin: "nav2_smac_planner/SmacPlannerHybrid" #Smac Hybrid-A* 规划器
  tolerance: 0.1                      # 如果无法达到精确的姿势，则规划的容差(单位m)
  downsample_costmap: false           # 是否将代价图下采样至另一个分辨率以进行搜索
  downsampling_factor: 1              # 对代价图进行下采样的乘数因子
  allow_unknown: false                # 允许在未知空间搜索
  max_iterations: 1000000             # 失败前搜索的最大迭代次数（以防无法到达），设置为 -1 以禁用
  max_on_approach_iterations: 1000    # 在公差范围内尝试达到目标的最大迭代次数，仅限 2D
  max_planning_time: 7.0              # 规划器进行规划、平滑和上采样的最大时间（以秒为单位）。
  motion_model_for_search: "REEDS_SHEPP"    # 2D Moore, Von Neumann; Hybrid Dubin, Redds-Shepp; State Lattice set internally
  cost_travel_multiplier: 2.0         #应用于搜索以避开高成本区域的成本乘数。
  # For 2D: Cost multiplier to apply to search to steer away from high cost areas. Larger values will place in the center of aisles more exactly (
  angle_quantization_bins: 64         #用于 SE2 搜索的角度箱数量。这可以是任何偶数，但好的基线是 64 或 72（以 5 度为增量）
  # For Hybrid/Lattice nodes: Number of angle bins for search, must be 1 for 2D node (no angle search)
  analytic_expansion_ratio: 4.0       #规划器将尝试以与该值和最小启发式方法成比例的频率完成分析扩展。
  # For Hybrid/Lattice nodes: The ratio to attempt analytic expansions during search for final approach.
  minimum_turning_radius: 0.6        #最小转弯半径
  # For Hybrid/Lattice nodes: minimum turning radius in m of path / vehicle
  reverse_penalty: 10.0               #如果反向搜索，则对 SE2 节点应用启发式惩罚。
  # For Reeds-Shepp model: penalty to apply if motion is reversing, must be => 1
  change_penalty: 1.5                #如果在搜索中改变方向（例如从左到右），则对 SE2 节点应用启发式惩罚。
  # For Hybrid nodes: penalty to apply if motion is changing directions, must be >= 0
  non_straight_penalty: 1.50          #如果在非直线方向上搜索，则对 SE2 节点应用启发式惩罚。
  # For Hybrid nodes: penalty to apply if motion is non-straight, must be => 1
  cost_penalty: 1.7                   #应用于 SE2 节点的姿势成本的启发式惩罚。
```

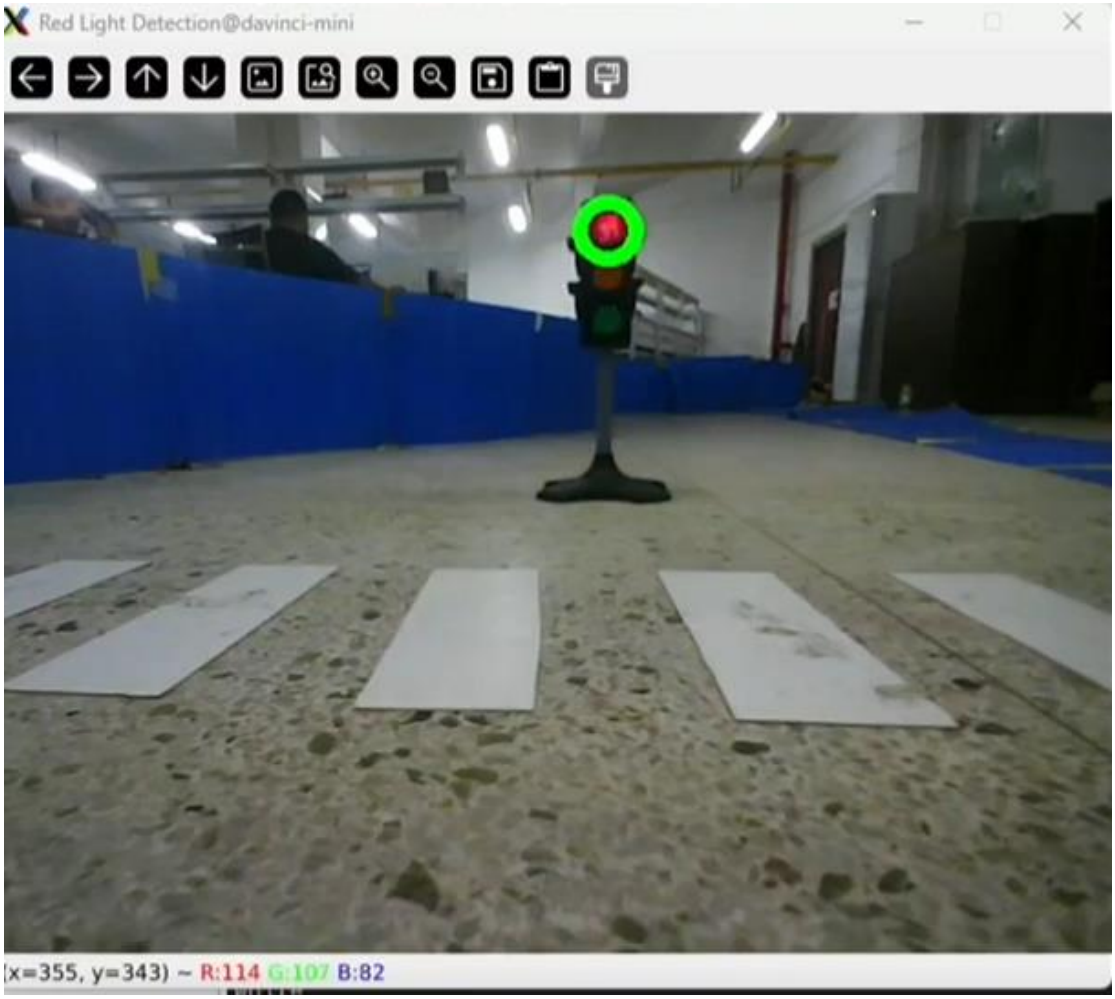
五、视觉参数的修改分析：

根据多次试验和测试，由于小车识别红绿灯或斑马线的时间有限，会导致识别失败，如下图。



我们训练了视觉模型的参数，尽可能地将识别控制在一个区域内，并将准确率和判别速度达到最佳。下图为调整后的参数。在调整后，打开小车的摄像头，小车的识别更加迅速和精准了，在运行过程中，小车的识别也更迅速和精准了。

```
smooth_kernel_size = 21
gaussian_blur_val = 15
morph_kernel_size = 5
min_area = 100
max_area = 3000

crop_percent = 0.3
median_blur_val = 15
gaussian_blur_val = 0
canny_threshold1 = 50
canny_threshold2 = 134
morph_kernel_size = 1
area_threshold = 3000
hough_threshold = 16
line_threshold = 10
```



第6章 作品总结

自四月中旬起，我们就投入到比赛的准备当中。我们的车是从上一届的学长手中接过来的，车内很多代码都有过他们的修改痕迹。在学长的指导和我们的探索下，再加上我们有组员有过比赛经验，我们顺利做完了大多数车辆基本调试和实验。但是有部分关键实验一直完成不了，比如红绿灯检测，单点导航失效，多点导航车辆未启动等等，花了很长一段时间也没有找到根本问题，这成为了我们完成这个比赛的最大阻碍，导致长时间没有进展。

后来，通过不断学习、查找资料和与其他同学探讨，一点点与其他组的车对比，我们才发现这辆车的系统未升级，大多数底层代码和指令与本次比赛并不匹配。这个时候已经是五月中下旬了，时间非常紧迫。我们不得不重新烧入新的镜像文件，从头开始调试做实验，好在有以前的经验，我们十分顺利地完成了所有实验。但是如何将实验适配场地又是一大难题，这不仅需要将已有知识和经验有机融合，还需要不断调试适配场地，解决一个又一个未知问题。由于小车识别的点位不精确，在地图构建和传感器配合等方面，我们基于自己的见解进行了大量的尝试与创新，最后将精确度大大提升；由于小车在直线状态下会莫名往左拐，我们拆了车的底盘重新进行了组装，解决了这个问题。但在这个尝试和创新过程中，由于组内的经验缺乏和对 ros 的知识不足，我们经历了多次失败，比如调试了不该调的参数，导致一段时间不仅没有进展反而是退步，比如尝试修改小车的代码，增加小车行驶的稳定性，但是却导致小车一段时间没有跑起来等等问题。

最大的两个问题则是小车无法准确识别出红绿灯或斑马线停车，并且反应有延迟，有几率失败，而小组成员对计算机视觉的经验和知识不足，难以解决这个问题，感谢和我们交流的老师和同学，提供了关键代码，解决了这个棘手的问题。还有一个问题则是小车莫名在结尾无法到达终点，总会偏离方向转圈，这个时候我们对 ros 有一定的知识储备了，尝试着去修改底层关键代码，经过多次尝试，在最后即将完赛的时候成功解决了这个问题。通过多次尝试和调试，最终总算给出了满意的结果。

此次比赛，我们投入了大量的时间和精力，收获的是宝贵的技能提升和一次完整的项目经历。我我们知道不积跬步，无以至千里，从最基础的调试、实验开始，坚持不懈，一点一点和车的传感器一起响应感受着终点；我们知道必须将理论和实际将结合起来，最初没有知识和理论的我们，也在日日夜夜前行中积攒了不少关于 ros 无人车、建图、计算机视觉的知识；我们知道实践是检验真理的唯一标准，调试修改后的个个参数和写下的代码都在激励着我们和车一起不断前行。

在这个过程中，假如没有老师给我们的设备支持、场地支持和精神上的鼓励，假如没有其他组给我们的场地支持、理解和帮助，假如没有我们三个人之间的努力、互相支持和陪伴，我们都很难完成这场比赛。在这里由衷地感谢他们，此时，我们要特别感谢交流群里的老师们。他们对我们的各种问题，无论是简单还是稍显稚嫩，都表现出了极高的耐心，为我们解决了诸多难题，使我们能够顺利度过备赛初期。

关于我们的智能车，我们坚信仍有巨大的改进空间。假如我们写下的代码时间复杂度，空间复杂度都低一些，参数调的更精准，那么结果肯定会更优异。假如我们能运用当今最先进的计算机视觉识别技术，小车的识别准确度和反应速度肯定会大大提升，在判别上做得更加精准。假如我们的智能车的芯片更高级，底层更先进，更轻量，小车的运行速度和反应速度也肯定会快不少，这能从客观上提升小车

的运行速度。而且当前的地图构建主要基于 2D 形式,，目前的导航方式依赖于多点导航，需要人工手动干预和输入指令，若能开发出基于地图的自动导航功能或者是基于视觉道路的自动识别功能，将显著提升智能车的自动化和智能化水平，若能开发出语音导航功能，智能车将会大大方便操作。虽然我们的小车仅仅运用于任务一，但是小车也能在很多地方发挥用途，例如固定点位的运输货物，例如自动驾驶的创新和运用，例如环境感知导航等等。

参考文献

[1] 姜明国,陆波.阿克曼原理与矩形化转向梯形设计[J].汽车技术,1994,(05):16-19.
[2] 基于激光测距与双目视觉信息融合的移动机器人SLAM研究[J]. 杜钊君;吴怀宇.计算机测量与控制,2013(01)
[3] 未知环境下的移动机器人SLAM方法[J]. 朱磊;樊继壮;赵杰;吴晓光.华中科技大学学报(自然科学版),2011(07)
[4] 大规模环境下基于图优化SLAM的后端优化方法[J]. 王忠立;赵杰;蔡鹤皋.哈尔滨工业大学学报,2015(07)
[5] 张帆,曹喜滨,邹经湘.一种新的全角度四元数与欧拉角的转换算法[J].南京理工大学学报(自然科学版),2002,(04):376-380.DOI:10.14177/j.cnki.32-1397n.2002.04.010.
[6] 周俞辰.基于激光三角测距法的激光雷达原理综述[J].电子技术与软件工程,2016,(19):94-95.DOI:10.20109/j.cnki.etse.2016.19.073.
[7] 基于单目视觉的SLAM算法研究[J]. 温丰;柴晓杰;朱智平;董小明;邹伟;原魁.系统科学与数学,2010(06)
[8] OctoMap: an efficient probabilistic 3D mapping framework based on octrees[J]. Armin Hornung;;Kai M. Wurm;;Maren Bennewitz;;Cyrill Stachniss;;Wolfram Burgard.Autonomous robots,2013
[9]大规模环境下基于图优化SLAM的后端优化方法[J]. 王忠立;赵杰;蔡鹤皋.哈尔滨工业大学学报,2015(07)
[10]1 year, 1000 km: The Oxford RobotCar dataset[J]. Will Maddern;;Geoffrey Pascoe;;Chris Linegar;;Paul Newman.The International Journal of Robotics Research,2017