EX #6- Extended Lab Reporting and Demonstration Follow-Up

NAME:

PRANJAL PIMPALE

Project Name: Advanced Driver Assistance System (AV) / (ADAS).

Question 1: Video presentation or annotated video showing results (also referenced in report)

1. **Demonstrate Application**
   Features: Introduction:
   The Advanced Driver Assistance System (AV) / (ADAS) we propose aims to enhance road safety and driver convenience through the integration of various advanced processing and detection technologies. ADAS leverages computer vision and machine learning algorithms to detect and interpret key elements of the driving environment, providing real-time feedback and assistance to the driver. Our system will focus on fve features: **lane detection, trafc light detection, stop sign detection, pedestrian detection, vehicle detection.**

   a) **<u>Detection of Pedestrians:</u>**
   - **Code Snippet:**

trainHog.cpp (~/Desktop/cv5/pedestrain) - gedit

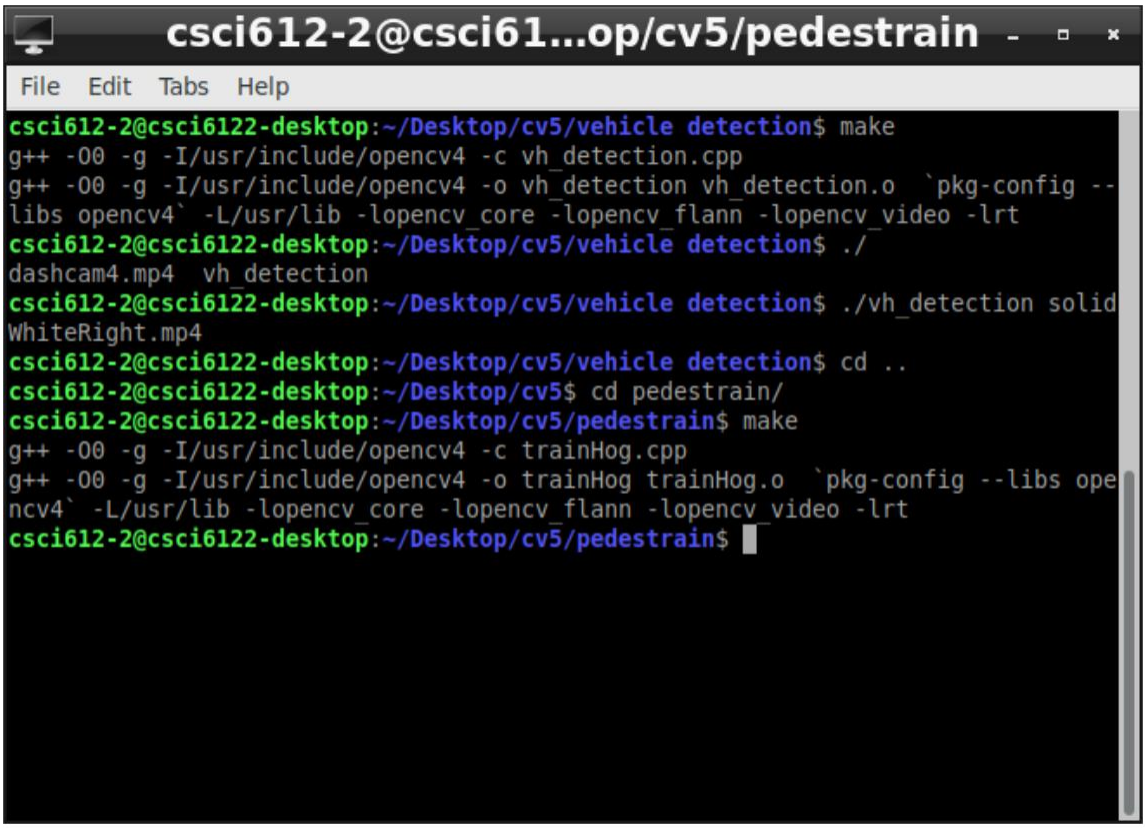
Open | trainHog.cpp ~/Desktop/cv5/pedestrain | Save

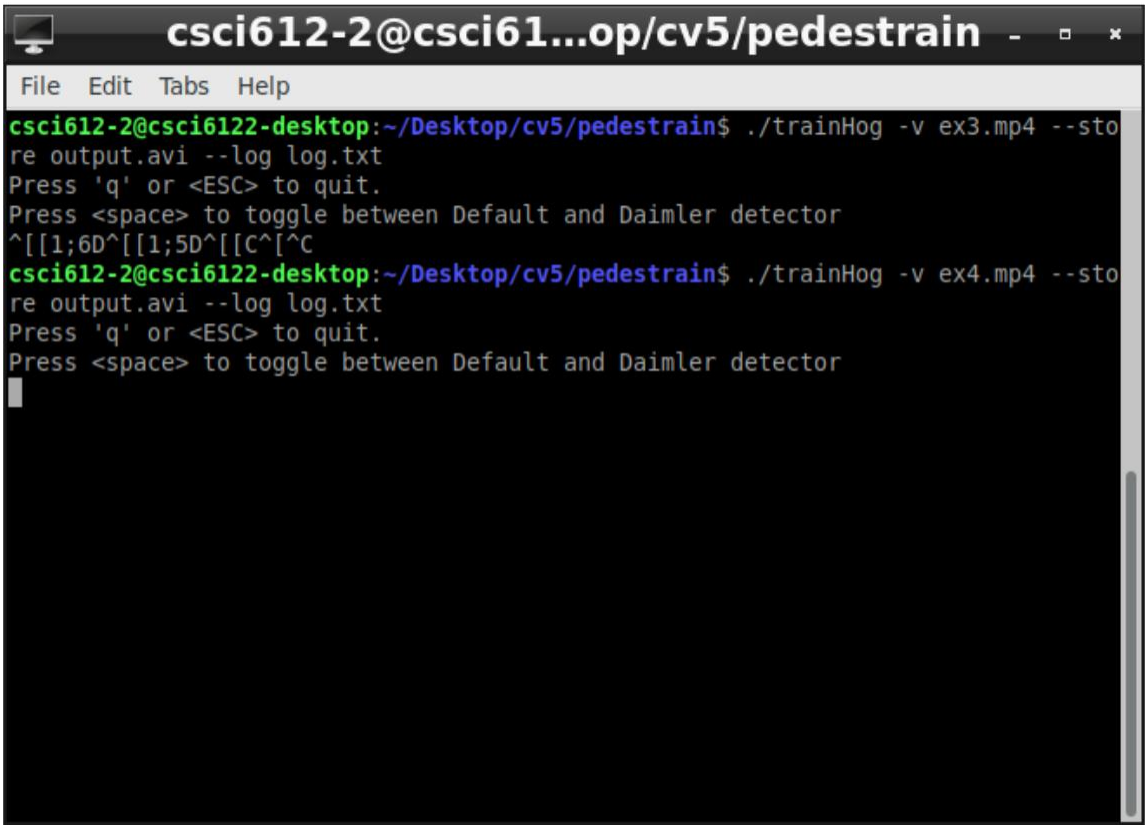
trainHog.cpp | vh_detection.cpp | vehical.cpp | Makefile | Makefile | main.cpp | trainHog.cpp

```
        r.height = cvRound(r.height * 0.8);
    }
};

static const string keys = "{ help h | | print help message }"
                           "{ camera c | 0 | capture video from camera (device index starting from 0) }"
                           "{ video v | | use video as input }"
                           "{ store s | | store results back in a video followed by the filename }"
                           "{ log l | | log the number of pedestrians detected and their coordinates in a file}";


int main(int argc, char **argv)
{
    string file;
    string storeFile;
    string logFile;

    // Parse command-line arguments
    for (int i = 1; i < argc; ++i)
    {
        if (strcmp(argv[i], "-v") == 0 && i + 1 < argc)
        {
            file = argv[i + 1];
            ++i;
        }
        else if (strcmp(argv[i], "--store") == 0 && i + 1 < argc)
        {
            storeFile = argv[i + 1];
            ++i;
        }
        else if (strcmp(argv[i], "--log") == 0 && i + 1 < argc)
        {
            logFile = argv[i + 1];
            ++i;
        }
    }

    // Check if a video file is provided
    if (file.empty())
    {
        cout << "No video file specified." << endl;
        return 1;
    }

    // Open video stream
    VideoCapture cap(file);
    if (!cap.isOpened())
    {
        cout << "Can not open video stream: '" << file << "'" << endl;
```

C++ | Tab Width: 8 | Ln 8, Col 4 | INS

- **Make and build of Pedestrian Detection:**





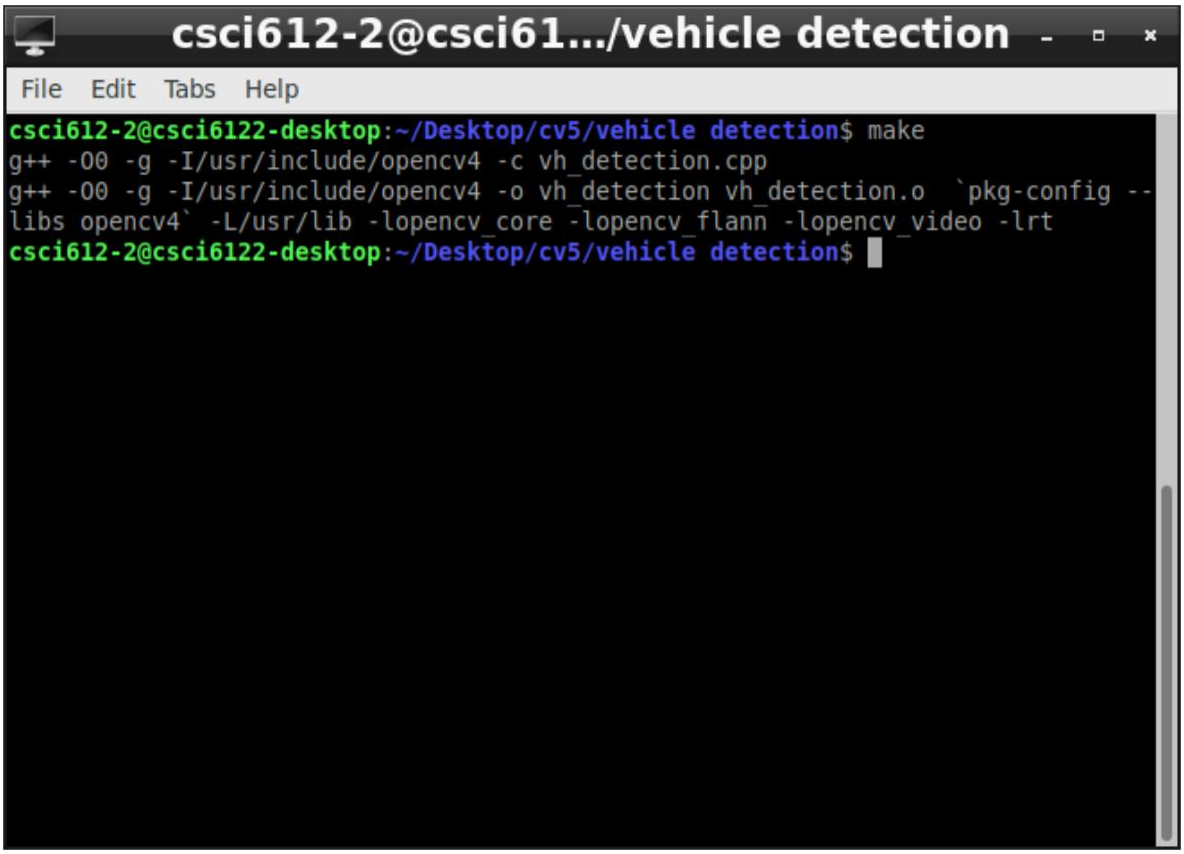
- **Output:**

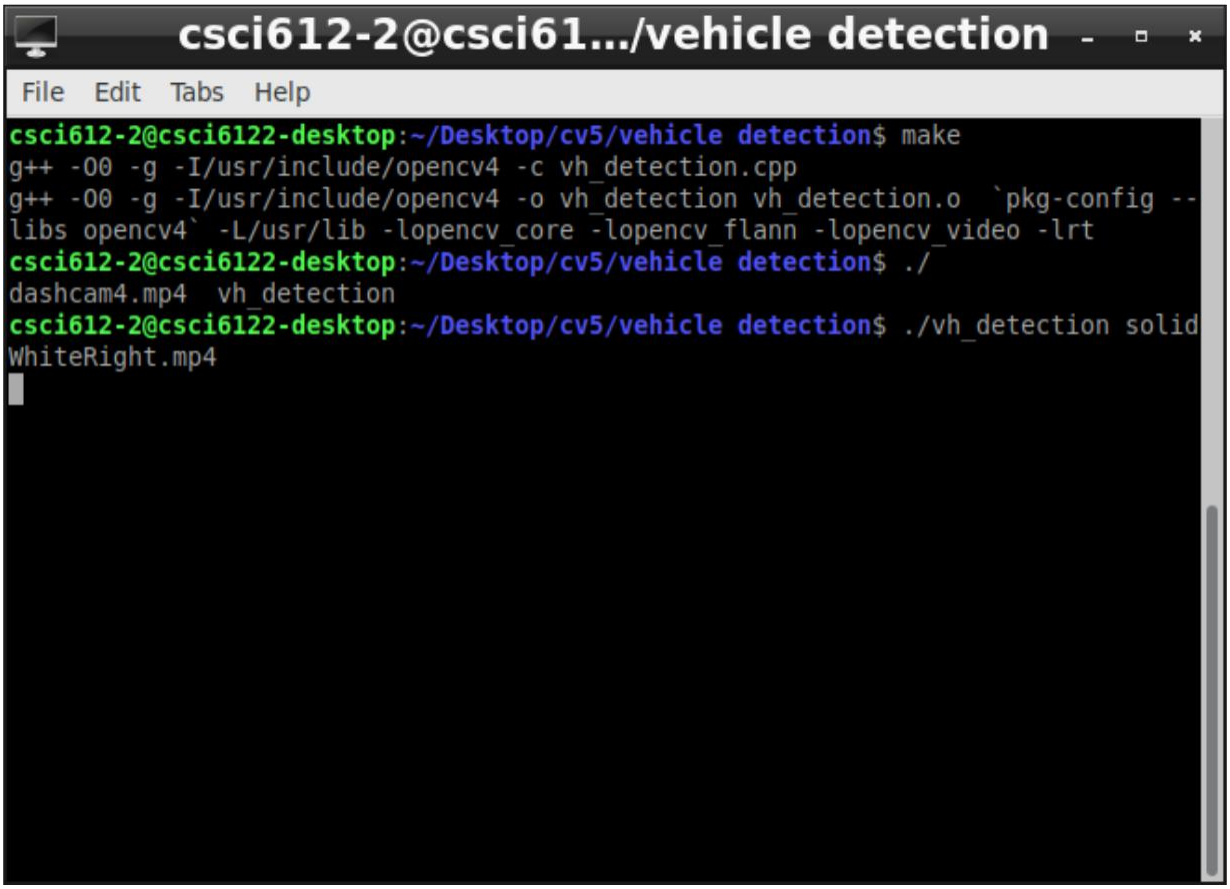
b) <u>Detection of Vehicles:</u>

- **Code Snippet:**

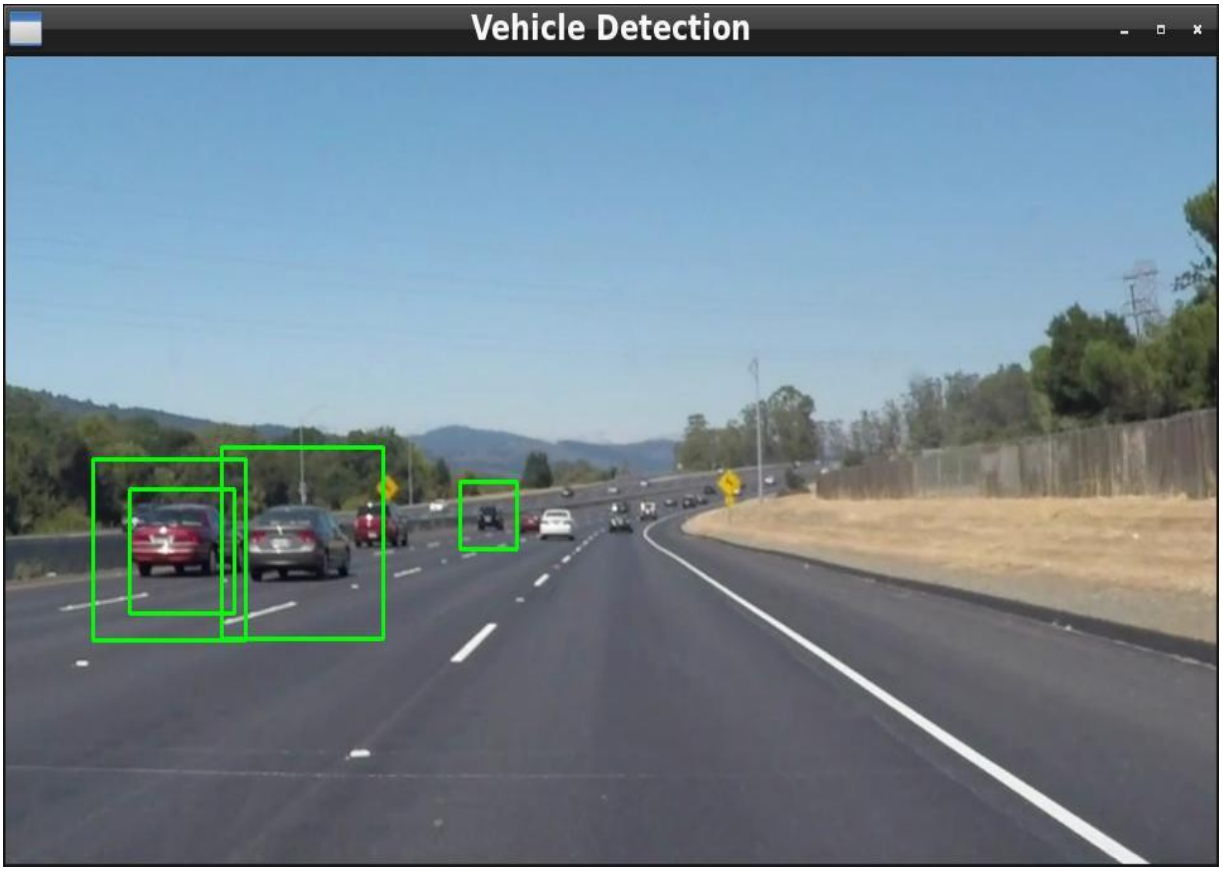
```
//////
int main(int argc, char** argv) {
    if (argc != 2) {
        cout << "Usage: ./vehicle_detection <video_path>" << endl;
        return -1;
    }

    // Load the video
    VideoCapture cap(argv[1]);
    if (!cap.isOpened()) {
        cout << "Error opening video file: " << argv[1] << endl;
        return -1;
    }

    // Load vehicle detection Haar cascade classifier
    CascadeClassifier vehicle_cascade;
    if (!vehicle_cascade.load("haarcascade_car.xml")) { // Make sure to have haarcascade_car.xml in the working directory
        cout << "Error loading vehicle cascade classifier. Make sure haarcascade_car.xml is in the working directory." << endl;
        return -1;
    }

    // Define output video writer
    VideoWriter outputVideo;
    int frame_width = cap.get(CAP_PROP_FRAME_WIDTH);
    int frame_height = cap.get(CAP_PROP_FRAME_HEIGHT);
    outputVideo.open("output.avi", VideoWriter::fourcc('M','J','P','G'), cap.get(CAP_PROP_FPS), Size(frame_width,frame_height));

    // Loop through each frame in the video
    while (true) {
        Mat frame;
        cap >> frame;

        // Check if the frame is empty (end of video)
        if (frame.empty())
            break;

        // Convert frame to grayscale
        Mat gray;
        cvtColor(frame, gray, COLOR_BGR2GRAY);

        // Detect vehicles in the frame
        vector<Rect> vehicles;
        vehicle_cascade.detectMultiScale(gray, vehicles, 1.1, 2, 0 | CASCADE_SCALE_IMAGE, Size(30, 30));

        // Draw bounding boxes around the detected vehicles
        for (const auto& vehicle : vehicles) {
            rectangle(frame, vehicle, Scalar(0, 255, 0), 2);
        }
```

- **Make and build of Vehicles Detection:**





- **Output:**

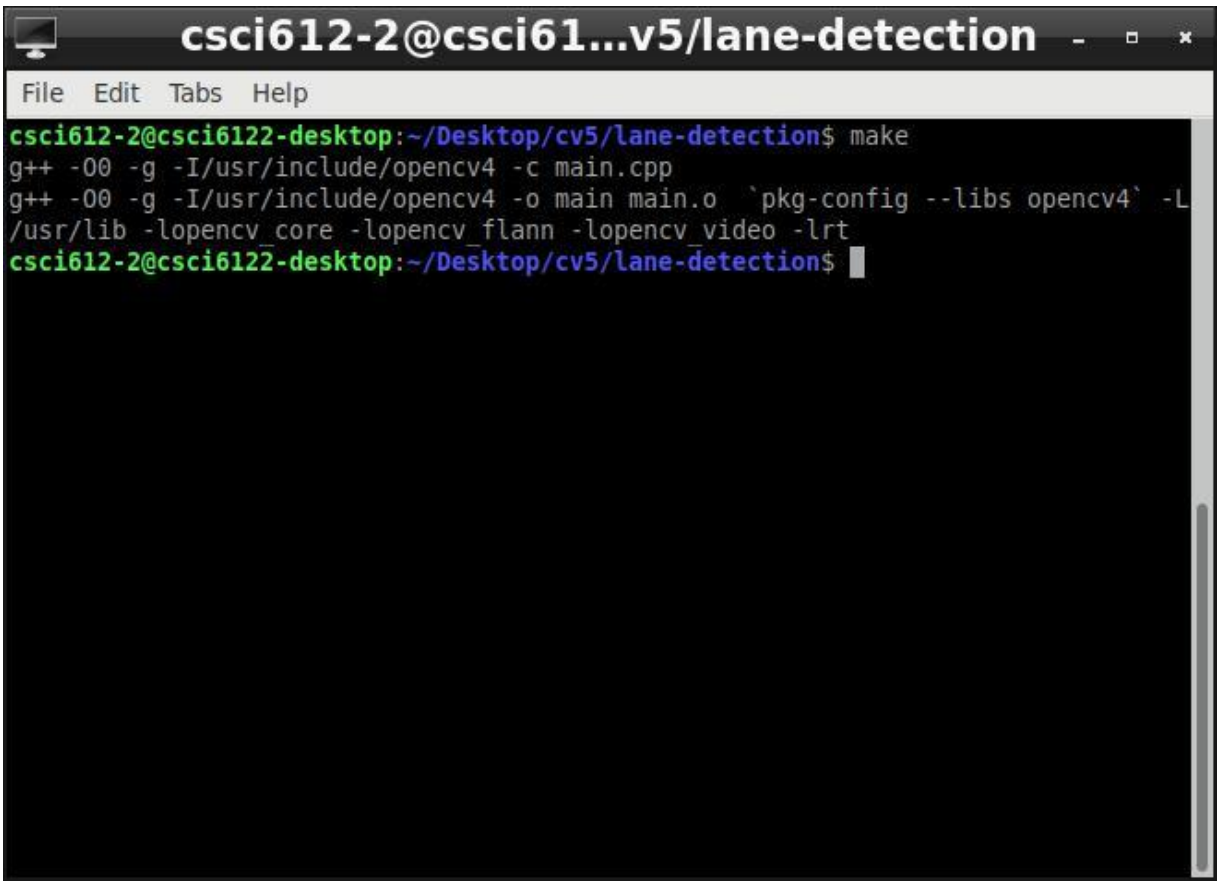


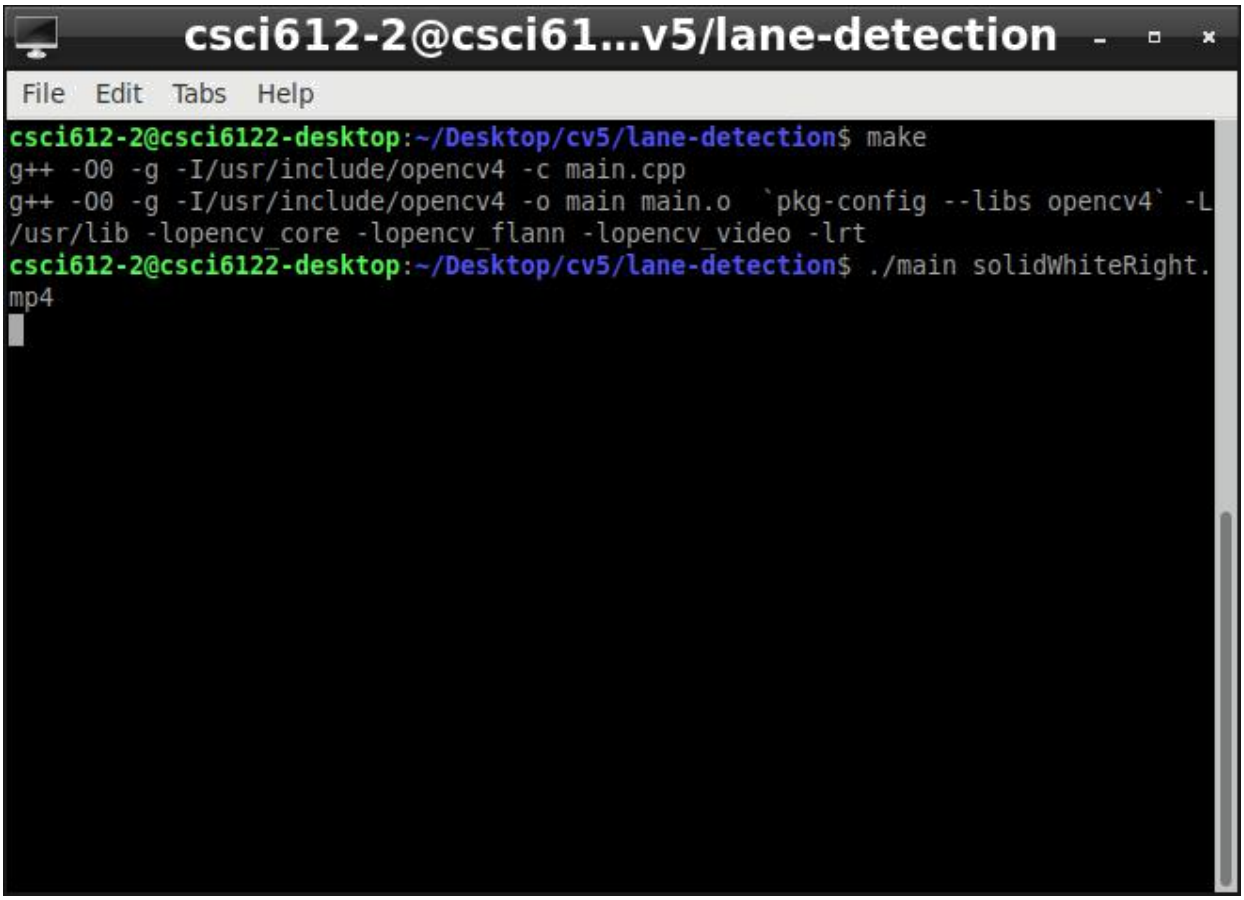
c) Lane Detection:

- **Code Snippet:**

```
#include <opencv2/highgui.hpp>
#include <opencv2/videoio.hpp>
///
cv::Mat loadImg(std::string img_path) {
    return cv::imread(img_path, cv::IMREAD_COLOR);
}

cv::Mat colorFilter(cv::Mat src) {
    cv::Mat hls, yellowmask, whitemask, mask, masked;
    cv::cvtColor(src, hls, cv::COLOR_RGB2HLS);
    cv::inRange(hls, cv::Scalar(10, 0, 90), cv::Scalar(50, 255, 255), yellowmask);
    cv::inRange(hls, cv::Scalar(0, 190, 0), cv::Scalar(255, 255, 255), whitemask);
    cv::bitwise_or(yellowmask, whitemask, mask);
    cv::bitwise_and(src, src, masked, mask = mask);
    return masked;
}

cv::Mat ROI(cv::Mat src) {
    int x = src.cols;
    int y = src.rows;
    cv::Point polygon_vertices[1][4];
    polygon_vertices[0][0] = cv::Point(0, y);
    polygon_vertices[0][1] = cv::Point(x, y);
    polygon_vertices[0][2] = cv::Point((int)std::round(0.55 * x), (int)std::round(0.6 * y));
    polygon_vertices[0][3] = cv::Point((int)std::round(0.45 * x), (int)std::round(0.6 * y));
    const cv::Point* polygons[1] = { polygon_vertices[0] };
    int n_vertices[] = { 4 };
    cv::Mat mask(y, x, CV_8UC1, cv::Scalar(0));
    int lineType = cv::LINE_8;
    cv::fillPoly(mask, polygons, n_vertices, 1, cv::Scalar(255, 255, 255), lineType);
    cv::Mat masked_image;
    cv::bitwise_and(src, src, masked_image, mask = mask);
    return masked_image;
}

cv::Mat grayscale(cv::Mat img) {
    cv::Mat gray_img;
    cv::cvtColor(img, gray_img, cv::COLOR_RGB2GRAY);
    return gray_img;
}

cv::Mat canny(cv::Mat img) {
    cv::Mat edges;
    cv::Canny(grayscale(img), edges, 50, 120);
    return edges;
}

float vectorAverage(std::vector<float> input_vec) {
    float average = std::accumulate(input_vec.begin(), input_vec.end(), 0.0) / input_vec.size();
    return average;
}
```

- **Make and build of lane Detection:**





- **Output:**

d) Stop SignsDetection:

- **Code Snippet:**

```
using namespace cv;
using namespace std;

class StopSignDetector {
private:
    CascadeClassifier cascade;

public:
    StopSignDetector() {
        cascade.load("stop_sign_classifier_2.xml");
    }

    vector<Rect> detectStopSigns(const Mat& img) {
        vector<Rect> found;
        cascade.detectMultiScale(img, found);

        // Filter out regions based on position and aspect ratio
        vector<Rect> filteredRegions;
        for (const auto& region : found) {
            double aspectRatio = static_cast<double>(region.width) / region.height;
            // Check if the aspect ratio is close to 1
            if (aspectRatio > 0.7 && aspectRatio < 1.3) {
                // Check if the region is not near the top of the frame
                if (region.y > img.rows / 3) {
                    filteredRegions.push_back(region);
                }
            }
        }

        return filteredRegions;
    }
};

int main()
{
    String videoFile = "stop_sign_video2.mp4";
    String outputFile = "output_video.mp4"; // Change the output file extension to .mp4

    VideoCapture cap(videoFile);
    if (!cap.isOpened())
    {
        cout << "Can not open video file: " << videoFile << endl;
        return -1;
    }

    // Get video properties
    int frame_width = cap.get(CAP_PROP_FRAME_WIDTH);
    int frame_height = cap.get(CAP_PROP_FRAME_HEIGHT);
    int fps = cap.get(CAP_PROP_FPS);
```

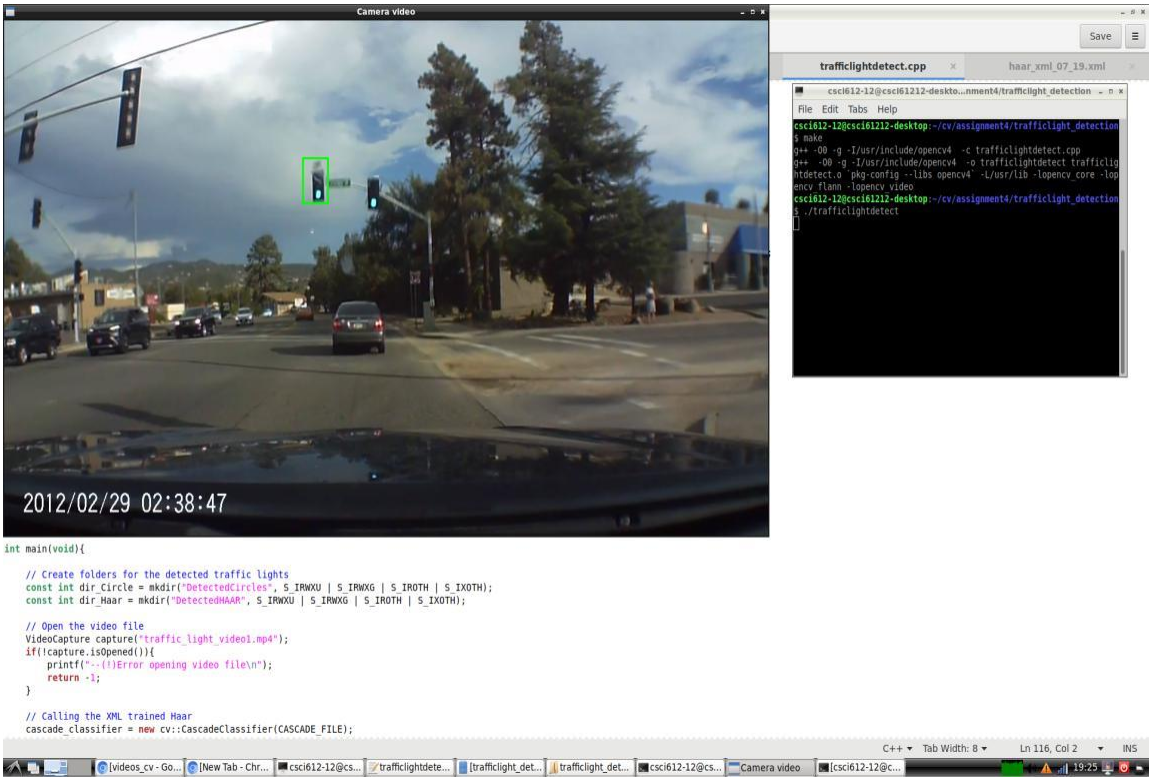
- **Make and build of stop signs Detection:**

e) Trafc Light Detection:

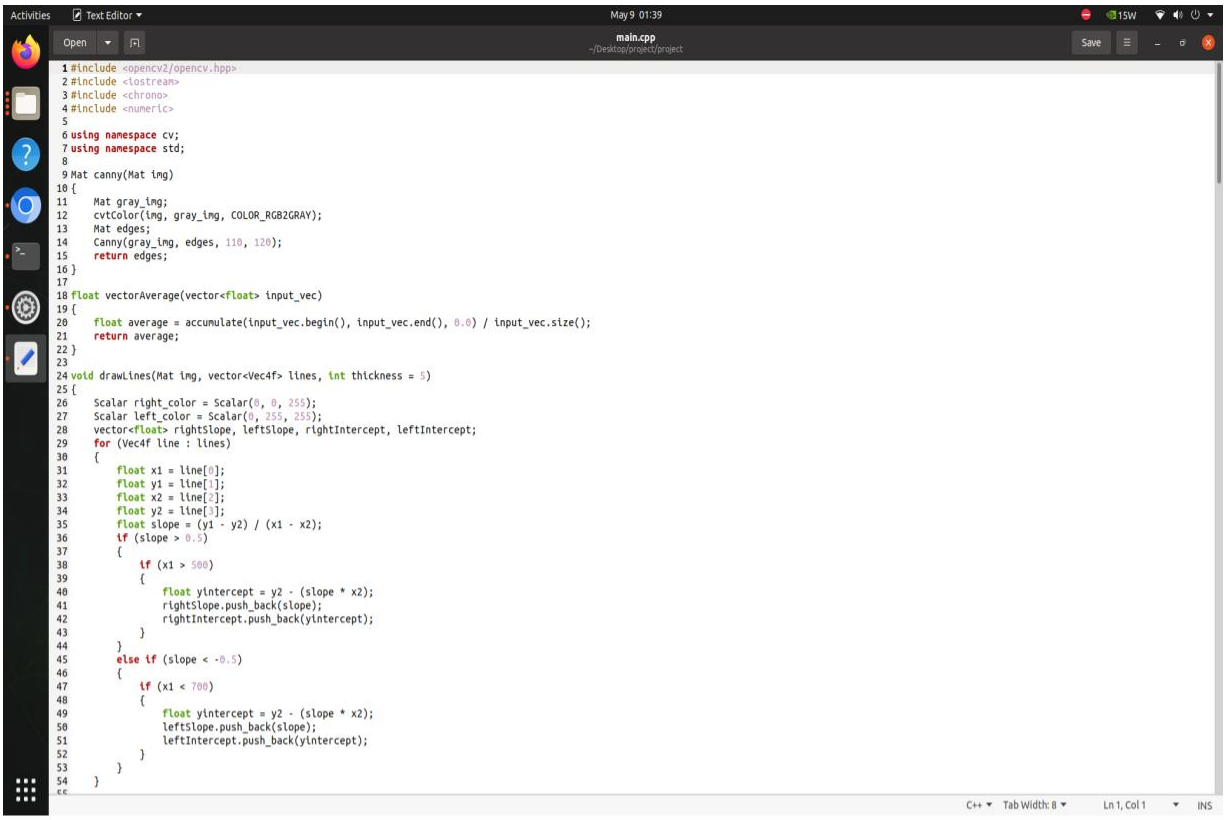
- **Code Snippet:**

```cpp
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>

#include <iostream>
#include <stdio.h>
#include <string>
#include <sys/stat.h>
#include <cstdlib>

using namespace std;
using namespace cv;

/** Function Headers */
void detectAndDisplay(Mat& frame, int nr, VideoWriter& out);
/** Global variables */
const cv::String WINDOW_NAME("Camera video");
/** Place the XML in the home directory */
const cv::String CASCADE_FILE("haar_xml_07_19.xml");
cv::CascadeClassifier *cascade_classifier;
cv::Mat frame, traffic_template;
cv::Mat trLightROI;

void showFrame(cv::Mat &frame)
{
    cv::imshow(WINDOW_NAME, frame);
    int c = waitKey(50);
    if ((char)c == 30) { exit(0);}
}

/** Circle Detection after detecting a traffic light */
void DetectCircles(cv::Mat &traffic_template, int nr, VideoWriter& out){
    std::vector<cv::Vec3f> VectorCir;
    std::vector<cv::Vec3f>::iterator iterCircles;
    Mat resultImg;

    // Save mat of detected circle in traffic lights
    string circle_name = "circle";
    string type_circle = ".jpg";
    string folderNameCircle = "DetectedCircles";
    stringstream ssfn_circle;
    ssfn_circle << folderNameCircle << "/" << circle_name << (nr) << type_circle;

    string fullpath_circle = ssfn_circle.str();
    ssfn_circle.str("");

    // Apply color map to search for certain color
    applyColorMap(traffic_template, traffic_template, COLORMAP_SUMMER);
    cv::inRange(traffic_template, cv::Scalar(0, 90, 90), cv::Scalar(204, 255, 255), resultImg);
    cv::GaussianBlur(resultImg, resultImg, cv::Size(9, 9), 0.5, 0.5);
    cv::HoughCircles(resultImg,
                     VectorCir,
```

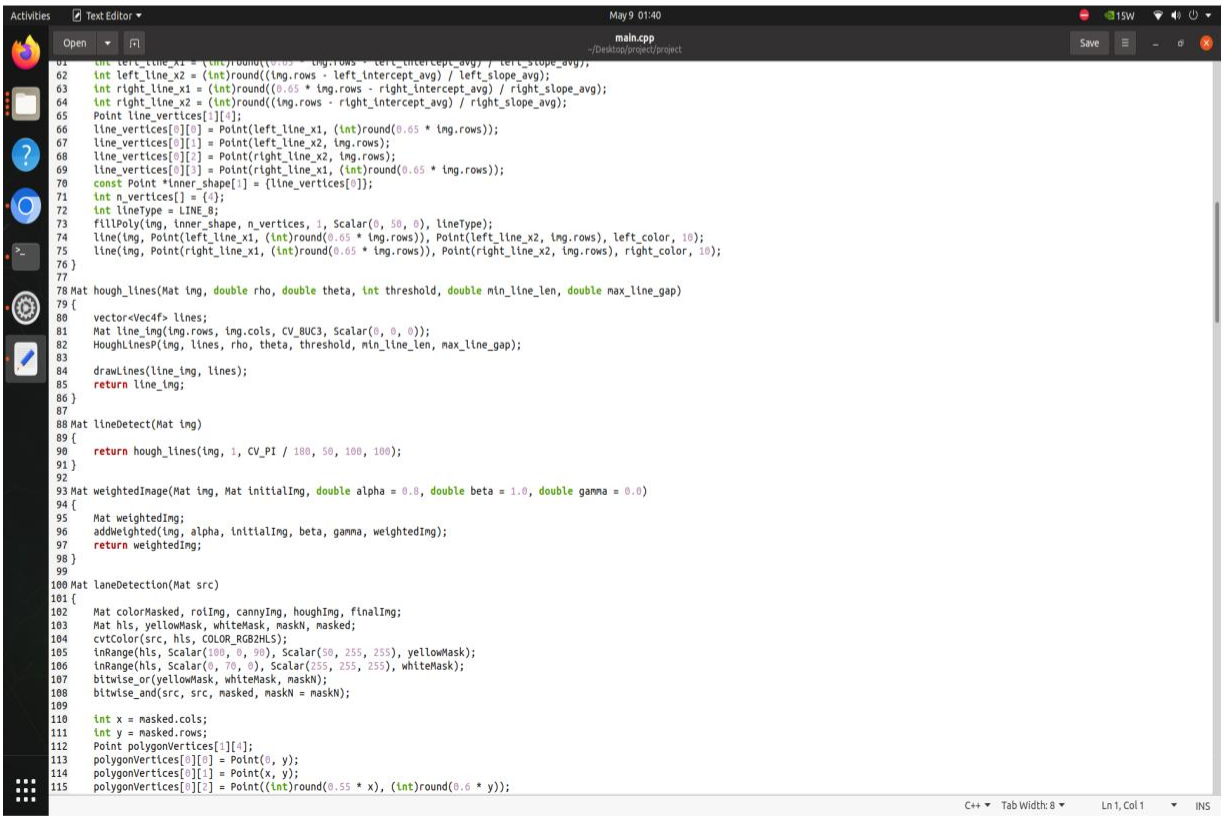
- **Make,build and output of Trafc Light Detection:**



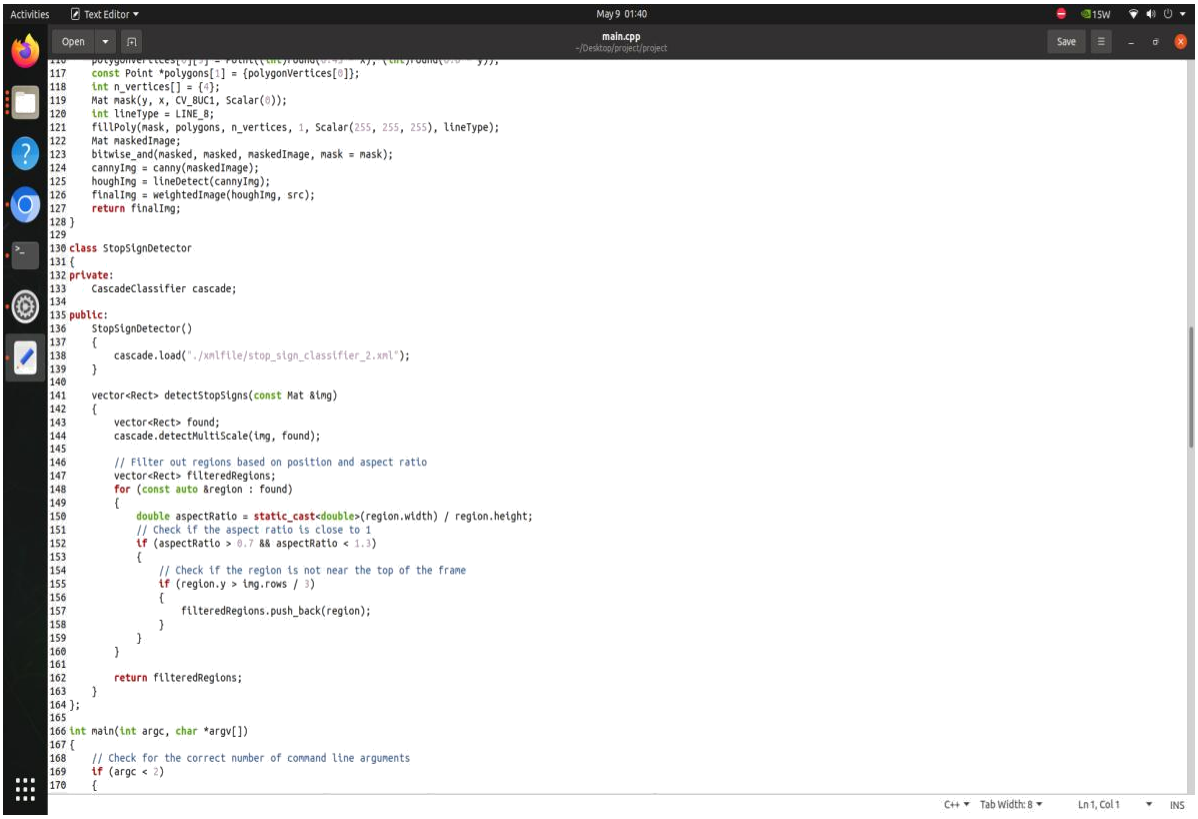
- **Integration of Features:**
  - **Code Snippet:**

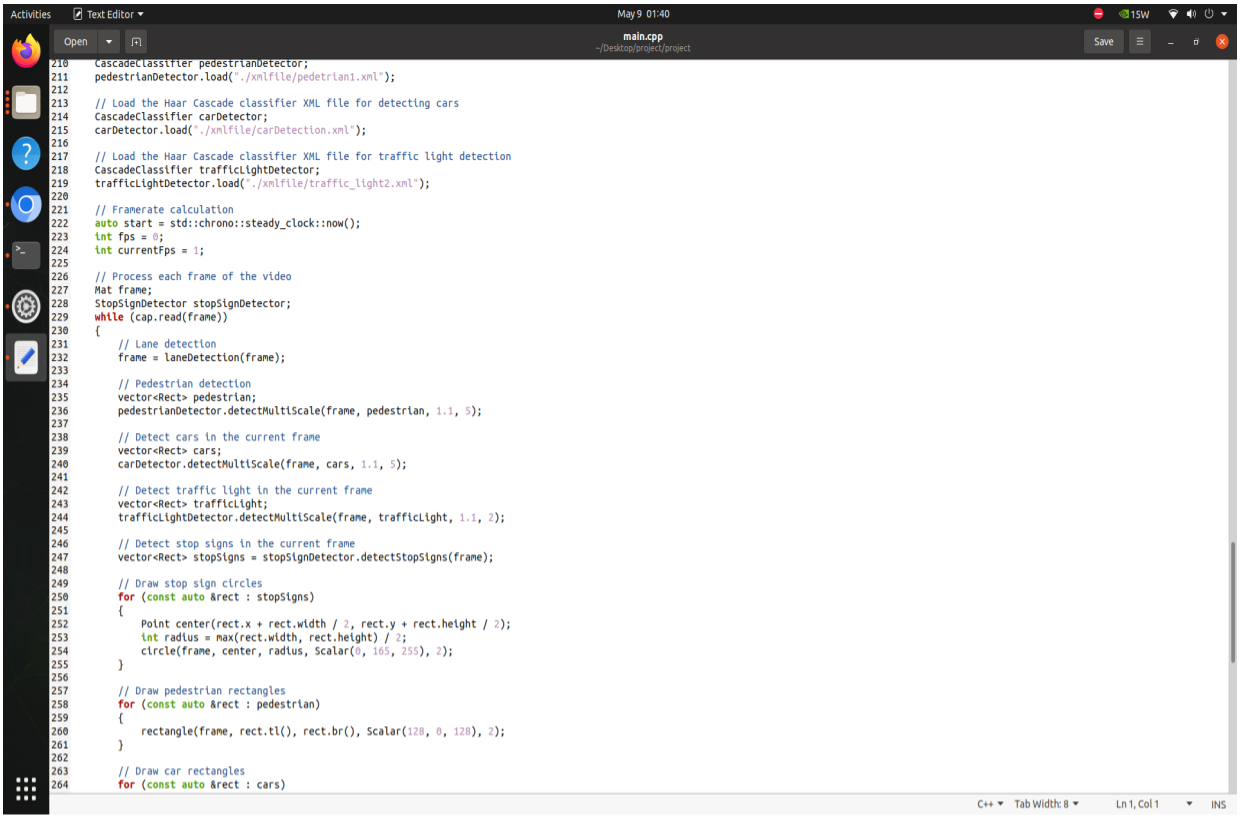


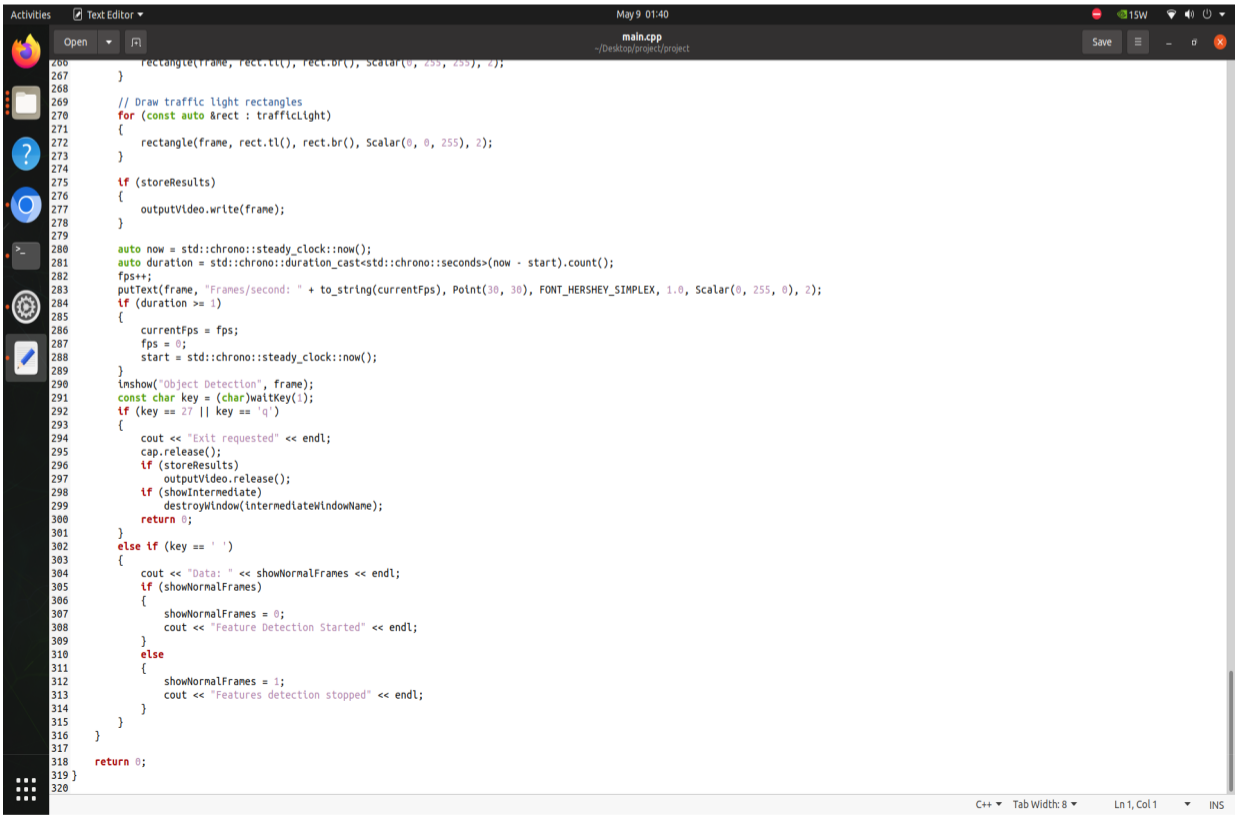

```cpp
#include <opencv2/opencv.hpp>
#include <iostream>
#include <chrono>
#include <numeric>

using namespace cv;
using namespace std;

Mat canny(Mat img)
{
    Mat gray_img;
    cvtColor(img, gray_img, COLOR_RGB2GRAY);
    Mat edges;
    Canny(gray_img, edges, 110, 120);
    return edges;
}

float vectorAverage(vector<float> input_vec)
{
    float average = accumulate(input_vec.begin(), input_vec.end(), 0.0) / input_vec.size();
    return average;
}

void drawLines(Mat img, vector<Vec4f> lines, int thickness = 5)
{
    Scalar right_color = Scalar(0, 0, 255);
    Scalar left_color = Scalar(0, 255, 255);
    vector<float> rightSlope, leftSlope, rightIntercept, leftIntercept;
    for (Vec4f line : lines)
    {
        float x1 = line[0];
        float y1 = line[1];
        float x2 = line[2];
        float y2 = line[3];
        float slope = (y1 - y2) / (x1 - x2);
        if (slope > 0.5)
        {
            if (x1 > 500)
            {
                float yintercept = y2 - (slope * x2);
                rightSlope.push_back(slope);
                rightIntercept.push_back(yintercept);
            }
        }
        else if (slope < -0.5)
        {
            if (x1 < 700)
            {
                float yintercept = y2 - (slope * x2);
                leftSlope.push_back(slope);
                leftIntercept.push_back(yintercept);
            }
        }
    }
```

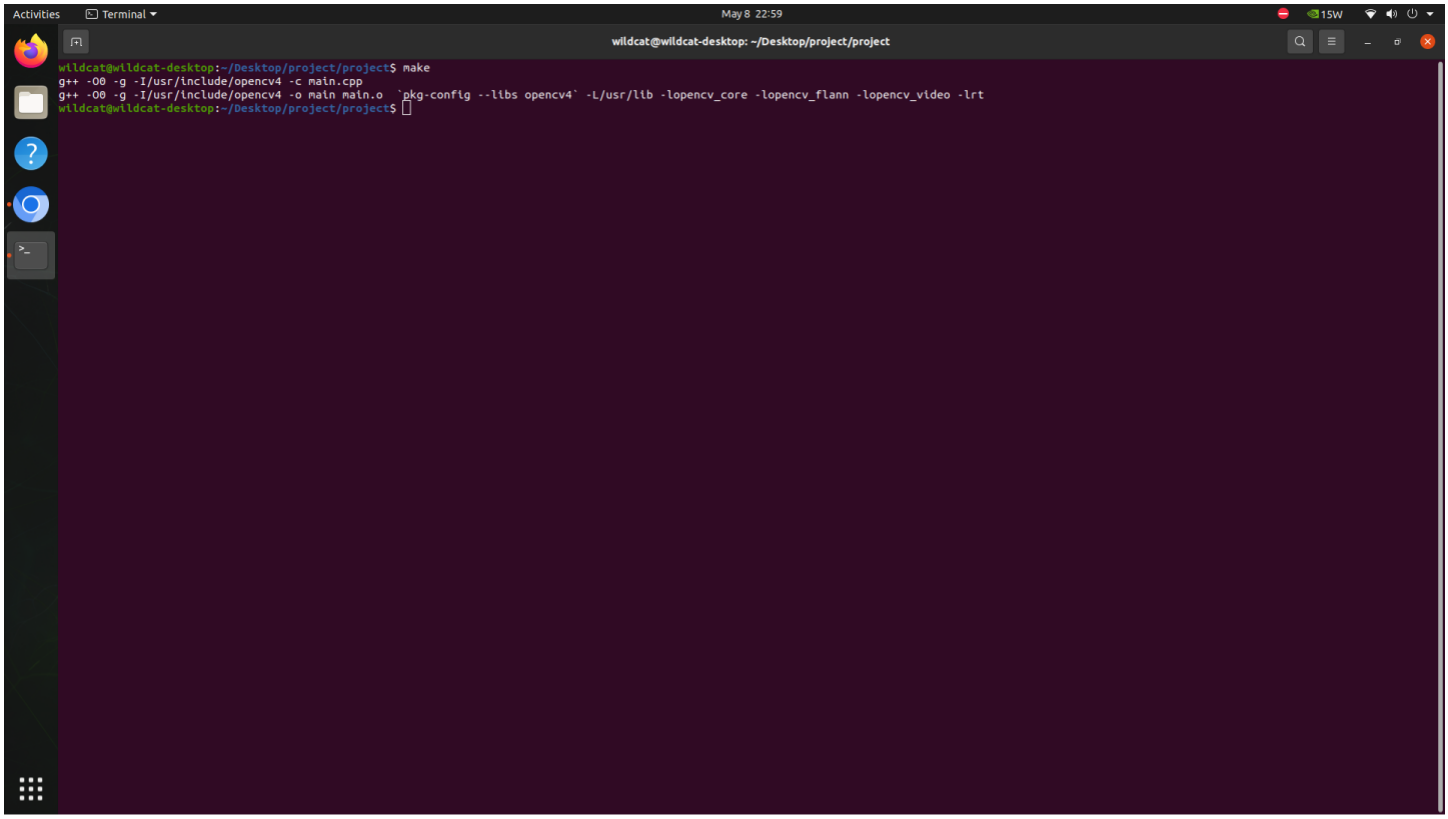

```cpp
    int left_line_x2 = (int)round((img.rows - left_intercept_avg) / left_slope_avg);
    int right_line_x1 = (int)round((0.65 * img.rows - right_intercept_avg) / right_slope_avg);
    int right_line_x2 = (int)round((img.rows - right_intercept_avg) / right_slope_avg);
    Point line_vertices[1][4];
    line_vertices[0][0] = Point(left_line_x1, (int)round(0.65 * img.rows));
    line_vertices[0][1] = Point(left_line_x2, img.rows);
    line_vertices[0][2] = Point(right_line_x2, img.rows);
    line_vertices[0][3] = Point(right_line_x1, (int)round(0.65 * img.rows));
    const Point *inner_shape[1] = {line_vertices[0]};
    int n_vertices[] = {4};
    int lineType = LINE_8;
    fillPoly(img, inner_shape, n_vertices, 1, Scalar(0, 50, 0), lineType);
    line(img, Point(left_line_x1, (int)round(0.65 * img.rows)), Point(left_line_x2, img.rows), left_color, 10);
    line(img, Point(right_line_x1, (int)round(0.65 * img.rows)), Point(right_line_x2, img.rows), right_color, 10);
}

Mat hough_lines(Mat img, double rho, double theta, int threshold, double min_line_len, double max_line_gap)
{
    vector<Vec4f> lines;
    Mat line_img(img.rows, img.cols, CV_8UC3, Scalar(0, 0, 0));
    HoughLinesP(img, lines, rho, theta, threshold, min_line_len, max_line_gap);

    drawLines(line_img, lines);
    return line_img;
}

Mat lineDetect(Mat img)
{
    return hough_lines(img, 1, CV_PI / 180, 50, 100, 100);
}

Mat weightedImage(Mat img, Mat initialImg, double alpha = 0.8, double beta = 1.0, double gamma = 0.0)
{
    Mat weightedImg;
    addWeighted(img, alpha, initialImg, beta, gamma, weightedImg);
    return weightedImg;
}

Mat laneDetection(Mat src)
{
    Mat colorMasked, roiImg, cannyImg, houghImg, finalImg;
    Mat hls, yellowMask, whiteMask, maskN, masked;
    cvtColor(src, hls, COLOR_RGB2HLS);
    inRange(hls, Scalar(100, 0, 90), Scalar(50, 255, 255), yellowMask);
    inRange(hls, Scalar(0, 70, 0), Scalar(255, 255, 255), whiteMask);
    bitwise_or(yellowMask, whiteMask, maskN);
    bitwise_and(src, src, masked, maskN = maskN);

    int x = masked.cols;
    int y = masked.rows;
    Point polygonVertices[1][4];
    polygonVertices[0][0] = Point(0, y);
    polygonVertices[0][1] = Point(x, y);
    polygonVertices[0][2] = Point((int)round(0.55 * x), (int)round(0.6 * y));
```

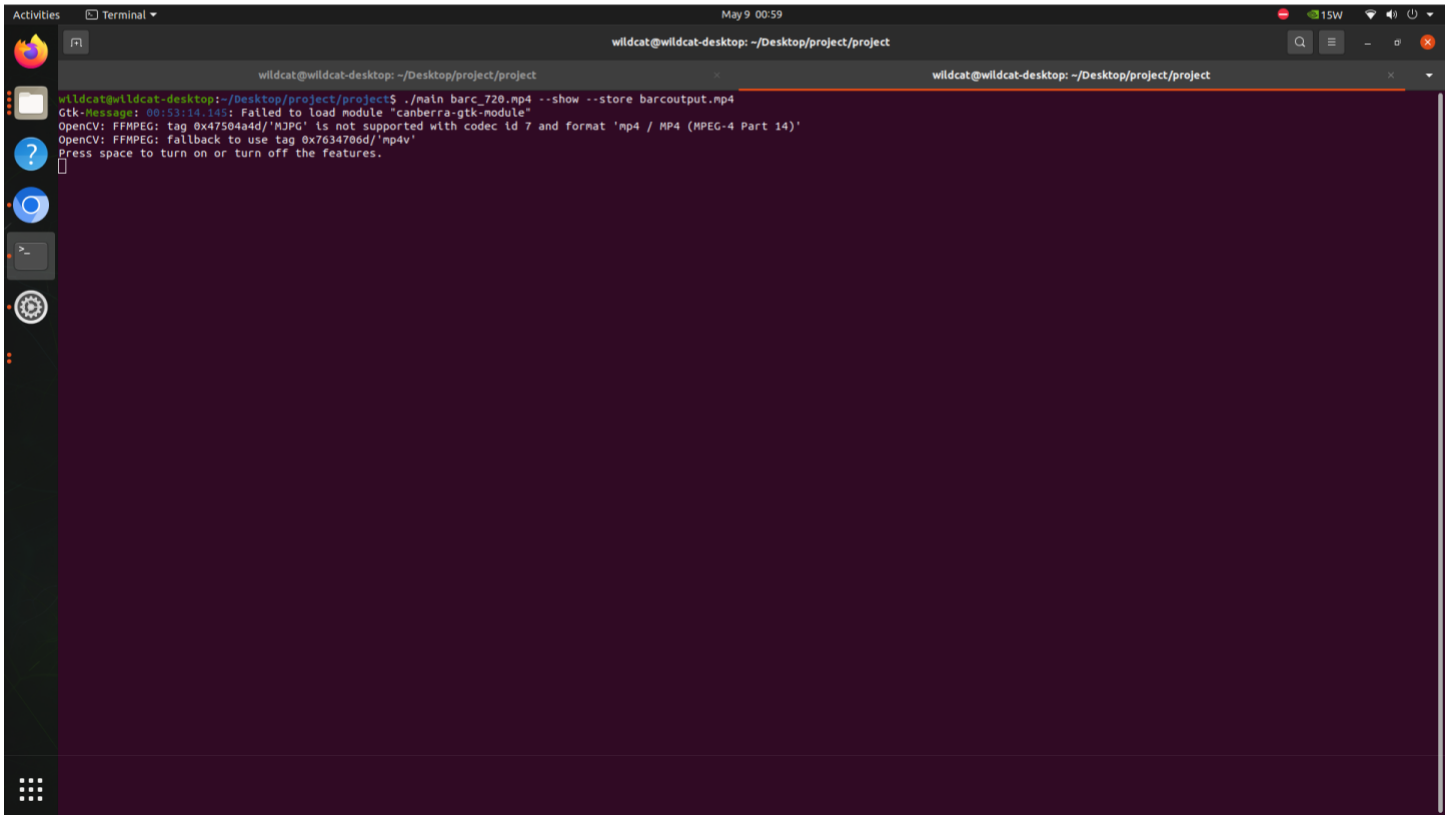

Activities
Text Editor
May 9 01:40
Open
main.cpp
~/Desktop/project/project
Save
117 const Point *polygons[1] = {polygonVertices[0]};
118 int n_vertices[] = {4};
119 Mat mask(y, x, CV_8UC1, Scalar(0));
120 int lineType = LINE_8;
121 fillPoly(mask, polygons, n_vertices, 1, Scalar(255, 255, 255), lineType);
122 Mat maskedImage;
123 bitwise_and(masked, masked, maskedImage, mask = mask);
124 cannyImg = canny(maskedImage);
125 houghImg = lineDetect(cannyImg);
126 finalImg = weightedImage(houghImg, src);
127 return finalImg;
128 }
129
130 class StopSignDetector
131 {
132 private:
133 CascadeClassifier cascade;
134
135 public:
136 StopSignDetector()
137 {
138 cascade.load("./xmlfile/stop_sign_classifier_2.xml");
139 }
140
141 vector<Rect> detectStopSigns(const Mat &img)
142 {
143 vector<Rect> found;
144 cascade.detectMultiScale(img, found);
145
146 // Filter out regions based on position and aspect ratio
147 vector<Rect> filteredRegions;
148 for (const auto &region : found)
149 {
150 double aspectRatio = static_cast<double>(region.width) / region.height;
151 // Check if the aspect ratio is close to 1
152 if (aspectRatio > 0.7 && aspectRatio < 1.3)
153 {
154 // Check if the region is not near the top of the frame
155 if (region.y > img.rows / 3)
156 {
157 filteredRegions.push_back(region);
158 }
159 }
160 }
161
162 return filteredRegions;
163 }
164 };
165
166 int main(int argc, char *argv[])
167 {
168 // Check for the correct number of command line arguments
169 if (argc < 2)
170 {
C++
Tab Width: 8
Ln 1, Col 1
INS


Activities
Text Editor
May 9 01:40
Open
main.cpp
~/Desktop/project/project
Save
CascadeClassifier pedestrianDetector;
pedestrianDetector.load("./xmlfile/pedetrian1.xml");
// Load the Haar Cascade classifier XML file for detecting cars
CascadeClassifier carDetector;
carDetector.load("./xmlfile/carDetection.xml");
// Load the Haar Cascade classifier XML file for traffic light detection
CascadeClassifier trafficLightDetector;
trafficLightDetector.load("./xmlfile/traffic_light2.xml");
// Framerate calculation
auto start = std::chrono::steady_clock::now();
int fps = 0;
int currentFps = 1;
// Process each frame of the video
Mat frame;
StopSignDetector stopSignDetector;
while (cap.read(frame))
{
// Lane detection
frame = laneDetection(frame);
// Pedestrian detection
vector<Rect> pedestrian;
pedestrianDetector.detectMultiScale(frame, pedestrian, 1.1, 5);
// Detect cars in the current frame
vector<Rect> cars;
carDetector.detectMultiScale(frame, cars, 1.1, 5);
// Detect traffic light in the current frame
vector<Rect> trafficLight;
trafficLightDetector.detectMultiScale(frame, trafficLight, 1.1, 2);
// Detect stop signs in the current frame
vector<Rect> stopSigns = stopSignDetector.detectStopSigns(frame);
// Draw stop sign circles
for (const auto &rect : stopSigns)
{
Point center(rect.x + rect.width / 2, rect.y + rect.height / 2);
int radius = max(rect.width, rect.height) / 2;
circle(frame, center, radius, Scalar(0, 165, 255), 2);
}
// Draw pedestrian rectangles
for (const auto &rect : pedestrian)
{
rectangle(frame, rect.tl(), rect.br(), Scalar(128, 0, 128), 2);
}
// Draw car rectangles
for (const auto &rect : cars)
C++
Tab Width: 8
Ln 1, Col 1
INS

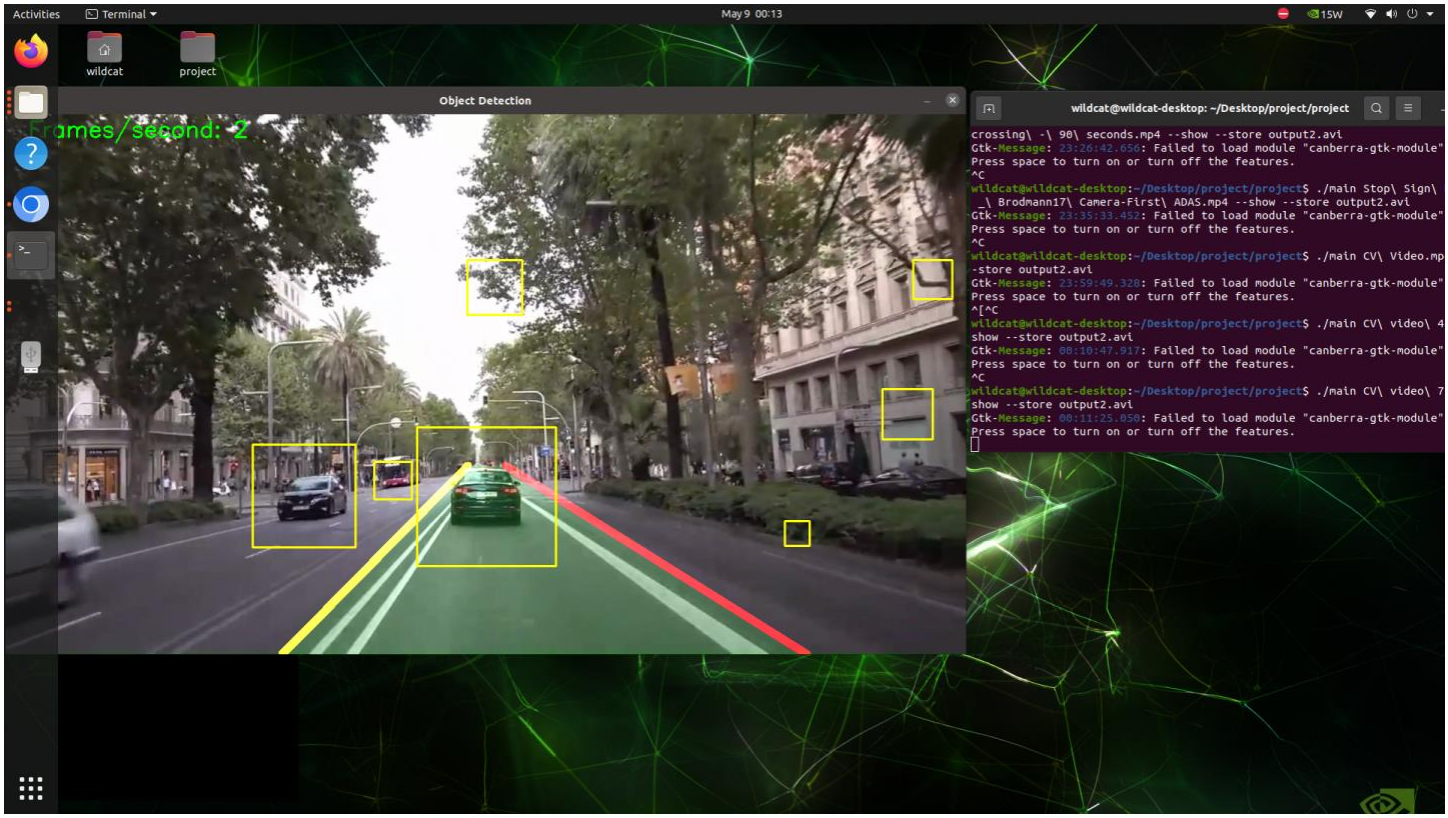

Activities
Text Editor
May 9 01:40
Open
main.cpp
~/Desktop/project/project
Save
}
// Draw traffic light rectangles
for (const auto &rect : trafficLight)
{
rectangle(frame, rect.tl(), rect.br(), Scalar(0, 0, 255), 2);
}
if (storeResults)
{
outputVideo.write(frame);
}
auto now = std::chrono::steady_clock::now();
auto duration = std::chrono::duration_cast<std::chrono::seconds>(now - start).count();
fps++;
putText(frame, "Frames/second: " + to_string(currentFps), Point(30, 30), FONT_HERSHEY_SIMPLEX, 1.0, Scalar(0, 255, 0), 2);
if (duration >= 1)
{
currentFps = fps;
fps = 0;
start = std::chrono::steady_clock::now();
}
imshow("Object Detection", frame);
const char key = (char)waitKey(1);
if (key == 27 || key == 'q')
{
cout << "Exit requested" << endl;
cap.release();
if (storeResults)
outputVideo.release();
if (showIntermediate)
destroyWindow(intermediateWindowName);
return 0;
}
else if (key == ' ')
{
cout << "Data: " << showNormalFrames << endl;
if (showNormalFrames)
{
showNormalFrames = 0;
cout << "Feature Detection Started" << endl;
}
else
{
showNormalFrames = 1;
cout << "Features detection stopped" << endl;
}
}
}
return 0;
}
C++
Tab Width: 8
Ln 1, Col 1
INS

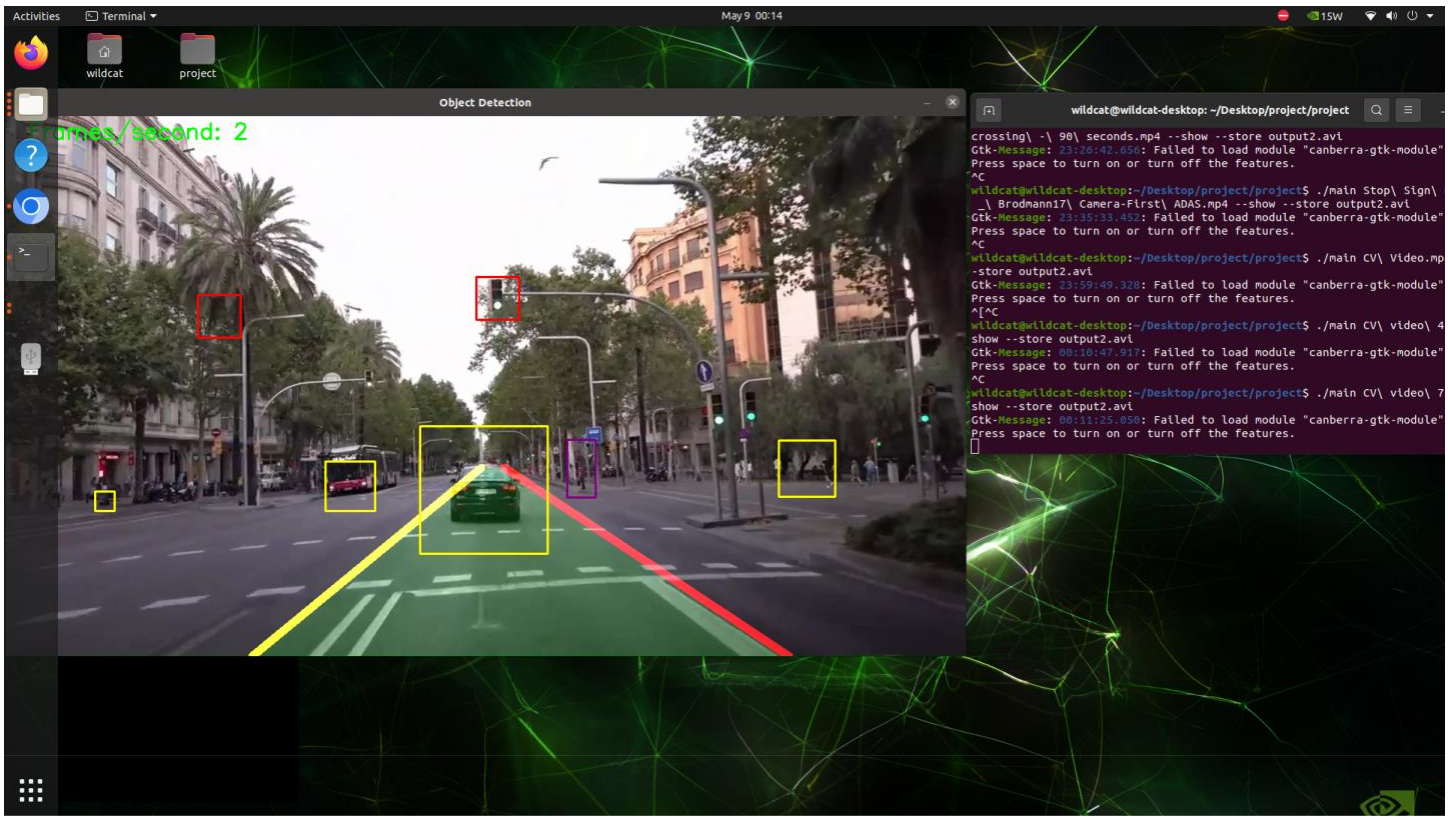
- **Make and build of Integrated features:**

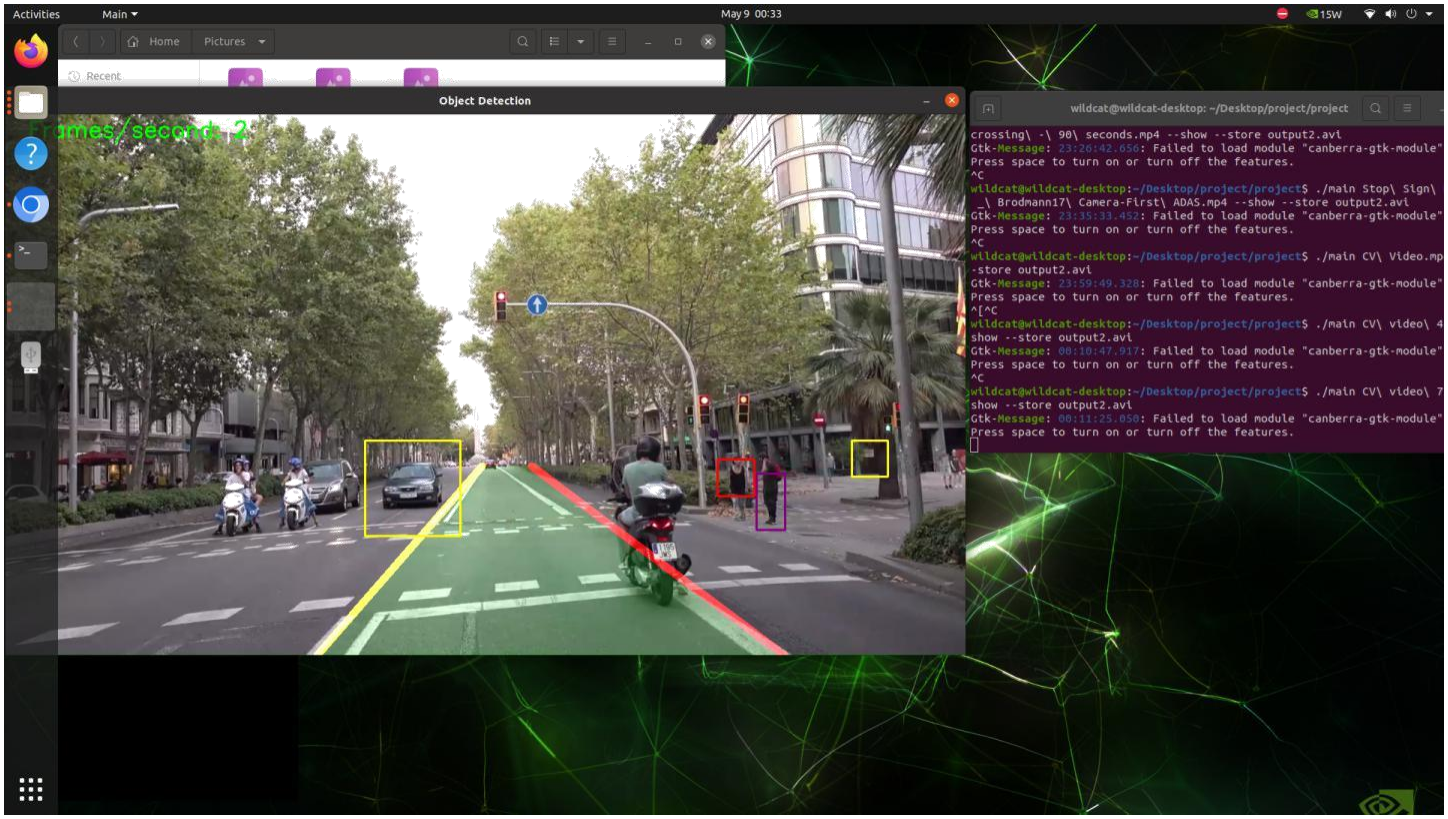


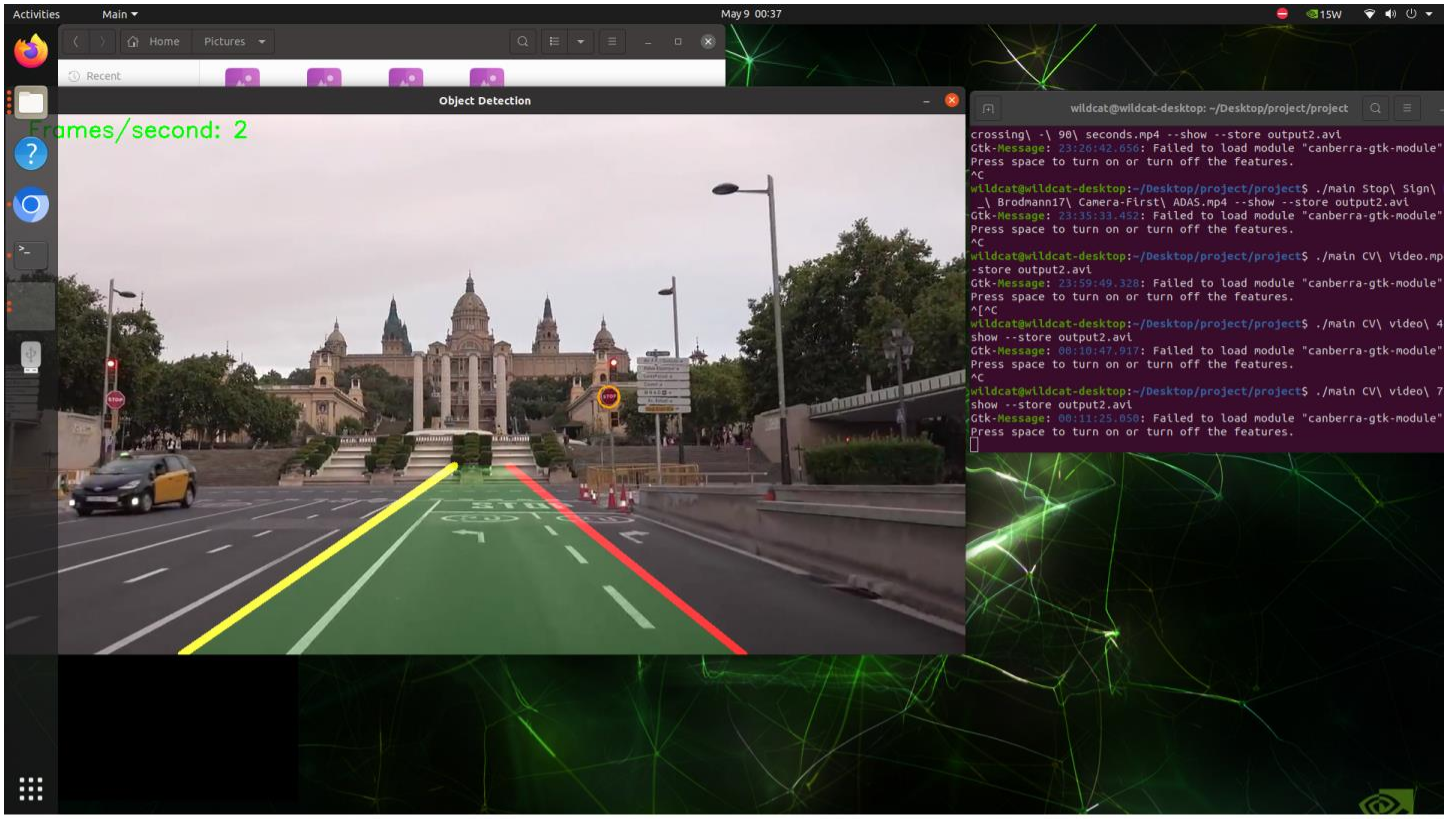


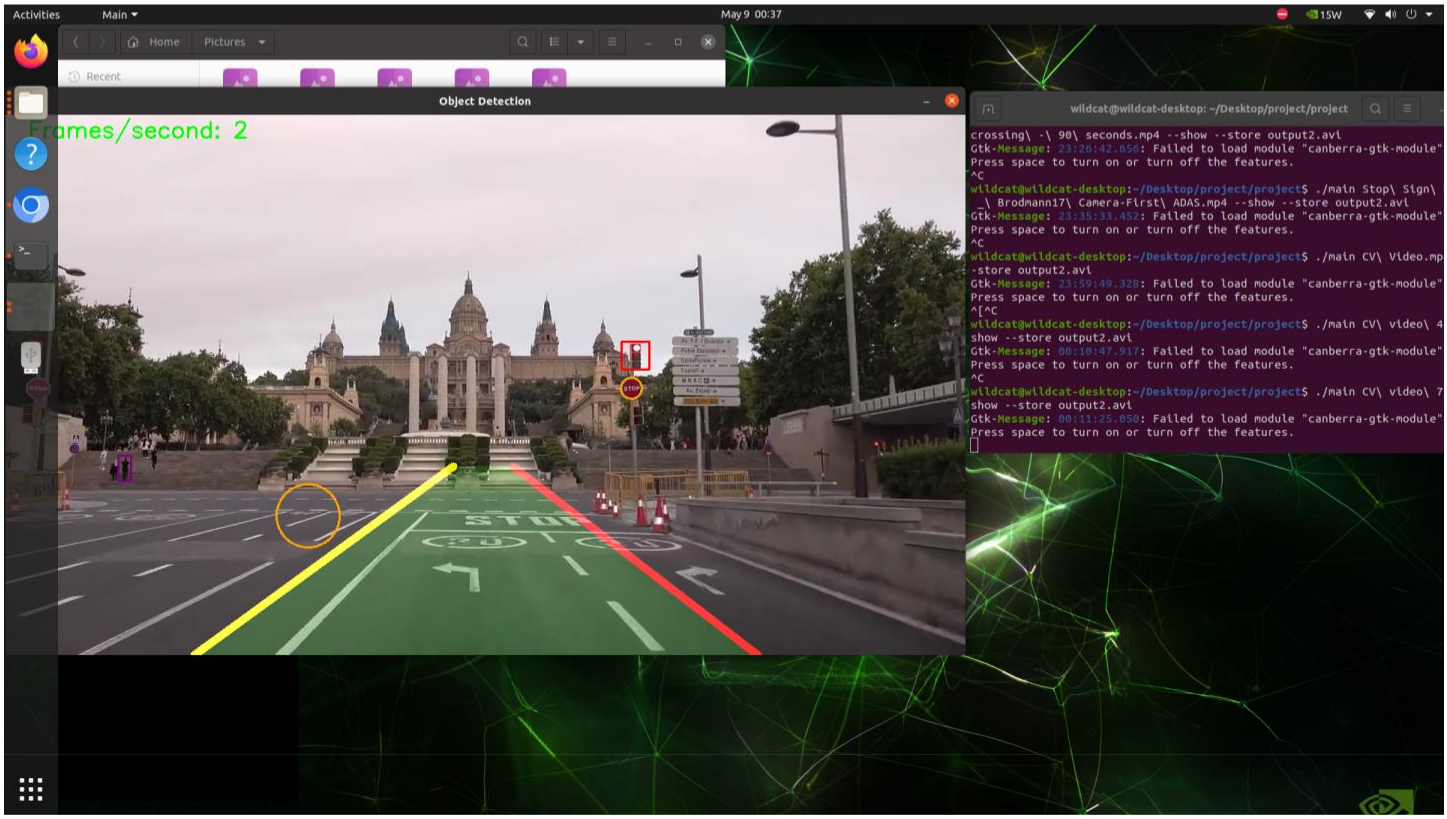
- **Output:**


Activities
Terminal
May 9 00:13
wildcat
project
Object Detection
Frames/second: 2
wildcat@wildcat-desktop: ~/Desktop/project/project
crossing\ -\ 90\ seconds.mp4 --show --store output2.avi
Gtk-Message: 23:26:42.656: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.
^C
wildcat@wildcat-desktop:~/Desktop/project/project$ ./main Stop\ Sign\
_\ Brodmann17\ Camera-First\ ADAS.mp4 --show --store output2.avi
Gtk-Message: 23:35:33.452: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.
^C
wildcat@wildcat-desktop:~/Desktop/project/project$ ./main CV\ Video.mp
-store output2.avi
Gtk-Message: 23:59:49.328: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.
^[^C
wildcat@wildcat-desktop:~/Desktop/project/project$ ./main CV\ video\ 4
show --store output2.avi
Gtk-Message: 00:10:47.917: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.
^C
wildcat@wildcat-desktop:~/Desktop/project/project$ ./main CV\ video\ 7
show --store output2.avi
Gtk-Message: 00:11:25.050: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.


Activities
Terminal
May 9 00:14
wildcat
project
Object Detection
Frames/second: 2
wildcat@wildcat-desktop: ~/Desktop/project/project
crossing\ -\ 90\ seconds.mp4 --show --store output2.avi
Gtk-Message: 23:26:42.656: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.
^C
wildcat@wildcat-desktop:~/Desktop/project/project$ ./main Stop\ Sign\
_\ Brodmann17\ Camera-First\ ADAS.mp4 --show --store output2.avi
Gtk-Message: 23:35:33.452: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.
^C
wildcat@wildcat-desktop:~/Desktop/project/project$ ./main CV\ Video.mp
-store output2.avi
Gtk-Message: 23:59:49.328: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.
^[^C
wildcat@wildcat-desktop:~/Desktop/project/project$ ./main CV\ video\ 4
show --store output2.avi
Gtk-Message: 00:10:47.917: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.
^C
wildcat@wildcat-desktop:~/Desktop/project/project$ ./main CV\ video\ 7
show --store output2.avi
Gtk-Message: 00:11:25.050: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.


Activities
Main
May 9 00:33
15W
Home
Pictures
Recent
Object Detection
mes/second: 2
wildcat@wildcat-desktop: ~/Desktop/project/project
crossing\ -\ 90\ seconds.mp4 --show --store output2.avi
Gtk-Message: 23:26:42.656: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.
^C
wildcat@wildcat-desktop:~/Desktop/project/project$ ./main Stop\ Sign\
_\ Brodmann17\ Camera-Firsr\ ADAS.mp4 --show --store output2.avi
Gtk-Message: 23:35:33.452: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.
^C
wildcat@wildcat-desktop:~/Desktop/project/project$ ./main CV\ Video.mp
-store output2.avi
Gtk-Message: 23:59:49.328: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.
^[^C
wildcat@wildcat-desktop:~/Desktop/project/project$ ./main CV\ video\ 4
show --store output2.avi
Gtk-Message: 00:10:47.917: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.
^C
wildcat@wildcat-desktop:~/Desktop/project/project$ ./main CV\ video\ 7
show --store output2.avi
Gtk-Message: 00:11:25.050: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.





Output Video: Video Link is provided below in the proof of concept.

2. Address Development Challenges Faced:

- **Performance Optimization:**
  - **Computational Overhead:** Integrating multiple detection modules can signifcantly increase computational complexity, potentially leading to slower processing speed, especially on resource-constrained devices.
  - **Real-Time Processing:** Ensuring real-time performance becomes challenging, especially when dealing with high-resolution video streams or processing multiple frames concurrently.
  - **Algorithm Efciency:** Each detection module may have its own set of algorithms and parameters, requiring careful optimization to minimize processing time while maintaining accuracy.

- **Accurate Detection:**
  - **False Positives/Negatives:** Combining multiple detection modules may lead to increased false positives or negatives, where objects are incorrectly identifed or missed altogether.
  - **Feature Interference:** The outputs of diferent detection modules may interfere with each other, causing inaccuracies in detection. For example, the detection of lane markings might interfere with the detection of pedestrian or vehicle objects.

- **Integration Complexity:**
  - **Data Compatibility:** Ensuring compatibility between diferent data formats and representations used by each detection module can be challenging. For instance, integrating outputs from modules operating on diferent color spaces or image resolutions.
  - **Synchronization:** Coordinating the execution of multiple detection modules and integrating their results cohesively requires careful synchronization to avoid confcts and ensure smooth operation.
  - **Parameter Tuning:** Each detection module may have its own set of parameters that need to be fne-tuned for optimal performance. Managing these parameters collectively can be complex and time-consuming.

- **Resource Management:**
  - **Memory Usage:** Integrating multiple detection modules may lead to increased memory usage, especially when processing high-resolution images or video streams. Managing memory efciently becomes crucial to avoid crashes or slowdowns.
  - **Hardware Dependency:** The performance of the integrated system may vary depending on the underlying hardware capabilities, such as CPU, GPU, or memory capacity. Ensuring compatibility and optimal utilization across diferent hardware confgurations is essential.

- **Testing and Validation:**
  - **Testing Coverage:** Validating the integrated system requires comprehensive testing to ensure that all scenarios are adequately covered. This includes testing under various lighting conditions, camera angles, object sizes, and environmental factors.

- **Ground Truth Establishment:** Establishing ground truth data for evaluation becomes challenging when dealing with multiple object classes and detection modalities. Creating accurate annotations and benchmarks for validation is crucial but time-consuming.

- **Maintenance and Updates:**
  - **Modular Maintenance:** Updating or replacing individual detection modules while maintaining system integrity can be challenging. Changes in one module may require adjustments in others to ensure compatibility and functionality.
  - **Algorithm Evolution:** As new algorithms and techniques emerge, integrating them into the existing system requires ongoing maintenance and updates to stay competitive and address emerging challenges.

- **Frame Rate Improvement:**
  - **Slow Video Processing:** Combining all fve features can make the system slow, causing delays in showing the video. To make it faster, we need to fnd ways to speed up how the system works.
  - **Identifying Delays:** We need to fgure out where the system gets stuck to make it faster. This might mean fnding parts of the system that are taking a long time to do their job.
  - **Using Better Technology:** We can try using better technology to speed things up, like special computer chips or using the computer's graphics card more efciently. But making these changes can be complicated.
  - **Smarter Ways of Working:** We can also try to make the system's software work smarter, fnding ways to do things faster without making mistakes.
  - **Sharing the Load:** Instead of making one part of the system do all the work, we can split the work between diferent parts. This can help speed things up, but it's tricky to make sure everything works together smoothly.

Question 2 : Accuracy analysis and verifcation methods, code, and discussion (from report):

Functionality	Actual Values	TP	TN	FP	FN	Accuracy	Recall	Precision	F1-Score
Pedestrians	15	7	1680	10	8	0.99	0.47	0.41	0.44
Vehicle detection	40	31	160	17	17	0.85	0.65	0.65	0.65
Stop Signal	14	9	3520	22	22	0.99	0.29	0.29	0.29
Stop Signs	1	1	240	9	9	0.93	0.10	0.10	0.10
Lane Detection	3	3	560	2	2	0.99	0.60	0.60	0.60

1. Accuracy Analysis:

Vehicle Detection: The accuracy here is relatively high at 85%. However, the precision is quite low at 65%, suggesting a high rate of false positives. This could be a focus area for improvement.

Signal Detection: This shows a very high accuracy of 99%, but again, precision is low (29%), indicating issues with false positive detections in trafc signal recognition.

Pedestrian Detection: Similarly, high accuracy at 99% but low precision at 40%, suggesting that while the system efectively avoids false negatives, it struggles with false positives.

Lane Detection: The accuracy is high at 99% with a recall of 60%, indicating that the system detects the majority of actual lane markings. However, there is room for improvement in identifying all lane markings accurately.

Stop Sign: The accuracy for stop sign detection is 93%, indicating that the system correctly identifies the majority of stop signs present in the environment. The precision for stop sign detection is relatively low at 10%, suggesting a high rate of false positives. There is a considerable chance it might not be a genuine stop sign.

2. Continuous Processing Capability:

Frame Rates: The project aims to make video processing faster. Right now, it's meeting the target requirements, but it's not as fast as we want it to be for the best performance. We need to make it even faster to get the results we're looking for.

3. Stability and Depth Estimation:

The system has difculty staying consistent in diferent environmental situations, especially when the weather is extreme or the lighting is inconsistent.

Recommendations for Improvement:

Improving Algorithms: Make the object recognition algorithms better at identifying objects accurately and reducing mistakes. This could mean using more advanced techniques like deeper neural networks or smarter ways of processing images to tell apart important objects from ones that aren't important.

Adding More Training Data: Include a bigger variety of situations in the data used to train the system, especially ones that are difcult like extreme weather conditions.

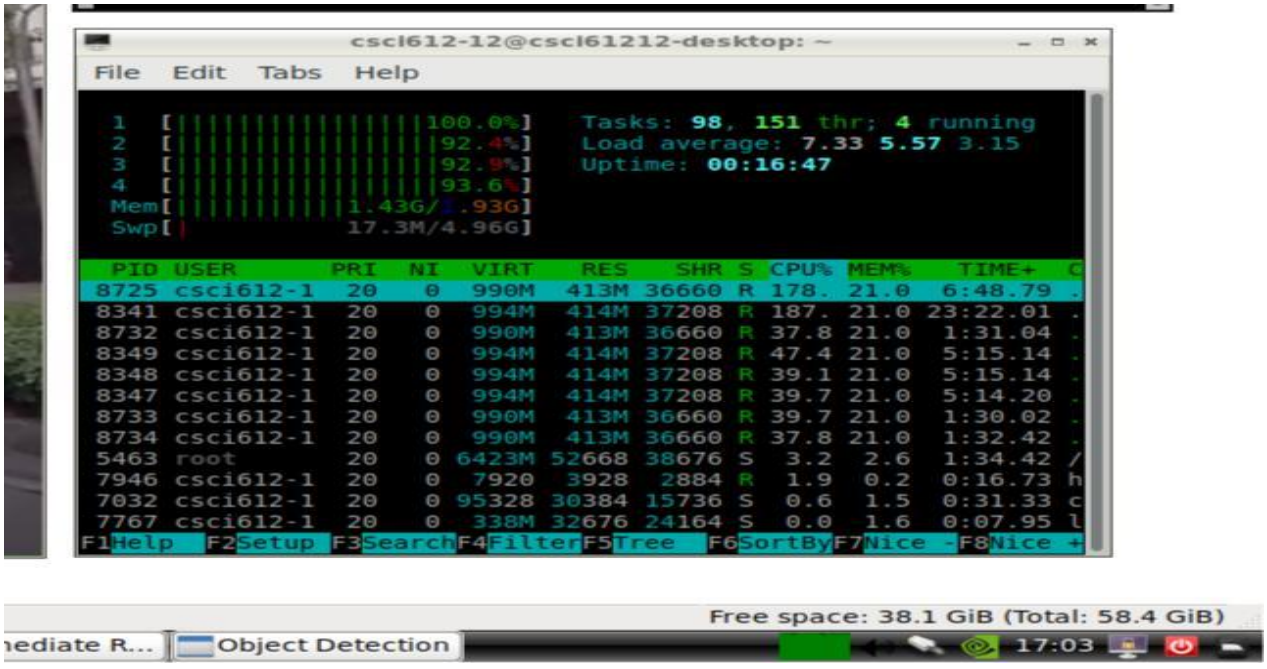
Documentation and Testing:

Based on our testing, our code meets the target standards set for it. We're aiming to achieve the desired and Optimal levels of performance while also keeping false positives as low as possible.

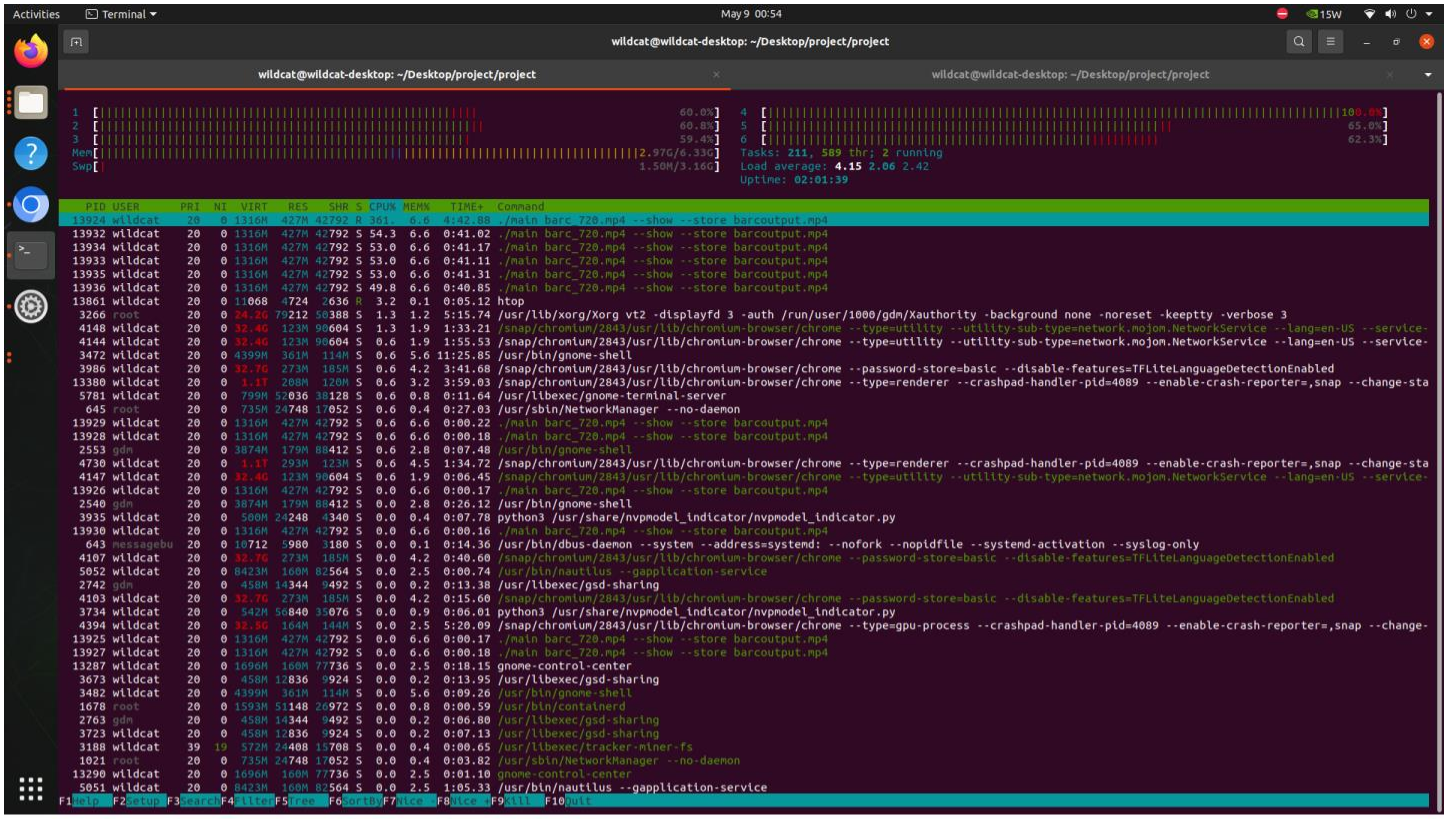
Technical Reporting:

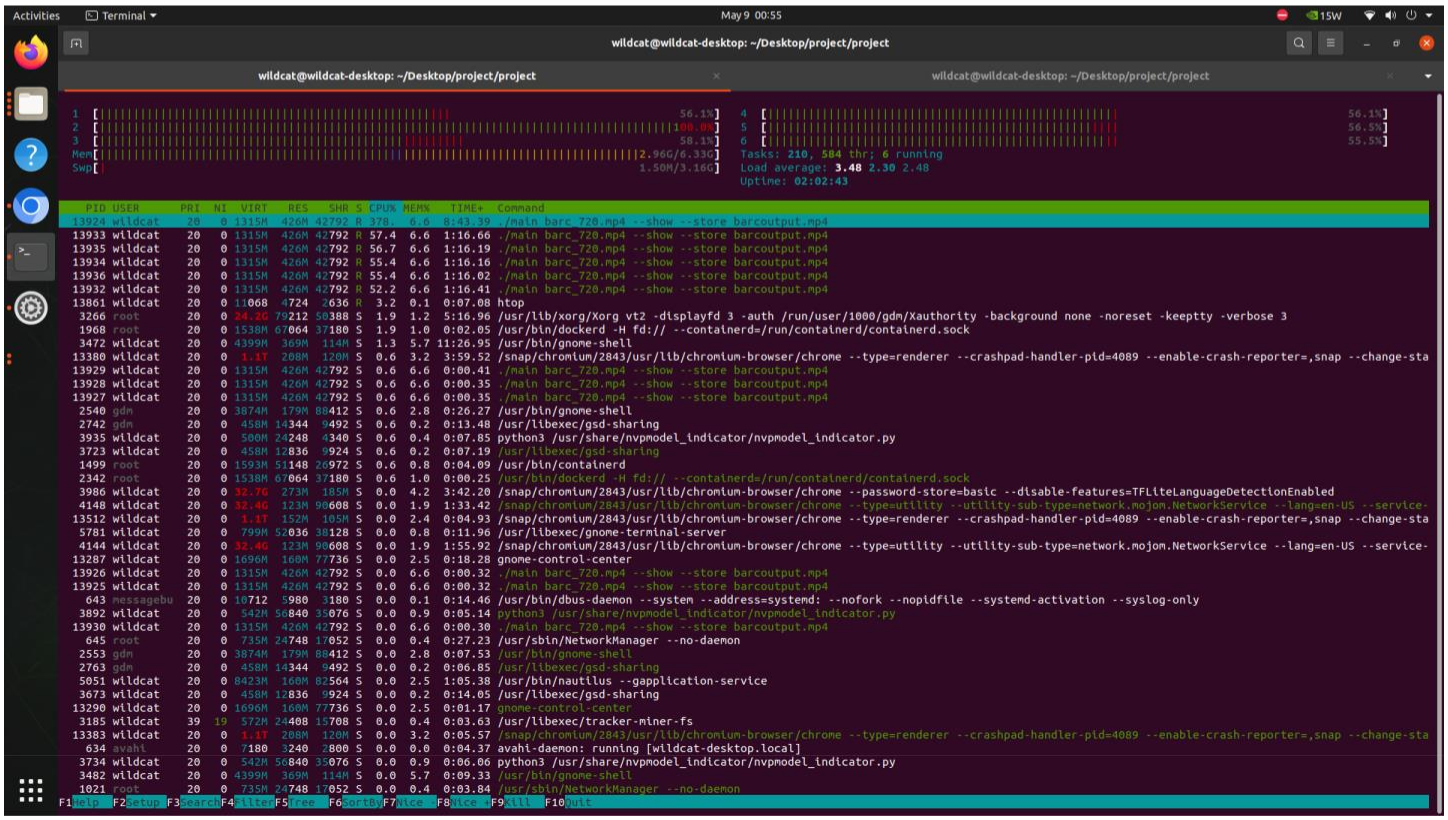
The report includes all the converted frames with annotations attached for reference.

Embedded Performance Analysis (HTOP GPU Status of Jetson Nano 2g):



Embedded Performance Analysis (HTOP GPU Status of Jetson Orin):





Question 3 : Report by required section:

Cover Page

Project Name:

Advanced Driver Assistance System (AV) / (ADAS)

PRANJAL PIMPALE

Introduction

Our team proposes an Advanced Driver Assistance System (ADAS) aimed at enhancing road safety and driver convenience by integrating advanced processing and detection technologies. ADAS utilizes computer vision and machine learning algorithms to detect and interpret key elements of the driving environment, providing real-time feedback and assistance to drivers. Our system focuses on crucial features such as lane detection, traffic light detection, stop sign detection, pedestrian detection, and vehicle detection.

The foundation of ADAS lies in progressively achieving set objectives, starting with basic detection capabilities and advancing towards more sophisticated functionalities. These objectives include ensuring accurate detection under different conditions, timely alerts to drivers, and consistent performance across diverse scenarios. As we strive towards more ambitious goals, such as predictive algorithms and cooperative collision avoidance, our vision for ADAS expands to offer comprehensive assistance and contribute to safer transportation systems.

This report outlines the design and development process of ADAS, detailing functional requirements, machine vision, and machine learning specifications, as well as implementation and testing phases. Through thorough analysis, design, and testing, our goal is to deliver an intelligent ADAS solution that sets new standards in road safety and driver assistance.

Functional (capability) Requirements

---

1. Lane Detection

- Technology Used: Hough Transform

- **Current Capability:**
    - Lane detection meets minimum and target levels. Currently working towards achieving optimal precision, recall, and throughput levels.

- Minimum Level (Achieved):
    - Achieved lane detection on well-marked roads under various lighting and weather conditions.
    - Throughput: 1 fps
    - Precision (P): ≥ 0.3
    - Recall (R): ≥ 0.3

- Target Level (Achieved):
    - Extend lane detection to various road types including freeways, urban roads, and intersections.
    - Enhance precision to over 0.5 and recall to over 0.5.
    - Throughput: 2-5 fps

- Optimal Level:
    - Implement advanced lane tracking algorithms for precise positioning within lanes.
    - Achieve throughput of 6-9 fps with a precision of at least 0.7 and a recall of at least 0.7.

2. Traffic Light Detection

- Technology Used: Haar Cascades

- **Current Capability:**
    - Traffic light detection meets minimum and target levels. Currently focusing on improving precision, recall, and throughput.

- Minimum Level (Achieved):
    - Detects traffic lights and their states (RED, YELLOW, GREEN) before the vehicle reaches intersections.
    - Throughput: 1 fps
    - Precision (P): ≥ 0.7
    - Recall (R): ≥ 0.3

- Target Level (Achieved):
    - Differentiate between traffic light states with high accuracy.

    - Enhance precision to over 0.25 and recall to over 0.2.
    - Throughput: 2-5 fps

- Optimal Level:
    - Implement deep learning models for precise delineation of traffic light regions.
    - Achieve throughput of 6-9 fps with a precision of at least 0.35 and a recall of at least 0.3.

3. Stop Sign Detection

- Technology Used: Haar Cascades

- **Current Capability:**
    - Stop sign detection meets minimum and target levels. Currently striving to achieve optimal precision, recall, and throughput.

- Minimum Level (Achieved):
    - Detect stop signs and provide timely alerts to the driver.
    - Throughput: 1 fps
    - Precision (P): ≥ 0.03
    - Recall (R): ≥ 0.03

- Target Level (Achieved):
    - Enhance detection range and angle coverage.
    - Achieve throughput of 2-5 fps with a precision of over 0.09 and a recall of over 0.08.

- Optimal Level:
    - Integrate stop sign recognition with advanced driver assistance features.
    - Achieve throughput of over 6-9 fps with a precision of at least 0.12 and a recall of at least 0.11.

4. Pedestrian Detection

- Technology Used: Haar Cascades

- **Current Capability:**
    - Pedestrian detection meets minimum and target levels. Currently working towards achieving optimal precision, recall, and throughput.

- Minimum Level (Achieved):
    - Detects pedestrians in the vicinity of the vehicle.
    - Throughput: 1 fps
    - Precision (P): ≥ 0.35
    - Recall (R): ≥ 0.3

- Target Level (Achieved):
    - Extend pedestrian detection capabilities to recognize various poses and behaviors.
    - Achieve throughput of 2-5 fps with a precision of over 0.4 and a recall of over 0.45.

- Optimal Level:
  - Deploy 3D object detection techniques for accurate pedestrian localization.
  - Achieve throughput of over 6-9 fps with a precision of at least 0.45 and a recall of at least 0.5.

5. Vehicle Detection

- Technology Used: Haar Cascades

- Current Capability:
  - Vehicle detection meets minimum and target levels. Currently focusing on achieving optimal precision, recall, and throughput.

- Minimum Level (Achieved):
    - Detects vehicles in the surroundings of the host vehicle.
    - Throughput: 1 fps
    - Precision (P): ≥ 0.5
    - Recall (R): ≥ 0.5

- Target Level (Achieved):
    - Enhance detection range and accuracy to distinguish between different types of vehicles.
    - Achieve throughput of 2-5 fps with a precision of 0.62 and a recall of 0.6.

- Optimal Level:
    - Implement vehicle re-identification techniques for robust tracking across multiple camera viewpoints.
    - Achieve throughput of 6-9 fps with a precision of at least 0.7 and a recall of at least 0.65.

Functionality	Actual Values	TP	TN	FP	FN	Accuracy	Recall	Precision	F1-Score
Pedestrians	15	7	1680	10	8	0.99	0.47	0.41	0.44
Vehicle detection	40	31	160	17	17	0.85	0.65	0.65	0.65
Stop Signal	14	9	3520	22	22	0.99	0.29	0.29	0.29
Stop Signs	1	1	240	9	9	0.93	0.10	0.10	0.10
Lane Detection	3	3	560	2	2	0.99	0.60	0.60	0.60

Machine Vision and Machine Learning Requirements

Machine Vision Requirements

Image Processing:

1. Real-time Processing Capabilities: The system must be capable of handling streaming video data at a minimum of 1 frame per second (fps) to ensure timely response and accurate detection of dynamic elements in the driving environment.

2. Robust Algorithms: Algorithms for image segmentation, object detection, and feature extraction should demonstrate robustness under varying lighting and weather conditions. This robustness is essential for maintaining consistent performance and accuracy across diferent driving scenarios.

3. Implementation of Hough Transforms: Hough transforms should be implemented for lane detection, enabling the system to accurately identify lane markings even in challenging conditions. Additionally, the use of Haar cascades and Histogram of Oriented Gradients (HOG) for object recognition tasks enhances the system's ability to detect and classify relevant objects in the scene.

Environmental Adaptability:

1. Adaptation to Diferent Lighting Conditions: Algorithms must be capable of adapting to diferent lighting conditions, including low-light, backlight, and direct sunlight scenarios. This adaptability ensures reliable performance under diverse environmental conditions encountered during driving.

2. Handling Visual Obstructions: The system should be able to handle visual obstructions such as shadows and occlusions without signifcant degradation in performance. Robust algorithms capable of detecting and mitigating the efects of obstructions ensure consistent and accurate detection results.

Machine Learning Requirements

Model Training:

1. Supervised Learning Models: Models should be trained using supervised learning techniques on a diverse dataset comprising various trafc scenarios, road types, and weather conditions. This dataset serves as the foundation for training models to recognize and interpret diferent elements in the driving environment.

2. Annotated Dataset: A large annotated dataset should be used for training and validating the models to ensure broad generalization capabilities. The dataset, consisting of 100 false and 100 positive images for each feature, provides ample training examples to enable the models to learn complex patterns and variations inherent in real-world driving scenarios.

Model Performance:

1. Precision and Recall Rate: Models must achieve a precision and recall rate of at least 60% in controlled test environments. This performance metric ensures that the models can efectively detect and classify relevant objects while minimizing false positives and negatives, thereby enhancing the system's overall reliability and accuracy.

Model Deployment:

1. Efcient Model Architecture: The system requires an efcient model architecture suitable for real-time inference on embedded systems with limited computational resources such as the NVIDIA Jetson or Raspberry Pi. Optimized model architectures enable seamless deployment of the ADAS system in resource-constrained environments without compromising performance or functionality.

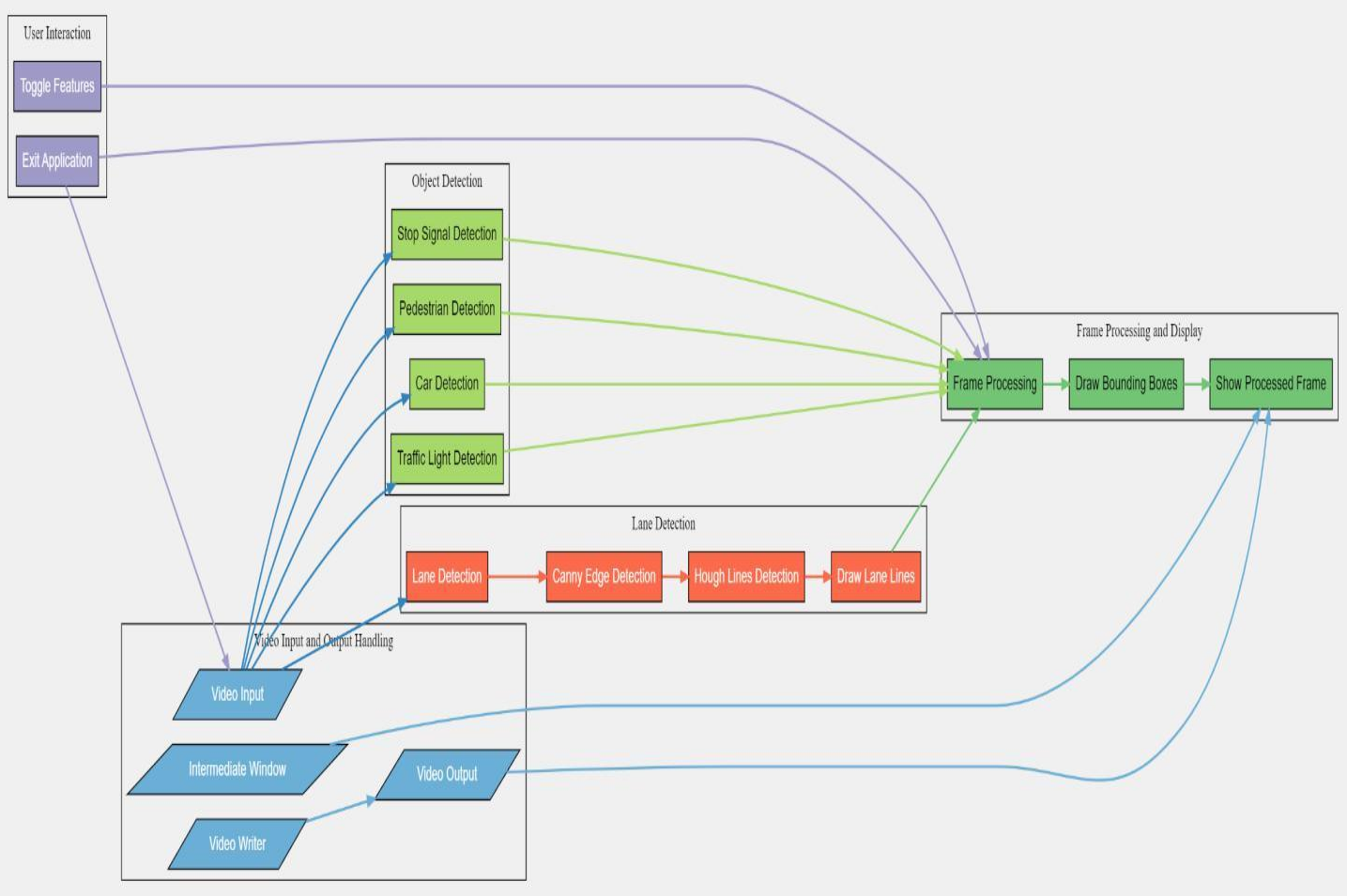
Functionality	Actual Values	TP	TN	FP	FN	Accuracy	Recall	Precision	F1-Score
Pedestrians	15	7	1680	10	8	0.99	0.47	0.41	0.44
Vehicle detection	40	31	160	17	17	0.85	0.65	0.65	0.65
Stop Signal	14	9	3520	22	22	0.99	0.29	0.29	0.29
Stop Signs	1	1	240	9	9	0.93	0.10	0.10	0.10
Lane Detection	3	3	560	2	2	0.99	0.60	0.60	0.60

\

Functional Design Overview and Diagrams

---

1. Block Diagram:

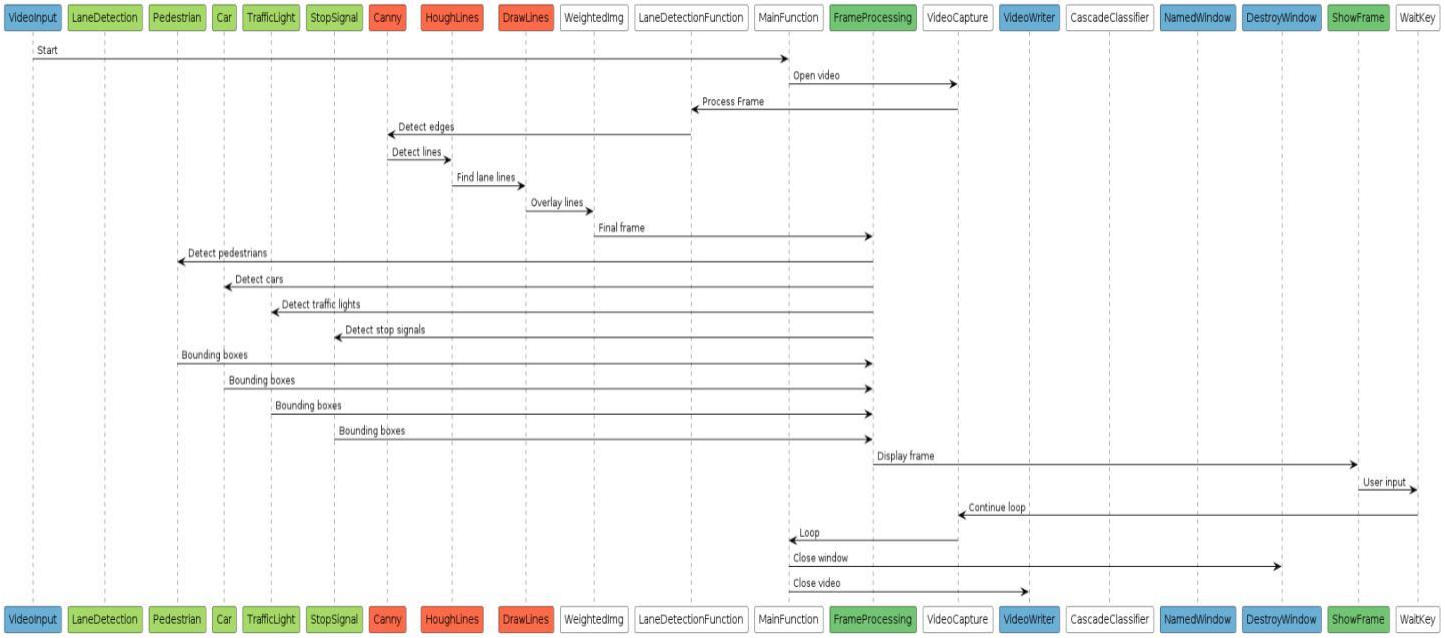


Block Diagram Description:

- **User Interaction:** This is where you, as the user, control how the system behaves. You can turn diferent features on or of and exit the application when you're done.
- **Video Input:** This is where the system gets the video data it needs to work with. This could be from a live camera feed, a video fle you've already recorded, or some other video source.
- **Intermediate Processing:** Before the video data moves on to the next stages, it might go through some initial adjustments or transformations. Think of this like preparing the video for further analysis.
- **Video Output:** Once the video data has been processed, this component handles how it's presented to you. It could be displayed on your screen, saved to a fle, or sent to another location.
- **Object Detection:** Here's where the system identifes diferent objects in the video, like stop signs, pedestrians, cars, and trafc lights. It uses special algorithms to do this, which are trained to recognize these objects.

- **Lane Detection:** This part focuses specifcally on fnding the lanes on the road in the video. It uses techniques like edge detection and line detection to do this, which are common tools in computer vision.
- **Frame Processing:** Once objects and lanes are detected, they're added to the video frames. For example, bounding boxes might be drawn around objects, and lane lines might be overlaid on the lanes.
- **Show Processed Frame:** Finally, the processed video frames are shown to you. This could be in a window on your screen or in some other format, depending on how you've set up the system.

2. Sequence Diagram:

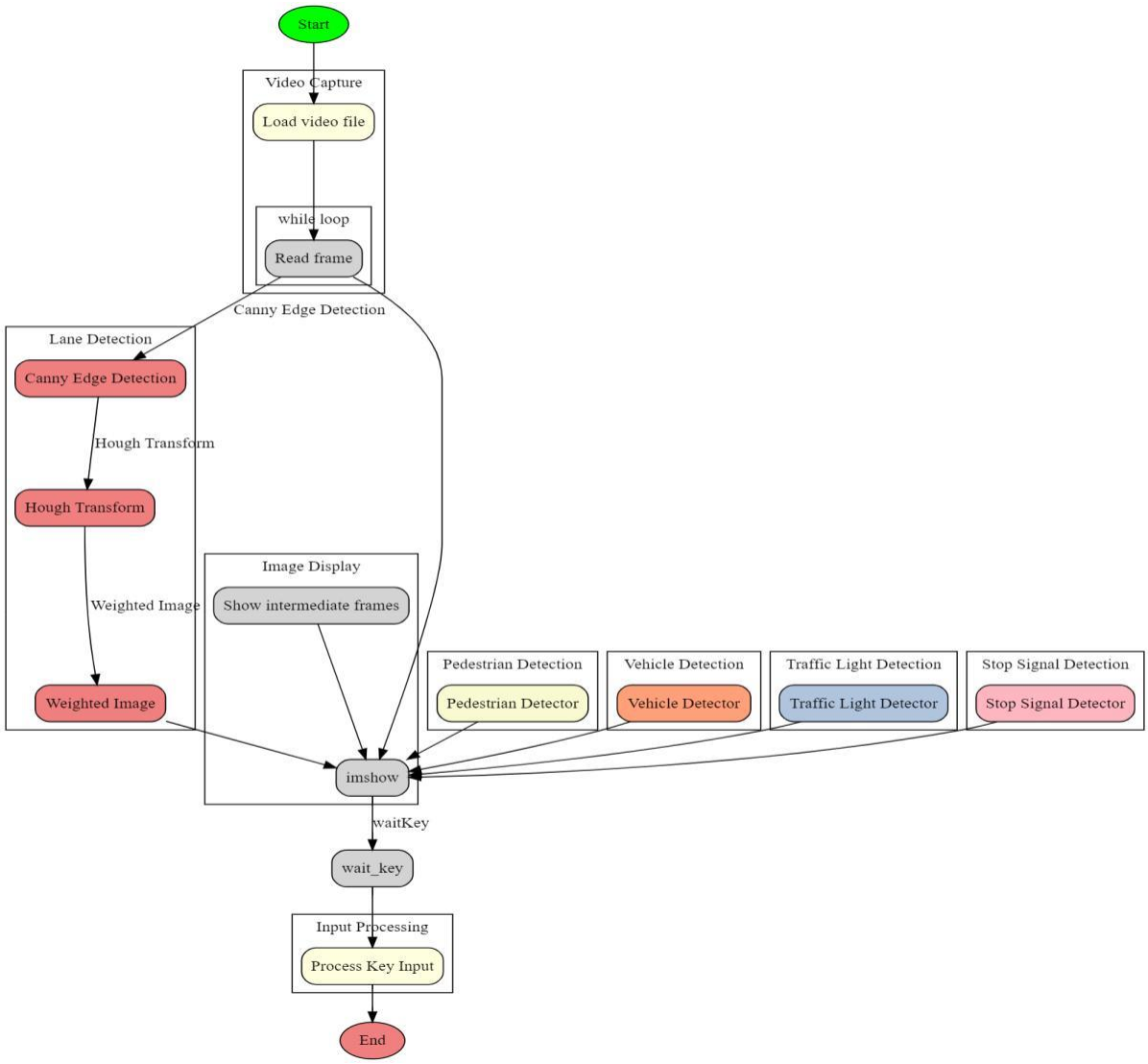


Sequence Diagram Description:

- **Start:** The process begins with the initiation of the video processing system. This could be triggered by a user action such as pressing a start button or launching a program.
- **Open Video:** The system opens the video fle or stream that is to be processed. This could involve reading a video fle from a disk, opening a live video stream from a camera, or accessing a video from a network location.
- **Detecting Pedestrians:** In this step, the system applies a pedestrian detection algorithm to each frame of the video. This could involve using machine learning models trained to recognize pedestrians in images. The output of this step would be coordinates of bounding boxes around detected pedestrians in the frame.
- **Detected Cars:** Similarly, the system applies a car detection algorithm to each frame. This could involve a diferent machine learning model trained specifcally to recognize cars. The output would be bounding boxes around detected cars.
- **Detect Stop Signals:** The system also detects stop signals in each frame. This could involve yet another machine learning model or a color and shape-based detection algorithm, depending on the specifc requirements of the system.

- **Bounding Boxes:** For each detected object (pedestrians, cars, stop signals), the system draws bounding boxes around them. These boxes serve to highlight the detected objects in the video.
- **Overlay Lines:** In addition to object detection, the system also detects lines in the video frames. These could represent lane markings on a road, for example. The detected lines are then overlaid onto the video frames.
- **Final Frame:** After all detections and overlays have been added, the system prepares the fnal processed frame. This frame now contains the original video data plus the added bounding boxes and overlays.
- **Display Frame:** The fnal processed frame is then displayed to the user. This could be in a dedicated video player window, for example.
- **Continue Loop:** The system continues to process the video in a loop, frame by frame. It applies all the detection and overlay steps to each frame in turn.
- **Close Window:** If the user decides to stop the video processing, they can trigger an action (such as pressing a stop button) that causes the video display window to close.
- **Close Video:** Finally, the system stops processing the video and releases any resources it was using for the video processing.

3. Flow Diagram:



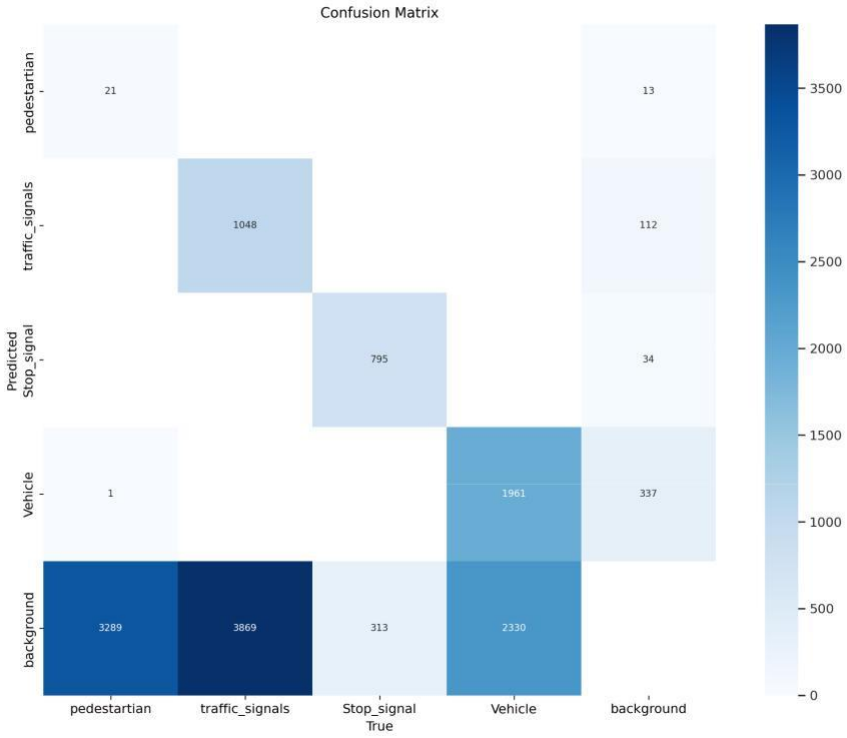
Description of Flow Diagram:

- **Start:** The process begins with the initiation of the video processing system. This could be triggered by a user action such as pressing a start button or launching a program.
- **Video Capture or Load Video File:** The system opens the video fle or stream that is to be processed. This could involve reading a video fle from disk, opening a live video stream from a camera, or accessing a video from a network location.
- **Read Frame:** In this step, the system reads each frame of the video in a loop for processing. This is the fundamental unit of operation since all subsequent operations are performed on these individual frames.
- **Canny Edge Detection:** This is a popular edge detection algorithm in image processing. It's used here to detect edges in each frame, which can be useful for various subsequent tasks like lane detection.
- **Hough Transform:** This is a feature extraction technique used in image analysis. In the context of this system, it's likely being used to detect straight lines in the image, such as lane markings on a road.
- **Weighted Image:** This could be a step where the outputs of the edge detection and line detection are combined with the original image to create a "weighted" image. This might involve overlaying the detected lines onto the original image.
- **Image Display (Show intermediate frames):** The system displays the intermediate frames after they have been processed. This could be useful for visualizing the output of the edge and line detection steps.
- **Object Detection:** The system includes separate detectors for pedestrians, vehicles, trafc lights, and stop signals. Each detector would be responsible for identifying its respective object in each frame.
- **imshow:** This is a function in many image processing libraries that displays an image in a window. In this system, it's used to display the fnal output after all detections have been made.
- **waitKey (wait_key):** This is a common function in video processing which waits for a keyboard event. It's used here to allow the user to control the fow of the video, such as pausing the video or ending the loop.
- **Input Processing (Process Key Input):** Depending on the key pressed by the user, diferent actions are taken. For example, the user could pause the video, skip to the next frame, or end the video.
- **End:** The process ends when the user decides to stop the video processing. This could be when the video has been fully processed, or when the user decides to prematurely end the process.

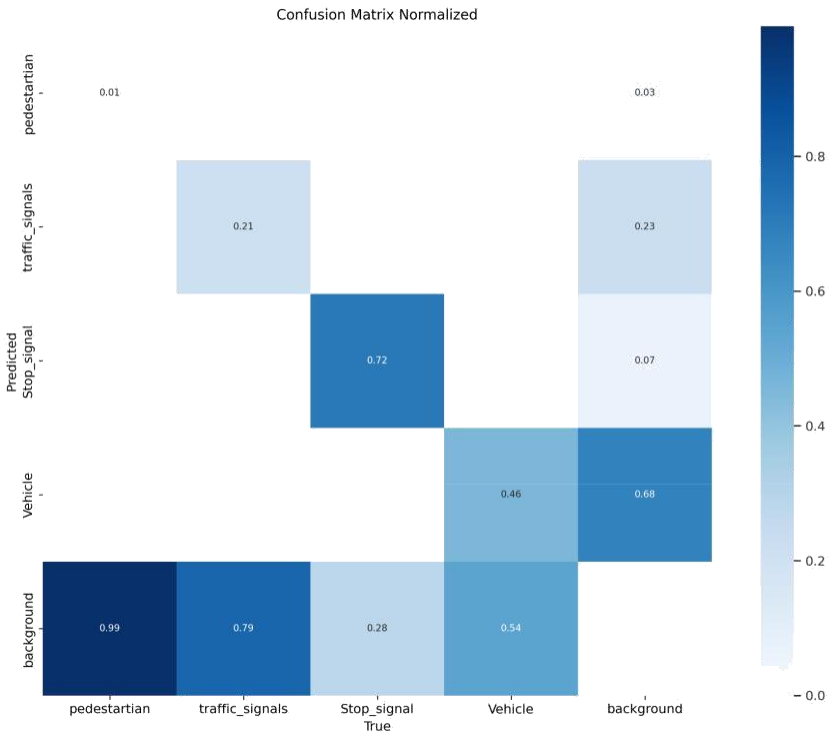
ACV Analysis and Design

- **ROC Curve :**
- **Confusion Matrix :**

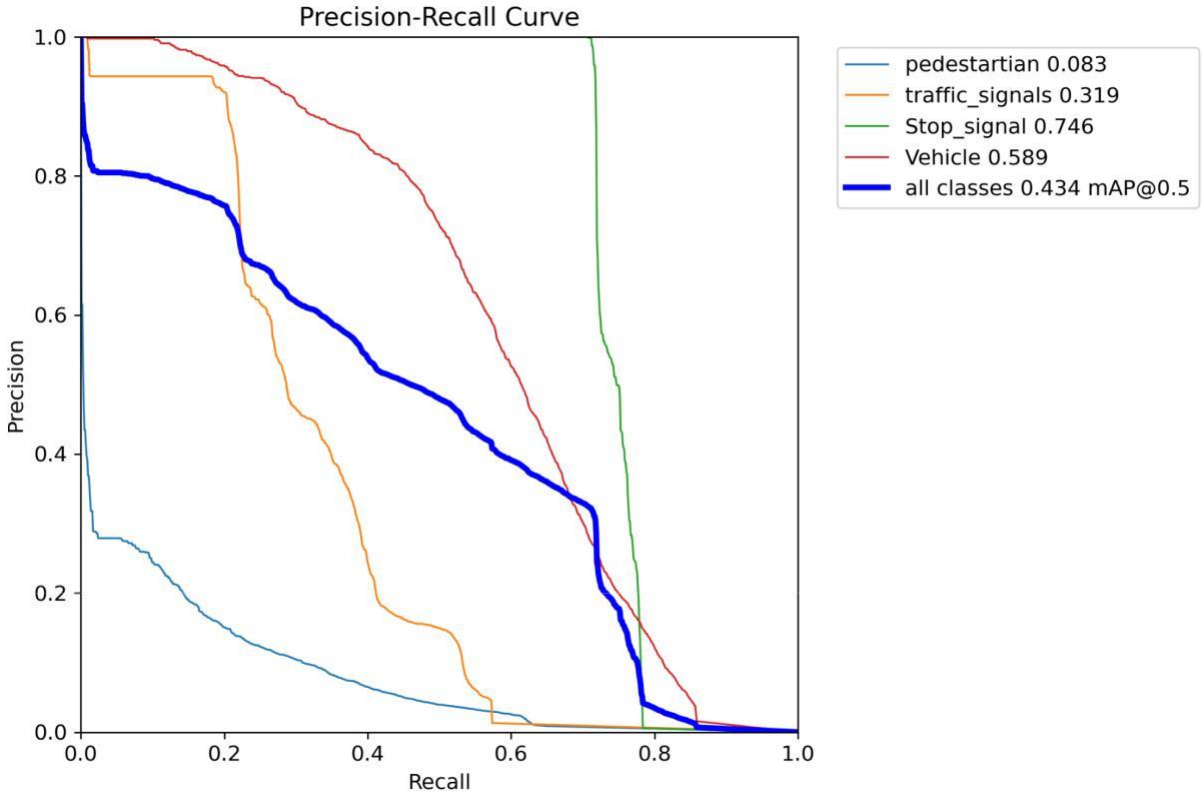
  All the below matrix and Curve are of the training data that we will be planning to use for the yolo model.



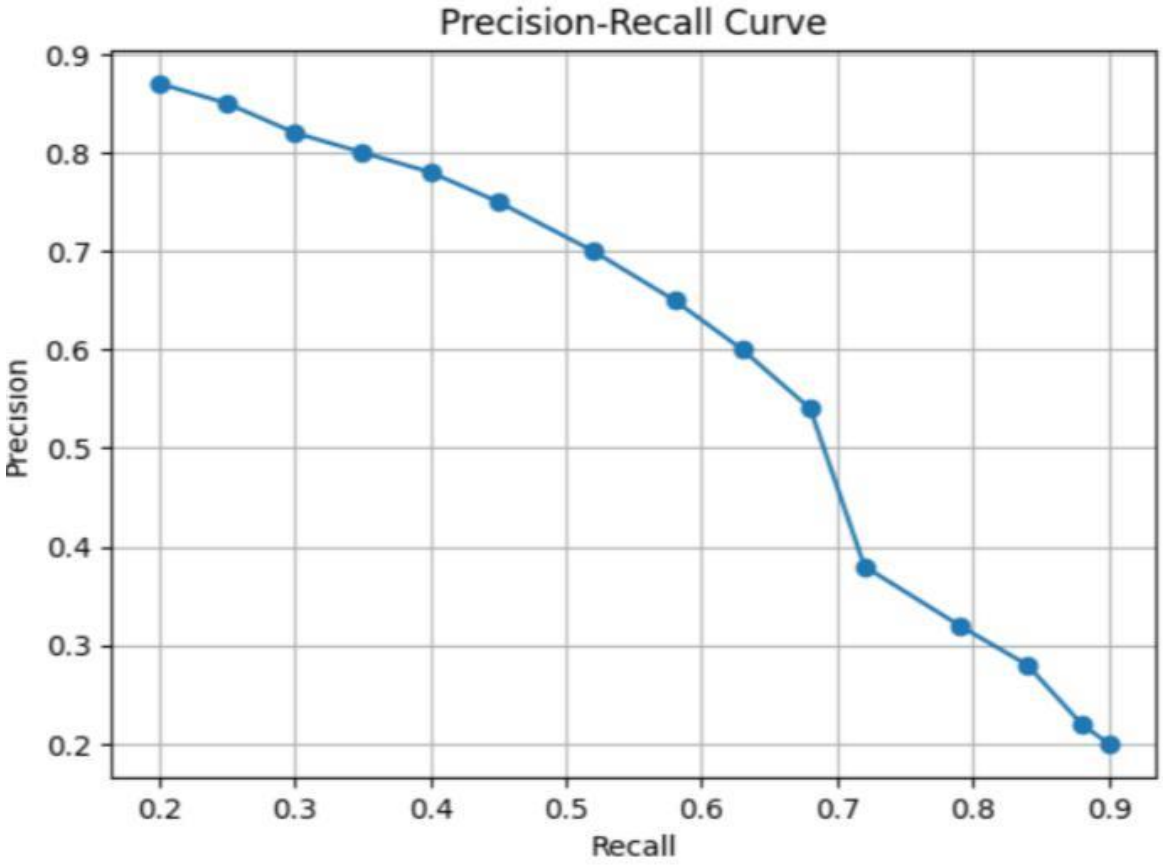
- **Confusion Matrix Normalized :**



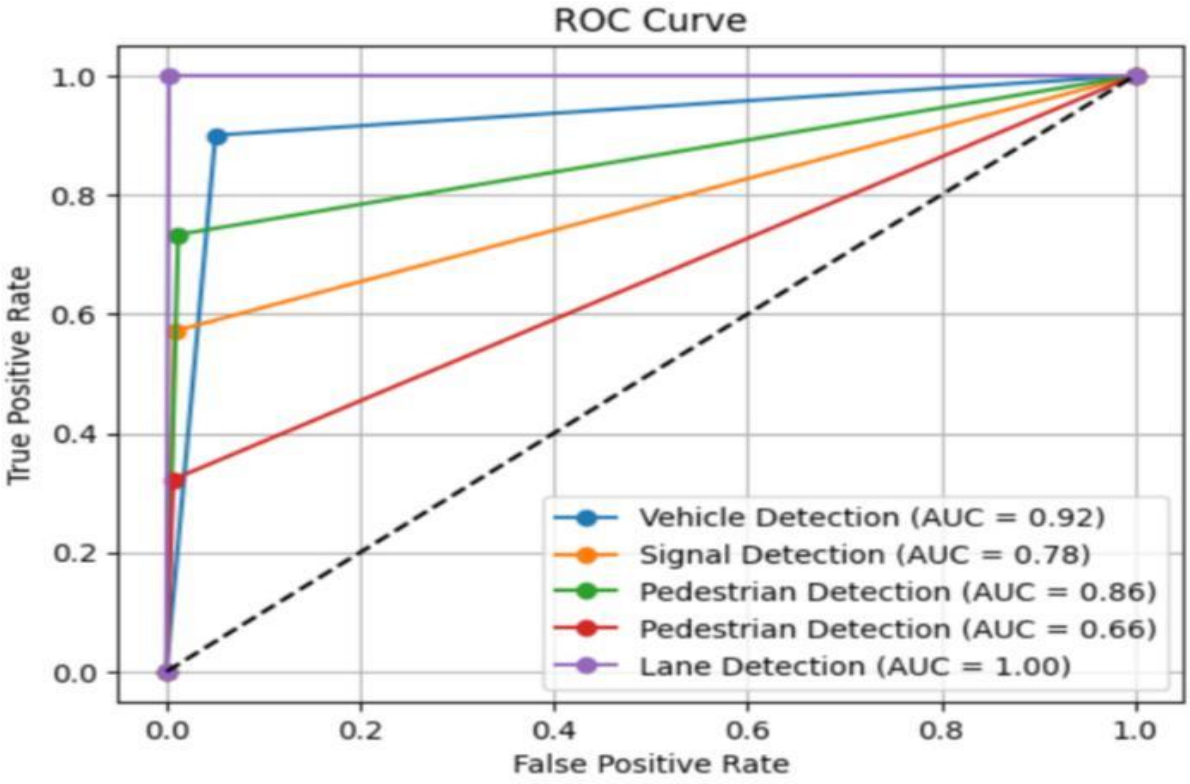
- **PR Curve of training data :**



- **PR Curve for all features that we have integrated for AV/ADAS:**



- **ROC Curve representing the various classes that we have integrated for AV/ADAS:**



Proof-of-Concept with Example Output and Tests Completed - (Must contain fnal annotated MPEG)

Code Snippet:

```
void drawLines(Mat img, vector<Vec4f> lines, int thickness = 5)
{
    Scalar right_color = Scalar(0, 0, 255);
    Scalar left_color = Scalar(0, 255, 255);
    vector<float> rightSlope, leftSlope, rightIntercept, leftIntercept;
    for (Vec4f line : lines)
    {
        float x1 = line[0];
        float y1 = line[1];
        float x2 = line[2];
        float y2 = line[3];
        float slope = (y1 - y2) / (x1 - x2);
        if (slope > 0.5)
        {
            if (x1 > 500)
            {
                float yintercept = y2 - (slope * x2);
                rightSlope.push_back(slope);
                rightIntercept.push_back(yintercept);
            }
        }
        else if (slope < -0.5)
        {
            if (x1 < 700)
            {
                float yintercept = y2 - (slope * x2);
                leftSlope.push_back(slope);
                leftIntercept.push_back(yintercept);
            }
        }
    }

    float left_intercept_avg = vectorAverage(leftIntercept);
    float right_intercept_avg = vectorAverage(rightIntercept);
    float left_slope_avg = vectorAverage(leftSlope); float
    right_slope_avg = vectorAverage(rightSlope);
```

```
        int left_line_x1 = (int)round((0.65 * img.rows - left_intercept_avg) /
left_slope_avg);
    int left_line_x2 = (int)round((img.rows - left_intercept_avg) / left_slope_avg);
       int right_line_x1 = (int)round((0.65 * img.rows - right_intercept_avg) /
right_slope_avg);
          int right_line_x2 = (int)round((img.rows - right_intercept_avg) /
right_slope_avg);
    Point line_vertices[1][4];
    line_vertices[0][0] = Point(left_line_x1, (int)round(0.65 * img.rows));
    line_vertices[0][1] = Point(left_line_x2, img.rows);
    line_vertices[0][2] = Point(right_line_x2, img.rows);
    line_vertices[0][3] = Point(right_line_x1, (int)round(0.65 * img.rows));
    const Point *inner_shape[1] = {line_vertices[0]};
    int n_vertices[] = {4};
    int lineType = LINE_8;
    fillPoly(img, inner_shape, n_vertices, 1, Scalar(0, 50, 0), lineType);
     line(img, Point(left_line_x1, (int)round(0.65 * img.rows)), Point(left_line_x2,
img.rows), left_color, 10);
    line(img, Point(right_line_x1, (int)round(0.65 * img.rows)), Point(right_line_x2,
img.rows), right_color, 10);
}

Mat hough_lines(Mat img, double rho, double theta, int threshold, double
min_line_len, double max_line_gap)
{
    vector<Vec4f> lines;
    Mat line_img(img.rows, img.cols, CV_8UC3, Scalar(0, 0, 0));
    HoughLinesP(img, lines, rho, theta, threshold, min_line_len, max_line_gap);

    drawLines(line_img, lines);
    return line_img;
}

Mat lineDetect(Mat img)
{
    return hough_lines(img, 1, CV_PI / 180, 50, 100, 100);
}

Mat weightedImage(Mat img, Mat initialImg, double alpha = 0.8, double beta = 1.0,
double gamma = 0.0)
{
    Mat weightedImg;
    addWeighted(img, alpha, initialImg, beta, gamma, weightedImg);
```

```
    return weightedImg;
}

Mat laneDetection(Mat src)
{
    Mat colorMasked, roiImg, cannyImg, houghImg,
    finalImg; Mat hls, yellowMask, whiteMask, maskN,
    masked; cvtColor(src, hls, COLOR_RGB2HLS);
    inRange(hls, Scalar(100, 0, 90), Scalar(50, 255, 255),
    yellowMask); inRange(hls, Scalar(0, 70, 0), Scalar(255, 255, 255),
    whiteMask); bitwise_or(yellowMask, whiteMask, maskN);
    bitwise_and(src, src, masked, maskN = maskN);

    int x = masked.cols;
    int y = masked.rows;
    Point polygonVertices[1][4];
    polygonVertices[0][0] = Point(0, y);
    polygonVertices[0][1] = Point(x, y);
    polygonVertices[0][2] = Point((int)round(0.55 * x), (int)round(0.6 * y));
    polygonVertices[0][3] = Point((int)round(0.45 * x), (int)round(0.6 * y));
    const Point *polygons[1] = {polygonVertices[0]}; int n_vertices[] = {4};

    Mat mask(y, x, CV_8UC1, Scalar(0));
    int lineType = LINE_8;
    fillPoly(mask, polygons, n_vertices, 1, Scalar(255, 255, 255), lineType);
    Mat maskedImage;
    bitwise_and(masked, masked, maskedImage, mask =
    mask); cannyImg = canny(maskedImage);
    houghImg = lineDetect(cannyImg);
    finalImg = weightedImage(houghImg,
    src); return finalImg;
}

class StopSignDetector
{
private:
    CascadeClassifier cascade;

public:
    StopSignDetector()
    {
        cascade.load("./xmlfile/stop_sign_classifier_2.xml");
    }
```

```
    vector<Rect> detectStopSigns(const Mat &img)
    {
        vector<Rect> found;
        cascade.detectMultiScale(img, found);

        // Filter out regions based on position and aspect ratio
        vector<Rect> filteredRegions;
        for (const auto &region : found)
        {
            double aspectRatio = static_cast<double>(region.width) / region.height;
            // Check if the aspect ratio is close to 1
            if (aspectRatio > 0.7 && aspectRatio < 1.3)
            {
                // Check if the region is not near the top of the
                frame if (region.y > img.rows / 3)
                {
                    filteredRegions.push_back(region);
                }
            }
        }

        return filteredRegions;
    }
};

int main(int argc, char *argv[])
{
    // Check for the correct number of command line
    arguments if (argc < 2)
    {
            cout << "Usage: " << argv[0] << " <video_file_name> [--show] [--store
<output_file_name>]" << endl;
        return -1;
    }

    // Load the video file
    VideoCapture cap(argv[1]);

    if (!cap.isOpened())
    {
        cout << "Error: Unable to open video file." << endl;
        return -1;
```

```
    }

    // Create a window for displaying intermediate results if --show flag is provided
    bool showIntermediate = false;
    string intermediateWindowName = "Intermediate Results";
    if (argc > 2 && string(argv[2]) == "--show")
    {
        showIntermediate = true;
        namedWindow(intermediateWindowName, WINDOW_AUTOSIZE);
    }

    // Create a video writer if --store flag is provided
    bool storeResults = false;
    VideoWriter outputVideo;
    string outputFileName;
    if (argc > 3 && string(argv[3]) == "--store" && argc > 4)
    {
        storeResults = true;
        outputFileName = argv[4];
        int codec = VideoWriter::fourcc('M', 'J', 'P', 'G');
                    Size frameSize = Size((int)cap.get(CAP_PROP_FRAME_WIDTH),
(int)cap.get(CAP_PROP_FRAME_HEIGHT));
         outputVideo.open(outputFileName,  codec,  cap.get(CAP_PROP_FPS),  frameSize,
true);
    }

    bool showNormalFrames = false;
    cout << "Press space to turn on or turn off the features." << endl;

    // Pedestrian detection CascadeClassifier
    pedestrianDetector;
    pedestrianDetector.load("./xmlfile/pedetrian1.xml");

    // Load the Haar Cascade classifier XML file for detecting cars
    CascadeClassifier carDetector;
    carDetector.load("./xmlfile/carDetection.xml");

    // Load the Haar Cascade classifier XML file for traffic light detection
    CascadeClassifier trafficLightDetector;
    trafficLightDetector.load("./xmlfile/traffic_light2.xml");

    // Framerate calculation
    auto start = std::chrono::steady_clock::now();
```

```
int fps = 0;
int currentFps = 1;

// Process each frame of the
video Mat frame;
StopSignDetector stopSignDetector;
while (cap.read(frame))
{
    // Lane detection
    frame = laneDetection(frame);

    // Pedestrian detection
    vector<Rect> pedestrian;
    pedestrianDetector.detectMultiScale(frame, pedestrian, 1.1, 5);

    // Detect cars in the current frame
    vector<Rect> cars;
    carDetector.detectMultiScale(frame, cars, 1.1, 5);

    // Detect traffic light in the current frame vector<Rect>
    trafficLight; trafficLightDetector.detectMultiScale(frame,
    trafficLight, 1.1, 2);

    // Detect stop signs in the current frame
    vector<Rect> stopSigns = stopSignDetector.detectStopSigns(frame);

    // Draw stop sign circles
    for (const auto &rect : stopSigns)
    {
        Point center(rect.x + rect.width / 2, rect.y + rect.height /
        2); int radius = max(rect.width, rect.height) / 2;
        circle(frame, center, radius, Scalar(0, 165, 255), 2);
    }

    // Draw pedestrian rectangles
    for (const auto &rect : pedestrian)
    {
        rectangle(frame, rect.tl(), rect.br(), Scalar(128, 0, 128), 2);
    }

    // Draw car rectangles
    for (const auto &rect : cars)
    {
```

```
            rectangle(frame, rect.tl(), rect.br(), Scalar(0, 255, 255), 2);
        }

        // Draw traffic light rectangles
        for (const auto &rect : trafficLight)
        {
            rectangle(frame, rect.tl(), rect.br(), Scalar(0, 0, 255), 2);
        }

        if (storeResults)
        {
            outputVideo.write(frame);
        }

        auto now = std::chrono::steady_clock::now();
             auto duration = std::chrono::duration_cast<std::chrono::seconds>(now -
start).count();
        fps++;
           putText(frame, "Frames/second: " + to_string(currentFps), Point(30, 30),
FONT_HERSHEY_SIMPLEX, 1.0, Scalar(0, 255, 0), 2);
        if (duration >= 1)
        {
            currentFps = fps;
            fps = 0;
            start = std::chrono::steady_clock::now();
        }
        imshow("Object Detection", frame);
        const char key = (char)waitKey(1);
        if (key == 27 || key == 'q')
        {
            cout << "Exit requested" << endl;
            cap.release();
            if (storeResults)
                outputVideo.release();
            if (showIntermediate)
                destroyWindow(intermediateWindowName);
            return 0;
        }
        else if (key == ' ')
        {
            cout << "Data: " << showNormalFrames << endl;
            if (showNormalFrames)
            {
```

```
                showNormalFrames = 0;
                cout << "Feature Detection Started" << endl;
            }
            else
            {
                showNormalFrames = 1;
                cout << "Features detection stopped" << endl;
            }
        }
    }

    return 0;
}
```

Code Description:

- **Header Includes:**
  - Includes necessary header fles for OpenCV, standard input/output operations, time measurement, and vector manipulation.

- **Namespace Declarations:**
  - `using namespace cv;` and `using namespace std;` to avoid writing `cv::` and `std::` prefxes before OpenCV and standard C++ functions, respectively.

- **Function Defnitions:**
  - `canny`: Converts the input image to grayscale and applies Canny edge detection to identify edges.
  - `vectorAverage`: Calculates the average value of elements in a vector.
  - `drawLines`: Draws lane lines on an image based on detected lines from Hough transform.
  - `hough_lines`: Detects lines in an edge-detected image using the Hough transform.
  - `lineDetect`: Detects lines in an image by calling `hough_lines`.
  - `weightedImage`: Combines two images with diferent weights to produce a weighted image.
  - `laneDetection`: Performs lane detection by combining edge detection, line detection, and image blending techniques.
  - `StopSignDetector`: Class for detecting stop signs using a Haar Cascade classifer, with a method `detectStopSigns` to identify stop signs in an image.

- **Main Function:**
  - Parses command line arguments to specify the input video fle, display option, and output fle storage.
  - Feature Initialization:
  - Loads XML fles for pedestrian, car, and trafc light detection using Haar Cascade classifers.
  - Frame Processing Loop:

- Iterates through each frame of the input video.
- Performs lane detection, pedestrian detection, car detection, trafc light detection, and stop sign detection on each frame.
- Draws rectangles or circles around detected objects.
- Calculates and displays frames per second (FPS) information.
- Allows toggling of feature detection using the spacebar.
- Optionally stores the processed video if specifed in the command line arguments.
- Resource Release:
- Properly releases resources (e.g., video capture, video writer) and exits the program when requested.

- **Object Detection:**
  - Utilizes Haar Cascade classifers for pedestrian, car, and trafc light detection.
  - Detects stop signs based on predefned criteria such as aspect ratio and position within the frame.

- **Lane Detection:**
  - Implements lane detection using a combination of edge detection (Canny), line detection (Hough transform), and image blending techniques.
  - Draws detected lane lines on the frame.

- **User Interaction:**
  - Allows the user to quit the program, toggle feature display, and view FPS information using keyboard inputs.

- **Overall Functionality:**
  - Performs real-time video processing for lane detection and object detection (pedestrians, cars, trafc lights, and stop signs).
  - Provides options for displaying intermediate results and storing the processed video.
  - Supports user interaction for controlling feature display and quitting the program.

How the code is capable of detecting and objects and lane:

a) Lane Detection:

- **Functionality:** Lane detection is performed using the following steps:
- **Preprocessing:** Convert the input frame to HLS color space and apply color masking to isolate white and yellow lane markings.
- **Edge Detection:** Apply Canny edge detection to the masked image to detect edges.
- **Line Detection:** Use the Hough transform to detect lines in the edge-detected image.
- **Lane Identifcation:** Based on the detected lines, identify left and right lane lines.
- **Drawing:** Draw the identifed lane lines on the original frame.
- **Implementation:** The laneDetection function orchestrates these steps to detect and draw lanes on each frame.

b) Trafc Signal Detection:

- **Functionality:** Detection of trafc signals involves identifying regions in the frame that correspond to trafc lights.
- **Implementation:** This functionality is achieved using a Haar Cascade classifer loaded from an XML fle (trafc_light2.xml). The classifer is applied to each frame to detect trafc lights, and rectangles are drawn around the detected regions.

c) Stop Sign Detection:

- **Functionality:** Detecting stop signs entails identifying regions in the frame that represent stop signs.
- **Implementation:** Similar to trafc signals, stop sign detection is accomplished using a Haar Cascade classifer loaded from an XML fle (stop_sign_classifer_2.xml). Detected stop signs are fltered based on position and aspect ratio criteria, and circles are drawn around the detected regions.

d) Pedestrian Detection:

- **Functionality:** Pedestrian detection involves identifying regions in the frame corresponding to pedestrians.
- **Implementation:** A Haar Cascade classifer (pedetrian1.xml) is loaded to detect pedestrians. Detected pedestrian regions are represented by rectangles drawn around them.

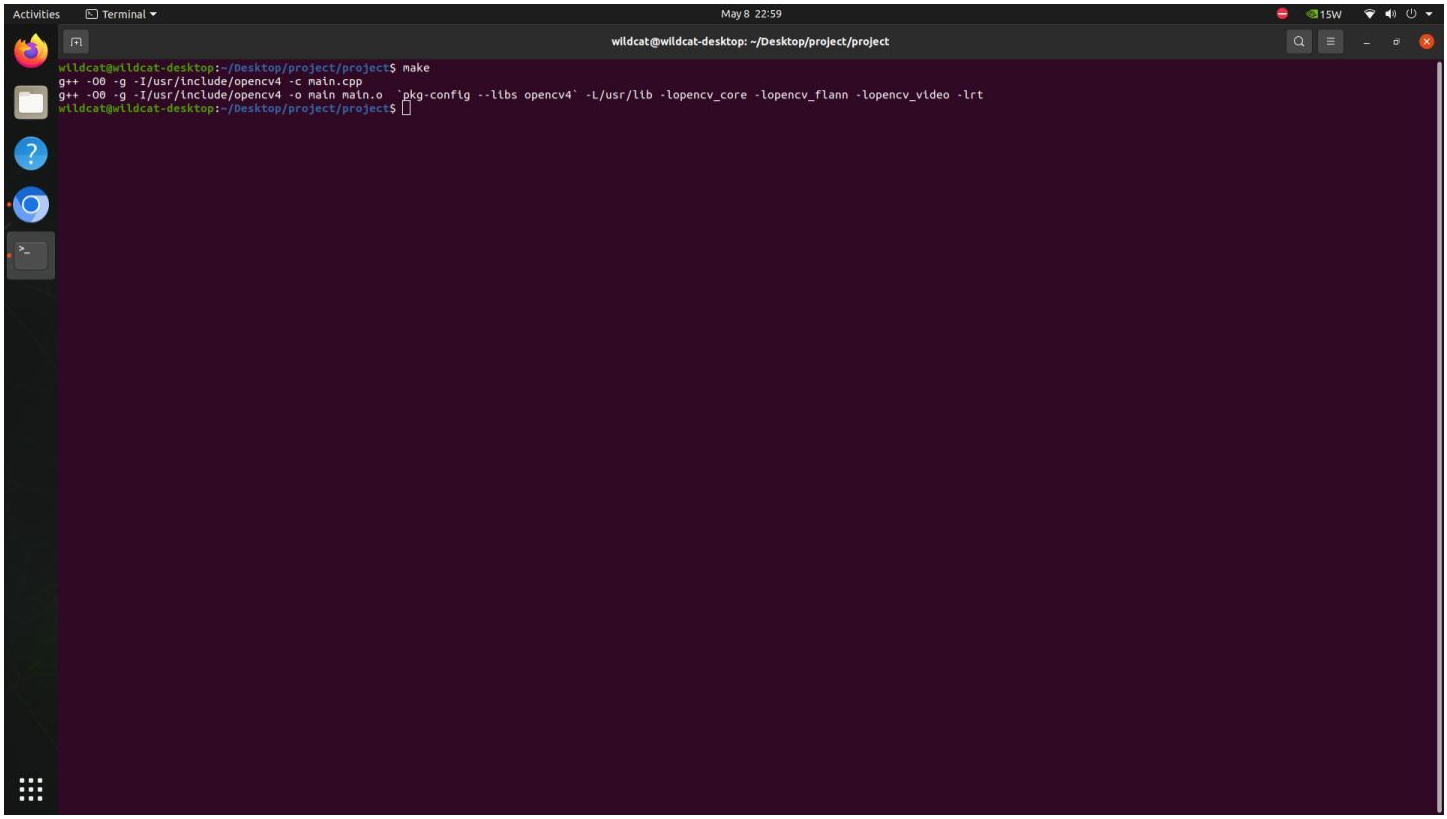
e) Vehicle Detection:

- **Functionality:** Detecting vehicles entails identifying regions in the frame that correspond to vehicles.
- **Implementation:** Similar to pedestrian detection, vehicle detection is performed using a Haar Cascade classifer (carDetection.xml). Detected vehicles are represented by rectangles drawn around them.

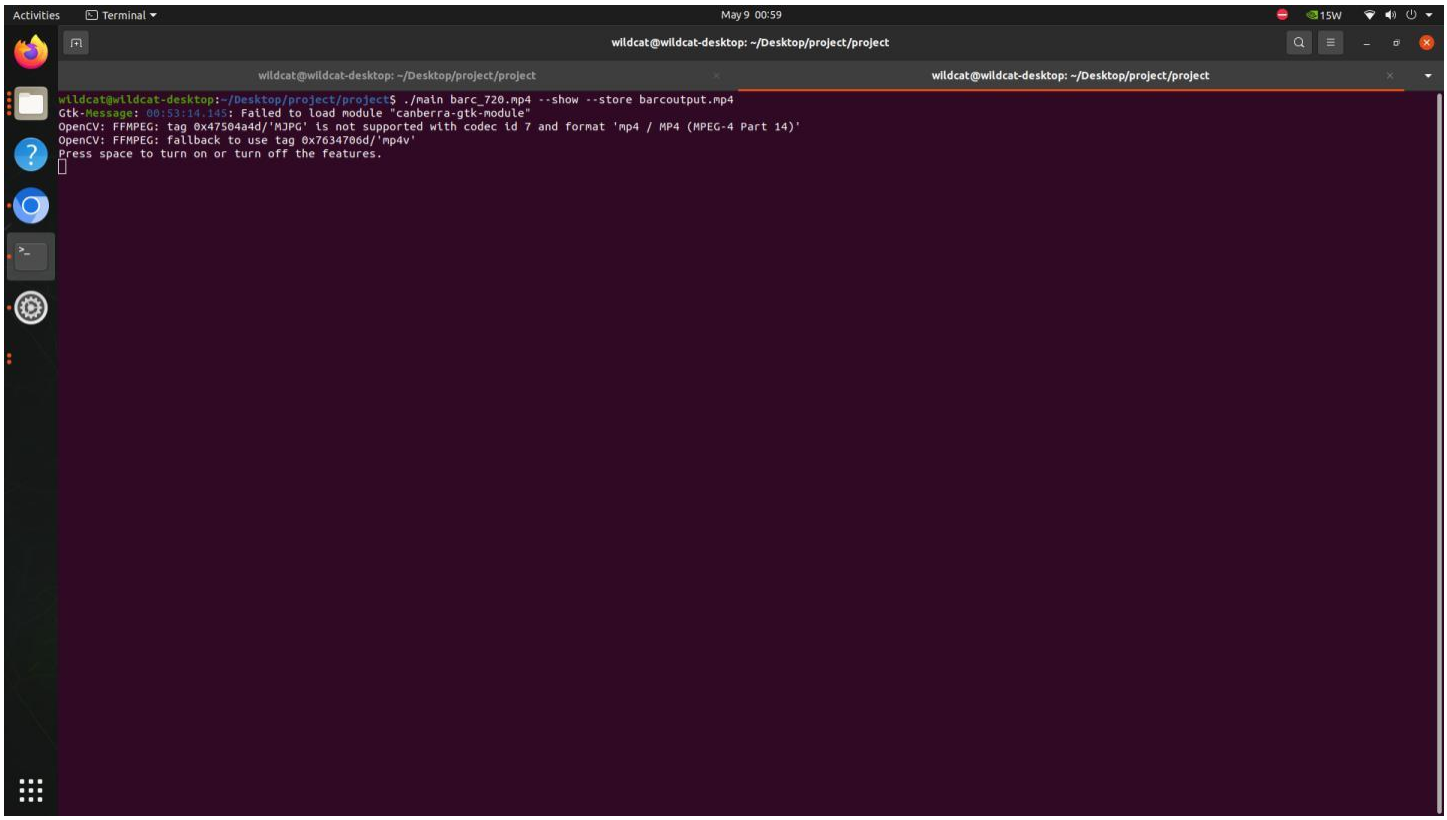
f) Integration and Real-time Processing:

- **Main Loop:** The main function orchestrates the entire process by reading frames from the input video, applying the various detection methods, and drawing corresponding annotations on each frame.
- **User Interaction:** The program allows the user to toggle feature display (e.g., lane detection, object detection) using the spacebar and provides real-time feedback by displaying the processed frames with annotations.
- **Video Output:** Optionally, the processed video can be stored if specifed via command line arguments.

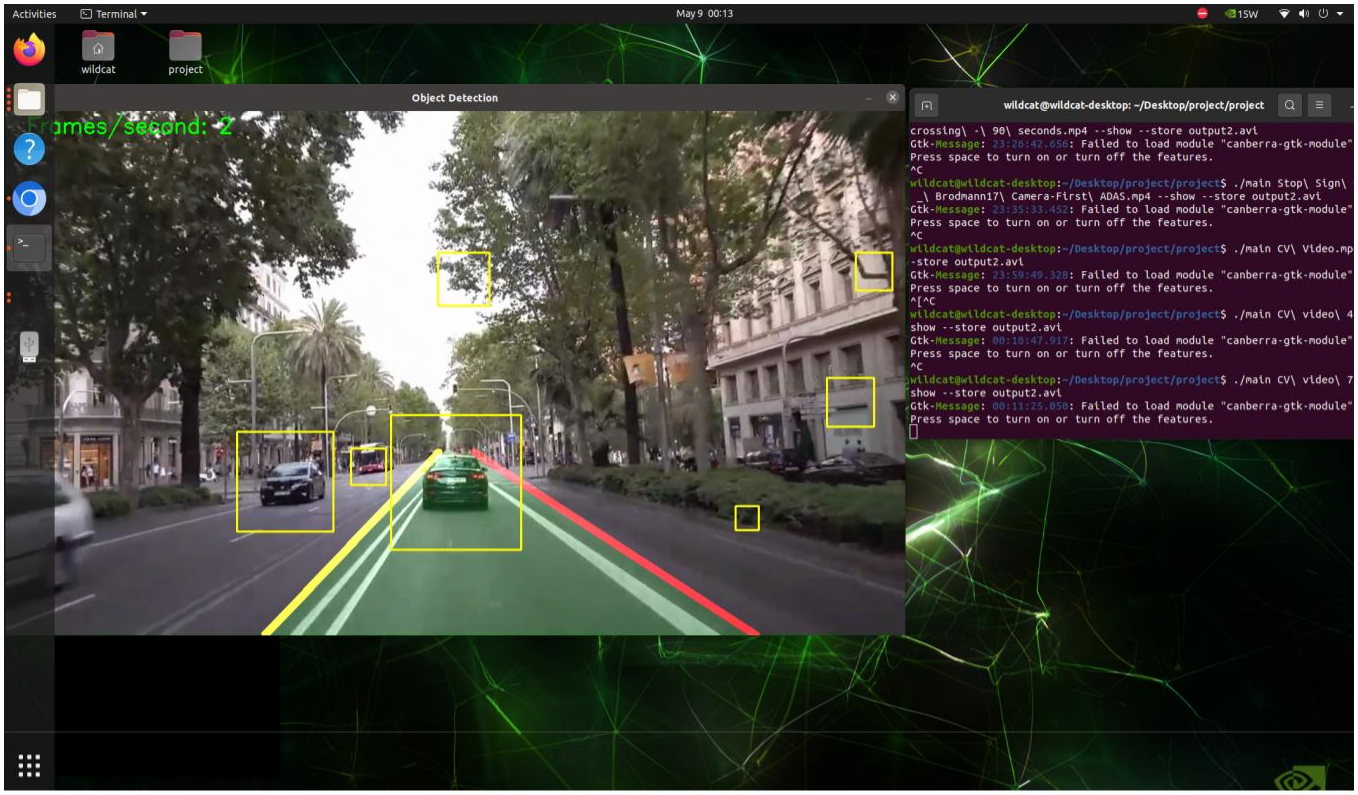
Make of the integrated features fle:

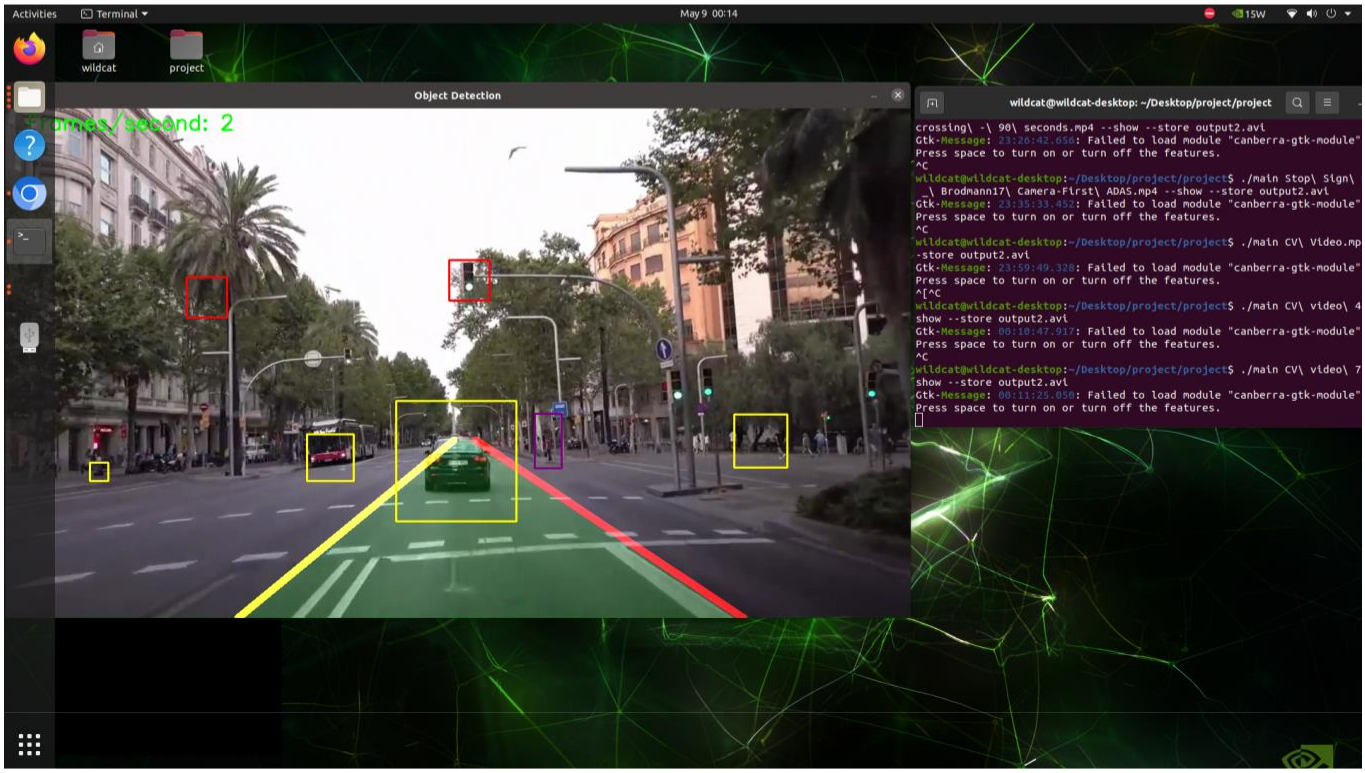


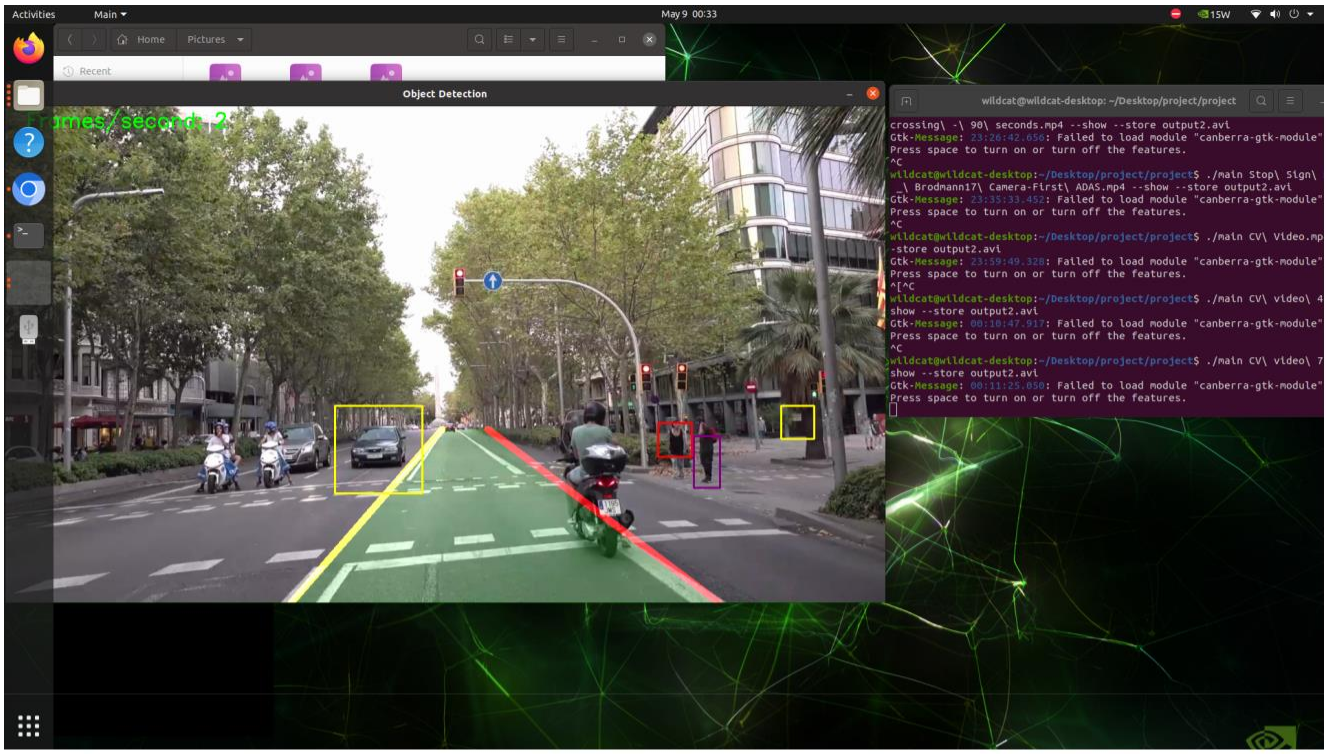
Run of the integrated features fle:

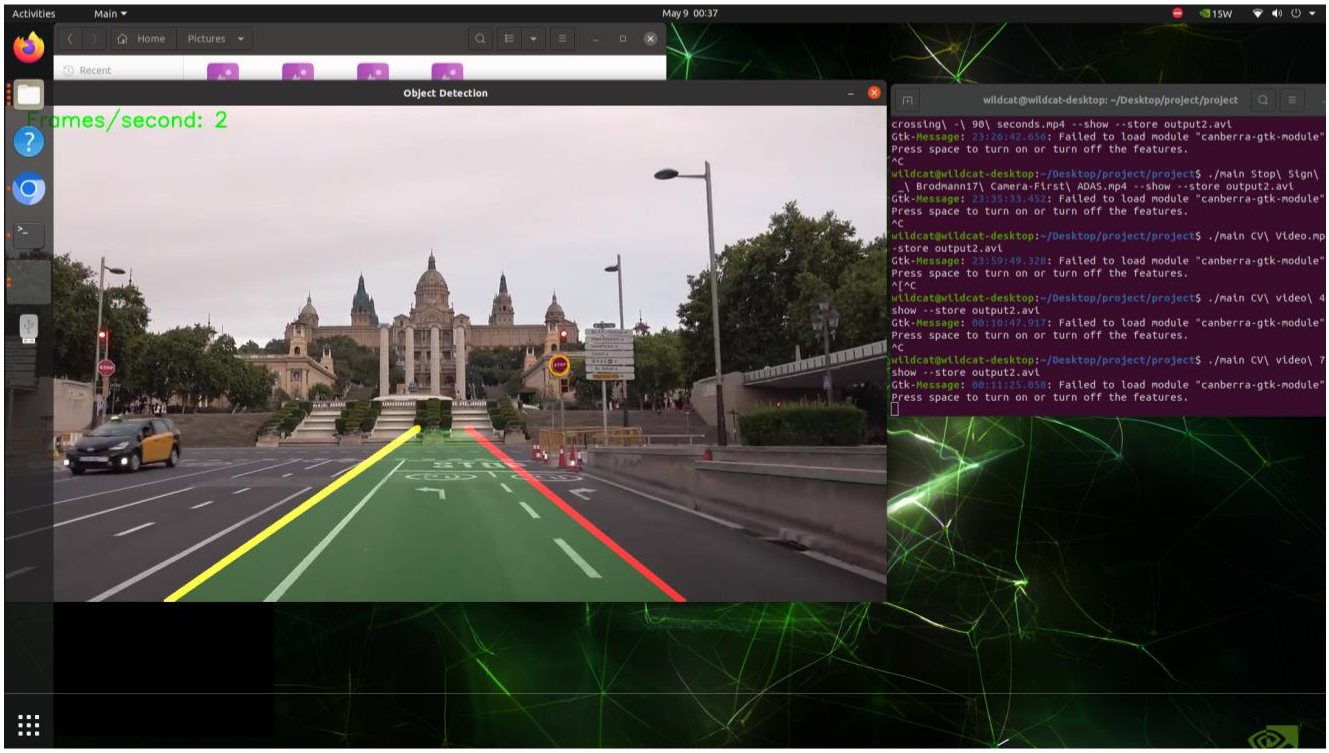


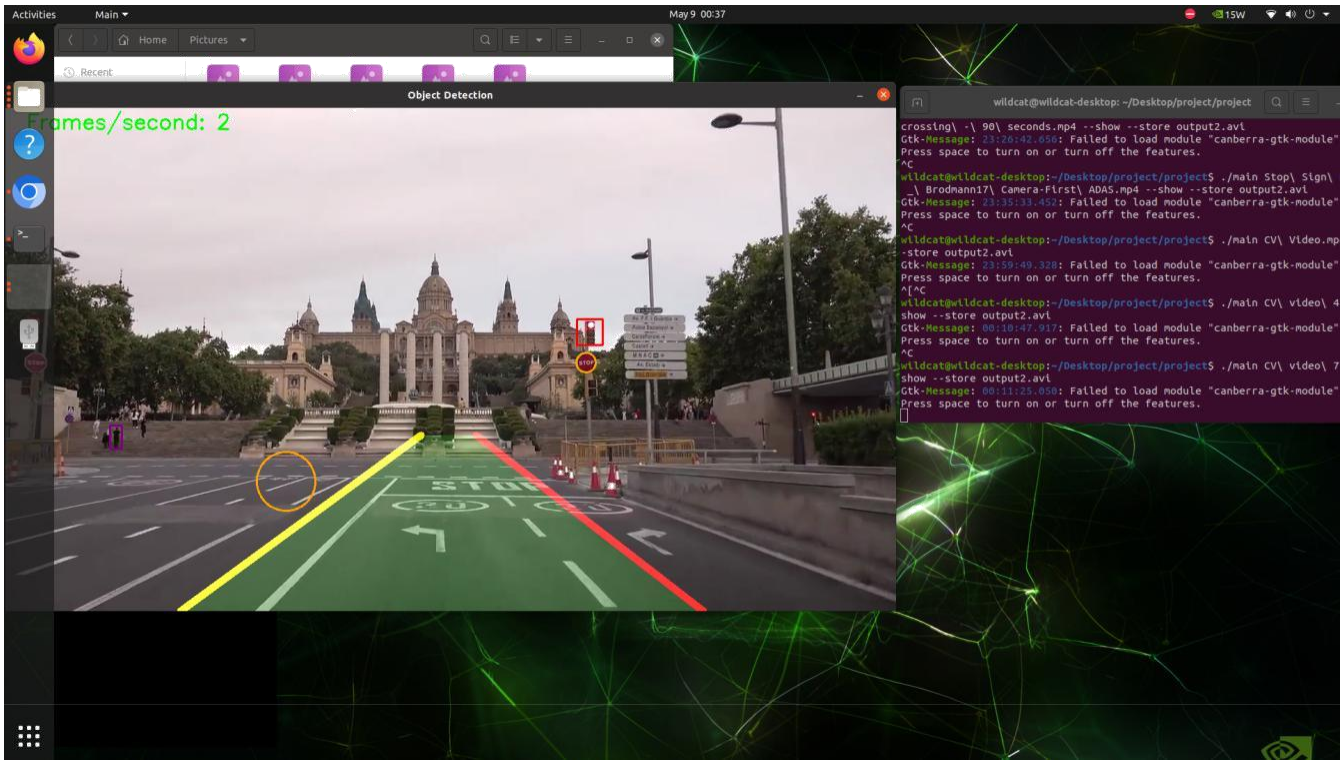
Output showing all features:






Activities
Main
May 9 00:33
Object Detection
Frames/second: 2
wildcat@wildcat-desktop: ~/Desktop/project/project
crossing\ -\ 90\ seconds.mp4 --show --store output2.avi
Gtk-Message: 23:26:42.656: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.
wildcat@wildcat-desktop:~/Desktop/project/project$ ./main Stop\ Sign\
_\ Brodmann17\ Camera-First\ ADAS.mp4 --show --store output2.avi
Gtk-Message: 23:35:33.452: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.
wildcat@wildcat-desktop:~/Desktop/project/project$ ./main CV\ Video.mp
-store output2.avi
Gtk-Message: 23:59:49.328: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.
wildcat@wildcat-desktop:~/Desktop/project/project$ ./main CV\ video\ 4
show --store output2.avi
Gtk-Message: 00:10:47.917: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.
wildcat@wildcat-desktop:~/Desktop/project/project$ ./main CV\ video\ 7
show --store output2.avi
Gtk-Message: 00:11:25.850: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.


Activities
Main
May 9 00:37
Object Detection
Frames/second: 2
wildcat@wildcat-desktop: ~/Desktop/project/project
crossing\ -\ 90\ seconds.mp4 --show --store output2.avi
Gtk-Message: 23:26:42.656: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.
wildcat@wildcat-desktop:~/Desktop/project/project$ ./main Stop\ Sign\
_\ Brodmann17\ Camera-First\ ADAS.mp4 --show --store output2.avi
Gtk-Message: 23:35:33.452: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.
wildcat@wildcat-desktop:~/Desktop/project/project$ ./main CV\ Video.mp
-store output2.avi
Gtk-Message: 23:59:49.328: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.
wildcat@wildcat-desktop:~/Desktop/project/project$ ./main CV\ video\ 4
show --store output2.avi
Gtk-Message: 00:10:47.917: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.
wildcat@wildcat-desktop:~/Desktop/project/project$ ./main CV\ video\ 7
show --store output2.avi
Gtk-Message: 00:11:25.850: Failed to load module "canberra-gtk-module"
Press space to turn on or turn off the features.



Output Video Link:

https://drive.google.com/file/d/1F6XCW8ejpDvji6DAaEQPkPBcnGsQ6Xp5/view?usp=sharing

Conclusion

The integration of critical functionalities such as lane detection, pedestrian identifcation, stop sign recognition, trafc light detection, and vehicle tracking in the system establishes a comprehensive perception framework essential for autonomous vehicle navigation and safety. This multifaceted approach ensures precise identifcation of lane markings, proactive measures to avoid collisions with pedestrians, adherence to trafc regulations through stop signs and trafc light detection, and efcient vehicle tracking for trafc analysis and collision avoidance. Despite achieving processing speeds of 2-5 frames per second (fps), ongoing optimization eforts are directed towards algorithmic enhancements, parallel processing techniques, and hardware acceleration to further enhance system efciency and responsiveness.

In conclusion, the system represents a foundational framework for advanced autonomous driving systems, ofering a robust approach to perception and navigation. Continued refnement and optimization will ensure readiness for real-world deployment, fostering the realization of safe, efcient, and autonomous transportation solutions.

References

1. “An open source advanced driver assistance system (ADAS) that uses Jetson Nano as the hardware.” Available at: https://github.com/vietanhdev/open-adas
2. “ADAS Stereo Vision: Pioneering Depth Perception Beyond LiDAR.” Available at:https://learnopencv.com/adas-stereo-vision/
3. “Real-Time Lane Detection for Self-Driving Cars using OpenCV.” Available at:https://www.labellerr.com/blog/real-time-lane-detection-for-self-driving-cars-using-opencv/
4. “OpenCV modules.” Available at:https://docs.opencv.org/master/index.html
5. “OpenCV - Open Computer Vision Library.” Available at:https://opencv.org/
6. “GitHub - opencv/opencv: Open Source Computer Vision Library.” Available at:https://github.com/opencv/opencv
7. “CUDA C++ Programming Guide - NVIDIA Documentation Hub.” Available at:https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html
8. “CUDA C++ Best Practices Guide - NVIDIA Documentation Hub.” Available at:https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html
9. “Tutorial 02: CUDA in Actions - CUDA Tutorial.” Available at:https://cuda-tutorial.readthedocs.io/en/latest/tutorials/tutorial02/
10. “Advanced Driver Assistance System | Every ADAS Levels in the Car Explained.” Available at:https://www.youtube.com/watch?v=EiWl5PAtfYA
11. “ADAS/AV Global Development & Testing Data Garage with AWS and dSPACE.” Available at:https://www.youtube.com/watch?v=I-NagJdVXq4
12. “AVL Facts4You | ADAS Testing.” Available at: https://www.youtube.com/watch?v=YXJ_o_uF1mE

Appendices

---

1. The chart showing the TP,FP and Accuracy, Precision, Recall and F-1 Score:

Functionality	Actual Values	TP	TN	FP	FN	Accuracy	Recall	Precision	F1-Score
Pedestrians	15	7	1680	10	8	0.99	0.47	0.41	0.44
Vehicle detection	40	31	160	17	17	0.85	0.65	0.65	0.65
Stop Signal	14	9	3520	22	22	0.99	0.29	0.29	0.29
Stop Signs	1	1	240	9	9	0.93	0.10	0.10	0.10
Lane Detection	3	3	560	2	2	0.99	0.60	0.60	0.60