ECEN 5623
REAL TIME EMBEDDED SYSTEMS
FINAL REPORT

REAL TIME LANE DETECTION AND BLIND SPOT MONITORING FOR SEMI AUTONOMOUS VEHICLE SAFETY

SUBMITTED BY:

MALOLA SIMMAN
SHRINITHI VENKATESAN
RAGHU SAI PHANI SRIRAJ VEMPARALA

MAY 7, 2023

UNIVERSITY OF COLORADO BOULDER

1.INTRODUCTION:

Blind spots in cars are areas around the vehicle that are not visible to the driver, either directly or through the mirrors. These blind spots can be a significant hazard for drivers, especially when changing lanes or making turns. According to the National Highway Traffic Safety Administration (NHTSA), over 800,000 accidents occur every year due to blind spot-related issues.

Blind spots can be particularly dangerous for larger vehicles such as trucks, SUVs, and vans, as they have bigger blind spots compared to smaller cars. Motorcycles are also at a higher risk of accidents caused by blind spots, as they are smaller and can easily be obscured from the driver's view.

The proposed solution aims to develop a real-time embedded system that will help drivers to avoid lane departure and blind spot-related accidents. The system will consist of sensors and algorithms that will detect the lane and the presence of vehicles in blind spots. The system will provide both visual and audible alerts to the driver if the driver is about to leave the lane or if there is a vehicle in the blind spot.

Our project is “Real-time Lane Detection and Blind spot monitoring for Enhanced Semi-Autonomous Vehicle Safety” helps solving this issue by development of blind spot detection algorithms using ultrasonic sensors and also detection of lanes using the Logitech C270 camera and the OpenCV libraries to detect dashed and dotted lanes and warns the driver.

2.CAPABILITY REQUIREMENTS:

Lane detection: The system must be capable of consistently and accurately detecting both continuous and dotted lines on the road. The lane detecting method ought to be tuned for minimum computing overhead and real-time processing.

Blind-spot detection: The system must be capable of reliably and correctly detecting cars in the vehicle's blind spot. The blind-spot detection method ought to be tuned for quick calculation and minimal computational overhead.

Alarm system: When a lane departure or a vehicle in the blind spot is identified, the system must be able to promptly and effectively inform the driver via an alarm system. In the event that the alarm system fails, the system should also contain a fail-safe component.

Real-time processing: Within a specified deadline, the system must be able to process the discovered data in real-time. In order to reduce processing time, the system should be built with a low-latency architecture, utilizing efficient algorithms and hardware acceleration as appropriate. For safety and dependability, the system should also include a way to deal with unforeseen delays or processing errors.

PERFORMANCE REQUIREMENTS:

The **lane detection** algorithm shall start detecting the lane for dotted and continuous lines within a maximum processing time of **125ms**.

+ Continuous lines - Flag set and processed in alarm thread.
+ Dotted lines - No warning.

The **ultrasonic sensors** shall start detecting for obstacles in the threshold range, which shall process within **20.7ms**.

+ Vehicles in blind spot - Flag set and processed in alarm thread.
+ No vehicles in blind spot - No warning.

The **alarm system** shall alert the user based on the response from the flags, which shall be processed in less than <**1ms**.

The overall system shall finish computations by a deadline of **300ms** to ensure that the system can operate in real-time and respond to lane departure and blind-spot detection quickly.

3. REAL-TIME REQUIREMENTS:

SERVICE 1: Alarm System:
SERVICE 2: Blind Spot Detection:
SERVICE 3: Lane Assistance.

Thread	WCET(ms)	Deadline / Period(ms)	Priority
Lane Detection thread	**125**	**300**	**3**
Ultrasonic Sensor thread	**20**	**60**	**2**
Alarm thread	**<1**	**5**	**1**

Table 1.1 Real time requirements

The given system has three main components: Alarm System, Blind Spot Detection, and Lane Assistance.

SERVICE 1: Alarm System:

The alarm system has a computation time (Ci) of less than 1ms, and its deadline (Di) and period (Ti) are both 5ms. This indicates that the alarm system needs to perform its computation and respond within 1ms to ensure that it meets the 5ms deadline for the next period. This thread takes very less time which will be in microseconds or nanoseconds. The deadline is set based on the assumption that the driver should react as fast as possible and hence we have chosen 5 milliseconds for a quick response.

SERVICE 2: Blind Spot Detection:

The blind spot detection system has a Ci of 20.7ms, and its Di and Ti are both 60ms. The Ci was calculated based on the time required for the ultrasonic sensor to generate eight clocks at a frequency of 40KHz, trigger time of 10us, and the time taken for the sound to bounce back from the object (3.5/170). The ultrasonic sensor data sheet suggests that the system needs to update every 60ms to detect objects in blind spots.

When the distance it can measure is at its greatest, this thread's WCET is highest. The sensor's measurement precision was up to 350 meters, hence the WCET was determined to be 20.7 milliseconds. According to the data sheet for the HCSR04 ultrasonic sensor, it is ideal to transmit the ultrasound signal every 60 ms, thus we've set it as our deadline.

SERVICE 3: Lane Assistance.

The Lane Assistance component has a Ci of 125ms, and a Di and Ti of 300ms. This component detects lane departures using a camera and processes the data to determine if the vehicle has crossed the lane markings. The Lane Assistance component must be able to process the video data and detect lane departures in real-time, with a margin for error, and alert the driver via the Alarm System component.

The lane detection thread has the following functionalities implemented.

1. Read Frame
2. Remove Noise
3. Detecting Edges using thresholding
4. Obtaining the Region of Interest
5. Hough line detection
6. Plotting the lane

Initially we have only taken Canny edge detection and Hough line transform into consideration for lane detection and estimated the WCET to be 134ms but as we are working on it, we have used thresholding technique and we have obtained the WCET as 125ms. The deadline of 300ms is computed based on the total deadline of the system which is 300ms

In summary, the system must be able to meet the real-time requirements of all three components to ensure safe and reliable operation. The Blind Spot Detection and Lane Assistance components have relatively longer Ci and Di/Ti requirements, indicating that they may require more computational power and resources compared to the Alarm System component. The overall system design and selection of components, including the real-time operating system and scheduling policy, must be carefully considered to ensure efficient and reliable operation.

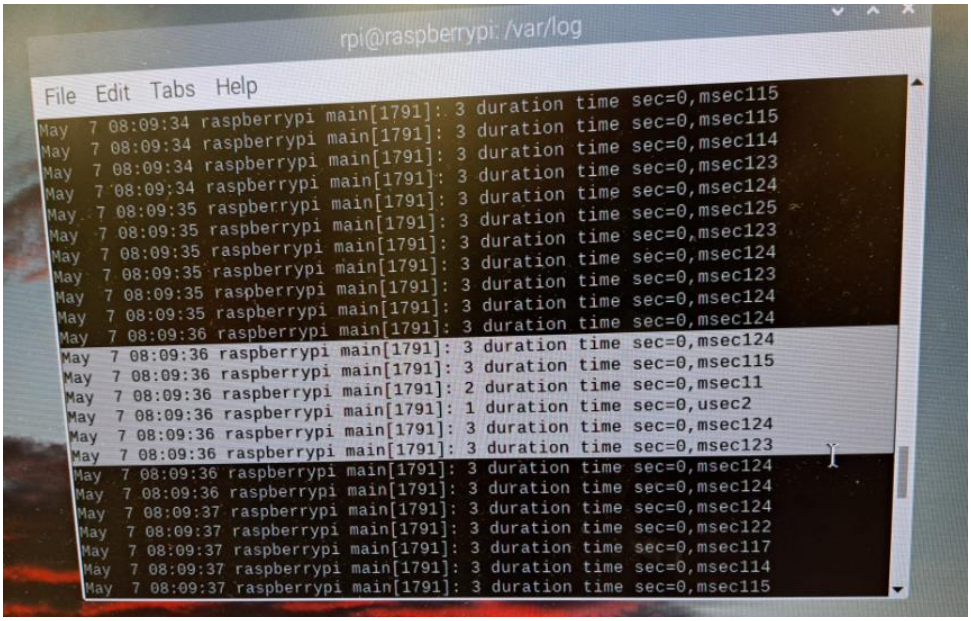


Fig 1.1 WCET Output

4. Functional Design Overview and Diagrams

Fig 1.2 Hardware Block Diagram

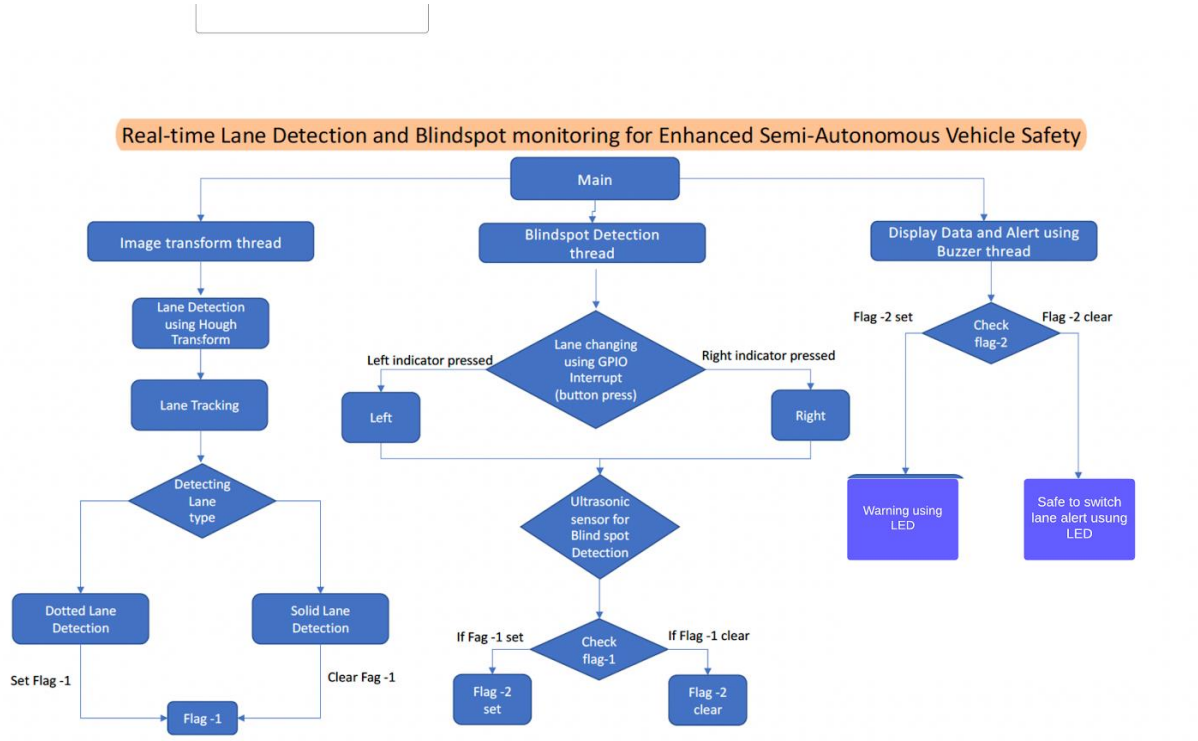


Fig 1.3 Software Flow chart

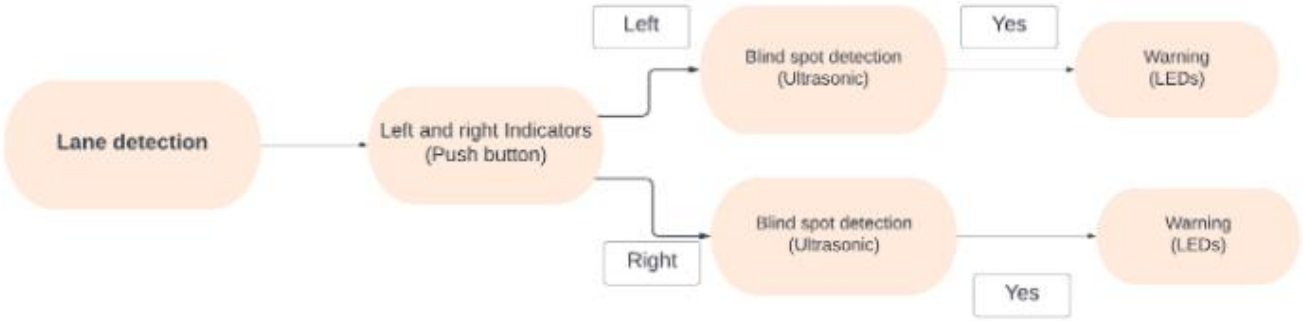


Fig 1.4 Data Flow chart

Lane Assistance:

Lane detection, which is used to recognize and follow the lanes on the road, is a crucial component of autonomous driving systems. The Hough transform is one of the most often used approaches for lane detection in this procedure, which makes use of computer vision algorithms. The well-known open-source computer vision package OpenCV may be used to put this method into practice.

A strong method for recognizing forms in a picture, such as lines, circles, or ellipses, is the Hough transform. The Hough transform is employed in lane detection to locate the lines that match the lane markers on the road. There are various steps in the procedure.

The first step in implementing lane detection using the Hough transform and OpenCV is to convert the input image to grayscale. The Hough transform works on grayscale images, so this step is necessary. Once

the image has been converted, an edge detection algorithm is applied to identify the edges of the lanes. There are various edge detection algorithms available, such as the Canny edge detector.

After the edges have been identified, the Hough transform is applied to the edge image to identify the lines that correspond to the lane markings. OpenCV provides a convenient function, HoughLines, for implementing the Hough transform. This function takes the edge image as input and outputs an array of lines that correspond to the lane markings.

Following the detection of the lines, the lane markings may be painted on the original picture. Although it is not required, this step might be useful for seeing the discovered lanes. The discovered lines may contain noise or be erroneous, depending on the original image's quality and the edge detection technique employed. The accuracy of the lane detection may be increased using a variety of filtering and refining approaches, such as thresholding, morphological procedures, or curve fitting.

Overall, the Hough transform and OpenCV are effective tools for lane detection, allowing for the development of reliable systems. It is feasible to create a dependable lane detecting system that may be employed in autonomous driving applications by following the above-mentioned methods.

Blind Spot Detection:

For drivers who might not have a good vision of their surroundings while driving, blind spot detection is a crucial safety element. Due to its dependability and accuracy in detecting things around the vehicle, ultrasonic sensors are frequently utilized in blind spot detection systems.

It's crucial to comprehend how the system operates in order to comprehend how blind spot identification utilizing ultrasonic sensors functions. The vehicle's sides and back normally have sensors fitted, and some systems also include sensors in the front. These sensors produce ultrasonic waves that are reflected off nearby objects by the vehicle. The distance between the vehicle and the sensor is determined by timing when the waves return to the sensor.

The technology warns the driver visually or audibly if an item is found in the car's blind zone. Depending on how the system is built, these signals may take the shape of a warning light or an audible beep. To aid the driver in avoiding crashes, certain cutting-edge systems can also display further details like the separation between the car and the object.

The use of ultrasonic sensors for blind spot identification can enhance driver safety and reduce collisions. It gives drivers who might be prone to missing things in their blind zones an added layer of awareness. Drivers may experience greater comfort and security on the road thanks to this technology, which eventually lowers the danger of crashes and accidents.

GPIO Interrupt Switch:

The GPIO interrupt switch is a powerful feature of the Raspberry Pi 3B that allows drivers to indicate their intention to change lanes. This is achieved by connecting an interrupt switch to the board's GPIO pins, which can be used to detect user input or monitor external sensors.

The switch is connected to a specific GPIO pin and configured to generate an interrupt when its state changes. This event can then trigger a function or program to respond accordingly, such as sending a signal to the vehicle's lane departure warning system or activating the turn signal.

By using the GPIO interrupt switch to indicate a lane change, drivers can communicate their intentions

more effectively and potentially prevent accidents caused by sudden lane changes. Overall, the GPIO interrupt switch is a versatile and valuable tool for enhancing driver safety and improving the functionality of a vehicle.

Synchronization:

The software has three threads, which are in sync with one another.

Without any semaphore, thread 1, lane detection runs constantly and is given the lowest priority.

According to the Rate Monotonic Policy, Thread 3 with an LED triggering mechanism is given the greatest priority, followed by Thread 2 which gets its value from an ultrasonic sensor.

Semaphores are posted in the following way to keep to the timetable.

To ensure proper scheduling, semaphores are used in the following manner: when the push button is pressed, a flag is set, which triggers the posting of semaphores. The program then goes to sleep while the semaphores run to completion. This cycle is repeated throughout the number of frames captured in the video.

Once the ultrasonic thread has determined the distance, it posts the semaphore for the alarm thread, which decides whether or not to sound the alarm. When the push button is depressed, a flag is set, which posts the semaphores for the ultrasonic thread. According to the cheddar analysis, the three thread sets may be scheduled.

Overall, the Rate Monotonic Policy is planned to be used since the three threads are independent and running based on First-In-First-Out (FIFO). The Worst Case Execution Time (WCET), Period (T), and Deadline (D) are calculated accordingly to be scheduled in the project. The three threads are schedulable with respect to the cheddar analysis made. Thus, the system is designed to meet the project's requirements.

By using proper synchronization techniques and scheduling policies, the program can effectively manage the execution of multiple threads and prevent any potential conflicts or race conditions. This ensures that the program runs smoothly and meets its intended purpose.

5. REAL TIME ANALYSIS AND DESIGN WITH TIMING DIAGRAMS:

A crucial component of our project, the overall timeframe was chosen after careful consideration and investigation. Our team did extensive study to determine how long it should take to make a lane change. The study's findings revealed that, depending on a number of variables including driving history, speed, and road conditions, this time ranges from 300 milliseconds to three seconds.

We made the decision to use a hard deadline approach and gave our application a deadline of 300 milliseconds in order to make sure that our system is effective and efficient. This deadline was selected taking into account both the system's overall needs and the least amount of time needed for a lane shift.

By establishing this deadline, we hoped to make sure that our system could identify lane departures and cars in the blind area in real-time and respond appropriately, which is crucial for the driver's safety and the protection of other road users. In order for our system to achieve the processing and response time criteria within the allotted time frame, we also had to optimize its design and execution.

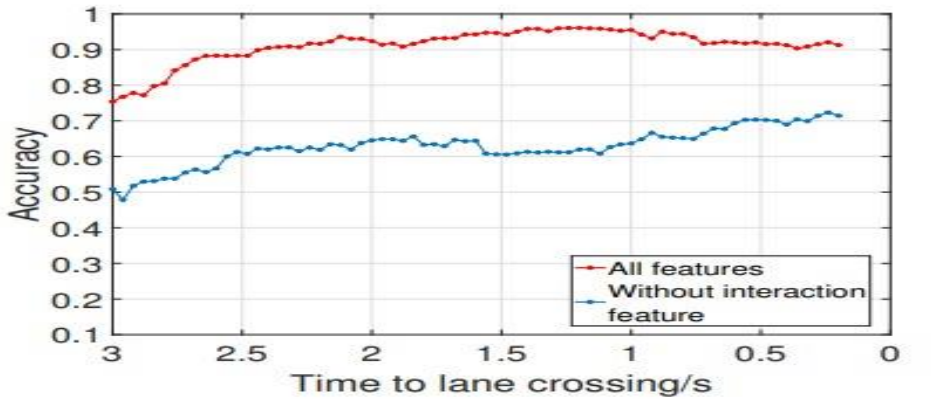


Fig 1.5 WCET from research

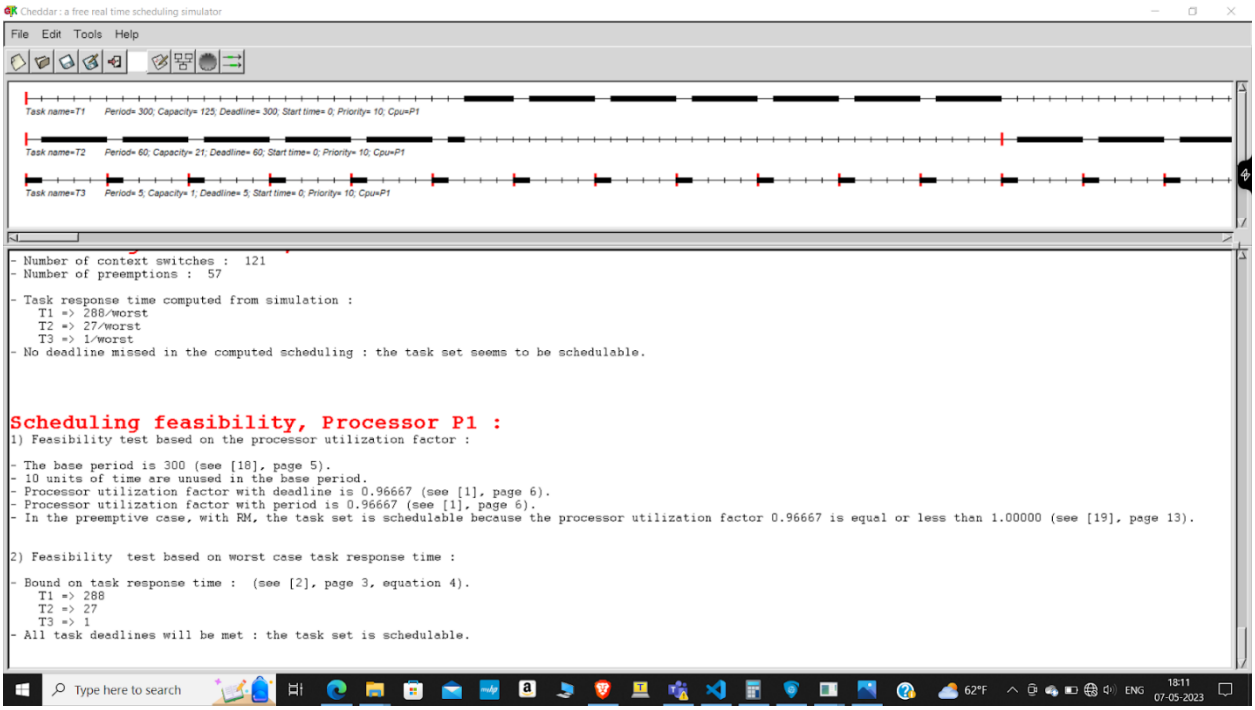


Fig 1.6 Feasibility and Cheddar Analysis

After conducting the cheddar analysis, we determined that the process utilization is 96.66%, which is less than 1. This means that the system is not feasible under the RM approach. However, when we applied the Rate Monotonic scheduling policy, we found that the system is schedulable.

SAFETY MARGIN:

Any real-time system should take a safety margin into account to make sure it can continue to function properly even in unforeseen situations. The safety margin is a portion of the overall processing time that is reserved for unanticipated occasions or additional processing requirements.

The system's utilization factor for this project is 96.66%, which indicates that the threads are using nearly all of the processing time that is available. A safety margin of 4% has been taken into account to guarantee that the system may be scheduled securely. This indicates that 4% of the overall processing time is reserved to deal with any unforeseen occurrences or additional processing requirements that could arise.

By establishing a safety margin, we can make sure that the system can work within its capabilities and that it can cope with any unforeseen occurrences that could arise while it is in use. This aids in avoiding system malfunctions, collisions, and other problems that can endanger the car or its occupants.

In conclusion, the safety margin is a vital factor to take into account in real-time systems and is essential to ensure that the system can function successfully and securely in all operating conditions. We can make sure the system can function within its capabilities and manage any unforeseen occurrences that may arise while operating by establishing a safety margin of 4%.

Feasibility tests:

Scheduling Point feasibility analysis:

The scheduling point feasibility was run from Assignment-2 and the system is feasible.
Though, the RM utilization greater than LUB value, Task sets are seems to be schedulable and certain scheduling conditions such as meeting their deadlines and not having release jitter, then there is a good chance that the tasks can be scheduled in a predictable and timely manner.

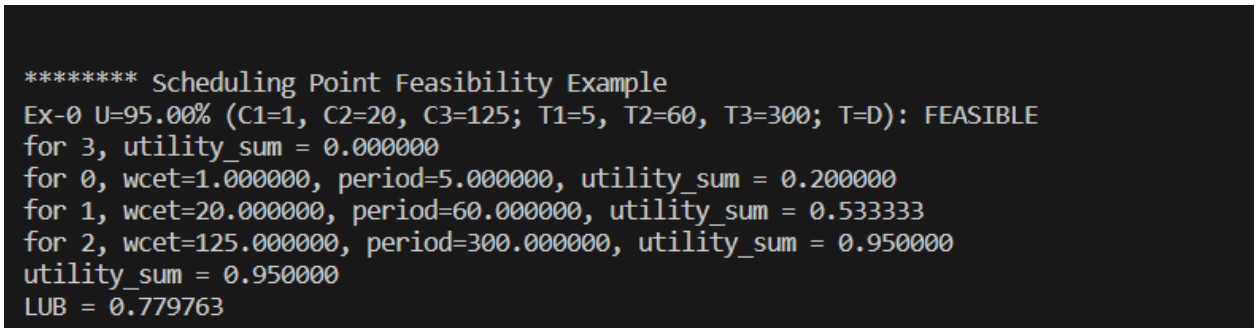


Fig 1.7 Scheduling Point feasibility analysis

Completion Point feasibility test:

The completion point feasibility was run from Assignment-2 and the system is feasible.
The completion test indicates that all tasks will complete their execution before their respective deadlines, then the set of tasks can be considered schedulable by the RM algorithm. Therefore, while the completion test is a useful tool for evaluating the schedulability of a set of tasks under the RM algorithm, it should be used in conjunction with other schedulability tests and techniques to ensure that the system is predictable and meets its timing requirements.

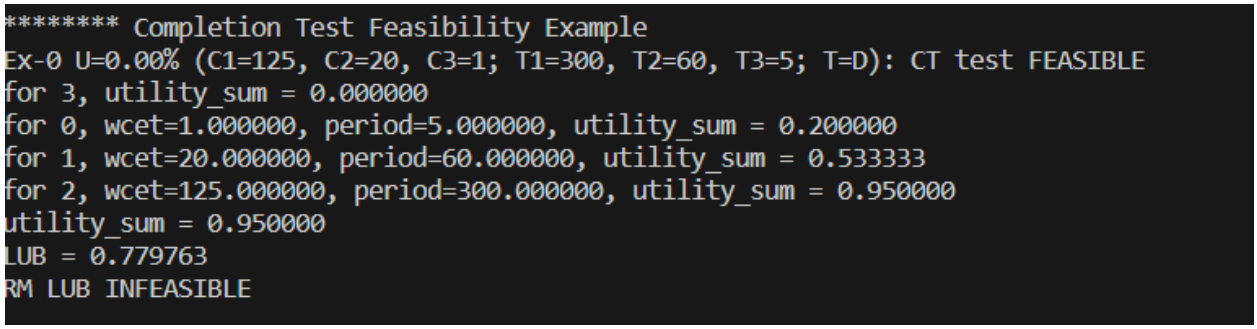


Fig 1.8 Completion Point feasibility analysis

6. PROOF OF CONCEPT WITH EXAMPLES AND OUTPUT:

We have used a video from online and ran it in our code. We are able to demonstrate from the video that we are able to detect dotted and dashed lanes. We have used the following techniques for Lane Detection.

1. **Read Frame**: We are able to read each frame of data from the video.
2. **Noise Removal:** We have removed the unwanted noise using Gaussian Blur technique. The following image shows the result of Gaussian Blur.
   The function used is cv::GaussianBlur(in, out, cv::Size(3, 3), 0, 0); It removes unnecessary noise by using standard deviation.

Fig 1.9 Noise Removal

3. **Detecting Edges using thresholding:**
   This technique detects the lane from the image by using a thresholding technique which sets a threshold value for detecting the lanes.

Fig 1.10 Detecting Edges using thresholding

4. **Masking:** Masking is performed to obtain the lane which we are interested in. We have achieved this by selecting the region of interest which is done by using the x and y coordinates of the array and taking only the points in the array which match with the right lane in our case.

Fig 1.11 Masking

5. **Lane Detection:** The Lane detection is performed by using Hough Lines which detects the lane. It is very important to provide the parameters in the function which is very crucial to detect Dashed and Dotted lanes. The function used is HoughLinesP(in, lines, 1, CV_PI / 180, 20, 20,30);

6. **Dash and Dotted Detection:** This was achieved by segregating the left and right lane by using ROI. Once the data is segregated we are using push_back method and pushing the coordinates into an array and using the normalization and distinguish the lane type.

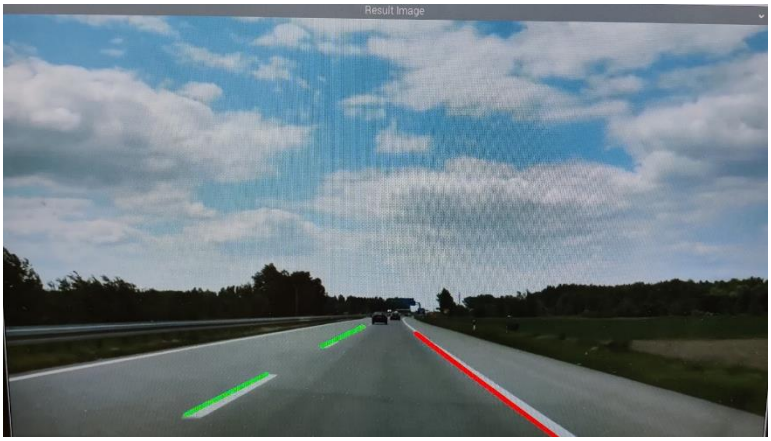


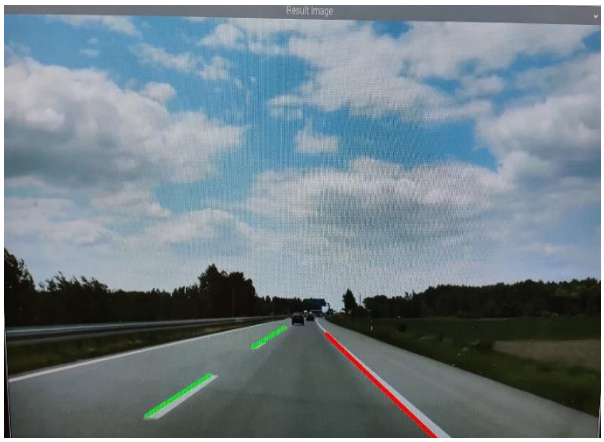
Fig 1.12 Dash and Dotted Detection

Blind spot detection:

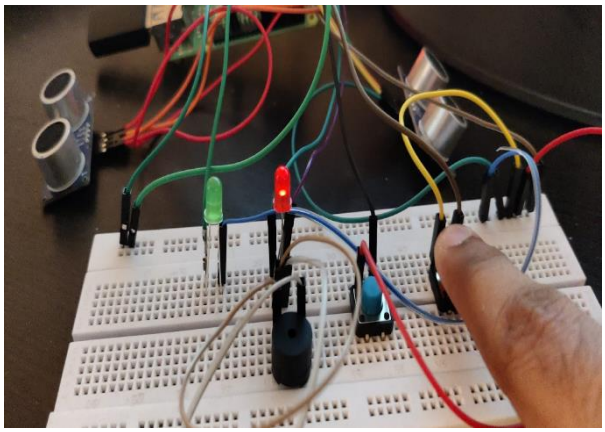
Blind spot detection is performed by using ultrasonic sensors. In the car we can place this sensor in between the doors. In real time the width of a single lane is 3.5 meters to 4 meters so our sensor should detect up to 3.5 meters including the car width and the sensor can achieve it.

For demonstration purposes we have taken the 3.5 meters length detection range as 30 cm . If a car is present in the 30 cm range then the red LED is triggered. If there is no blind spot then the Green LED is turned ON mentioning it is safe to change the lane. The following are the results obtained.

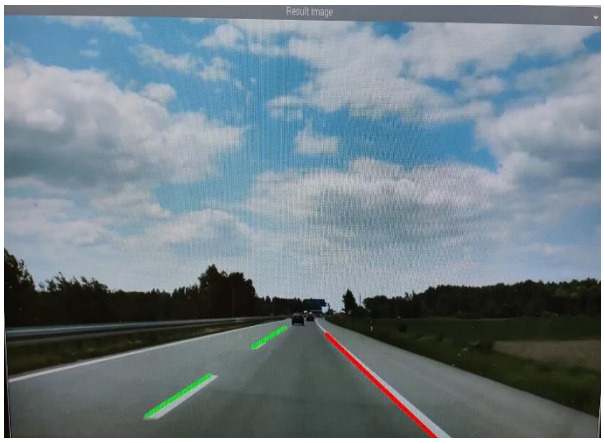
The following are the results obtained.



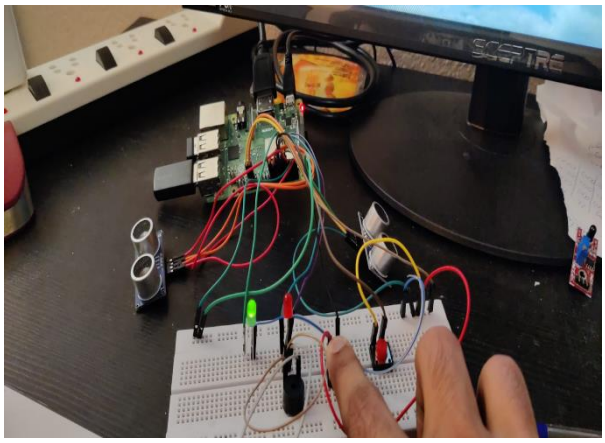
Solid Lane detected on the right



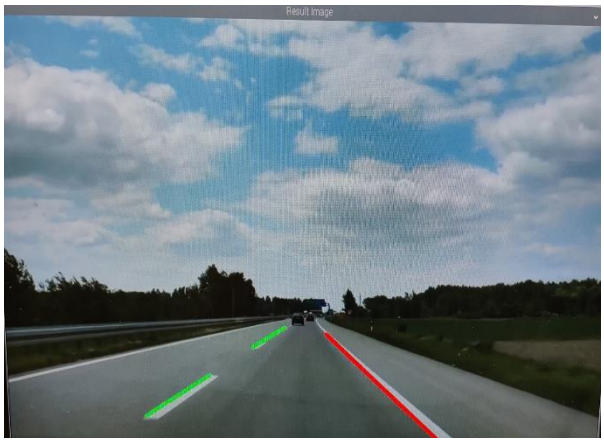
RED led is triggered when the right lane change is requested using the push button



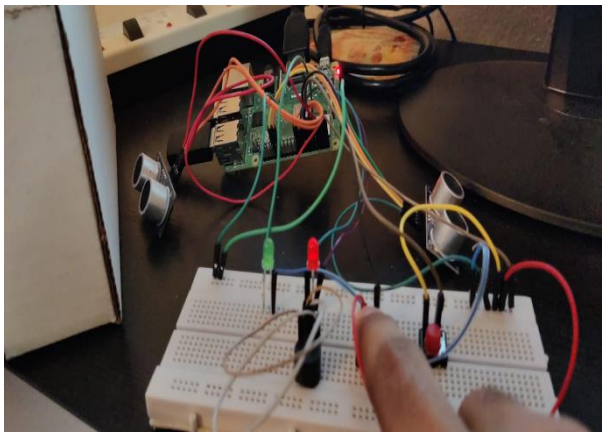
Dotted Lane is detected on the left



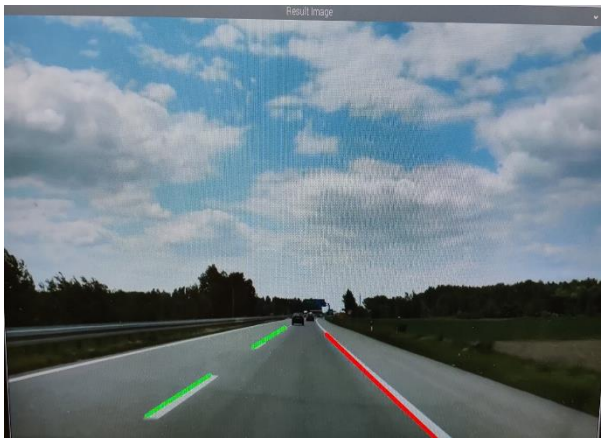
GREEN led is triggered when the dashed lane is detected and there is no object in the blind spot



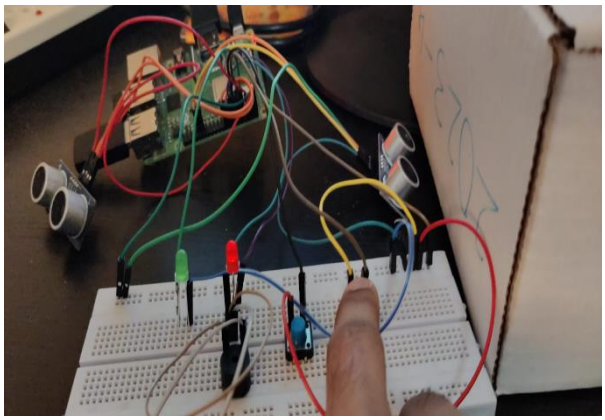
Dotted Lane is detected on the left



RED led is triggered since there is an object detected in the blind spot on the left



Solid Lane is detected on the right



Red light is triggered; it is a solid lane and an object in the blind spot. Object is depicted in the above snapshot using white box.

VERIFICATION PLAN:

Requirements Review:

The system requirements for real-time lane detection and blind spot monitoring, including the needed accuracy and precision of the sensors and the minimal reaction time for warnings, were examined as part of the requirements evaluation. The review procedure made sure that the specifications are precise, unambiguous, and practical within the parameters of the project. It made sure that the requirements are testable, which means that a test plan could be created to validate each need.

Design Evaluation:

The software architecture, lane detection and blind spot monitoring algorithms, and the implementation of the hardware components were all examined as part of the design review. The review process made sure the design is suitable for fulfilling the needs of the system and is scalable, maintainable, and reliable. It also ensured that the code follows coding standards and best practices, and that there are no bugs or errors that could cause system failure or unexpected behaviour.

Code review:
After writing each part of the code, it was reviewed to ensure that the code followed neat organization, was free of bugs, had effectively moved between different threads and properly synchronized data flow as it was supposed to.

Unit testing and Integration testing:
Initially each part of the code was separately created and tested for unit tests. Ultrasonic was the first thread we tested, following the push button interrupts and then followed by the lane detection thread and finally our alert system. Overall system was integrated together and was synchronized. Scheduling was implemented assigning priorities for the threads and overall system was tested properly after integration for expected code and design flow.

System Testing:
System testing is the process of testing the entire system as a whole to ensure that it functions correctly in all operating scenarios and meets all the test cases. This includes testing for functionality, reliability, usability, and performance. The system was tested multiple times to ensure that it was not a one time show and corner cases were set to test all the possible ways to validate our proof of concept.

Components and tests performed:

Ultrasonic sensors: The ultrasonic sensors were tested for their minimum and maximum distance detection capabilities, threshold detection, and response time. Multiple tests were conducted to ensure obstacle detection at any point within the measurable distance range, and different types of obstacles were used to verify the sensor's accuracy and reliability.

Push Buttons: The push buttons were tested for proper functioning, including multiple interrupts and fast response time. Their connection to the LEDs was also verified to ensure that the LEDs were properly triggered when an alert was generated.

LEDs: The LEDs were tested for proper functioning, ensuring that they produced the correct output when an alert was generated and responded quickly to the signal.

Camera and Image Processing: The algorithm for image processing was tested to ensure its proper functioning. The camera was verified to be powered properly and integrated seamlessly with the board. The frame rate was measured, and frame processing within the specified timeline was confirmed. Image transformation procedures were also tested, and the quick response time of the camera and image processing system was verified.

Overall, a comprehensive testing approach was adopted to verify the proper functioning of all the components and to ensure that the system met all the specified requirements.

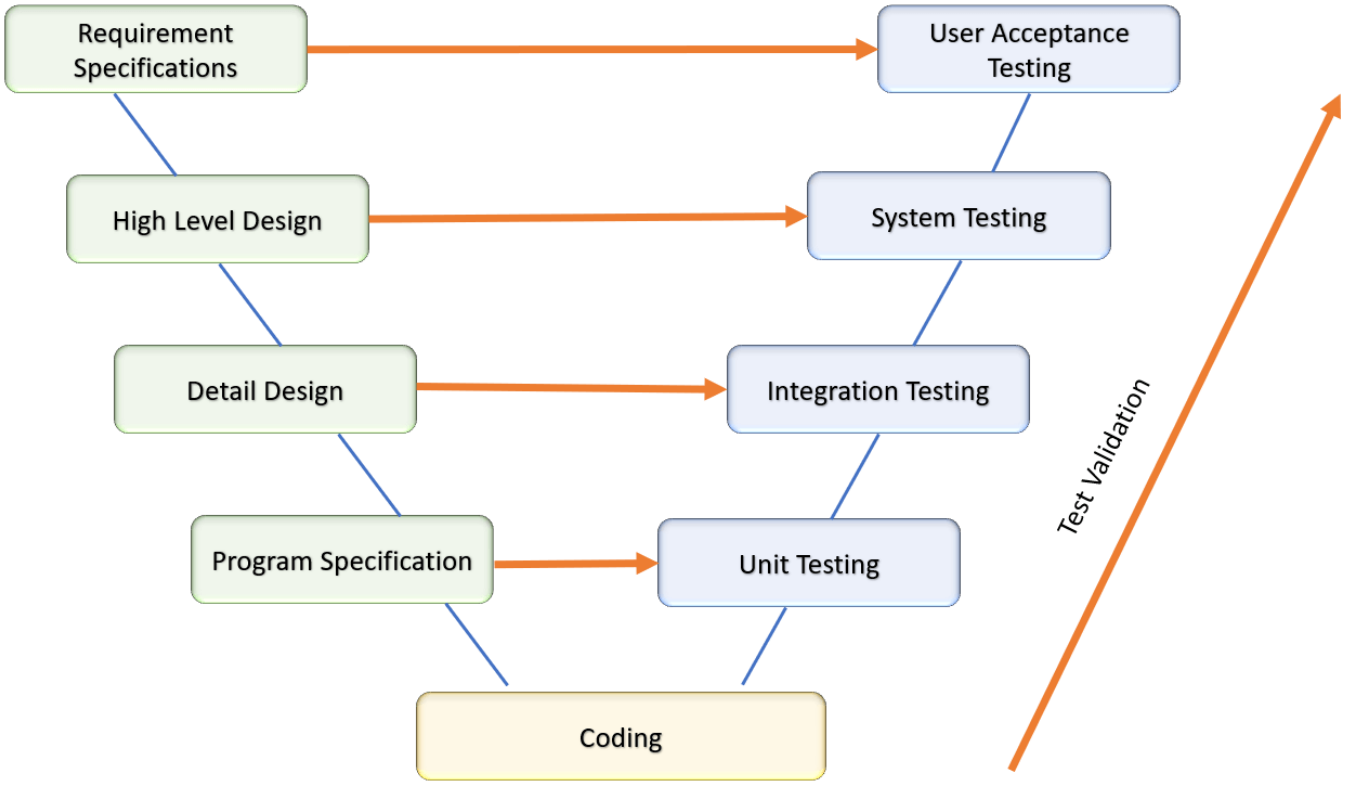


Fig 1.13 V MODEL TESTING PLAN

7. CONCLUSION:

The real-time lane identification and blind spot monitoring system for semi-autonomous vehicle safety, in conclusion, is a significant step towards improving the safety and dependability of autonomous cars. The real-time lane detection and blind spot monitoring system developed for this research makes use of computer vision algorithms and ultrasonic sensors.

Edge detection, the Hough transform, and color filtering are some of the techniques used by the lane identification algorithm to effectively recognize lane markers and give the driver feedback in real-time. The ability to maintain awareness of the vehicle's location on the road and respond appropriately when necessary are made possible by this characteristic, which is crucial in semi-autonomous cars.

On the other hand, the blind spot monitoring system makes use of ultrasonic sensors to find surrounding cars that the driver might not be able to see. In order to prevent crashes, the system warns the driver when an item is detected in the blind spot using visual or audio signals and offers an indicator of the distance between the vehicle and the identified object.

The project used a Rate Monotonic Policy for thread scheduling and a GPIO interrupt switch to signal the driver's desire to change lanes in order to guarantee the system runs smoothly and effectively. To alert the driver of possible threats like lane departure or an oncoming car in the blind zone, the system also has an alarm feature.

All things considered, this study effectively illustrates the possibility of real-time lane identification and blind spot monitoring in semi-autonomous cars. The technology improves the car's dependability and safety while giving the driver the information they need to make wise choices when they are behind the wheel. This technology has the potential to completely transform the automobile industry and pave the road for the mass adoption of autonomous cars with future research and improvement.

8. FORMAL REFERNCES:

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9385665

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8924448

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9860558

https://github.com/ChipHunter/LaneDetection_with_openCV/tree/master/Project1/src

Demo link:

https://www.youtube.com/watch?v=ujftCr-hPOU

Appendix:

LaneDetector.h

```
/*
 * file name    : lanedetector.h
 * Hardware     : Raspberry Pi 3B
 * project name : Real-time Lane Detection and Blindspot
                  monitoring for Enhanced Semi-Autonomous Vehicle Safety
 * author       : Malola Simman Srinivasan Kannan, Shrinithi Venkatesan, Sriraj Vemparala
 * Reference    : https://github.com/ChipHunter/LaneDetection_with_openCV/tree/master/Project1/src
 */
#include "opencv2/opencv.hpp"
#include <string>

class laneDetector {
public:
    laneDetector(std::string path);

     void Noise_reduction(cv::Mat& in, cv::Mat& out);
     void edgeDetection(cv::Mat& in, cv::Mat& out);
     void mask_requiredlane(cv::Mat& in, cv::Mat& out);
     void houghLines(cv::Mat& in, std::vector<cv::Vec4i>& lines, std::vector<cv::Vec4i>& solidlines,
std::vector<cv::Vec4i>& dottedlines);
     void readFrame(cv::Mat& frame);
     void plot(std::vector<cv::Vec4i>& lines,  std::vector<cv::Vec4i>& solidlines,
std::vector<cv::Vec4i>& dottedlines,cv::Mat& frame);

private:
    cv::VideoCapture _cap;

};
```

LaneDectector.c

```
/*
 * file name    : Lanedetector.cpp
 * Hardware     : Raspberry Pi 3B
 * project name : Real-time Lane Detection and Blindspot
                  monitoring for Enhanced Semi-Autonomous Vehicle Safety
 * author       : Malola Simman Srinivasan Kannan, Shrinithi Venkatesan, Sriraj Vemparala
 * Reference    :https://github.com/ChipHunter/LaneDetection_with_openCV/tree/master/Project1/src
 */


#include "laneDetector.h"
#include <vector>
#include <stdexcept>
#include <string>
#include <vector>
#include "opencv2/opencv.hpp"
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <string>
```

```
#include <vector>
#include "opencv2/opencv.hpp"
#include <semaphore.h>

extern int dotted;
extern int solid;
extern int left_flag;
extern int right_flag;
extern int thd1flag;
extern sem_t sem_blind_spot;

laneDetector::laneDetector(std::string path) {

    _cap.open(path);

    if (!_cap.isOpened())
        throw std::runtime_error("Problem openning the video!");
        }


void laneDetector:: Noise_reduction(cv::Mat& in, cv::Mat& out) {

cv::GaussianBlur(in, out, cv::Size(3, 3), 0, 0);
}

void laneDetector::mask_requiredlane(cv::Mat& in, cv::Mat& out) {


    cv::Mat mask = cv::Mat::zeros(in.size(), in.type());
    cv::Point pts[4] = {
        cv::Point(150, 720),
        cv::Point(530, 500),
        cv::Point(700, 500),
        cv::Point(1100, 720)
    };

    cv::fillConvexPoly(mask, pts, 4, cv::Scalar(255, 0, 0));

    cv::bitwise_and(in, mask, out);
}


void laneDetector::houghLines(cv::Mat& in, std::vector<cv::Vec4i>& lines, std::vector<cv::Vec4i>&
solidlines, std::vector<cv::Vec4i>& dottedlines) {

       HoughLinesP(in, lines, 1, CV_PI / 180, 20, 20,30);
    solidlines.clear();
    dottedlines.clear();
    int center = in.cols / 2;
    int min_line_length = 30;

    for (const cv::Vec4i& line : lines)
    {
        double slope = static_cast<double>(line[3] - line[1]) / static_cast<double>(line[2] - line[0]);
        double length = cv::norm(cv::Point(line[0], line[1]) - cv::Point(line[2], line[3]));
        double dashed_threshold = 2;
        if (length < min_line_length)
        {
            continue;
        }
        if (slope < 0 && line[2] < center && line[0] < center)
```

```
        {
             // Classify dashed or solid line based on length
            if (length < dashed_threshold * min_line_length) {
                dottedlines.push_back(line);
                dotted = 1;
                solid = 0;
            } else {
                solidlines.push_back(line);
                solid = 1;
                dotted = 0;
            }

        }
        else if (slope > 0 && line[2] > center && line[0] > center)
        {
            // Classify dashed or solid line based on length
            if (length < dashed_threshold * min_line_length) {
                dottedlines.push_back(line);
                dotted = 1;
                solid = 0;
                if(thd1flag ==1){
                sem_post(&sem_blind_spot);
                thd1flag=0;
                }
            } else {
                solidlines.push_back(line);
                solid = 1;
                dotted = 0;
                if(thd1flag ==1){
                sem_post(&sem_blind_spot);
                thd1flag=0;
                }
            }
        }

    }

}

void laneDetector::readFrame(cv::Mat& frame) {

   if (!_cap.read(frame))
        throw std::runtime_error("Problem reading a frame");

}

void laneDetector::plot(std::vector<cv::Vec4i>& lines, std::vector<cv::Vec4i>& solidlines,
std::vector<cv::Vec4i>& dottedlines,cv::Mat& frame) {


   for (size_t i = 0; i < solidlines.size(); i++)
        {
            cv::Vec4i l = solidlines[i];
            cv::line(frame, cv::Point(l[0], l[1]), cv::Point(l[2], l[3]), cv::Scalar(0, 0, 255), 5,
CV_AA);

        }
        for (size_t i = 0; i < dottedlines.size(); i++)
        {
            cv::Vec4i k = dottedlines[i];
```

```
            cv::line(frame, cv::Point(k[0], k[1]), cv::Point(k[2], k[3]), cv::Scalar(0, 255,0), 5,
CV_AA);

        }


    cv::imshow("Result Image", frame);

}

void laneDetector::edgeDetection(cv::Mat& in, cv::Mat& out) {

     cv::Mat kernel;
    cv::Point anchor;

    cv::cvtColor(in, out, cv::COLOR_RGB2GRAY);

    cv::threshold(out, out, 140, 255, cv::THRESH_BINARY);

    anchor = cv::Point(-1, -1);
    kernel = cv::Mat(1, 3, CV_32F);
    kernel.at<float>(0, 0) = -1;
    kernel.at<float>(0, 1) = 0;
    kernel.at<float>(0, 2) = 1;

    cv::filter2D(out, out, -1, kernel, anchor, 0, cv::BORDER_DEFAULT);
}
```

Main.c

```
/*
 * file name    : main.cpp
 * Hardware     : Raspberry Pi 3B
 * project name : Real-time Lane Detection and Blindspot
                  monitoring for Enhanced Semi-Autonomous Vehicle Safety
 * author       : Malola Simman Srinivasan Kannan, Shrinithi Venkatesan, Sriraj Vemparala
 * Reference    :https://github.com/ChipHunter/LaneDetection_with_openCV/tree/master/Project1/src

 */

#include <stdio.h>
#include <pigpio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <iostream>
#include <time.h>
#include <pthread.h>
#include <signal.h>
#include <sched.h>
#include <syslog.h>
#include <errno.h>
#include <semaphore.h>
#include <string>
#include <sys/sysinfo.h>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/opencv.hpp>
```

```
#include "laneDetector.h"
#include <memory>
#include <time.h>
#include <sys/time.h>
#include <sys/sysinfo.h>

#define BUTTON_PIN_1 17
#define BUTTON_PIN_2 15
#define BUZZER 18
#define LED 4
#define TRIG_PIN 23
#define ECHO_PIN 24
#define TRIG_PIN_US2 5
#define ECHO_PIN_US2 6
#define ERROR (-1)
#define OK (0)
#define NUM_THREADS (4)
#define NUM_CPUS (4)
#define NSEC_PER_SEC (1000000000)
#define NSEC_PER_MSEC (1000000)
#define NSEC_PER_MICROSEC (1000)
#define DELAY_TICKS (1)
#define ERROR (-1)
#define OK (0)
#define TIMER_INTERVAL 1000000 // Timer interval in microseconds


typedef struct
{
    int threadIdx; //thread index
} threadParams_t;

int numberofprocessors = 1;
struct timespec start_time = {0, 0};
struct timespec end_time = {0, 0};
struct timespec delta_time;


pthread_t thread_blind_spot, thread_alarm, thread_lanedetection;
threadParams_t threadParams_blind_spot, threadParams_alarm, threadParams_lanedetection;
struct sched_param rt_param_blind_spot, rt_param_alarm, rt_param_lanedetection;
pthread_attr_t thread_attr_blind_spot, thread_attr_alarm, thread_attr_lanedetection;

sem_t sem_blind_spot, sem_alarm, sem_img;

double deadline;
struct sched_param rt_attr_transform;
struct sched_param rt_attr_log_time;

pthread_attr_t rt_sched_attr_blind_spot;
pthread_attr_t rt_sched_attr_alarm;
pthread_attr_t rt_sched_attr_lanedetection;
int rt_max_prio, rt_min_prio;
int interval=1000;
pid_t mainpid;
int numberOfProcessors=NUM_CPUS;
int dotted=0;
int solid=0;
int flag2=0;
int button1_count = 0;
int button2_count =0;
```

```
int left_flag =0;
int right_flag =0;
int thd1flag=0;
unsigned timerHandle;


void button1_callback(int gpio, int level, uint32_t tick)
 {
    if (button1_count == 0) {

        thd1flag=1;
         left_flag =1;
         button1_count = 1;
       }
    else
    {
       button1_count = 0;
    }
}
void button2_callback(int gpio, int level, uint32_t tick) {
   if (button2_count == 0) {

       thd1flag=1;
         right_flag=1;
         button2_count = 1;
       }
    else
    {
       button2_count = 0;
    }
}

void gpio_init()
{
    gpioInitialise();
    gpioSetMode(TRIG_PIN, PI_OUTPUT);
    gpioSetMode(ECHO_PIN, PI_INPUT);

}


int delta_t(struct timespec *stop, struct timespec *start, struct timespec *delta_t)
{
  int dt_sec=stop->tv_sec - start->tv_sec;
  int dt_nsec=stop->tv_nsec - start->tv_nsec;


  // case 1 - less than a second of change
  if(dt_sec == 0)
  {

      if(dt_nsec >= 0 && dt_nsec < NSEC_PER_SEC)
      {
          delta_t->tv_sec = 0;
          delta_t->tv_nsec = dt_nsec;
      }

      else if(dt_nsec > NSEC_PER_SEC)
      {
          delta_t->tv_sec = 1;
          delta_t->tv_nsec = dt_nsec-NSEC_PER_SEC;
```

```
      }

      else // dt_nsec < 0 means stop is earlier than start
      {
         return(ERROR);
      }
  }

  // case 2 - more than a second of change, check for roll-over
  else if(dt_sec > 0)
  {

      if(dt_nsec >= 0 && dt_nsec < NSEC_PER_SEC)
      {
          delta_t->tv_sec = dt_sec;
          delta_t->tv_nsec = dt_nsec;
      }

      else if(dt_nsec > NSEC_PER_SEC)
      {
          delta_t->tv_sec = delta_t->tv_sec + 1;
          delta_t->tv_nsec = dt_nsec-NSEC_PER_SEC;
      }

      else // dt_nsec < 0 means roll over
      {
          delta_t->tv_sec = dt_sec-1;
          delta_t->tv_nsec = NSEC_PER_SEC + dt_nsec;
      }
  }

  return(OK);
}



void *alarm(void *)
{
    struct timespec start_time, end_time,delta_time;
    gpioSetMode(BUTTON_PIN_1, PI_INPUT);
    gpioSetPullUpDown(BUTTON_PIN_1, PI_PUD_UP);
    gpioSetMode(BUTTON_PIN_2, PI_INPUT);
    gpioSetPullUpDown(BUTTON_PIN_2, PI_PUD_UP);
    gpioSetMode(BUZZER, PI_OUTPUT);
    gpioSetMode(LED, PI_OUTPUT);
    gpioSetAlertFunc(BUTTON_PIN_1, button1_callback);
    gpioSetAlertFunc(BUTTON_PIN_2, button2_callback);

    while (1) {

       sem_wait(&sem_alarm);
        clock_gettime(CLOCK_REALTIME ,&start_time);
       int count = 0;
       if(flag2)
       {

           gpioWrite(LED, 1);
           clock_gettime(CLOCK_REALTIME ,&end_time);
            usleep(2000);
           gpioWrite(LED, 0);
           usleep(2000);
```

```
          right_flag=0;
          left_flag=0;

       }
       else
       {

           gpioWrite(BUZZER, 1);
           clock_gettime(CLOCK_REALTIME ,&end_time);
            usleep(2000);
            gpioWrite(BUZZER, 0);
            usleep(2000);


          right_flag=0;
          left_flag=0;

        }

      delta_t(&end_time,&start_time,&delta_time);
      syslog(LOG_CRIT,"1 duration time
sec=%d,usec%d",(delta_time.tv_sec),((delta_time.tv_nsec)/NSEC_PER_MICROSEC));
    }

    gpioTerminate();
    return 0;
}




void *blind_spot(void *)
{

   float distance;
    uint32_t start_time_us, end_time_us;

    struct timespec start_time, end_time,delta_time;

    while (1) {
        int count = 0;
        sem_wait(&sem_blind_spot);
        clock_gettime(CLOCK_REALTIME ,&start_time);
        // Send a 10 us pulse to the trigger pin
        if(left_flag)
        {
        gpioWrite(TRIG_PIN, 1);
        usleep(1000);
        gpioWrite(TRIG_PIN, 0);

        // Wait for the echo pin to go high
        while (gpioRead(ECHO_PIN) == 0);
        start_time_us = gpioTick();

        // Wait for the echo pin to go low
        while (gpioRead(ECHO_PIN) == 1);
        end_time_us = gpioTick();
        // Calculate the distance in cm
```

```
        distance = ((end_time_us - start_time_us) / 1000000.0) * 17150.0;
        if((dotted==1) && (distance>30))
        {
            flag2=1;
            sem_post(&sem_alarm);
        }
        else
        {
             flag2=0;
             sem_post(&sem_alarm);
        }

       }
       else if(right_flag)
       {
        gpioWrite(TRIG_PIN_US2, 1);
        usleep(1000);
        gpioWrite(TRIG_PIN_US2, 0);

        // Wait for the echo pin to go high
        while (gpioRead(ECHO_PIN_US2) == 0);
        start_time_us = gpioTick();

        // Wait for the echo pin to go low
        while (gpioRead(ECHO_PIN_US2) == 1);
        end_time_us = gpioTick();
        // Calculate the distance in cm
        distance = ((end_time_us - start_time_us) / 1000000.0) * 17150.0;
        if((dotted==1)&&(distance>30))
        {
            flag2=1;
            sem_post(&sem_alarm);
        }
        else
        {
             flag2=0;
             sem_post(&sem_alarm);
        }

       //|
        //printf("US2_distance=%.2f\n",distance);
      }
      clock_gettime(CLOCK_REALTIME ,&end_time);
      delta_t(&end_time,&start_time,&delta_time);
      syslog(LOG_CRIT,"2 duration time
sec=%d,msec%d",(delta_time.tv_sec),((delta_time.tv_nsec)/NSEC_PER_MSEC));

   }
    gpioTerminate();

    return 0;
}


void *image_processing(void *)
{
    struct timespec start_time, end_time,delta_time;

   cv::Mat frame, img_reduce_noise, img_edge_detect, img_mask;
    std::vector<cv::Vec4i> lines, solidlines, dottedlines;
```

```
    try {
    laneDetector *lane_detection;
 lane_detection = new laneDetector("/home/rpi/Desktop/theVideo.mp4");

        while (true) {
             clock_gettime(CLOCK_REALTIME ,&start_time);
            lane_detection->readFrame(frame);

            lane_detection->Noise_reduction(frame, img_reduce_noise);

            lane_detection->edgeDetection(img_reduce_noise, img_edge_detect);

            lane_detection->mask_requiredlane(img_edge_detect, img_mask);

            lane_detection->houghLines(img_mask, lines,solidlines, dottedlines);

            clock_gettime(CLOCK_REALTIME ,&end_time);
            delta_t(&end_time,&start_time,&delta_time);
            syslog(LOG_CRIT,"3 duration time
sec=%d,msec%d",(delta_time.tv_sec),((delta_time.tv_nsec)/NSEC_PER_MSEC));
            if (!lines.empty()) {

                lane_detection->plot(lines, solidlines, dottedlines,frame);

            }

            cv::waitKey(5);

        }
    } catch(std::exception& e) {

        std::cout << "some exception: " << e.what() << std::endl;

    } catch (...) {

        std::cout << "other exception: " << std::endl;

    }

return 0;
}

int main(int argc, char** argv)
{
    int rc;
     int i;
    cpu_set_t threadcpu;
    int coreid;
    gpio_init();

   numberOfProcessors = get_nprocs_conf();

    openlog(NULL, LOG_PID , LOG_USER);
    sem_init(&sem_blind_spot, 0, 0);
    sem_init(&sem_alarm, 0, 0);
    sem_init(&sem_img, 0, 0);


   mainpid=getpid(); // Stores the thread's Process ID in the `mainpid` variable

   rt_max_prio = sched_get_priority_max(SCHED_FIFO); // getting the maximum priority for the SCHED_FIFO
```

```
   rt_min_prio = sched_get_priority_min(SCHED_FIFO); // getting the minimum priority for the SCHED_FIFO

       CPU_ZERO(&threadcpu);
       coreid=i%numberOfProcessors;  // calculating the core id
       printf("Setting thread %d to core %d\n", i, coreid);
       CPU_SET(coreid, &threadcpu);

       rc=pthread_attr_init(&rt_sched_attr_lanedetection);                                    //
Initialize the attributes
       rc=pthread_attr_setinheritsched(&rt_sched_attr_lanedetection, PTHREAD_EXPLICIT_SCHED);  // Set the
inheritance attribute
       rc=pthread_attr_setschedpolicy(&rt_sched_attr_lanedetection, SCHED_FIFO);             // Set the
scheduling policy to MY_SCHEDULER
       rc=pthread_attr_setaffinity_np(&rt_sched_attr_lanedetection, sizeof(cpu_set_t), &threadcpu);
//sets the thread's affinity to a specific core

       rc=pthread_attr_init(&rt_sched_attr_blind_spot);                                      // Initialize
the attributes
       rc=pthread_attr_setinheritsched(&rt_sched_attr_blind_spot, PTHREAD_EXPLICIT_SCHED);  // Set the
inheritance attribute
       rc=pthread_attr_setschedpolicy(&rt_sched_attr_blind_spot, SCHED_FIFO);             // Set the
scheduling policy to MY_SCHEDULER
       rc=pthread_attr_setaffinity_np(&rt_sched_attr_blind_spot, sizeof(cpu_set_t), &threadcpu); //sets
the thread's affinity to a specific core

       rc=pthread_attr_init(&rt_sched_attr_alarm);                                         // Initialize the
attributes
       rc=pthread_attr_setinheritsched(&rt_sched_attr_alarm, PTHREAD_EXPLICIT_SCHED);  // Set the
inheritance attribute
       rc=pthread_attr_setschedpolicy(&rt_sched_attr_alarm, SCHED_FIFO);             // Set the
scheduling policy to MY_SCHEDULER
       rc=pthread_attr_setaffinity_np(&rt_sched_attr_alarm, sizeof(cpu_set_t), &threadcpu); //sets the
thread's affinity to a specific core
       rt_param_blind_spot.sched_priority=rt_max_prio-2;                              // Set the scheduling
priority
       rt_param_alarm.sched_priority = rt_max_prio-1;
       rt_param_lanedetection.sched_priority = rt_max_prio - 3;
       pthread_attr_setschedparam(&rt_sched_attr_blind_spot, &rt_param_blind_spot);
       pthread_attr_setschedparam(&rt_sched_attr_alarm, &rt_param_alarm);
       pthread_attr_setschedparam(&rt_sched_attr_lanedetection, &rt_param_lanedetection);


    if(pthread_create(&thread_lanedetection,&thread_attr_lanedetection,image_processing,NULL) !=0)
    {
        syslog(LOG_ERR,"pthread_create error from timestamp");
        exit(1);
    }

    if(pthread_create(&thread_blind_spot,&thread_attr_lanedetection,blind_spot,NULL) !=0)

    {
        syslog(LOG_ERR,"pthread_create error from timestamp");
        exit(1);
    }

    if(pthread_create(&thread_alarm,&thread_attr_alarm,alarm, NULL) !=0)
    {
        syslog(LOG_ERR,"pthread_create error from timestamp");
        exit(1);
    }
```

```
       if(pthread_join(thread_lanedetection, NULL) !=0 )
    {
        syslog(LOG_ERR,"pthread_join");
        exit(1);
    }

    if(pthread_join(thread_blind_spot, NULL) !=0 )
    {
        syslog(LOG_ERR,"pthread_join");
        exit(1);
    }

    if (pthread_join(thread_alarm, NULL) !=0 )
    {
        syslog(LOG_ERR,"pthread_join");
        exit(1);
    }
    gpioTerminate();


    sem_destroy(&sem_blind_spot);
    sem_destroy(&sem_alarm);
    sem_destroy(&sem_img);
    closelog();
return 0;
}
```