Lanes Detection Task

2024233125 Chen Xuanxin

References

Link: SI360-Task2

Guidance Book: ROS2_fishor

My Experiment

Main Process

1. Play information from given `culane_test.bag` to topic `detection_target`
2. Subscribe Image Data in `detection_target` and convert ROS data to CV data, then use detection algorithm with specific parameters to obtain detected lanes for constructing results and drawing result images.
3. Publish these results into provided topics with given message format.
4. Record `detection_result` topic to a bag as the final output.

Details

Focus on `lane_detection_starter_cpp/src/detect.cpp`

Publisher and Subscriber

From the codes, we already know that there are provided two publisher `result_pub_`, `detected_img_pub_` and one subscriber `target_sub_`.
`result_pub_` will publish `DetectionResult` to `detection_result` topic.
`detected_img_pub_` will publish `Image` to `detected_image` topic.
`target_sub_` will subscribe `DetectionTarget` from `detection_target` topic, which are bound to function `target_callback`.

Message

DetectionResult

```
uint32 frame_id // index of frame
Lane[] lanes // detected lanes set
Lane[] gt_lanes // real lanes set
float64 run_time // detector running time
```

DetectionTarget

```
std_msgs/Header header // header
uint32 frame_id // index of frame
sensor_msgs/Image image_raw // raw image
Lane[] gt_lanes // real lanes set
```

Lane

```
uint32[] coordinates // lane coordinates
```

Function Instructions

1. `target_callback` - Main Function

- record running time and print log information
- assemble `DetectionResult` construct, initialize in detection_result(header, frame_id, gt_lanes), update in detection_result.lanes
- detect lanes using `lane_detection`
- draw new images with detected lanes using `draw_lanes`
- let 2 publisher to publish messages

2. `cv_bridge`

Convert ROS2 Image messages to OpenCV messages.

3. `lane_detection`

It applies 3 functions to detect lanes, which respectively is `canny_edge_detection`, `region_of_interest` and `HoughLinesP`.
Ultimately, it returns processed lines in `vector`.
And this template just give some random parameters for you, you should update them based on your test sets.

- canny edge detection (3 parts):
  First, it turns images to Grayscales.
  Then it makes Gaussian Blur to Grayscales.
  Finally, it use canny edge detection.
- find interest region:
  Through padding ploygons to generate regions, then remain edges of these areas.
- HoughLinesP:
  To detect straight lines.

4. `draw_lane`
   Draw real lanes and detected lanes in the same image.

Focus on `culane_to_bag/culane_to_bag/convert.py`

Generate culane_test.bag for playing in `detect_target` topic and play it in /detection_target

Convert OpenCV to ROS image messages.
In your workspace: mine is `/home/cxx/SMART_VEHICLES/Detect_Vehicle_Lanes/templates/cxx`

load lane_detection_msgs

build: `colcon build --packages-select lane_detection_msgs`

source workspace: `source install/setup.bash`

generate bag

build: `colcon build --packages-select culane_to_bag`

source workspace: `source install/setup.bash`

common usage: `ros2 run culane_to_bag convert /path/to/culane/dataset /path/to/output.bag`

mine: `ros2 run culane_to_bag convert /home/cxx/SMART_VEHICLES/Detect_Vehicle_Lanes/templates/cxx/Test_dataset/test_images /home/cxx/SMART_VEHICLES/Detect_Vehicle_Lanes/templates/cxx/Test_dataset/test_bags.bag`

play bag

`ros2 bag play /home/cxx/SMART_VEHICLES/Detect_Vehicle_Lanes/templates/cxx/Test_dataset/test_bags.bag --topic /detection_target`

Testing

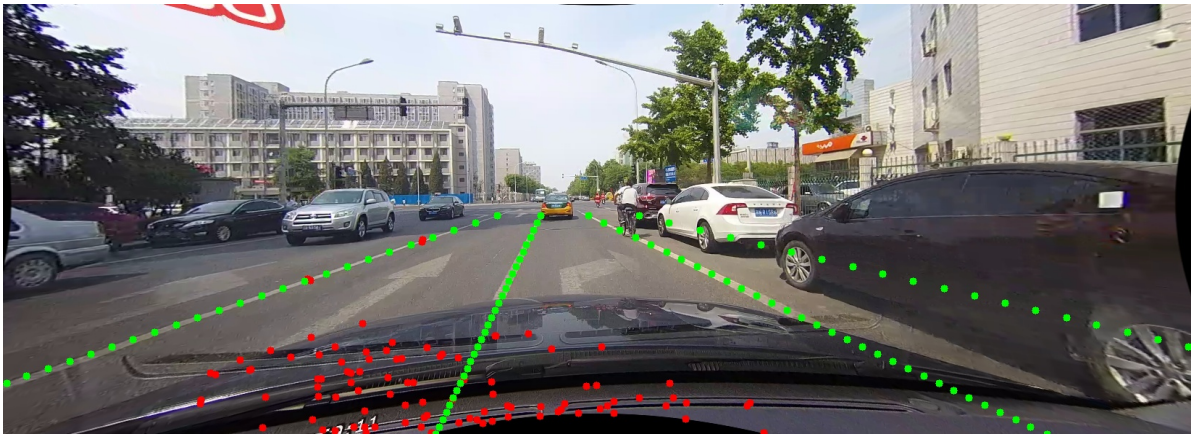
Use UI Window to check detected results

Add UI Window for showing and check the items for more information. In detected images, **Red** dots are detected lanes, while **Green dots** are ground-truth lanes.

It is obvious that the detected performance with initial parameters is terrible from the following example: it failed to detect any lanes and detect the wrong interest regions - a car hood.

Canny_edges:

Detected_lanes:



Analyze failure reason

canny_edge_detect

`cv::GaussianBlur(gray, blur, cv::Size(5, 5), 0)`: blur core is 5 X 5. Larger core will reduce details(sounds) in image. The blur core setting can be updated or change another method to create blur images.

`cv::Canny(blur, canny, 50, 150)`: low threshold is 50, high threshold is 150. Pixel between two bounds will be an possible edge.The parameters can be updated. Too high or too low may result in extra sounds and details loss.

region_of_interest

Define a region of interest. The area may limit following analysis.

```
cv::Point pts[1][3];
pts[0][0] = cv::Point(200, canny.rows);
pts[0][1] = cv::Point(1100, canny.rows);
pts[0][2] = cv::Point(550, 250);
```

Fill this region and ignore other areas:
`cv::fillPoly(mask, ppt, npt, 1, cv::Scalar(255, 255, 255), cv::LINE_8);`

HoughLinesP

`cv::HoughLinesP(cropped_image, lines, 2, CV_PI / 180, 100, 40, 5);`

cropped_image: masked edge image.

lines: output detected line segments, including the coordinates of each line segment.

2: accumulator resolution, which indicates the distance accuracy of the line when detecting, in pixels. Larger values can detect thicker lines.

CV_PI / 180: angular resolution, which indicates the angle accuracy when detecting lines, in radians. Here CV_PI / 180 means 1 degree accuracy.

100: threshold, which indicates the minimum number of votes required to detect a line. A higher threshold means that only clear lines will be detected.

40: minimum line length, which indicates the shortest distance that can be detected as a line segment, in pixels. Line segments below this length will be ignored.

5: maximum gap, if the gap between two lines is smaller than this value, they will be connected into a single line.

Update parameters in `lane_detection` and change structure

Update canny_edge_detection

Replace `GaussianBlur` with `bilateralFilter`. In order to balance images that are light or dark instead of using the same mode, to compute adaptive thresholds for canny edge detection.

Here are paramters of `bilateralFilter`:
`cv::bilateralFilter(gray, blur, d, sigmaColor, sigmaSpace)`

`d`: This parameter represents the diameter of the filter, which affects the size of the neighborhood. The larger the value, the larger the neighborhood range, and the stronger the filtering effect.

`sigmaColor`: This is the standard deviation in the color space. It controls the influence of color differences within the neighborhood. The larger the value, the more the filter will smooth regions with significant color differences.

`sigmaSpace`: This is the standard deviation in the coordinate space. It controls the influence of spatial distance. The larger the value, the more distant pixels will be considered neighbors, increasing the degree of blurring.

Then it applies `Sobel` to enhance edge to obtain a sharpened grayscale. Finally, it uses above components to compute canny values.

`cv::Sobel(blur, sobelX, ...)` computes the horizontal gradients in the image. This highlights edges that are vertical in the image.

`cv::Sobel(blur, sobelY, ...)` computes the vertical gradients in the image. This highlights edges that are horizontal in the image.

Update region_of_interest

In the original template, it provide a triangle area to refine computation range of images. But according to my experiment, it can extend to a polygon to cover more regions to include more details. In this way, my task designs a hexagon area to create a filled polygon for next step.

HoughLinesP parameters

In order to get more dots and improve the detection of continuous lines in the original image, the parameters in the `HoughLinesP` can be adjusted as follows:

Threshold: Increasing the threshold makes the algorithm more selective, only detecting lines with stronger evidence in the image. This helps reduce the number of detected short or weak lines, filtering out noisy dots.

Min Line Length: Setting a larger minimum line length ensures that shorter, disconnected lines (like small dots) are ignored, favoring longer, continuous lines.

Max Line Gap: This allows for connecting points into a single line, but by keeping it relatively small, it prevents over-connecting lines that are too far apart, reducing false connections.

In this way, my work selects (110, 30, 10), according to my analysis principle, it aims to detect as many potential lanes as possible. The threshold of 110 filters out weaker lines, but still allows the detection of significant lanes. The minimum line length of 30 ensures that only substantial lane segments are considered, avoiding short or noisy detections. The maximum line gap of 10 allows for detecting lanes that may be broken or dashed, ensuring continuity between segments. Through comparison with detected results with other parameters, it can be a best solution for this test set.

Launch whole process

Environment & requirements

- Ubuntu 22.04
- ROS2 humble version
- `python` >= 3.10.1
- `numpy` == 1.21.2
- `opencv-python` == 4.10.0.84
- `VSCode`

Launch script

According to my task structure, my main workspace is `lane_detection_starter_cpp`. So it is necessary to edit `CMakeLists.txt` and `package.xml` to remain build process successful.

In `lane_detection_starter_cpp/src`, copy `detector.cpp` and rename `my_detector.cpp`. Add my update codes to original template.

In `lane_detection_starter_cpp/`, create a new folder `launch` and create your script in it, mine is `lane_detection_launch.py`.

In `CMakeLists.txt`, add necessaries to build `my_detector` and `launch` folder.
Add executions and their dependencies, then create link manually:

```
add_executable(mydetector src/my_detector.cpp)
ament_target_dependencies(mydetector rclcpp cv_bridge lane_detection_msgs)
target_link_libraries(mydetector ${OpenCV_LIBS})
```

Add install items:

```
install(TARGETS mydetector
  DESTINATION lib/${PROJECT_NAME})
install(
  DIRECTORY launch/
  DESTINATION share/${PROJECT_NAME}/launch
)
```

Here are my commands that needs to be lanuched by scripts:

1. build and convert (contents in `build.sh`)

```
#!/bin/bash
# build packages
colcon build --packages-select culane_to_bag --allow-overriding culane_to_bag
colcon build --packages-select lane_detection_msgs --allow-overriding
lane_detection_msgs
colcon build --packages-select lane_detection_starter_cpp --allow-overriding
lane_detection_starter_cpp
# source workspace
source
/home/cxx/SMART_VEHICLES/Detect_Vehicle_Lanes/templates/cxx/install/setup.bash
# convert test_bag.bag
ros2 run culane_to_bag convert
/home/cxx/SMART_VEHICLES/Detect_Vehicle_Lanes/templates/cxx/Test_dataset/test_im
ages
/home/cxx/SMART_VEHICLES/Detect_Vehicle_Lanes/templates/cxx/Test_dataset/test_ba
gs.bag
# exit terminal
exit 0
```

2. play bag

```
ros2 bag play
/home/cxx/SMART_VEHICLES/Detect_Vehicle_Lanes/templates/cxx/Test_dataset/test_ba
gs.bag --topic /detection_target
```

3. run `my_detector` node and record bag

```
ros2 run lane_detection_starter_cpp mydetector & ros2 bag record
/detection_result -o
/home/cxx/SMART_VEHICLES/Detect_Vehicle_Lanes/templates/cxx/Test_dataset/detecti
on_result.bag
```

To run launcher scripts:

Commands:

```
cd /home/cxx/SMART_VEHICLES/Detect_Vehicle_Lanes/templates/cxx/

colcon build --packages-select lane_detection_starter_cpp

source install/setup.bash

ros2 launch lane_detection_starter_cpp lane_detection_launch.py
```

Results:

```
root@cxx-virtual-machine:~/SMART_VEHICLES/Detect_Vehicle_Lanes/templates/cxx# ros2 launch lane_
detection_starter_cpp lane_detection_launch.py
[INFO] [launch]: All log files can be found below /home/cxx/.ros/log/2024-10-21-00-33-34-221642
-cxx-virtual-machine-281682
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [xterm-1]: process started with pid [281683]
[INFO] [xterm-1]: process has finished cleanly [pid 281683]
[INFO] [xterm-2]: process started with pid [282156]
[INFO] [xterm-3]: process started with pid [282159]
```

The lauched sequence is as follow:

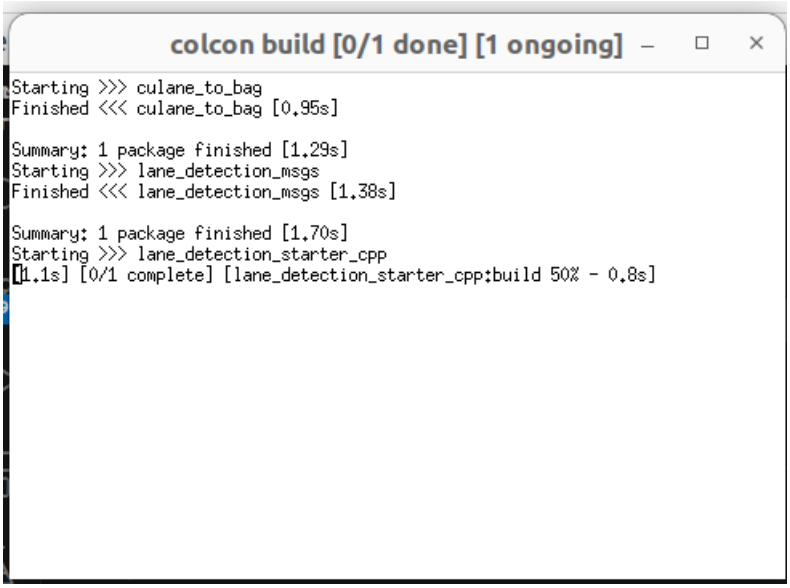
```
    return LaunchDescription([
        build_and_source,
        # 当 build.sh 完成后，执行转换和播放
        RegisterEventHandler(
            OnProcessExit(
                target_action=build_and_source,
                on_exit=[
                    TimerAction(period=0.0, actions=[convert_and_play_bag])  # 执行转换并播放 bag
                ]
            )
        ),
        # 当播放 bag 文件开始后，启动检测节点并记录 bag
        RegisterEventHandler(
            OnProcessStart(
                target_action=convert_and_play_bag,
                on_start=[run_detection_and_record_bag]
            )
        )
    ])
```

In order to control node, it updates `convert.py` as follow:

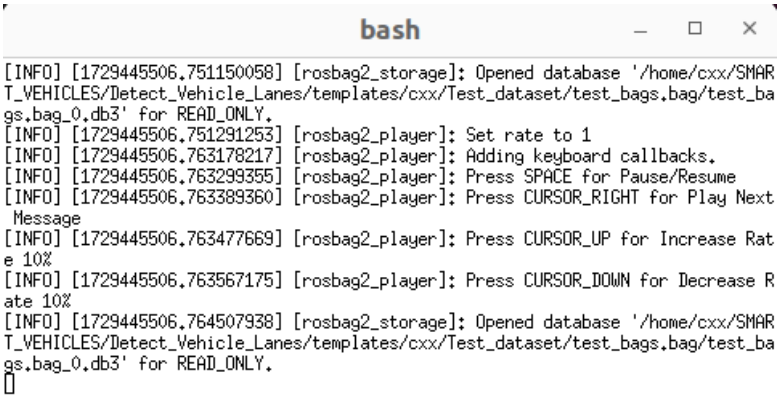
```
    while rclpy.ok():
        rclpy.spin_once(node, timeout_sec=0.1)
        if not os.path.exists(args.input_dir):
            break

    node.destroy_node()
    rclpy.shutdown()
```

Details

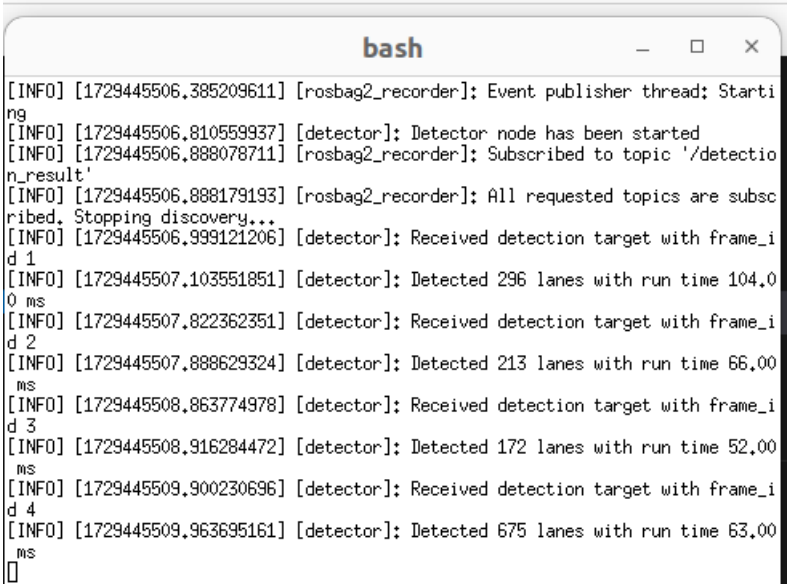
Build and convert process: (`Ctrl+C` to next step)



Play bag process:



Run node process:

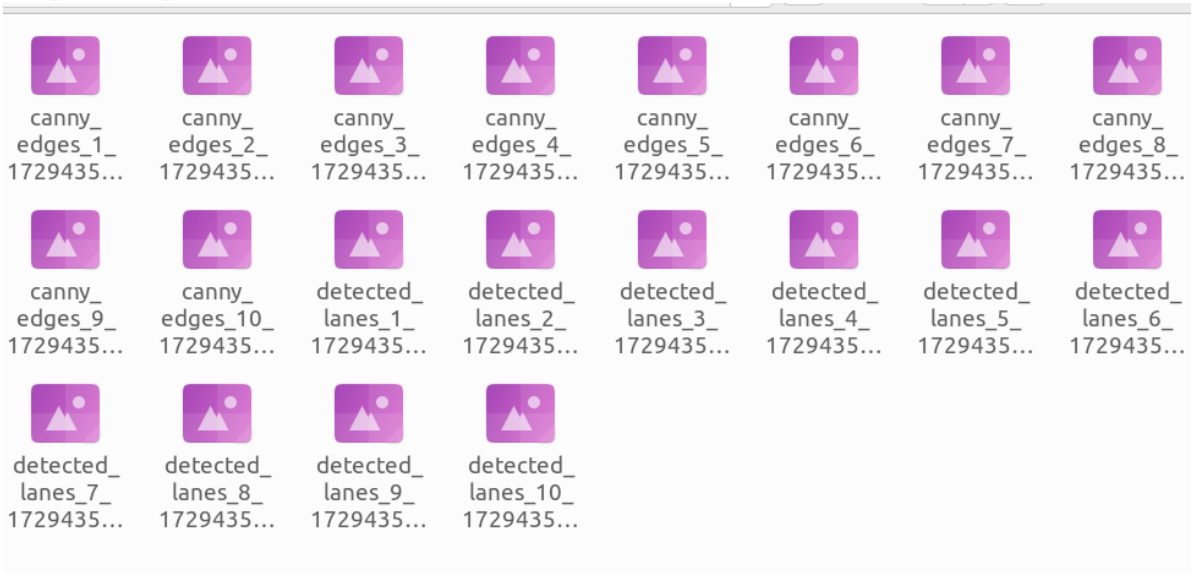


Stop recording bag: (`Ctrl+C` to stop)

```
bash
 ms
[INFO] [1729445513.040734831] [detector]: Received detection target with frame_i
d 7
[INFO] [1729445513.092321457] [detector]: Detected 153 lanes with run time 51.00
 ms
[INFO] [1729445514.097971300] [detector]: Received detection target with frame_i
d 8
[INFO] [1729445514.153304515] [detector]: Detected 160 lanes with run time 55.00
 ms
[INFO] [1729445515.132323404] [detector]: Received detection target with frame_i
d 9
[INFO] [1729445515.187676636] [detector]: Detected 277 lanes with run time 55.00
 ms
[INFO] [1729445516.170225659] [detector]: Received detection target with frame_i
d 10
[INFO] [1729445516.227002818] [detector]: Detected 284 lanes with run time 56.00
 ms
[INFO] [1729445539.085346666] [rclcpp]: signal_handler(signum=2)
[INFO] [1729445539.085688902] [rosbag2_cpp]: Writing remaining messages from cac
he to the bag. It may take a while
[INFO] [1729445539.087451784] [rosbag2_recorder]: Event publisher thread: Exitin
g
[INFO] [1729445539.088343156] [rosbag2_recorder]: Recording stopped
```

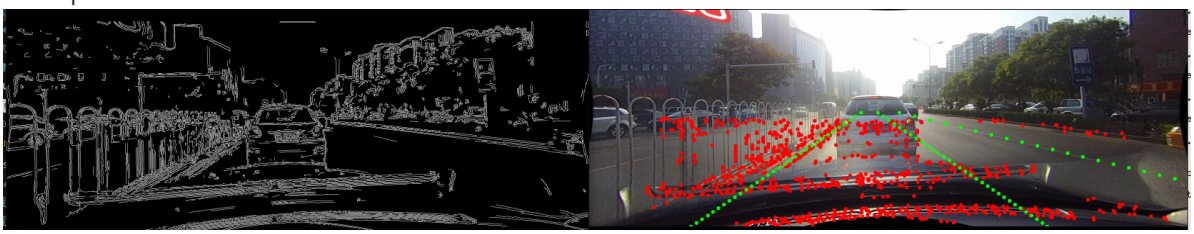
Outputs:

Images in single folder:

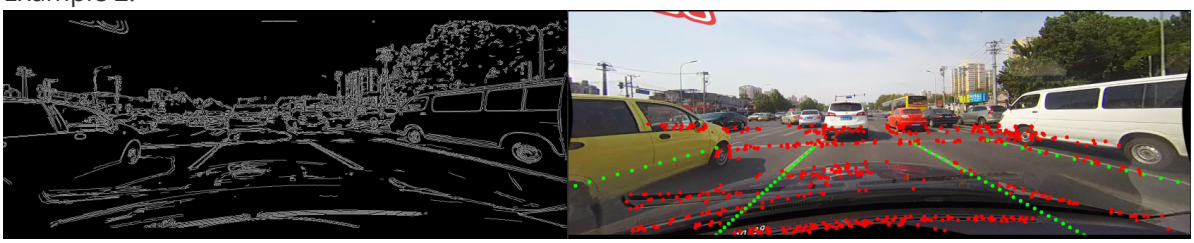


Canny image and Detected image:

Example 1:



Example 2:



Check for record bag:

```
root@cxx-virtual-machine:~/SMART_VEHICLES/Detect_Vehicle_Lanes/templates/cxx/res
# ros2 bag info detection_result.bag

Files:             detection_result.bag_0.db3
Bag size:          105.0 KiB
Storage id:        sqlite3
Duration:          9.126679156s
Start:             Oct 20 2024 23:20:11.341009820 (1729437611.341009820)
End:               Oct 20 2024 23:20:20.467688976 (1729437620.467688976)
Messages:          10
Topic information: Topic: /detection_result | Type: lane_detection_msgs/msg/Dete
ctionResult | Count: 10 | Serialization Format: cdr
```

Appendix

`my_detector.cpp`

```
#include <chrono>
#include <functional>
#include <memory>
#include <vector>
#include <cstdint>
#include <opencv2/opencv.hpp>
#include "rclcpp/rclcpp.hpp"
#include "rclcpp/qos.hpp"

#ifdef ROS_DISTRO_JAZZ
#include "cv_bridge/cv_bridge.hpp"
#else
#include "cv_bridge/cv_bridge.h"
#endif
#include "lane_detection_msgs/msg/detection_target.hpp"
#include "lane_detection_msgs/msg/detection_result.hpp"
#include "lane_detection_msgs/msg/lane.hpp"
#include "sensor_msgs/msg/image.hpp"

using std::placeholders::_1;
using namespace std::chrono_literals;

class Detector : public rclcpp::Node
{
public:
  Detector() : Node("detector")
  {
    RCLCPP_INFO(this->get_logger(), "Detector node has been started");
    result_pub_ = this-
>create_publisher<lane_detection_msgs::msg::DetectionResult>(
    "/detection_result", rclcpp::QoS(10));
    detected_img_pub_ = this->create_publisher<sensor_msgs::msg::Image>(
    "/detected_image", rclcpp::QoS(10));
    target_sub_ = this-
>create_subscription<lane_detection_msgs::msg::DetectionTarget>(
    "/detection_target", rclcpp::QoS(10), std::bind(&Detector::target_callback,
this, _1));
  }

private:
```

```
  void target_callback(const
lane_detection_msgs::msg::DetectionTarget::SharedPtr msg)
  {
    RCLCPP_INFO(this->get_logger(), "Received detection target with frame_id
%d", msg->frame_id);

    auto start_time = std::chrono::high_resolution_clock::now();
    lane_detection_msgs::msg::DetectionResult detection_result;
    detection_result.header = msg->header;
    detection_result.frame_id = msg->frame_id;
    detection_result.gt_lanes = msg->gt_lanes;

    cv_bridge::CvImagePtr cv_ptr;
    try {
        cv_ptr = cv_bridge::toCvCopy(msg->image_raw,
sensor_msgs::image_encodings::BGR8);
    } catch (cv_bridge::Exception& e) {
        RCLCPP_ERROR(this->get_logger(), "cv_bridge exception: %s", e.what());
        return;
    }
    // 动态生成唯一的文件名（根据 frame_id 或者时间戳）
    std::string frame_id_str = std::to_string(msg->frame_id);
    std::string timestamp_str = std::to_string(this->get_clock()-
>now().nanoseconds());
    // 文件保存路径前缀
    std::string base_path =
"/home/cxx/SMART_VEHICLES/Detect_Vehicle_Lanes/templates/cxx/Test_dataset/detect
ed_images/";
    // 执行车道检测
    auto detected_lanes = lane_detection(cv_ptr->image);
    // 构建检测结果
    for (const auto& lane : detected_lanes) {
        lane_detection_msgs::msg::Lane lane_msg;
        lane_msg.coordinates.assign(lane.begin(), lane.end());
        detection_result.lanes.push_back(lane_msg);
    }
    auto end_time = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::milliseconds>
(end_time - start_time);
    detection_result.run_time = duration.count();

    RCLCPP_INFO(this->get_logger(), "Detected %zu lanes with run time %.2f ms",
                detected_lanes.size(), detection_result.run_time);

    std::vector<std::vector<uint32_t>> gt_lanes;
    for (const auto& lane : msg->gt_lanes) {
        gt_lanes.push_back(lane.coordinates);
    }
    // 阶段2：边缘检测并保存
    cv::Mat canny_edges = canny_edge_detection(cv_ptr->image);
    std::string canny_filename = base_path + "canny_edges_" + frame_id_str + "_"
+ timestamp_str + ".jpg";
    // cv::imshow("Canny Edge Detection", canny_edges);
    // cv::waitKey(1);  // 确保图像显示
    cv::imwrite(canny_filename, canny_edges);  // 动态保存边缘检测图像
    // 绘制车道线
    draw_lanes(cv_ptr->image, detected_lanes, gt_lanes);
    // 阶段3：显示绘制车道后的图像并保存
```

```
    std::string lanes_filename = base_path + "detected_lanes_" + frame_id_str +
"_" + timestamp_str + ".jpg";
    // cv::imshow("Detected Lanes", cv_ptr->image);
    // cv::waitKey(1);  // 确保图像显示
    cv::imwrite(lanes_filename, cv_ptr->image);  // 动态保存车道线图像
    // 发布结果
    result_pub_->publish(detection_result);
    detected_img_pub_->publish(*(cv_ptr->toImageMsg()));
  }

    cv::Mat canny_edge_detection(const cv::Mat& img_cv)
    {
        cv::Mat gray, blur, canny;
        // 转换为灰度图
        cv::cvtColor(img_cv, gray, cv::COLOR_BGR2GRAY);
        // 应用双边滤波以减少噪声并保持边缘
        cv::bilateralFilter(gray, blur, 9, 75, 75);
        // 自适应阈值计算
        double median = cv::mean(blur)[0]; // 使用中值作为亮度的衡量
        double lower = std::max(0.0, 0.7 * median); // 设置较低的阈值
        double upper = std::min(255.0, 1.3 * median); // 设置较高的阈值
        // 使用Sobel滤波增强边缘
        cv::Mat sobelX, sobelY;
        cv::Sobel(blur, sobelX, CV_64F, 1, 0, 3);  // 水平梯度
        cv::Sobel(blur, sobelY, CV_64F, 0, 1, 3);  // 垂直梯度
        cv::Mat sobel = abs(sobelX) + abs(sobelY);
        cv::Mat sobel_8u;
        cv::convertScaleAbs(sobel, sobel_8u);
        // 应用Canny边缘检测
        cv::Canny(sobel_8u, canny, lower, upper, 3, true);
        return canny;
    }

    cv::Mat region_of_interest(const cv::Mat& canny)
    {
        cv::Mat mask = cv::Mat::zeros(canny.size(), canny.type());
        // 使用一个六边形的感兴趣区域，可以根据图片手动调整顶点坐标
        cv::Point pts[1][6];
        // 定义多边形的顶点，调整使其适应不同道路场景
        pts[0][0] = cv::Point(50, canny.rows);           // 左下角
        pts[0][1] = cv::Point(canny.cols - 50, canny.rows); // 右下角
        pts[0][2] = cv::Point(canny.cols - 200, canny.rows / 2); // 右中间
        pts[0][3] = cv::Point(canny.cols - 600, 300);     // 顶部中间靠右
        pts[0][4] = cv::Point(600, 300);                 // 顶部中间靠左
        pts[0][5] = cv::Point(200, canny.rows / 2);      // 左中间
        const cv::Point* ppt[1] = {pts[0]};
        int npt[] = {6};

        // 填充多边形区域
        cv::fillPoly(mask, ppt, npt, 1, cv::Scalar(255, 255, 255), cv::LINE_8);
        cv::Mat masked_image;
        cv::bitwise_and(canny, mask, masked_image);
        return masked_image;
    }
    std::vector<std::vector<uint32_t>> lane_detection(const cv::Mat& img_cv)
    {
        cv::Mat canny = canny_edge_detection(img_cv);
        cv::Mat cropped_image = region_of_interest(canny);
```

```
        std::vector<cv::Vec4i> lines;
        cv::HoughLinesP(cropped_image, lines, 1, CV_PI / 180, 110, 30, 10); //
Lower threshold and minLineLength

        std::vector<std::vector<uint32_t>> return_lines;
        for (const auto& line : lines)
        {
            return_lines.push_back({line[0], line[1], line[2], line[3]});
        }
        return return_lines;
    }

    void draw_lanes(cv::Mat& img_cv, const std::vector<std::vector<uint32_t>>&
lanes, const std::vector<std::vector<uint32_t>>& gt_lanes)
    {
        for (const auto& lane : lanes)
        {
            for (size_t i = 0; i < lane.size(); i += 2)
            {
                if (i + 1 < lane.size())
                {
                    int x = lane[i];
                    int y = lane[i + 1];
                    cv::circle(img_cv, cv::Point(x, y), 5, cv::Scalar(0, 0,
255), -1);
                }
            }
        }
        for (const auto& lane : gt_lanes)
        {
            for (size_t i = 0; i < lane.size(); i += 2)
            {
                if (i + 1 < lane.size())
                {
                    int x = lane[i];
                    int y = lane[i + 1];
                    cv::circle(img_cv, cv::Point(x, y), 5, cv::Scalar(0, 255,
0), -1);
                }
            }
        }
    }
  rclcpp::Publisher<lane_detection_msgs::msg::DetectionResult>::SharedPtr
result_pub_;
  rclcpp::Subscription<lane_detection_msgs::msg::DetectionTarget>::SharedPtr
target_sub_;
  rclcpp::Publisher<sensor_msgs::msg::Image>::SharedPtr detected_img_pub_;
};
int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);
  rclcpp::spin(std::make_shared<Detector>());
  rclcpp::shutdown();
  return 0;
}
```

lane_detection_launch.py

```
import time
from launch import LaunchDescription
from launch.actions import ExecuteProcess, RegisterEventHandler, TimerAction
from launch.event_handlers import OnProcessExit, OnProcessStart
def generate_launch_description():
    # 定义路径
    time_stamp = time.strftime("%Y%m%d_%H%M%S")
    test_images =
"/home/cxx/SMART_VEHICLES/Detect_Vehicle_Lanes/templates/cxx/Test_dataset/test_i
mages"
    test_bag =
"/home/cxx/SMART_VEHICLES/Detect_Vehicle_Lanes/templates/cxx/Test_dataset/test_b
ags.bag"
    detection_bag =
f"/home/cxx/SMART_VEHICLES/Detect_Vehicle_Lanes/templates/cxx/Test_dataset/detec
tion_result_{time_stamp}.bag"
    workspace_setup_script =
"/home/cxx/SMART_VEHICLES/Detect_Vehicle_Lanes/templates/cxx/install/setup.bash"

    # Part 1: 执行 build.sh 脚本
    build_and_source = ExecuteProcess(
        cmd=[
            'xterm', '-e',
            f"bash -c 'chmod +x
/home/cxx/SMART_VEHICLES/Detect_Vehicle_Lanes/templates/cxx/build.sh &&
/home/cxx/SMART_VEHICLES/Detect_Vehicle_Lanes/templates/cxx/build.sh'"
        ],
        shell=True
    )

    # Part 2: 转换图片到 bag 文件并播放（保持在同一终端）
    convert_and_play_bag = ExecuteProcess(
        cmd=[
            'xterm', '-hold', '-e',
            f"bash -c 'source {workspace_setup_script} && "
            # f"(timeout 1s ros2 run culane_to_bag convert {test_images}
{test_bag}); "  # 使用子 shell控制超时
            f"ros2 bag play {test_bag} --topic /detection_target'"
        ],
        shell=True
    )

    # Part 3: 启动检测节点并记录检测结果
    run_detection_and_record_bag = ExecuteProcess(
        cmd=[
            'xterm', '-hold', '-e',
            f"bash -c 'source {workspace_setup_script} && "
            f"ros2 run lane_detection_starter_cpp mydetector & "
            f"ros2 bag record -o {detection_bag} /detection_result'"
        ],
        shell=True
    )

    return LaunchDescription([
        build_and_source,
```

```
        # 当 build.sh 完成后，执行转换和播放
        RegisterEventHandler(
            OnProcessExit(
                target_action=build_and_source,
                on_exit=[
                    TimerAction(period=0.0, actions=[convert_and_play_bag])  # 执
行转换并播放 bag
                ]
            )
        ),
        # 当播放 bag 文件开始后，启动检测节点并记录 bag
        RegisterEventHandler(
            OnProcessStart(
                target_action=convert_and_play_bag,
                on_start=[run_detection_and_record_bag]
            )
        )
    ])
```