Autorace

Summary

1.Preliminary steps

First, we need to start roscore following by launching bringup for the robot and the camera.

```
roscore
```

To send a command to the turtlebot we use the `ssh ubuntu@192.168.0.200` command. Just after entering the password, we can bringup the robot and start the video acquisition with the two following commands:

```
roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

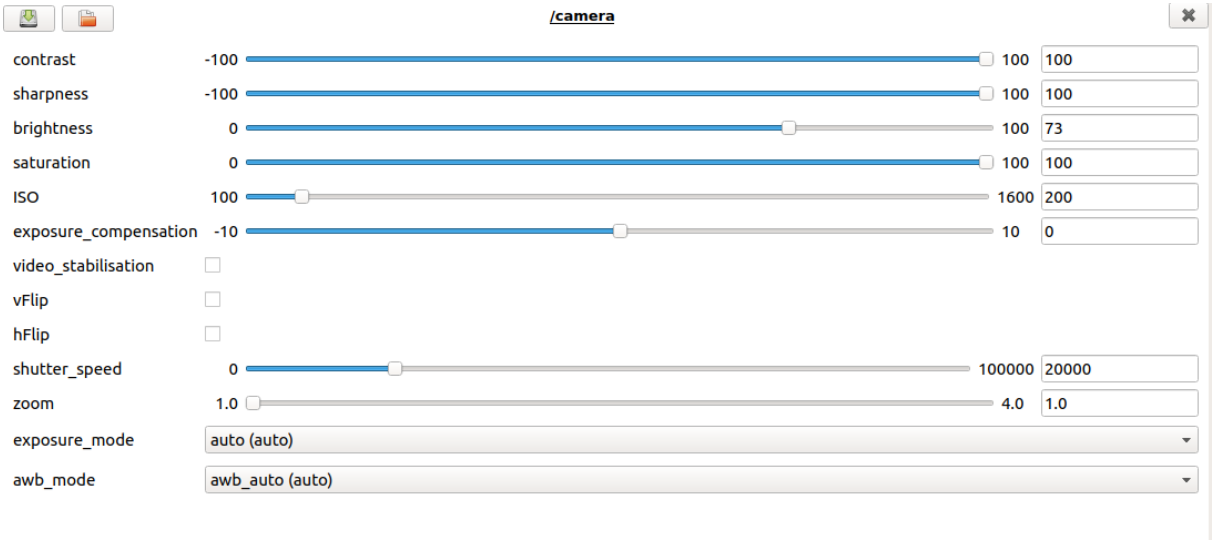
```
roslaunch turtlebot3_autorace_traffic_light_camera turtlebot3_autorace_camera_pi.launch
```

2.Calibration

The first step is to calibrate the different parameters of the camera which include the camera itself, the intrinsic parameters, and the extrinsic parameters.

2.1. Camera calibration

The calibration of the camera allows to obtain a sharper image with higher contrast. These will subsequently simplify the recognition of yellow and white colors and therefore lines. The setting is done through the rqt_reconfigure window which allows you to modify the values of the topics in use using sliders. We will deal here with the **/camera topic.**

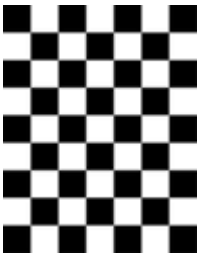


In our case here are the values for which we obtained a satisfactory image:

```
camera:
  ISO: 200
  awb_mode: awb_auto
  brightness: 73
  contrast: 100
  exposureCompensation: 0
  exposure_mode: auto
  hFlip: false
  saturation: 100
  sharpness: 100
  shutterSpeed: 20000
  vFlip: false
  videoStabilisation: false
  zoom: 1.0
```

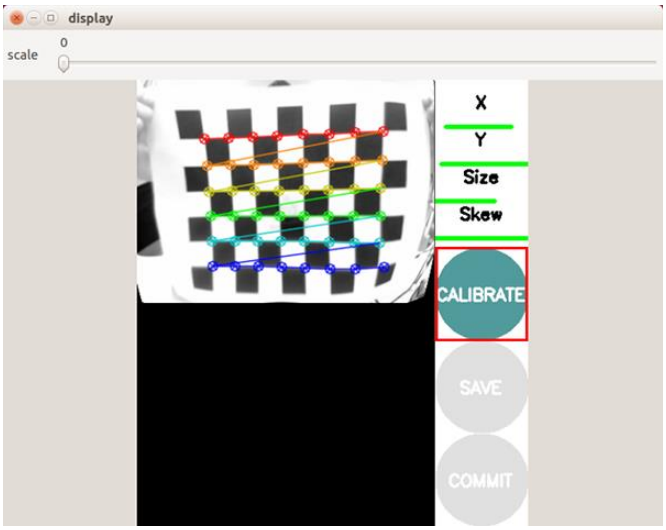
2.2. Intrinsic calibration

For the calibration of the intrinsic parameters of the camera we need a checkerboard.



Run intrinsic camera calibration launch file on Remote PC.

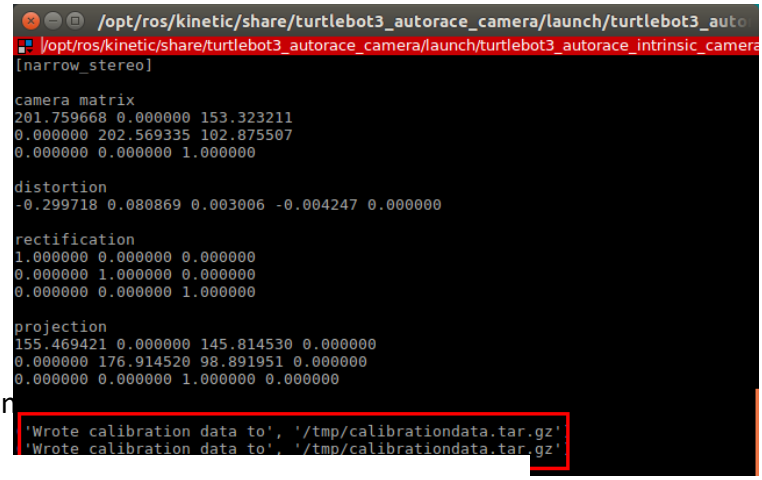
```
export AUTO_IN_CALIB=calibration

export GAZEBO_MODE=false

roslaunch turtlebot3_autorace_traffic_light_camera
turtlebot3_autorace_intrinsic_camera_calibration.launch
```



AUTO_IN_CALIB is a launch file variable defining the mode of use: calibration or action.

turtlebot3_autorace_intrinsic_camera_calibration.launch will, in calibration mode, launch the **cameracalibrator.py** node which subscribes to the raw ROS image subjects, and presents a calibration window. The calibration window shows the current camera images, highlighting the checkerboard. When the user presses the CALIBRATE button, the node calculates the camera's calibration parameters. When the user clicks COMMIT, the node downloads these new calibration settings to the camera driver using a service call. In action mode the launch file will retrieve the previously acquired data for use in the camera configuration.

calibrationdata.tar.gz folder will be created at **/tmp folder**.



Extract calibration

```
image_width: 320
image_height: 240
camera_name: narrow_stereo
camera_matrix:
  rows: 3
  cols: 3
  data: [ 151.92778,    0.      ,  155.93112,
            0.      ,  154.06819,  103.67015,
            0.      ,    0.      ,    1.     ]
camera_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.312921, 0.065575, 0.001588, 0.001507, 0.000000]
rectification_matrix:
  rows: 3
  cols: 3
  data: [ 1.,  0.,  0.,
          0.,  1.,  0.,
          0.,  0.,  1.]
projection_matrix:
  rows: 3
  cols: 4
  data: [ 103.28983,    0.      ,  157.72701,    0.      ,
            0.      ,  113.65757,   94.06372,    0.      ,
            0.      ,    0.      ,    1.     ,    0.      ]
```

2.3. Extrinsic calibration

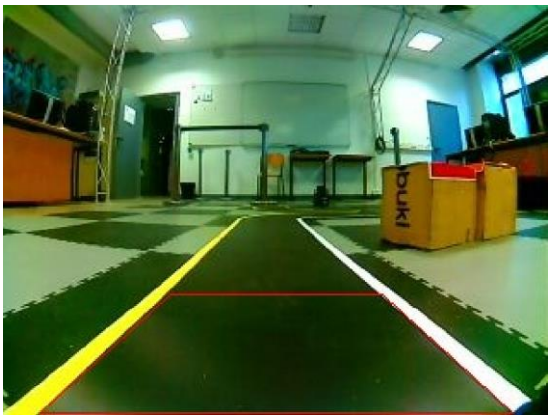
Use the following commands on remote PC:

```
export AUTO_IN_CALIB=calibration

export GAZEBO_MODE=false

roslaunch turtlebot3_autorace_traffic_light_camera
turtlebot3_autorace_intrinsic_camera_calibration.launch
```

```
export AUTO_EX_CALIB=calibration

roslaunch turtlebot3_autorace_traffic_light_camera
turtlebot3_autorace_extrinsic_camera_calibration.launch
```

```
rqt
```

```
rosrun rqt_reconfigure rqt_reconfigure
```



```
!!python/object/new:dynamic_reconfigure.encoding.Config
dictitems:
  bottom_x: 138
  bottom_y: 118
  groups: !!python/object/new:dynamic_reconfigure.encoding.Config
    dictitems:
      bottom_x: 138
      bottom_y: 118
      groups: !!python/object/new:dynamic_reconfigure.encoding.Config
        state: []
      id: 0
      name: Default
      parameters: !!python/object/new:dynamic_reconfigure.encoding.Config
        state: []
      parent: 0
      state: true
      top_x: 65
      top_y: 11
      type: ''
    state: []
  top_x: 65
  top_y: 11
state: []
```

parameter.yaml *file contains the dimension of the trapezoid.*

Intrinsic camera calibration will transform the image surrounded by the red rectangle and will show the image that looks from over the lane. This view will allow to have an undistorted image of the lines which will facilitate their recognition.

3. Lane following

3.1. Calibration

Lane detection package allows Turtlebot3 to drive between two lanes without external influence.

The following instructions describe how to use the lane detection feature and to calibrate camera via rqt.

We need to take care to the orientation. The white lane must be in the right of the turtlebot.

We run intrinsic camera calibration launch file followed by the extrinsic calibration on Remote PC as we see previously.

Then, we will run the the detection lane launch file :

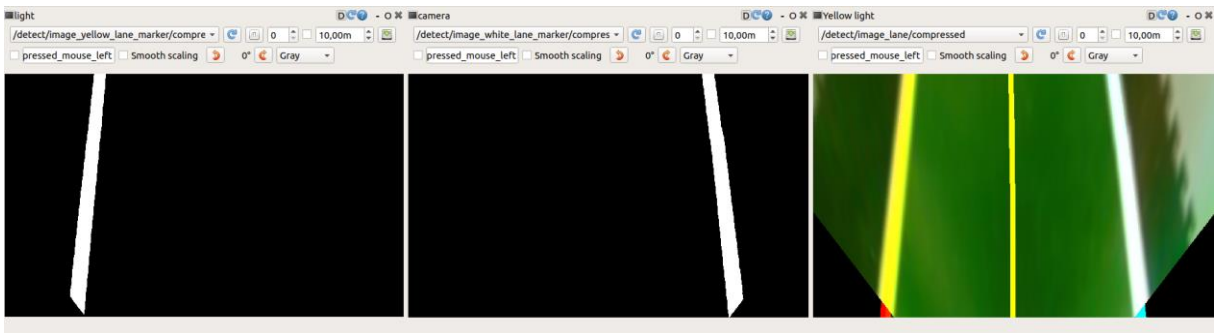
```
export AUTO_DT_CALIB=action

roslaunch turtlebot3_autorace_traffic_light_detect turtlebot3_autorace_detect_lane.launch
```

The line detection is done via three different topics:

- /detect/image_yellow_lane_marker/compressed
- /detect/image_white_lane_marker/compressed
- /detect/image_lane/compressed.

The first two will be set with rqt_reconfigure. We can then adjust low and high thresholds for hue, saturation and brightness. Depending on these parameters it is possible to isolate the color of the line, hence the interest of having a properly calibrated image as we saw in Part 1. The topic image_lane will retrieve the detection made by the two previous topics and return a fusion of the two and the image returned by the camera. Moreover, the center of the track is calculated according to the position of the yellow and white lines. When this one is perfectly calculated, the track becomes green as we can see on the right image below.

The process of calibrating the line color filtering is sometimes difficult due to the physical environment, such as room brightness, etc.



Once the calibration is done correctly, the hue, saturation and luminosity values are saved in the **lane.yaml** file. This same file will then be used by the launch file **turtlebot3_autorace_detect_lane then turtlebot3_autorace_control_lane** to correctly detect the lines and adjust the movement of the robot according to the circuit.

In our case we use these values :

```
detect:
  lane:
    white:
      hue_l: 0
      hue_h: 163
      saturation_l: 0
      saturation_h: 90
      lightness_l: 191
      lightness_h: 255
    yellow:
      hue_l: 0
      hue_h: 36
      saturation_l: 128
      saturation_h: 255
      lightness_l: 114
      lightness_h: 255
```

3.2. Action

Now that the lines of the track are perfectly detected, all that remains is to launch the robot.

To do this we will switch the launch file turtlebot3_autorace_detect_lane to action mode and then run it. Then we launch the control of the line.

```
export AUTO_DT_CALIB=action

roslaunch turtlebot3_autorace_traffic_light_detect turtlebot3_autorace_detect_lane.launch
```

```
roslaunch turtlebot3_autorace_traffic_light_control
turtlebot3_autorace_control_lane.launch
```

If everything has been correctly calibrated, the turtlebot will start up and follow the center line of the track.

3.2.1. Speed control

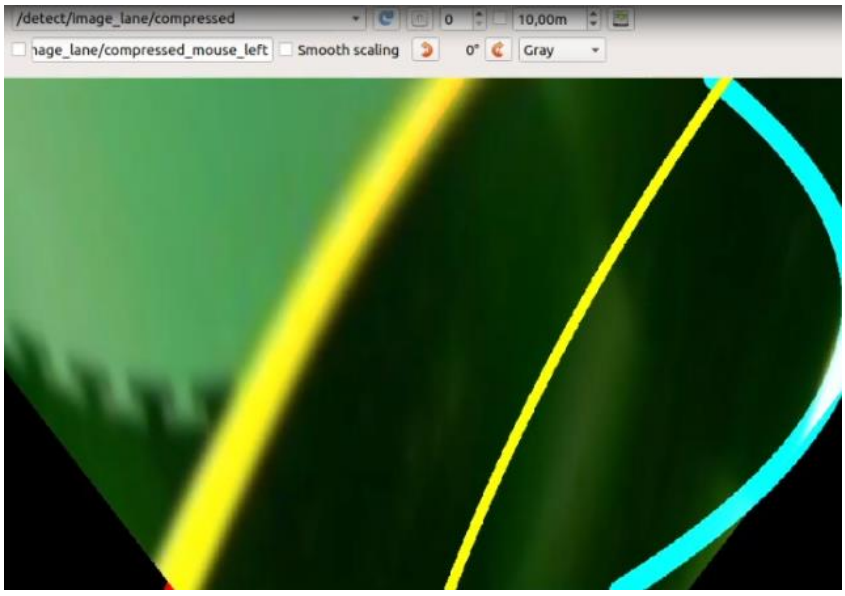
In the **control_lane** node there is a function called *FollowLane*. This is this function who managed the speed of the robot. To increase or decrease the speed we just change the value of the variable twist.linear.x. This function will publish a message to the topic **/cmd_vel** which will use the value of displacement on the x axis indicated by the user to control the speed of the wheel servomotors. Here we have a speed of 0.1.

When the launch file **control_lane** is killed the node sends a null value to the topic **/cmd_vel** which has the effect of stopping the robot.

3.2.2. Special cases

Turn left and right :

In curves, one of the lines may go out of the camera's field of view. In this case the **control_moving** node will keep the distance between the visible line and the last value of the center until the second line is detected again. For exemple in the image below, the robot is in the right turn so the white lane is lost. We can see the distance between the center lane and the yellow lane is keeped.



Tunnel :

Since line detection is dependent on ambient light, it can be lost when the environment is too dark, such as in a tunnel. To remedy this problem, it is sufficient to normalize the image acquired by the camera so that it adapts to the brightness of the room. On the other hand, the adjustment that we have made on the camera allows to attenuate the effects of a bright light directed towards the track and to keep a sufficient detection of the lines.

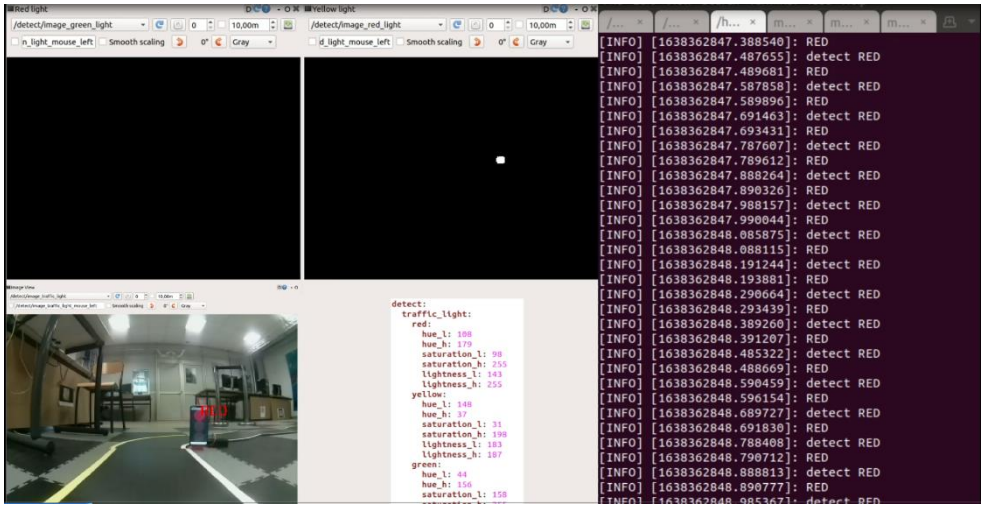
4. Traffic light detection

4.1. Calibration

The objective of this mission is to move the turtlebot only when the light is green. For this we need to detect the right color.

The topics **/detect/image_red_light**, **/detect/image_yellow_light**, **/detect/image_green_light** and **/detect/image_traffic_light** allow to visualize the different colors of the light and the topic **/detect_traffic_light** contains the parameters to adjust. With rqt_reconfigure we will be able to modify the latter in the same way as we did for the lane detection. The values are stored in the file **traffic_light.yaml**.

```
export AUTO_DT_CALIB=calibration

roslaunch turtlebot3_autorace_traffic_light_detect
turtlebot3_autorace_detect_traffic_light.launch
```



Only the green detection is necessary to give the start to the robot but we also wanted to calibrate the red in order to stop the robot after one lap. Unfortunately, this function could not be implemented as we will see later.

```
detect:
  traffic_light:
    red:
      hue_l: 108
      hue_h: 179
      saturation_l: 98
      saturation_h: 255
      lightness_l: 143
      lightness_h: 255
    yellow:
      hue_l: 148
      hue_h: 37
      saturation_l: 31
      saturation_h: 198
      lightness_l: 183
      lightness_h: 187
    green:
      hue_l: 44
      hue_h: 156
      saturation_l: 158
      saturation_h: 255
      lightness_l: 131
      lightness_h: 255
```

4.2. Action

In this part we still have to launch the intrinsic parameters file, but only the extrinsic calibration mode needs to be specified. Indeed the launch file turtlebot3_autorace_core will call it automatically after detecting the green light.

```
export AUTO_IN_CALIB=action

export GAZEBO_MODE=false

roslaunch turtlebot3_autorace_traffic_light_camera
turtlebot3_autorace_intrinsic_camera_calibration.launch
```

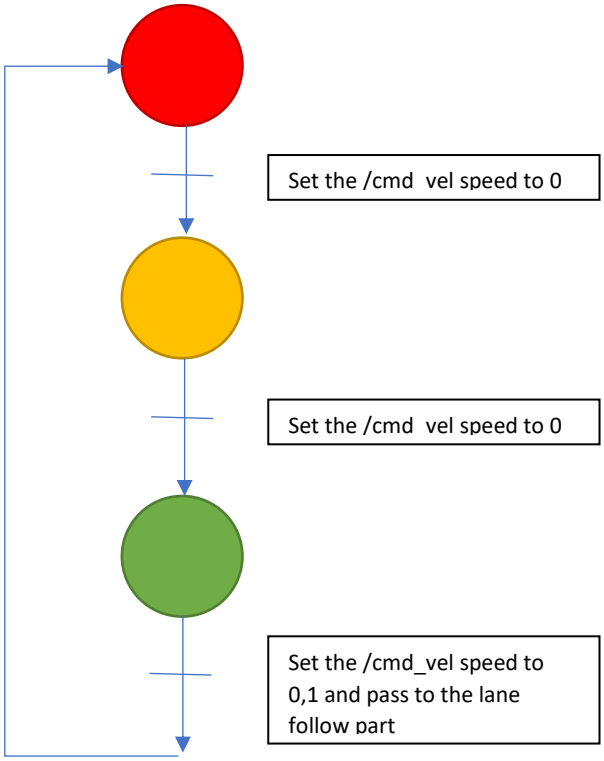
```
export AUTO_EX_CALIB=action

export AUTO_DT_CALIB=action

roslaunch turtlebot3_autorace_traffic_light_core turtlebot3_autorace_core.launch
```

After that, we publish a message to the decided_mode topic to indicate we want to use the traffic_light_detection.

```
rostopic pub -1 /core/decided_mode std_msgs/UInt8 "data: 3"
```

4.2.3. Detect_traffic_light

The management of the traffic light is a linear loop where each step is performed after the other. The loop will first check if the red light is detected and if so, apply the corresponding action. Same thing for the yellow. When the green is detected, another message is sent ordering the robot to move forward, and the loop stops so that the program can move to the lane follow part. The launch file **detect_lane** will then be executed followed by the launch file **control_lane**. If the green is not detected then the loop continues infinitely.



Note: Each color must be detected a certain number of times defined in the node before being validated and considered as detected

4.2.4. Problems

Wrong start

Although the calibration is rigorously carried out, when the action mode is used the program sometimes detects green elsewhere than on the light, which causes unexpected departures.

Stop function

This linear way of proceeding prevents us to realize the stop function initially wanted with the traffic light. Indeed, once out of the light detection loop, it is impossible to come back to it and the robot will continue to execute the line following function. One of the possible solutions would be to execute the detection of the traffic light permanently in the background but this would be a problem if the same red color was detected elsewhere than on the traffic light. Another solution would be to warn the robot via a traffic sign of the presence of the traffic light. So, this program would continuously search for the sign and when it would appear the line following would be interrupted to execute the detection of the traffic light.

5. Conclusion

The initial objective was to create a program that would allow a turtlebot to follow a track and this one is entirely fulfilled. Our robot is able to move autonomously along the track and to pass the different turns without any problem.

As we have seen, it is necessary to perform an excellent calibration in order to have a robust tracking in all circumstances. In this sense our calibration could be improved in order to better adapt to dark environments.

In addition to this basic mission we wanted to add a traffic light recognition function. This part, although functional, is still unfortunately unstable and a thorough study of the different function files is necessary to improve the stability and robustness of this function.

6. References

https://emanual.robotis.com/docs/en/platform/turtlebot3/autonomous_driving/#getting-started

http://wiki.ros.org/camera_calibration#cameracalibrator.py

ROS Basics in 5 days

Mastering with ROS: Turtlebot3

ROS Perception in 5 Days