TB3_Autorace 调试教程

2019 年 6 月

武汉京天电器有限公司
www.jingtianrobots.com

目　录

TB3_Autorace-硬件要求..... 3
TB3_Autorace-软件要求..... 3
TB3_Autorace-场景搭建..... 4
TB3_Autorace-图像校准..... 7
TB3_Autorace-相机内标定..... 8
TB3_Autorace-相机外标定..... 9
TB3_Autorace-车道识别..... 11
TB3_Autorace-交通信号识别..... 14
TB3_Autorace-交通灯识别..... 17
TB3_Autorace-自动泊车..... 19
TB3_Autorace-交通杆识别..... 20
TB3_Autorace-通过隧道..... 22
TB3_Autorace-完整测试..... 23

TB3_Autorace-硬件要求

硬件配置：

- 机器人：
  - TurtleBot3 Burger
- 远程 PC：
  - Remote PC (Laptop, Desktop, etc.)
- 辅助设备：
  - 鱼眼相机：Raspberry Pi Camera Type G
  - 相机固定架：Raspberry Pi Camera Mount
- 赛道结构和配件：
  - 裁判系统资源： https://github.com/ROBOTIS-GIT/autorace_referee
  - CAD 模型：https://github.com/ROBOTIS-GIT/autorace_track

TB3_Autorace-软件要求

软件配置：

- [Remote PC & TurtleBot SBC] 安装依赖

```
$ sudo apt-get install ros-kinetic-image-transport ros-kinetic-cv-bridge ros-kinetic-vision-opencv python-opencv libopencv-dev ros-kinetic-image-proc
```

- [Remote PC & TurtleBot SBC] 安装 autorace 包

```
$ cd ~/catkin_ws/src/

$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_autorace.git

$ cd ~/catkin_ws && catkin_make
```

- [Remote PC & TurtleBot SBC] 安装 raspicam_node

```
$ cd ~/catkin_ws/src/

$ git clone https://github.com/ncnynl/raspicam_node

$ cd ~/catkin_ws && catkin_make
```

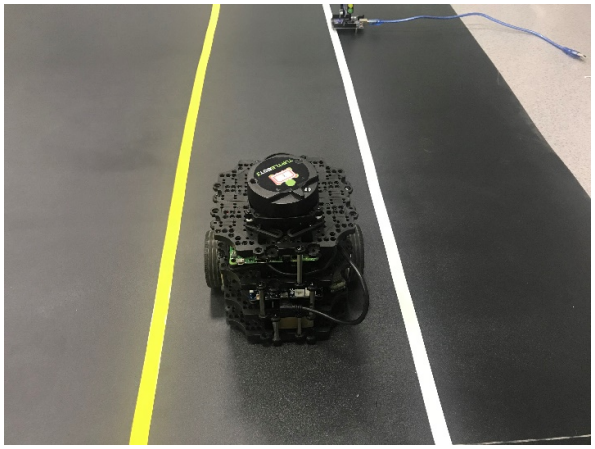
TB3_Autorace-场景搭建

场地要求

- 由于 Turtlebot3 测试自动驾驶受环境光线影响较大，所以测试环境应该设置在室内，且尽量保证早晚的环境光线保持一致
- 场地应该采用深色平坦，不能太过光滑的背景，建议背景色为黑色
- 下图是测试时搭建的场景

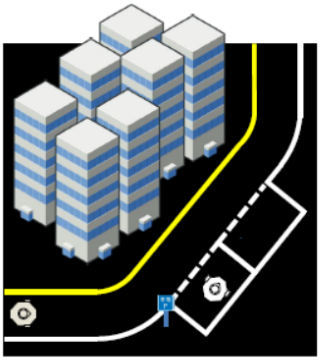
车道

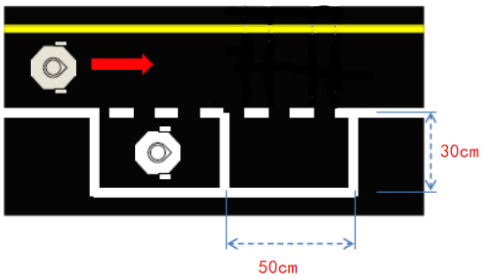
- 车道主要由两条车道线组成，基于机器人的位置判定，左边为黄色车道线，右边为白色车道线
- 建议黄色车道线与白色车道线的距离设置为 25cm
- 车道线的宽度为 2cm 或 3cm
- 参考图



停车位

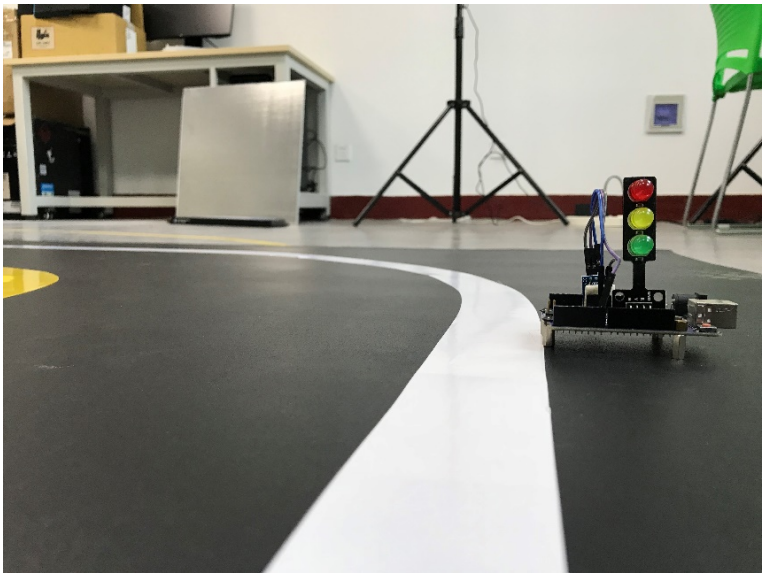
- 停车位上的停车虚线段长度在 2cm 或 3cm
- 每两段停车虚线段之间的距离不适宜太远，要确保进行停车时，topic：/detect/image_parking 上的摄像头检测到多个停车虚线段
- 参考图





交通灯

- 交通灯可以用红黄绿三种 LED 灯来代替，或者使用自行车青蛙灯
- 参考图

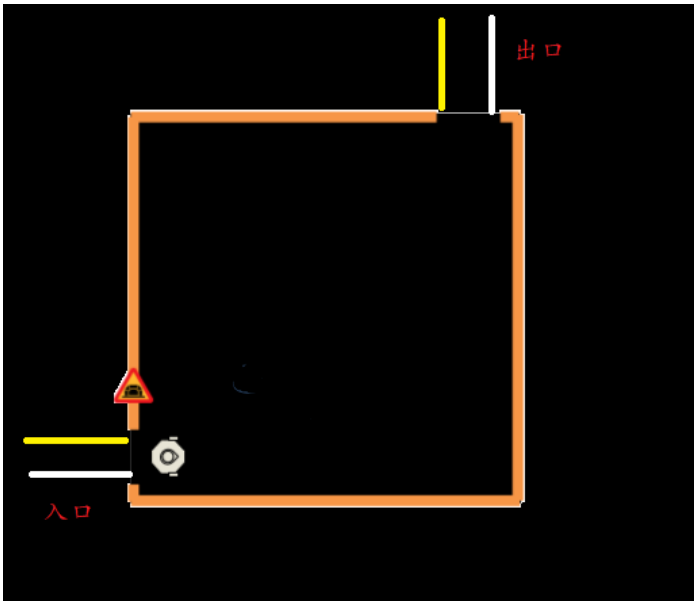


交通杆

- 交通杆的制作可以使用三片红色的纸片或丝带
- 建议色块大小为长 4cm，宽 2cm，两色块相距 4cm
- 交通杆应该放置在与摄像头水平的位置
- 参考图

隧道

- 隧道入口和出口的宽度应该大于车道的宽度，大约为 30cm
- 隧道可以使用木板或纸箱等高于 25cm 的物品围起来，只留出入口
- 隧道出口一般设置在整个自动驾驶流程的起点处
- 参考图



TB3_Autorace-图像校准

操作步骤

- [Remote PC] 新终端，启动 roscore

```
$ roscore
```

- [TurtleBot SBC] 新终端，启动相机

```
$ roslaunch turtlebot3_autorace_camera turtlebot3_autorace_camera_pi.launch
```

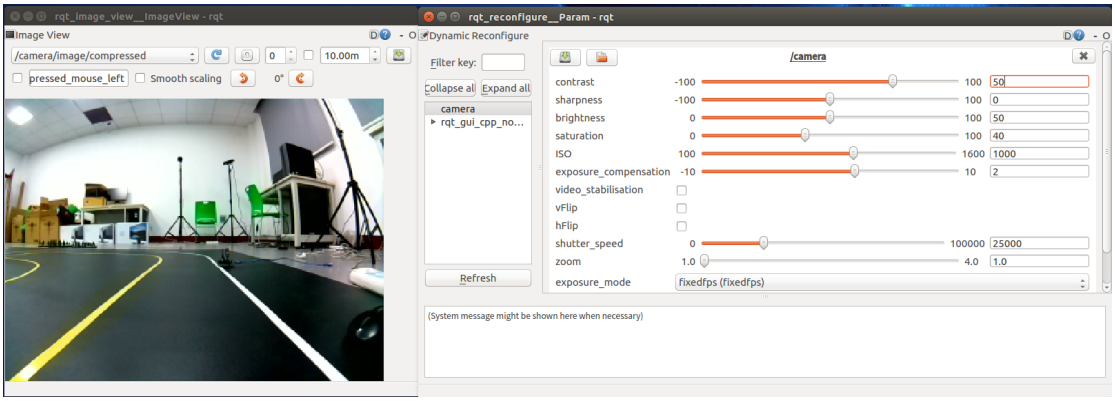
- [Remote PC] 新终端，浏览图像界面

```
$ rqt_image_view
```

- 选择 topic：/camera/image/compressed 或/camera/image/ 如果正常则可以看到图像
- [Remote PC] 新终端，打开配置界面

```
$ rosrun rqt_reconfigure rqt_reconfigure
```

- 选择 camera，调整参数值，使相机显示干净，足够明亮的图像。 之后，将每个值覆盖到树莓派上 turtlebot3_autorace_camera/calibration/camera_calibration 文件夹中 camera.yaml 里的参数值
- 下次启动即会使用新的参数值，以达到更好的显示效果。
- 若不能正常载入参数，可以尝试修改树莓上的 raspicam_node/cfg 文件夹中的 Camera.cfg 内的参数值
- 效果图



TB3_Autorace-相机内标定

操作步骤

- [Remote PC] 新终端，启动 roscore

```
$ roscore
```

- [TurtleBot SBC] 新终端，启动摄像头

```
$ roslaunch turtlebot3_autorace_camera turtlebot3_autorace_camera_pi.launch
```

- 将 PC 上的 turtlebot3_autorace_camera/data 文件夹里的 checkerboard_for_calibration.pdf 的内容打印到一张 A4 纸上
- [Remote PC] 根据打印出来的格子直径修改 turtlebot3_autorace_camera/launch 文件夹里的 turtlebot3_autorace_intrinsic_camera_calibration.launch 的--square 0.108

```
 <node if="$(eval calibration_mode == 'calibration')" pkg="camera_calibration" type="cameracalibrator.py" name="cameracalibrator" args="--size 8x6 --square 0.108 image:=/camera/image camera:=/camera" output="screen"/>


###将--square 0.108 修改为--square 0.0245


<node if="$(eval calibration_mode == 'calibration')" pkg="camera_calibration" type="cameracalibrator.py" name="cameracalibrator" args="--size 8x6 --square 0.0245 image:=/camera/image camera:=/camera" output="screen"/>
```

- [Remote PC] 新终端，执行 calibration 模式下的内标定程序

```
$ export AUTO_IN_CALIB=calibration

$ roslaunch turtlebot3_autorace_camera turtlebot3_autorace_intrinsic_camera_calibration.launch
```

- 可以参考官方教程进行标定：参考链接

- 标定完成之后将文件保存到 PC 下 turtlebot3_autorace_camera/calibration/intrinsic_calibration 文件夹下并命名为 camerav2_320x240_30fps.yaml，修改名称：camera_name:camera，同时将该文件保存到树莓派上的同样路径目录下。

TB3_Autorace-相机外标定

操作步骤

- [Remote PC] 新终端，启动 roscore

```
$ roscore
```

- [TurtleBot SBC] 新终端，启动摄像头

```
$ roslaunch turtlebot3_autorace_camera turtlebot3_autorace_camera_pi.launch
```

- [TurtleBot SBC] 新终端，启动 action 模式下的内标定

```
$ export AUTO_IN_CALIB=action

$ roslaunch turtlebot3_autorace_camera turtlebot3_autorace_intrinsic_camera_calibration.launch
```

- [Remote PC] 新终端，启动 calibration 模式下的外标定

```
$ export AUTO_EX_CALIB=calibration

$ roslaunch turtlebot3_autorace_camera turtlebot3_autorace_extrinsic_camera_calibration.launch
```

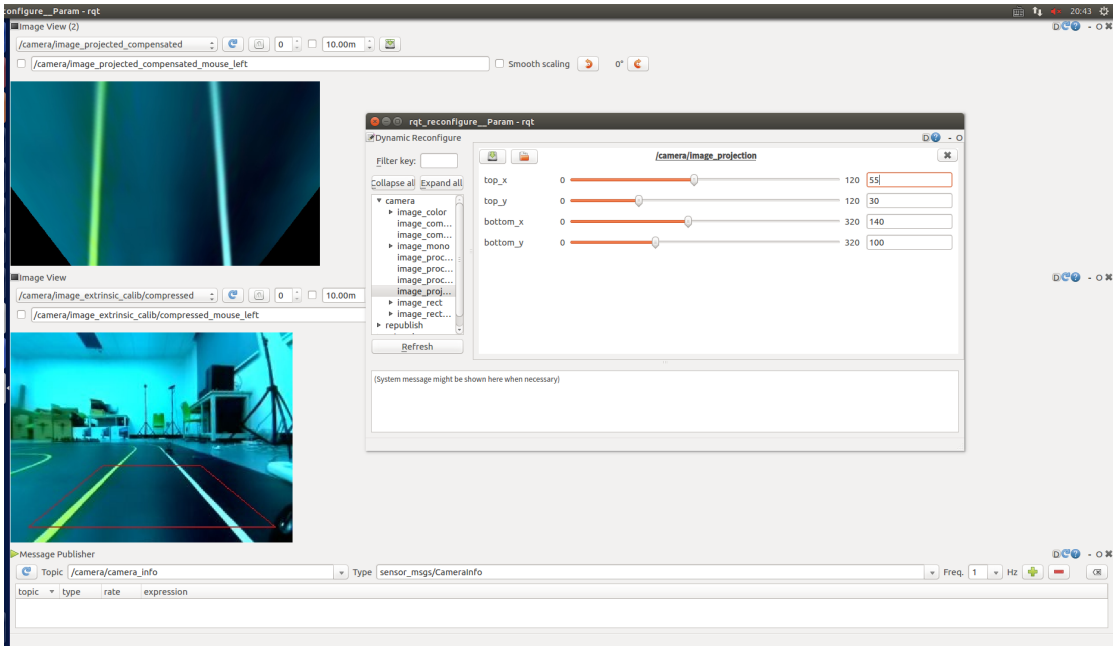
- [Remote PC] 新终端，打开 rqt

```
$ rqt
```

- 左上角选择 plugins -> visualization -> Image view 打开图像窗口
- 打开第二天图像窗口，选择 topic：/camera/image_extrinsic_calib/compressed
- 打开第三个图像窗口，选择 topic：/camera/image_projected_compensated
- [Remote PC] 新终端，打开参数配置

```
$ rosrun rqt_reconfigure rqt_reconfigure
```

- 调整参数值/camera/image_projection，/camera/image_compensation_projection
- 调整 image_projection 参数，会改变打开/camera/image_extrinsic_calib/compressed 的图像窗口的红色方框的形状
- 相机外标定将转换由红色矩形包围的图像，并将显示从通道上看的图像
- 把相应的参数值写入到 PC 上的 turtlebot3_autorace_camera/calibration/extrinsic_calibration/文件里的 projection.yaml 文件
- 下一次启动即会使用新的参数值
- 若无法载入参数值，可以尝试修改 PC 上的 turtlebot3_autorace_camera/nodes/文件里的 image_projection.py 文件内的相关参数值
-
- 效果图：



TB3_Autorace-车道识别

车道识别

操作步骤

- [Remote PC] 新终端，启动 roscore

```
$ roscore
```

- [TurtleBot SBC] 新终端，启动摄像头

```
$ roslaunch turtlebot3_autorace_camera turtlebot3_autorace_camera_pi.launch
```

- [TurtleBot SBC] 新终端，打开 action 模式下的内标定程序

```
$ export AUTO_IN_CALIB=action

$ roslaunch turtlebot3_autorace_camera turtlebot3_autorace_intrinsic_camera_calibration.launch
```

- [Remote PC] 新终端，打开 action 模式下的外标定程序

```
$ export AUTO_EX_CALIB=action

$ roslaunch turtlebot3_autorace_camera turtlebot3_autorace_extrinsic_camera_calibration.launch
```

- 主要调整 feature detector/color filter 来优化对象识别
- 将 tb3 放置在车道中间，左边为黄色线，右边为白色线
- [Remote PC] 新终端，启动 calibration 模式下的车道识别

```
$ export AUTO_DT_CALIB=calibration

$ roslaunch turtlebot3_autorace_detect turtlebot3_autorace_detect_lane.launch
```

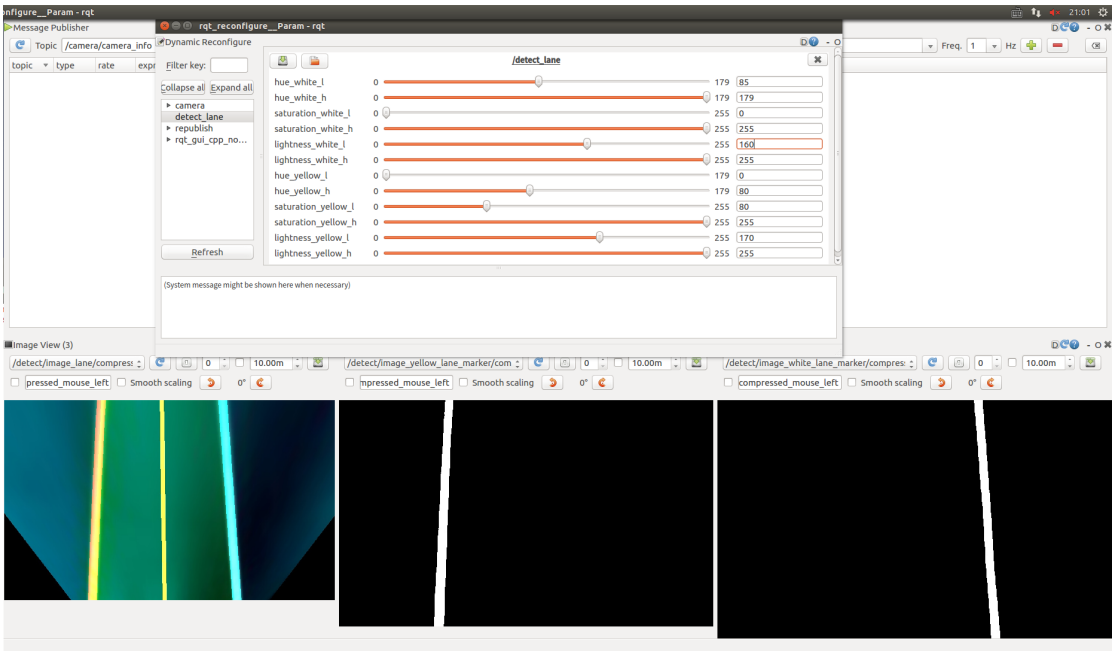
- [Remote PC] 新终端，启动 rqt

```
$ rqt
```

- 在左上角选择 plugins -> visualization -> Image view 新建图像窗口
- 打开第二个图像窗口，选择 topic：/detect/image_yellow_lane_marker/compressed
- 打开第三个图像窗口，选择 topic：/detect/image_lane/compressed
- 打开第四个图像窗口，选择 topic：/detect/image_white_lane_marker/compressed
- 如果一切正常，左右屏幕将显示黄线和白线的过滤图像，中心屏幕将显示机器人应该去的路径
- 在标定模式下，左右屏幕将显示白色，中央屏幕可能显示异常结果
- 应该调整滤镜参数以显示正确的线条和方向
- [Remote PC] 新终端启动 rqt_reconfigure

```
$ rosrun rqt_reconfigure rqt_reconfigure
```

- 由于实际物理环境包括房间内的光亮等，车道的彩色滤波的校准过程有时会非常困难
- 校准时可以参考 turtlebot3_autorace_detect/param/lane/lane.yaml 里面的参数值
- 首先校准色调低 - 高值。 色调值表示颜色，每种颜色，如黄色，白色，都有自己的色调值区域（参考 hsv 图）
- 然后校准饱和度低 - 高值。 每种颜色也有自己的饱和度。
- 最后，校准亮度低 - 高值。 将亮度高值设置为 255.清晰过滤的线条图像将为您提供清晰的通道结果。
- 效果图：



注意事项：

- 以上效果图仅供参考，请根据实际环境来设置参数

车道识别测试

- [Remote PC] 完成标定之后关闭 rqt_rconfigure and turtlebot3_autorace_detect_lane
- [Remote PC] 新终端，启动 action 模式下的车道识别测试

```
$ export AUTO_DT_CALIB=action

$ roslaunch turtlebot3_autorace_detect turtlebot3_autorace_detect_lane.launch
```

- [Remote PC] 新终端，启动车道识别控制程序

```
$ roslaunch turtlebot3_autorace_control turtlebot3_autorace_control_lane.launch
```

- [TurtleBot SBC] 新终端，启动 tb3

```
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

TB3_Autorace-交通信号识别

制作交通标志

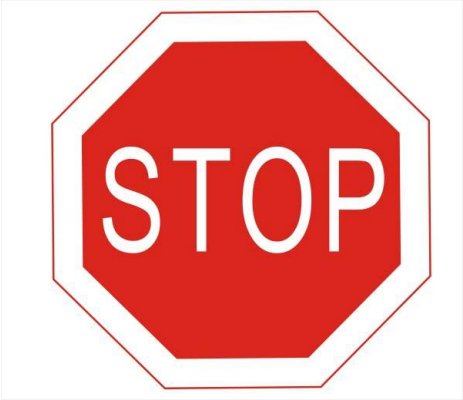
- 交通标志检测需要一些交通标志的图片。
- 节点使用 SIFT 算法查找交通标志
- 因此如果您想使用自定义交通标志（未在 autorace_track 中引入）
- 请注意交通标志中的更多边缘可提供更好的 SIFT 识别结果

方法一

- 启动摄像头后，通过 rqt_image_view 节点拍摄现有的交通标志图，并通过 linux 中的任何照片编辑器编辑它们的大小和形状
- 在远程 PC 上启动 rqt_image_view，选择话题/camera/image_compensated
- 通过 alt+print screen 拍摄照片，使用首选照片编辑器编辑拍摄的照片
- 之后，将图片放置到 PC 里的/turtlebot3_autorace/turtlebot3_autorace_detect/file/detect_sign/文件夹并根据所需修改对应的文件名
- 但是，如果要更改默认文件名，则应更改 detect_sign.py 中原本写入的文件名

方法二

- 下载以下三张交通信号标志图后，将图片放置到 PC 里的 /turtlebot3_autorace/turtlebot3_autorace_detect/file/detect_sign/文件夹并根据所需修改对应的文件名
- 默认的文件名为： parking.png,stop.png,tunnel.png,其在官方例程中分别应用于泊车、通过交通杆和隧道
- 但是，如果要更改默认文件名，则应更改 detect_sign.py 中原本写入的文件名
- 再将交通标志图打印出来，大小约为半张 A4 纸
- 示例图：



交通信号识别测试

- 把机器人放在车道上， 同时将交通标志应放置在机器人可以轻松看到的位置
- 确保尚未启动 turtlebot3_bringup 包的 turtlebot3_robot 节点
- [Remote PC] 新终端，启动 roscore

```
$ roscore
```

- [TurtleBot SBC] 新终端，启动摄像头

```
$ roslaunch turtlebot3_autorace_camera turtlebot3_autorace_camera_pi.launch
```

- [TurtleBot SBC] 新终端，打开 action 模式下的内标定程序

```
$ export AUTO_IN_CALIB=action

$ roslaunch turtlebot3_autorace_camera turtlebot3_autorace_intrinsic_camera_calibration.launch
```

- [Remote PC] 新终端，打开 action 模式下的外标定程序

```
$ export AUTO_EX_CALIB=action

$ roslaunch turtlebot3_autorace_camera turtlebot3_autorace_extrinsic_camera_calibration.launch
```

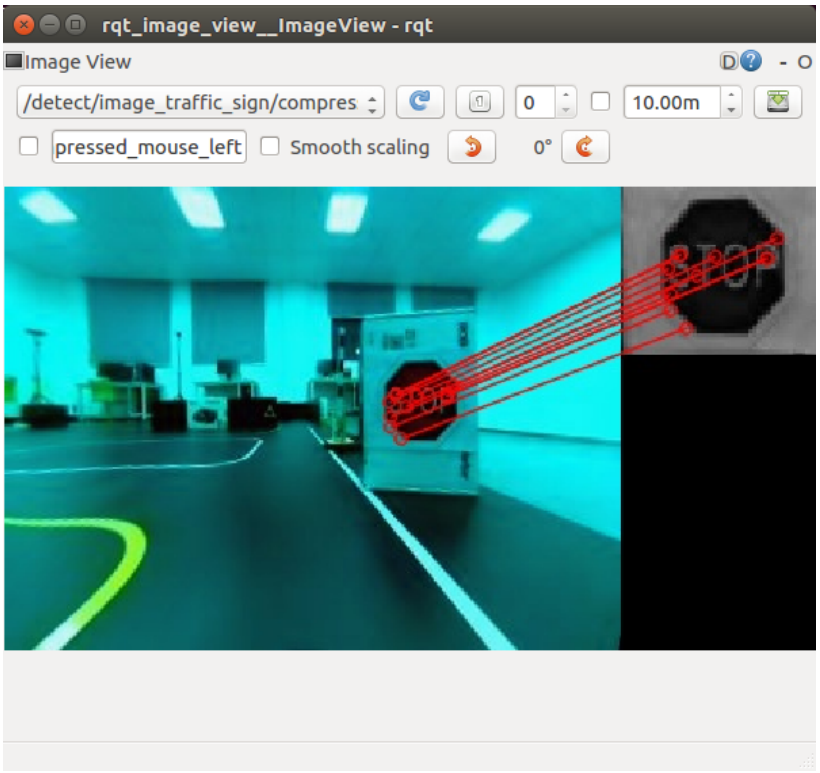
- 主要调整 feature detector/color filter 来优化对象识别
- [Remote PC] 新终端，启动交通信号识别程序

```
$ roslaunch turtlebot3_autorace_detect turtlebot3_autorace_detect_sign.launch
```

- [Remote PC] 新终端，启动 rqt_image_view

```
$ rqt_image_view
```

- 选择 topic：/detect/image_traffic_sign/compressed
- 如果它成功识别它，屏幕将显示交通标志检测的结果
- 效果图



TB3_Autorace-交通灯识别

操作步骤

- [Remote PC] 新终端启动 roscore

```
$ roscore
```

- [TurtleBot SBC] 新终端，启动摄像头

```
$ roslaunch turtlebot3_autorace_camera turtlebot3_autorace_camera_pi.launch
```

- [TurtleBot SBC] 新终端，启动 action 模式下的内标定程序

```
$ export AUTO_IN_CALIB=action

$ roslaunch turtlebot3_autorace_camera turtlebot3_autorace_intrinsic_camera_calibration.launch
```

- [Remote PC] 新终端，启动 action 模式下的外标定程序

```
$ export AUTO_EX_CALIB=action

$ roslaunch turtlebot3_autorace_camera turtlebot3_autorace_extrinsic_camera_calibration.launch
```

- 主要调整 feature detector / color filter 来优化对象识别
- [Remote PC] 新终端，启动 calibration 模式下的交通灯识别程序

```
$ export AUTO_DT_CALIB=calibration

$ roslaunch turtlebot3_autorace_detect turtlebot3_autorace_detect_traffic_light.launch
```

- [Remote PC] 新终端，启动 rqt

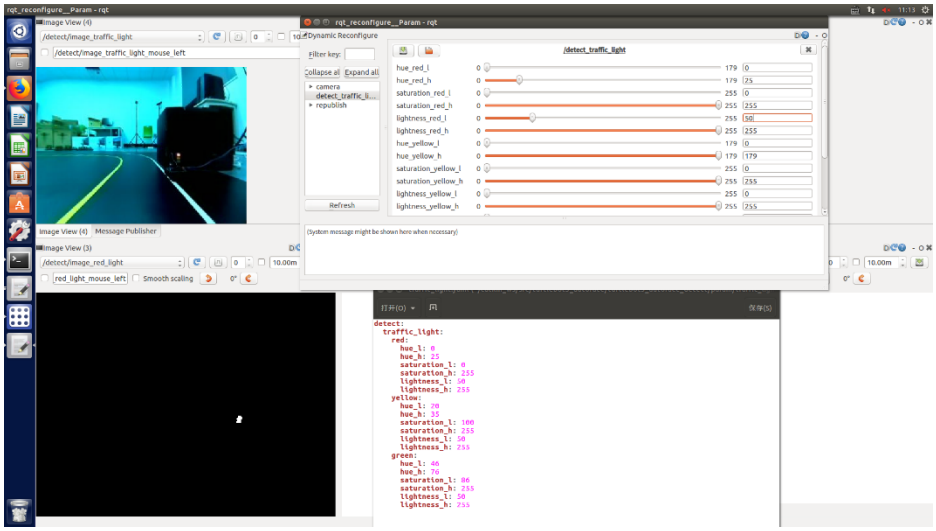
```
$ rqt
```

- 左上角菜单栏上选择 plugins -> visualization -> Image view
- 分别打开多个图像窗口选择不同话题
- 打开窗口，选择 topic：/detect/image_red_light
- 打开窗口，选择 topic：/detect/image_yellow_light
- 打开窗口，选择 topic：/detect/image_green_light
- 打开窗口，选择 topic：/detect/image_traffic_light

- 如果一切正常，三个屏幕将显示红色/黄色/绿色光的过滤图像，另一个将显示带有短字符串的识别颜色
- 在校准模式下，三个屏幕将显示白色，另一个屏幕可能显示明显的结果
- 从这里，您应该调整滤镜参数以显示正确的线条和方向。
- [Remote PC] 新终端，启动参数设置

```
$ rosrun rqt_reconfigure rqt_reconfigure
```

- 调整/detect_traffic_light 值，更改滤色器的值将显示每种颜色屏幕上过滤视图的更改
- 之后，将值覆盖到 turtlebot3_autorace_detect/param/traffic_light/中的 traffic_light.yaml 文件
- 但若参数不能正常载入，可以尝试直接修改 PC 上的 turtlebot3_autorace_detect/nodes/里的 detect_traffic_light 文件中的参数
- 效果图：



注意：以上效果图仅供参考，应该根据实际环境来设置参数

交通灯识别测试

- 覆盖校准文件后，关闭 rqt_rconfigure 和 turtlebot3_autorace_detect_traffic_light
- [Remote PC] 新终端启动

```
$ export AUTO_DT_CALIB=action

$ roslaunch turtlebot3_autorace_detect turtlebot3_autorace_detect_traffic_light.launch
```

- [Remote PC] 新终端启动 rqt_image_view

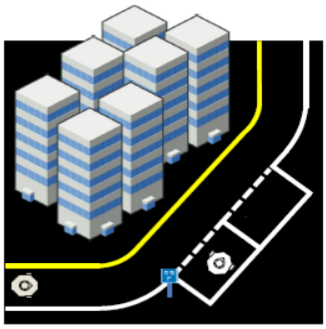
```
$ rqt_image_view
```

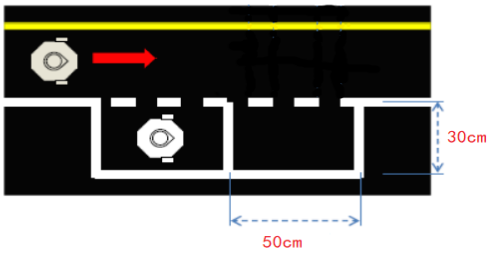
- 检测结果是否正确

TB3_Autorace-自动泊车

准备工作停车只需要一次交通标志识别

- 将一个机器人或者高于测试使用的机器人高度的物品放在其中一个停车位上，将要执行停车的机器人放车道上
- 将交通信号标志放置在停车位沿车道线的前方
- 机器人需要识别到停车标志，再进入到停车程序，所以机器人与停车标志要有一定的距离
- 示例图





- 测试时搭建的场景图

自动泊车测试

- 使用完整的自动驾驶程序进行测试： turtlebot3 自动驾驶

TB3_Autorace-交通杆识别

操作步骤

- 水平交通杆在水平面上找到 3 个红色矩形，并计算水平是打开还是关闭，以及机器人离交通杆有多远
- 将机器人放在车道上放置再关闭的水平交通杆面前
- [Remote PC] 新终端，启动 roscore

```
$ roscore
```

- [TurtleBot SBC] 新终端，启动摄像头

```
$ roslaunch turtlebot3_autorace_camera turtlebot3_autorace_camera_pi.launch
```

- [TurtleBot SBC] 新终端，启动 action 模式下的内标定程序

```
$ export AUTO_IN_CALIB=action

$ roslaunch turtlebot3_autorace_camera turtlebot3_autorace_intrinsic_camera_calibration.launch
```

- [Remote PC] 新终端，启动 action 模式下的外标定程序

```
$ export AUTO_EX_CALIB=action

$ roslaunch turtlebot3_autorace_camera turtlebot3_autorace_extrinsic_camera_calibration.launch
```

- 主要调整 feature detector / color filter 来优化对象识别
- [Remote PC]新终端，启动 calibration 模式的交通杆识别程序

```
$ export AUTO_DT_CALIB=calibration

$ roslaunch turtlebot3_autorace_detect turtlebot3_autorace_detect_level.launch
```

- [Remote PC] 新终端，启动 rqt

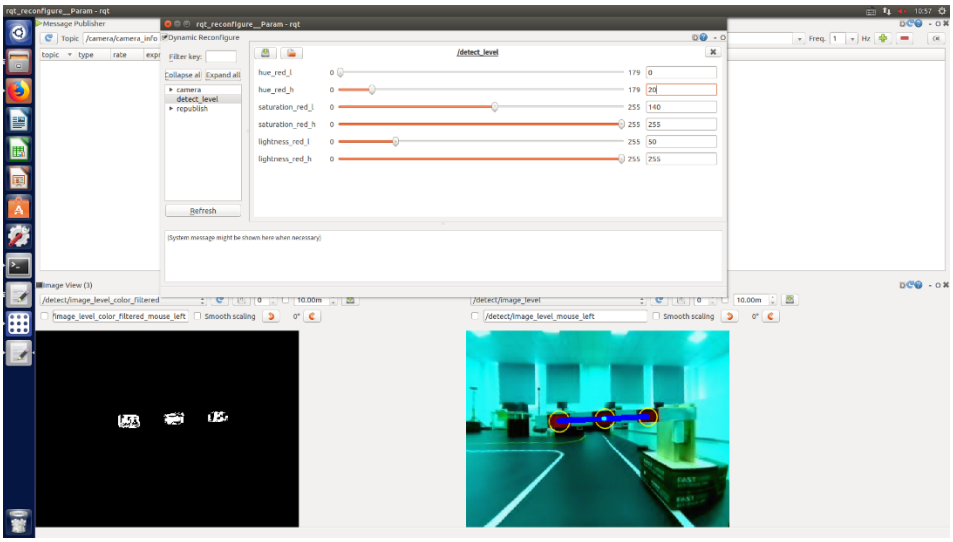
```
$ rqt
```

- 左上角菜单栏上选择 plugins -> visualization -> Image view
- 分别打开多个图像窗口选择不同 topic
- 打开新窗口，选择 topic：/detect/image_level_color_filtered

- 打开新窗口，选择 topic：/detect/image_level
- 如果一切正常，三个屏幕将显示红色矩形的过滤图像，另一个将绘制连接矩形的线条
- 在校准模式下，屏幕将显示白色，而另一个屏幕可能显示明显的结果
- 从这里，您应该调整滤镜参数以显示正确的线条和方向
- [Remote PC] 新终端，启动参数设置程序

```
$ rosrun rqt_reconfigure rqt_reconfigure
```

- 调整/detect_level 中的参数值
- 更改滤色器的值将显示每种颜色屏幕上过滤视图的更改
- 之后，将值覆盖到 PC 上 turtlebot3_autorace_detect/param/level/文件夹中的 level.yaml 文件
- 将在下次启动使用新参数值
- 效果图：



注意：以上效果图仅供参考，请根据实际环境来设置参数

交通杆识别测试

- [Remote PC] 覆盖校准文件后，关闭 rqt_rconfigure 和 turtlebot3_autorace_detect_level
- [Remote PC] 新终端，启动 action 模式的交通杆识别程序

```
$ export AUTO_DT_CALIB=action

$ roslaunch turtlebot3_autorace_detect turtlebot3_autorace_detect_level.launch
```

- [Remote PC] 新终端，启动 rqt_image_view

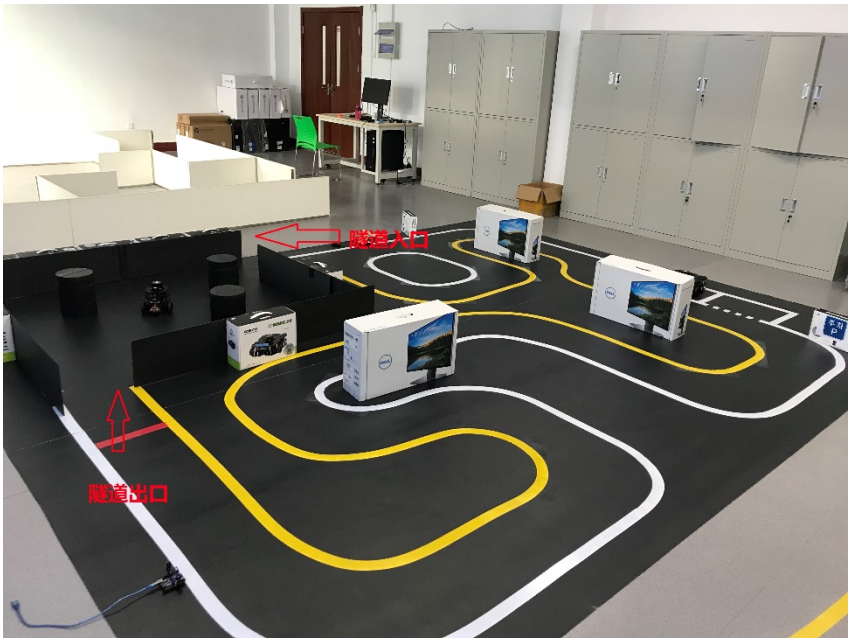
```
$ rqt_image_view
```

- 查看结果是否正确

TB3_Autorace-通过隧道

制作地图

- 首先要确定机器人在整个自动驾驶流程的起点位置，因为机器人是基于里程计的信息来制作地图
- 其次是确定隧道的所在位置，隧道可以用木板围起来，只留下出口和入口
- 测试搭时建的示例场景



- 接下来是通过 slam 来制作地图
- 建图流程：参考链接

注意：

- 机器人需要从起点位置通过键盘控制移动到隧道的中心位置，再启动 slam 程序，尽量不要扫描到太多隧道外围的地图
- 隧道入口是通过里程计反馈的坐标来确定，所以要确保机器人是从起点位置出发且出发时，里程计的坐标应该处于原点

相关设置

- [Remote PC] 将保存好的地图复制到 `turtlebot3_autorace/turtlebot3_autorace_control/maps` 文件夹下且分别修改文件名为 `tunnel.yaml` 和 `tunnel.pgm`，同时将 `tunnel.yaml` 里面的 image 路径修改为 `image: tunnel.pgm`
- [Remote PC] 更改 `turtlebot3_autorace/turtlebot3_autorace_detect/nodes/detect_tunnel` 文件夹下 `detect_tunnel.py` 的 fnPubGoalPose 函数里指定出口的位置及方向

```
def fnPubGoalPose(self):
    goalPoseStamped = PoseStamped()

    goalPoseStamped.header.frame_id = "map"
    goalPoseStamped.header.stamp = rospy.Time.now()

    goalPoseStamped.pose.position.x = 0.15
    goalPoseStamped.pose.position.y = -1.76
    goalPoseStamped.pose.position.z = 0.0

    goalPoseStamped.pose.orientation.x = 0.0
    goalPoseStamped.pose.orientation.y = 0.0
    goalPoseStamped.pose.orientation.z = 0.0
    goalPoseStamped.pose.orientation.w = 1.0
```

- 出口的位置及方向的获取可以通过使用已经建好的地图来进行导航，同时监听 topic：/move_base/current_goal，当在导航中指定目标时，监听的终端就会显示相应的坐标和方向

隧道测试

- 使用完整的自动驾驶程序进行测试：turtlebot3 自动驾驶

但若导航时出错，需要关闭树莓派后重启机器人，确保机器人在起点出发时，里程计信息已重

TB3_Autorace-完整测试

准备工作:

- 从现在开始，所有相关节点都将以 action 模式运行
- 如果有些尚未关闭，请关闭 Remote PC 和 TurtleBot SBC 上所有与 ROS 相关的程序和终端
- 然后将机器人放在车道中间上

操作步骤

- [Remote PC] 新终端，启动 roscore

```
$ roscore
```

- [TurtleBot SBC] 新终端，启动 tb3

```
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

- [TurtleBot SBC] 新终端，启动相机

```
$ roslaunch turtlebot3_autorace_camera turtlebot3_autorace_camera_pi.launch
```

- [TurtleBot SBC] 新终端，启动 action 模式下的内标定程序

```
$ export AUTO_IN_CALIB=action

$ roslaunch turtlebot3_autorace_camera turtlebot3_autorace_intrinsic_camera_calibration.launch
```

- [Remote PC] 新终端，启动 action 模式下的自动驾驶程序

```
$ export AUTO_EX_CALIB=action

$ export AUTO_DT_CALIB=action

$ export TURTLEBOT3_MODEL=burger

$ roslaunch turtlebot3_autorace_core turtlebot3_autorace_core.launch
```

- [Remote PC] 新终端，执行

```
$ rostopic pub -1 /core/decided_mode std_msgs/UInt8 "data: 2"
```

- turtlebot3_autorace_core 将控制包中的所有系统，打开和关闭包中的节点。