ECEN5623, Real-Time Embedded Systems

Automated Car Braking System

UNDER THE GUIDANCE OF:
PROFESSOR TIMOTHY SCHERR

University of Colorado Boulder

SUBMITTED BY:
AAKSHA JAYWANT
ATHARV DESAI

Date : 05/02/2020

Contents

I. Introduction:

- The primary objective of our project 'Autonomous Car Braking System' is to use OpenCv Object Detection tool on the Raspberry Pi interfaced with Rpi Camera for detecting road signs and regulating the car speed.
- The preliminary notion of this project is to incorporate smart brakes mode in autonomous cars which can be generally activated on 8-lane-highways.
- The idea came into our minds when we read about the accidents which take place when the cars are in driverless mode on 8-lane-highways.
- Following is an image of a car accident which took place due to the car on autopilot mode on Highway 101 (leveraged from theguardian.com)

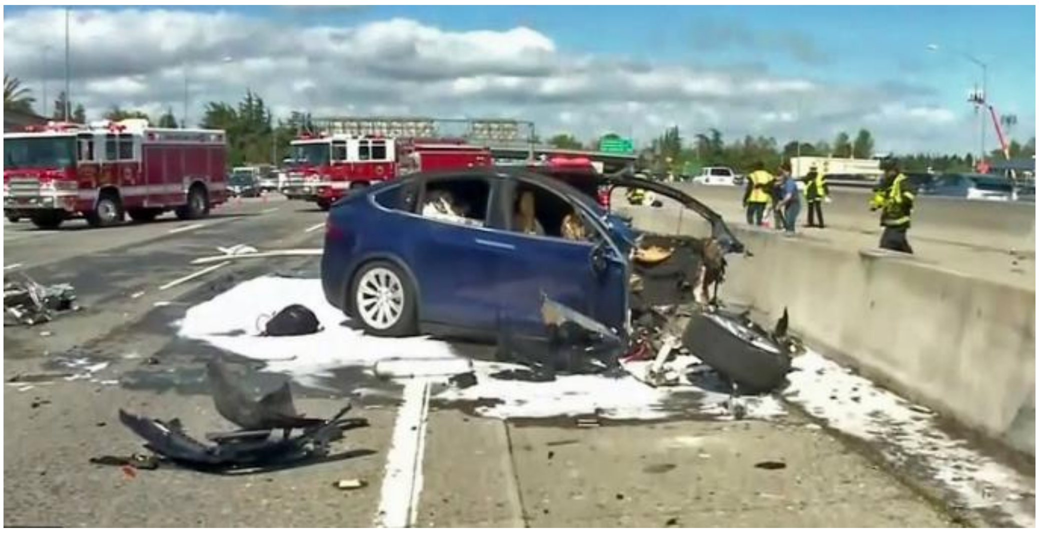


Fig. Accident in a Tesla Car while in auto pilot mode

- To avoid such potential accidents in future, we felt that OpenCV could be a great tool to detect the road signs on highways like the Stop Sign, Speed Limit Sign and should regulate the car speed without human intervention.

- ➔ The Raspberry Pi Camera would detect the Stop Sign as shown below.

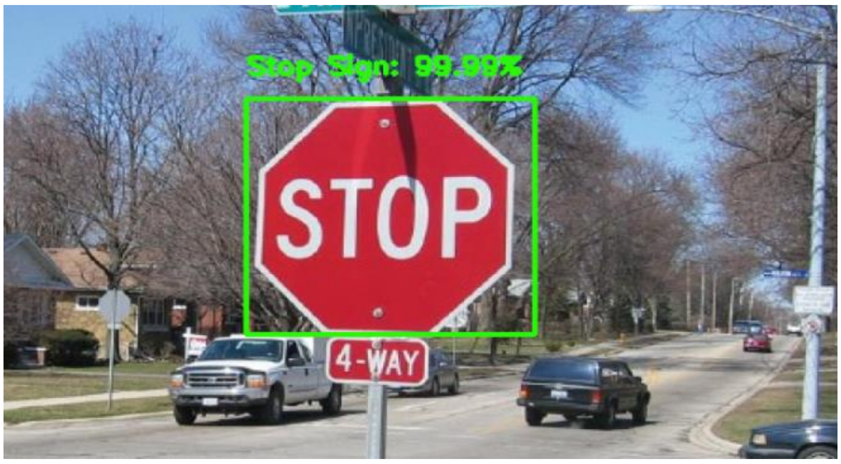


Fig. Stop Sign Detection

II. <u>Working Principle</u>

The project will be primarily divided into two real time services which are

1. Image Processing using OpenCV
2. Car Motor Speed Regulation

- ➔ Also, if time permits and depending on the sensor availability due to COVID-19, we plan to incorporate one more service as a future scope which can be interfacing Ultrasonic sensor to detect obstacle proximity, fingerprint sensor to apply immediate brakes or accelerometer to vary the speed based on the car elevation.
- ➔ We will be implementing a Rate Monotonic Algorithm to schedule the two real time services with predefined priorities to each of the service.
- ➔ Since the whole project is based on Jetson i.e Linux platform, we will be using SCHED_FIFO scheduling policy. We will be using Rate Monotonic Policy, to implement the two tasks namely the Camera task and the motor control task.
- ➔ Following are the Hardware Components which we will be incorporating in our project
  1) Raspberry Pi 3
  2) Raspberry Pi Camera
  3) Motor Driver & Motors

- ➔ Following are the tasks which will be running in the project using RMA policy.
  1) Scheduler Task
     - This task will be having the highest priority among all the tasks
     - This task will be controlling the order of execution of the other tasks
     - Also, this task will be using some locking mechanisms to ensure proper synchronization between the tasks.

  2) Object Detection Task

- This task will use OpenCV on Raspberry Pi Camera and will detect the human presence while crossing the road or the road stop sign on the 8-lane-highways.
- This task will have third highest priority i.e. the lowest priority among the other tasks.

3) Motor Speed Control Task
    - This task will be having second highest priority after the scheduler task and higher priority than the Object Detection task.
    - Whenever the Camera detects human presence or stop sign presence, the Motor speed control task would pre-empt the lower priority camera object detection task and it would regulate the motor speed to avoid accidents.

III. Individual Roles:

1) Aaksha Jaywant

➢ I created a Scheduler task which will execute tasks according to the SCHED_FIFO.

➢ The scheduler will use Rate Monotonic Scheduling algorithm for posting semaphores to the object detection task as well as motor speed control task according to their frequency of occurrence.

➢ The priority of the tasks was ensured by using locking mechanisms while the higher priority task is getting executed.

➢ Also, I will be working on an Object detection task using Raspberry Pi camera configuration on Raspberry Pi Board and getting OpenCV to work on it according to project requirements. I will be doing this task in collaboration with my partner Atharv Desai.

➢ Worked on creating the int main (void) function which had all the initializations of attributes and pthread creations.

2) Atharv Desai

➢ I worked on the Motor Speed Control Task based on the feedback received from the Object Detection from the Raspberry Pi 3.

➢ I incorporated a flag motor_Stop flag set by Object Detection Task and verified the flag in the motor speed taskcode.

➢ Also, I will be setting the GPIO pins on the Raspberry Pi to interface the Motor Driver to supply enough power to the motors.

➢ Therefore, I'll have to scrutinize and speculate the Motor Driver datasheet for the motor integration with Raspberry Pi.

➢ Mapping of the PWM duty cycle with the Object Detection to regulate motor speed

➢ Also, along with my partner in collaboration, I will be working on road sign detection in the OpenCV Object Detection task.

IV. Functional Capability Requirements

- The scheduler schedules the services based on the release of semaphore at a frequency.
- The motor of the car should keep on moving in forward direction until the motor stop flag is set.
- The motor should stop when the motor stop flag is set.
- The object detection must capture video frames and check for the threshold radius i.e. 15cm.
- The object detection must be able to make proper decisions on detecting the stop sign.
- The object detection task should set and reset the motor stop flag depending on the threshold radius.

V. Functional Design Overview and Diagrams

- Shown below is the model of our 'Autonomous Car Braking System'

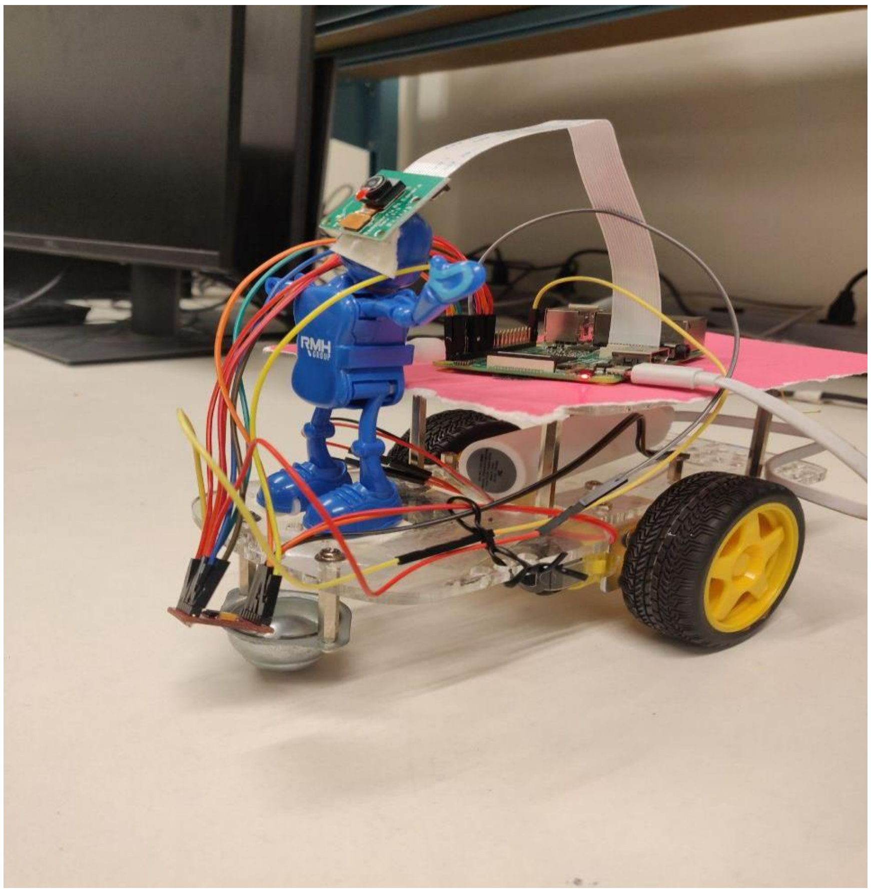


Fig. Autonomous Car Braking System Model

➢ To design this hard real time system, hardware requirements and software requirements are explained below

❑ HARDWARE REQUIREMENTS

1) Raspberry Pi 3

➢ Initially we were confused whether to use Jetson Nano or Raspberry Pi.

➢ We needed GPIO pins to interface the motor driver which is generally useful in stepping up the voltage to drive the motor since the drive voltage on the GPIO pins of a microcontroller is very less.

➢ Jetson Nano doesn't have GPIO APIs in it's device tree where as WiringPi is a GPIO library readily available in Raspberry Pi. Hence we decided to move ahead with Raspberry Pi.

Fig. Raspberry Pi 3

2) Raspberry Pi Camera

➢ We had a Raspberry Pi Camera v1.3 model available with us.

➢ It has a resolution of 5 Megapixels, C programming APIs are present and it's driver is available in the linux kernel.

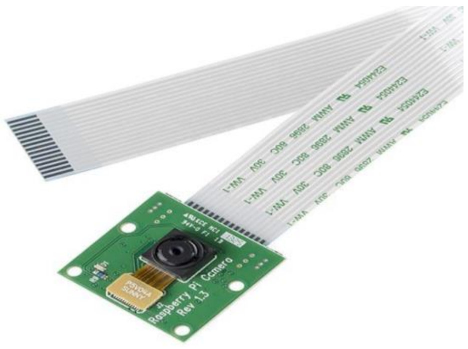


Fig. Raspberry Pi Camera v1.3

3) Motor Driver TB6612FNG

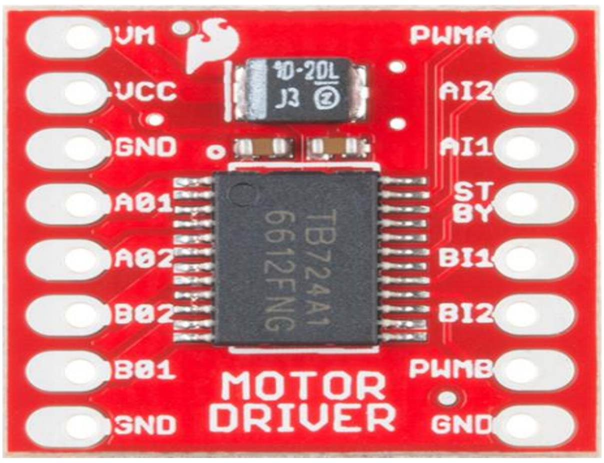


Fig. Motor Driver

- It is a dual motor driver with separately controlled outputs.
- VM pin is for providing power to the motors, PWM is for speed control, standy pin is for power saving whereas AI1, AI2, BI1, BI2 and AO1, AO2, BO1, BO2 are A and B input and output pins respectively.

4) DC Motor

- It has a “no load” speed of 140 RPM at a supply voltage of 4.5V.

Fig. DC Motors

❑ SOFTWARE REQUIREMENTS

1) OpenCV Library for Image processing
2) Wiring Pi Library for enabling GPIO
3) RaspiCam for C++ functions to operate Raspberry Pi camera
4) SSH Remote Login to Raspberry Pi

❑ FUNCTIONAL INTERFACES:

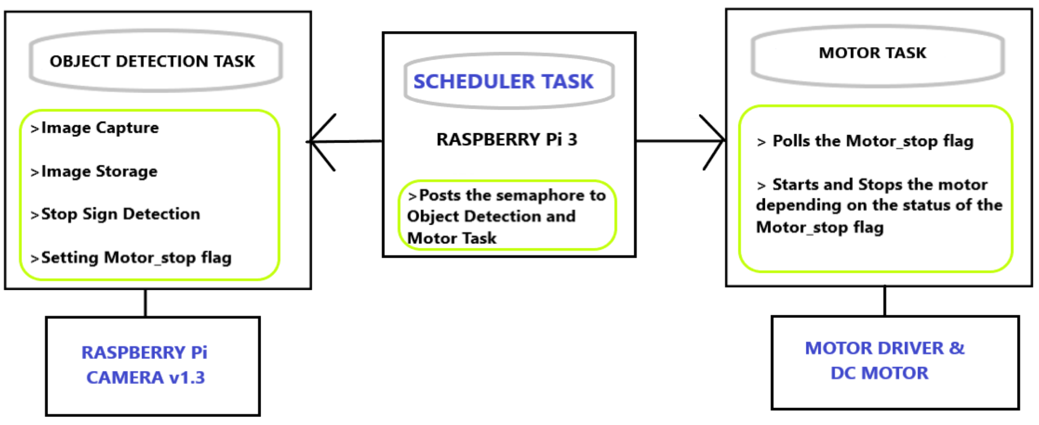
1) Motor and motor driver interface

- ➢ The motor is connected to the Raspberry Pi through the motor driver so that the motor driver can step up the input voltage across it's terminals and hence can provide enough voltage to drive the motors.
- ➢ The motor thread controls the motors by writing control values to the GPIO pins of the Raspberry Pi.
- ➢ The motor thread also polls the motor stop flags and stops and starts the motor according to the signal received.

2) Camera Interface for Object Detection

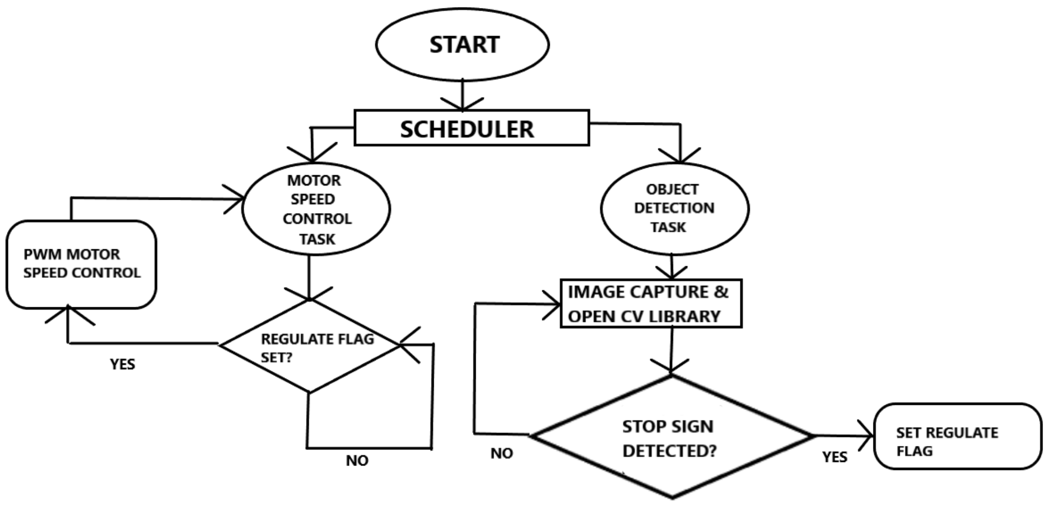
- ➢ The camera is interfaced with the Raspberry Pi so that it can detect stop sign.
- ➢ There is an object detection thread which loads the .xml file and opens the camera so that it can start capturing frames.
- ➢ The Opencv library functions are used to control the camera as per requirements.

❑ BLOCK DIAGRAM



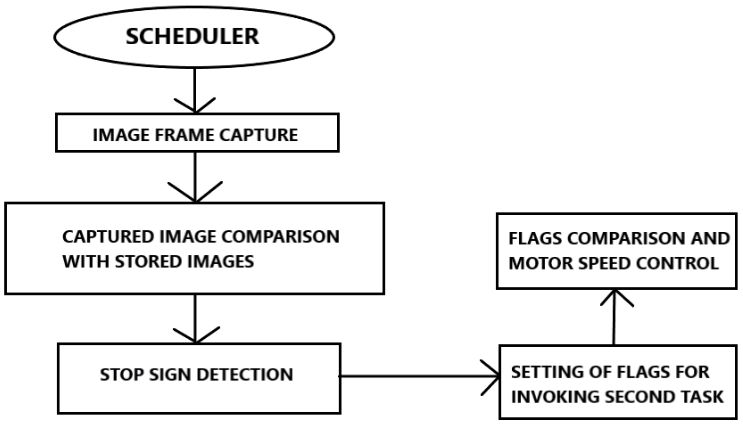
- ➢ The system block diagram has 3 main components.
- ➢ The Raspberry Pi 3 was used to spawn different tasks such as scheduler task,motor task and object detection task.
- ➢ The Raspberry Pi Camera was used to capture image frames at a rate of 25 FPS for object detection.
- ➢ The Motor driver and DC motor’s forward and stop motion was controlled by the object detection task.

❑ <u>CONDITIONAL FLOW DIAGRAM</u>



- ➢ The code flow is such that the scheduler task which is operating at 80Hz posts the semaphores to the motor task at 40Hz and to the Object detection task at 4Hz since the camera task should be given some more time processing and for sensing the video frames.
- ➢ The Motor task polls for the motor stop flag to be set which is set in the object detection task.
- ➢ The object detection task loads the stopsign19stages.xml file and starts capturing frames in a while loop. It constantly checks for the threshold radius of the stop sign i.e. 15cm . If the radius is greater than 15cm, the motor stop flag is set which in turn halts the motor.

❑ **SEQUENCE DIAGRAM:**


SCHEDULER
IMAGE FRAME CAPTURE
CAPTURED IMAGE COMPARISON WITH STORED IMAGES
STOP SIGN DETECTION
SETTING OF FLAGS FOR INVOKING SECOND TASK
FLAGS COMPARISON AND MOTOR SPEED CONTROL

➢ The above diagram shows the process execution flow.

VI. Real-Time Analysis and Design with Timing Diagrams

❑ REAL-TIME RESPONSE TIME REQUIREMENTS FOR MEETING DEADLINES:

➢ To design the Real-time Embedded System, the first step that needs to be done is to analyse the deadlines that are meant to be met in real-time for the system to work safely and avoid any catastrophic consequences.

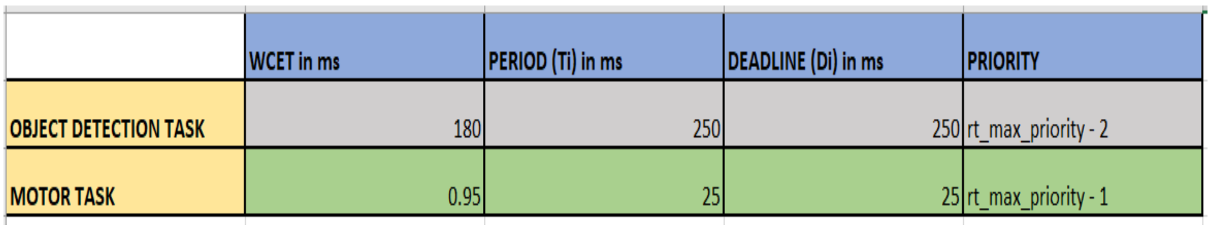
➢ So, to analyse the deadline which is required to be considered in this project, we researched and found that according to MIT news report for ‘how fast humans react to road hazards’ especially in semi autonomous cars, humans take around 390 to 600 milliseconds to detect and react to road hazards

➢ Therefore, based on this research we set our deadline for Camera Object Detection task to be around 250 ms and the Motor task to be 25 ms so that the autonomous braking system we design meets the deadline requirements in real world scenario.

❑ REAL-TIME SERVICES FEASIBILITY, SAFETY AND MARGIN:

1) WCET, Ti and Di estimation

➢ Depending on the response time 390 to 600ms we set the frequency of our motor task and object detection so that they could fall under the usual response time. We took various test cases into consideration wherein we adjusted the deadline of each task. The best we could come up with was around 250ms for object detection task and 25ms for motor task.



	WCET in ms	PERIOD (Ti) in ms	DEADLINE (Di) in ms	PRIORITY
OBJECT DETECTION TASK	180	250	250	rt_max_priority - 2
MOTOR TASK	0.95	25	25	rt_max_priority - 1

Fig. DCT Table

2) Cheddar Worst-Case and Simulation with Timing Diagram

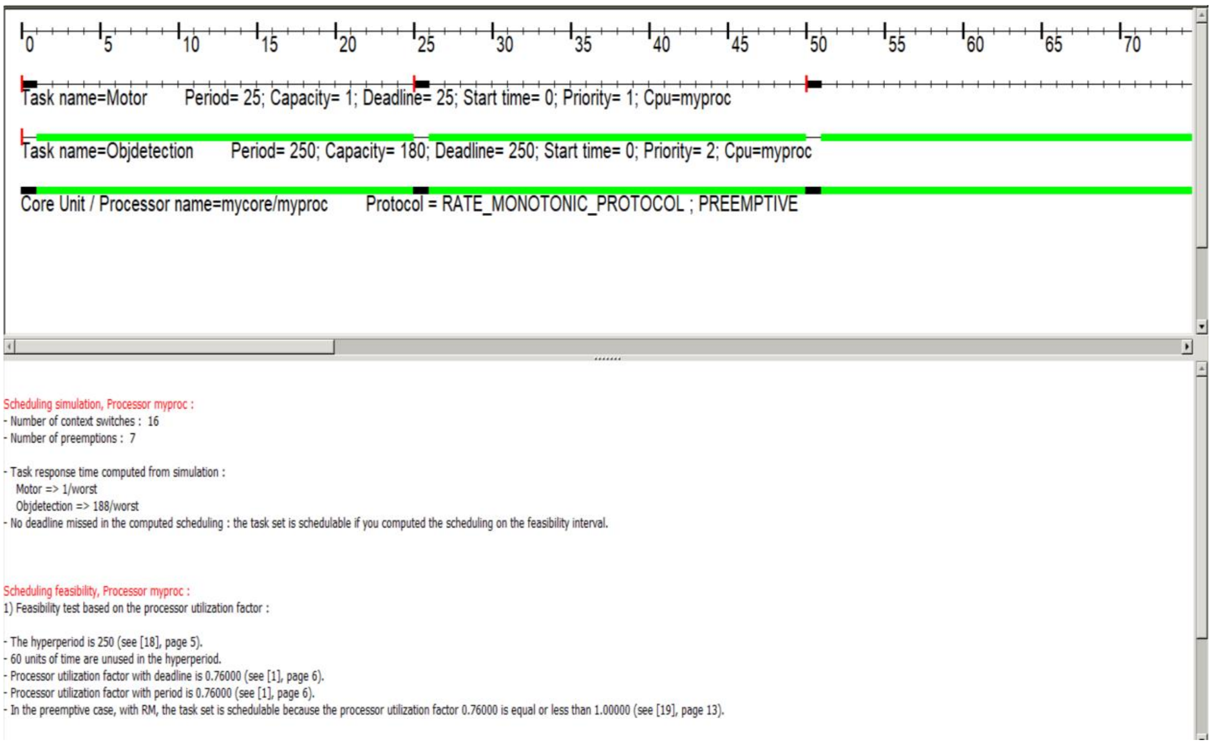
➢ The usual response time of the system that we had considered was verified by means of cheddar analysis.

➢ The WCET was calculated using clock_gettime function with clock id as CLOCK_REALTIME. Timestamps were incorporated at the start and end of each task to calculate the WCET.

```
➢ if ( ( end - start ) > wcet )
➢                 {
➢                         wcet = end - start;
➢                 }
```

➢ We fed the values from the above table into Cheddar software to check whether this system was feasible.

➢ Based upon the results, we understood that the real-time process is set schedulable. Hence we further went onto designing an efficient code so that we get a better response time.

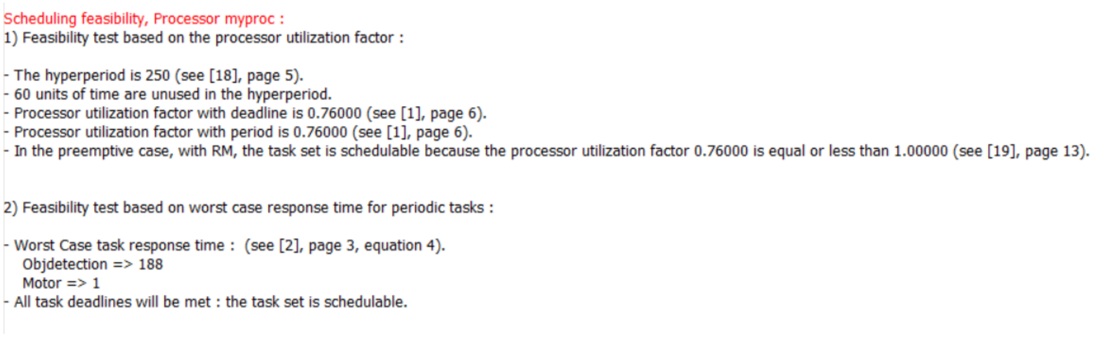


```
Task name=Motor        Period= 25; Capacity= 1; Deadline= 25; Start time= 0; Priority= 1; Cpu=myproc
Task name=Objdetection        Period= 250; Capacity= 180; Deadline= 250; Start time= 0; Priority= 2; Cpu=myproc
Core Unit / Processor name=mycore/myproc        Protocol = RATE_MONOTONIC_PROTOCOL ; PREEMPTIVE

Scheduling simulation, Processor myproc :
- Number of context switches :  16
- Number of preemptions :  7

- Task response time computed from simulation :
   Motor => 1/worst
   Objdetection => 188/worst
- No deadline missed in the computed scheduling : the task set is schedulable if you computed the scheduling on the feasibility interval.


Scheduling feasibility, Processor myproc :
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 250 (see [18], page 5).
- 60 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.76000 (see [1], page 6).
- Processor utilization factor with period is 0.76000 (see [1], page 6).
- In the preemptive case, with RM, the task set is schedulable because the processor utilization factor 0.76000 is equal or less than 1.00000 (see [19], page 13).
```

Fig. Cheddar Simulation

- The Utilization factor from Cheddar Analysis is around 76%.
-
- The least upper bound (LUB), according to Liu and Layland paper for safe utility is defined as follows:

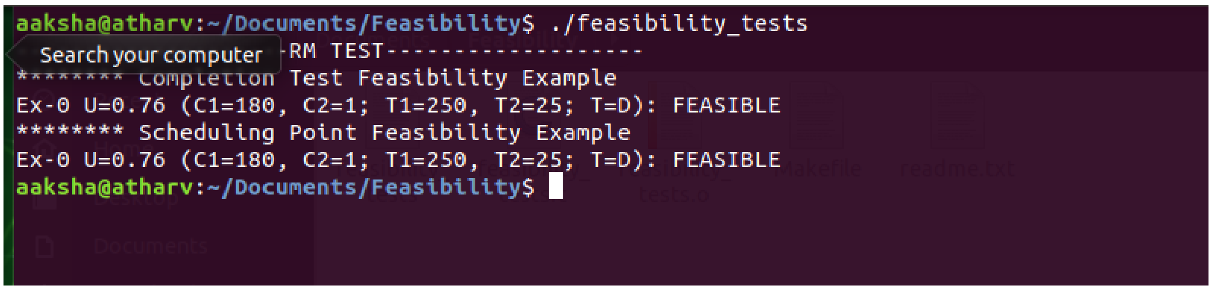
$$U = \sum_{i=1}^{m} (C_i / T_i) \leq m(2^{1/m} - 1)$$



```
Scheduling feasibility, Processor myproc :
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 250 (see [18], page 5).
- 60 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.76000 (see [1], page 6).
- Processor utilization factor with period is 0.76000 (see [1], page 6).
- In the preemptive case, with RM, the task set is schedulable because the processor utilization factor 0.76000 is equal or less than 1.00000 (see [19], page 13).


2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time :  (see [2], page 3, equation 4).
   Objdetection => 188
   Motor => 1
- All task deadlines will be met : the task set is schedulable.
```

Fig. Processor Utilization

4) Scheduling Point / Completion tests



```
aaksha@atharv:~/Documents/Feasibility$ ./feasibility_tests
                      -RM TEST------------------
******** Completion Test Feasibility Example
Ex-0 U=0.76 (C1=180, C2=1; T1=250, T2=25; T=D): FEASIBLE
******** Scheduling Point Feasibility Example
Ex-0 U=0.76 (C1=180, C2=1; T1=250, T2=25; T=D): FEASIBLE
aaksha@atharv:~/Documents/Feasibility$ 
```

Fig. Output of Feasibility Tests

- ➢ The scheduling point test and the completion point test are the two algorithms which are used for necessary and sufficient feasibility testing with RM policy. In this algorithm .
- ➢ The scheduling point test is based on Liu and Layland's paper which asserts that
- ➢ "In the worst case, all services might be requested at the same point in time".
- ➢ Hence this iterative test is done by the following following equation.

$$\forall i, 1 \le i \le n, \min \sum_{j=1}^{i} C_j \left\lceil \frac{(l)T_k}{T_j} \right\rceil \le (l)T_k$$

$$(k,l) \in R_i$$

$$R_i = \left\{ (k,l) \middle| 1 \le k \le i, l = 1, \dots, \left\lfloor \frac{T_i}{T_k} \right\rfloor \right\}$$

- ➢ The completion test is an alternative to the Scheduling point test. It's computation is based on this equation,

$$a_n(t) = \sum_{j=1}^{n} \left\lceil \frac{t}{T_j} \right\rceil C_j$$

➢ We took the completion test as well as the scheduling point test for our real-time system. The utilization factor as well the feasibility check was calculated through the feasibility test code.

5) Safety margin analysis

➢ To compute the Safety Margin of our Autonomous Car Braking System, we need to consider the Ci of the Camera Object Detection task and the Motor task.
➢ C1 → Camera Object Detection Task = 180 milliseconds
➢ C2 → Motor Task = 0.95 milliseconds

<u>Calculation</u>:

➢ C1 + C2 = 180 ms + 0.95 ms
➢ = 180.95 ms

➢ Period of Camera Object detection task is 250 ms. (Frequency is 4 Hz)
➢
➢ ∴ 1/(4 Hz) = 250 ms - 180.95 = 69.05 ms

➢ Therefore, the system meets the Safety Margin Requirements and can be considered as feasible.

VII. Design Documentation

❑ PRIORITY ANALYSIS:

- ➢ The priorities of different services are as follows:
- ➢ Priority of the Scheduler = rt_max_priority
- ➢ Priority of the Motor Task = rt_max_priority - 1
- ➢ Priority of the Object Detection Task = rt_max_priority - 2
- ➢ In SCHED_FIFO maximum priority is 99. Hence, rt_max_priority = 99.
- ➢ This system uses Rate Monotonic scheduling policy. Hence the task with the highest frequency gets the highest priority.
- ➢ Therefore the motor task is scheduled at higher frequency than the object detection task as it has to keep an eye on the motor stop flag whether it is set or reset.
- ➢ Hence the motor task will be scheduled with the 2nd highest priority as it will check the flag for changing it's motion.

❑ PIN DIAGRAM

- ➢ SHOWN BELOW IS THE TABLE FOR CONNECTIONS REQUIRED FOR HARDWARE INTERFACING OF THE PROJECT.

PINS	CONNECTION & FUNCTIONALITY
Raspberry Pi 3 Pin → 7	AI-1 pin of Motor Driver for Forward direction Input to Motor 1
Raspberry Pi 3 Pin → 16	AI-2 pin of Motor Driver for Reverse direction Input to Motor 1
Raspberry Pi 3 Pin → 18	PWM-A pin of Motor Driver for Varying PWM pulse Input to Motor 1
Raspberry Pi 3 Pin → 11	BI-1 pin of Motor Driver for Forward direction Input to Motor 2
Raspberry Pi 3 Pin → 13	BI-2 pin of Motor Driver for Reverse direction Input to Motor 2
Raspberry Pi 3 Pin → 15	PWM-B pin of Motor Driver for Varying PWM pulse Input to Motor 2

Motor Driver Pin → AO-1	DC Motor 1 Forward Direction Pin
Motor Driver Pin → AO-2	DC Motor 1 Reverse Direction Pin
Motor Driver Pin → BO-1	DC Motor 2 Forward Direction Pin
Motor Driver Pin → BO-2	DC Motor 2 Reverse Direction Pin

❑ **BUDGET ESTIMATES**

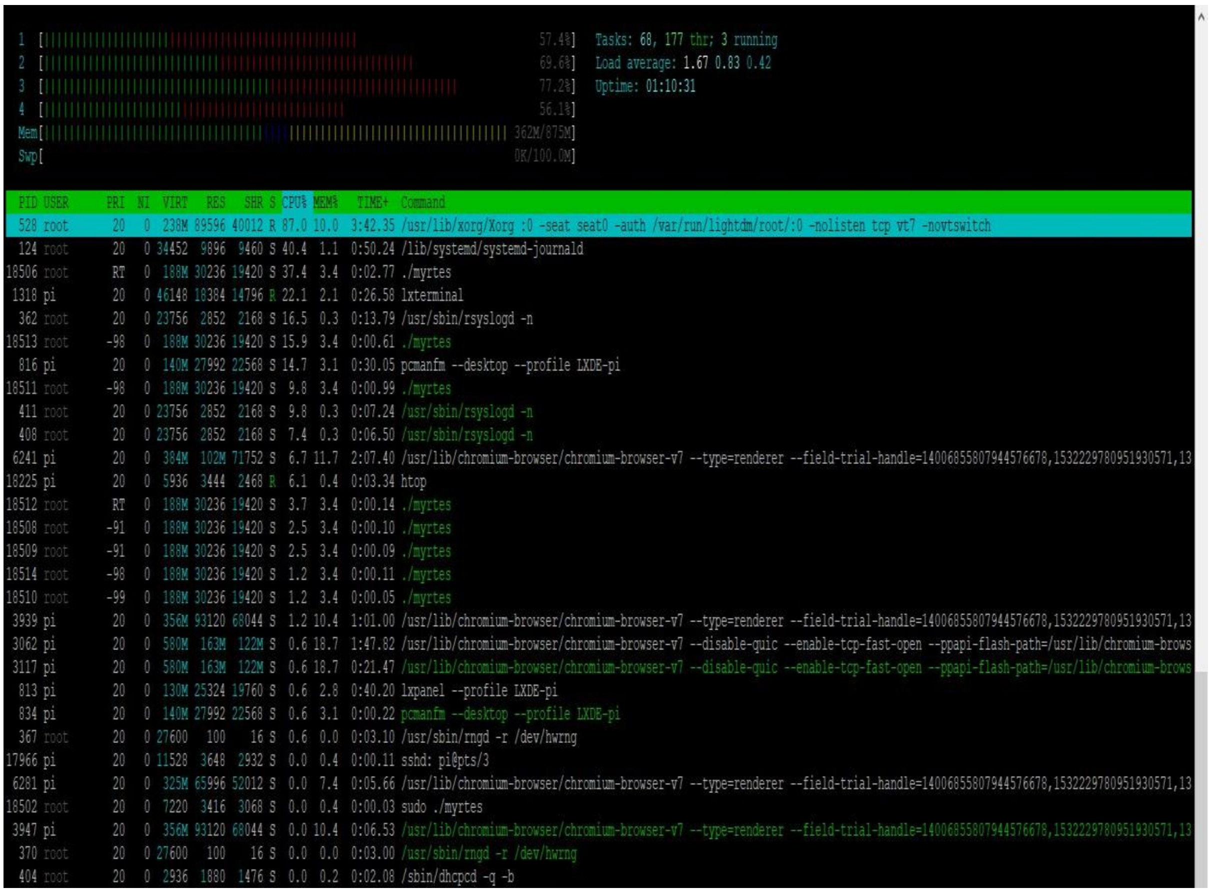
Sr No.	Component	Cost
1.	Raspberry Pi 3	$ 35
2.	R Pi Camera	$ 12.99
3.	DC motors	$ 4.11
4.	Motor Driver	$ 4.95
5.	Chassis	$ 6.99
	Total	$ 64.04

VIII. Proof-of-Concept with Example Output and Tests Completed

❑ **CPU UTILIZATION**

➢ The below image shows the CPU Utilization i.e. the load on the CPU core while the code is being executed.

➢ The CPU utilization is around 87%. The utilization calculation that was achieved through Cheddar software was 76%. This discrepancy might be because of the CPU loading caused by the background processes. The background processes might have introduced some more load on the CPU.

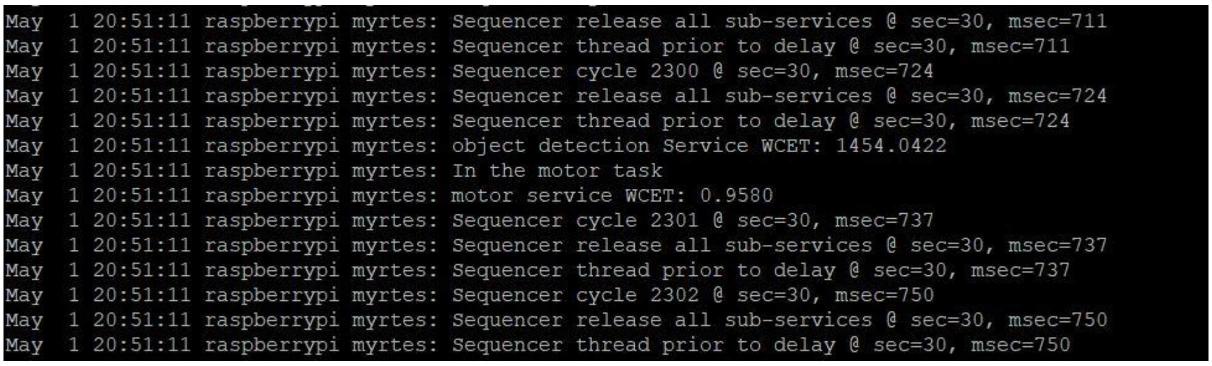


```
  1  [||||||||||||||||||||||||||||||||||||||||||||                      57.4%]   Tasks: 68, 177 thr; 3 running
  2  [|||||||||||||||||||||||||||||||||||||||||||||||||||||              69.6%]   Load average: 1.67 0.83 0.42
  3  [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||       77.2%]   Uptime: 01:10:31
  4  [||||||||||||||||||||||||||||||||||||||||||||                       56.1%]
  Mem[||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 362M/875M]
  Swp[                                                                0K/100.0M]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
  528 root       20   0  238M 89596 40012 R 87.0 10.0  3:42.35 /usr/lib/xorg/Xorg :0 -seat seat0 -auth /var/run/lightdm/root/:0 -nolisten tcp vt7 -novtswitch
  124 root       20   0 34452  9896  9460 S 40.4  1.1  0:50.24 /lib/systemd/systemd-journald
18506 root       RT   0  188M 30236 19420 S 37.4  3.4  0:02.77 ./myrtes
 1318 pi         20   0 46148 18384 14796 R 22.1  2.1  0:26.58 lxterminal
  362 root       20   0 23756  2852  2168 S 16.5  0.3  0:13.79 /usr/sbin/rsyslogd -n
18513 root      -98   0  188M 30236 19420 S 15.9  3.4  0:00.61 ./myrtes
  816 pi         20   0  140M 27992 22568 S 14.7  3.1  0:30.05 pcmanfm --desktop --profile LXDE-pi
18511 root      -98   0  188M 30236 19420 S  9.8  3.4  0:00.99 ./myrtes
  411 root       20   0 23756  2852  2168 S  9.8  0.3  0:07.24 /usr/sbin/rsyslogd -n
  408 root       20   0 23756  2852  2168 S  7.4  0.3  0:06.50 /usr/sbin/rsyslogd -n
 6241 pi         20   0  384M  102M 71752 S  6.7 11.7  2:07.40 /usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=14006855807944576678,15322297809519305571,13
18225 pi         20   0  5936  3444  2468 R  6.1  0.4  0:03.34 htop
18512 root       RT   0  188M 30236 19420 S  3.7  3.4  0:00.14 ./myrtes
18508 root      -91   0  188M 30236 19420 S  2.5  3.4  0:00.10 ./myrtes
18509 root      -91   0  188M 30236 19420 S  2.5  3.4  0:00.09 ./myrtes
18514 root      -98   0  188M 30236 19420 S  1.2  3.4  0:00.11 ./myrtes
18510 root      -99   0  188M 30236 19420 S  1.2  3.4  0:00.05 ./myrtes
 3939 pi         20   0  356M 93120 68044 S  1.2 10.4  1:01.00 /usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=14006855807944576678,15322297809519305571,13
 3062 pi         20   0  580M  163M  122M S  0.6 18.7  1:47.82 /usr/lib/chromium-browser/chromium-browser-v7 --disable-quic --enable-tcp-fast-open --ppapi-flash-path=/usr/lib/chromium-brows
 3117 pi         20   0  580M  163M  122M S  0.6 18.7  0:21.47 /usr/lib/chromium-browser/chromium-browser-v7 --disable-quic --enable-tcp-fast-open --ppapi-flash-path=/usr/lib/chromium-brows
  813 pi         20   0  130M 25324 19760 S  0.6  2.8  0:40.20 lxpanel --profile LXDE-pi
  834 pi         20   0  140M 27992 22568 S  0.6  3.1  0:00.22 pcmanfm --desktop --profile LXDE-pi
  367 root       20   0 27600   100    16 S  0.6  0.0  0:03.10 /usr/sbin/rngd -r /dev/hwrng
17966 pi         20   0 11528  3648  2932 S  0.0  0.4  0:00.11 sshd: pi@pts/3
 6281 pi         20   0  325M 65996 52012 S  0.0  7.4  0:05.66 /usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=14006855807944576678,15322297809519305571,13
18502 root       20   0  7220  3416  3068 S  0.0  0.4  0:00.03 sudo ./myrtes
 3947 pi         20   0  356M 93120 68044 S  0.0 10.4  0:06.53 /usr/lib/chromium-browser/chromium-browser-v7 --type=renderer --field-trial-handle=14006855807944576678,15322297809519305571,13
  370 root       20   0 27600   100    16 S  0.0  0.0  0:03.00 /usr/sbin/rngd -r /dev/hwrng
  404 root       20   0  2936  1880  1476 S  0.0  0.2  0:02.08 /sbin/dhcpcd -q -b
```

Fig. CPU Utilization

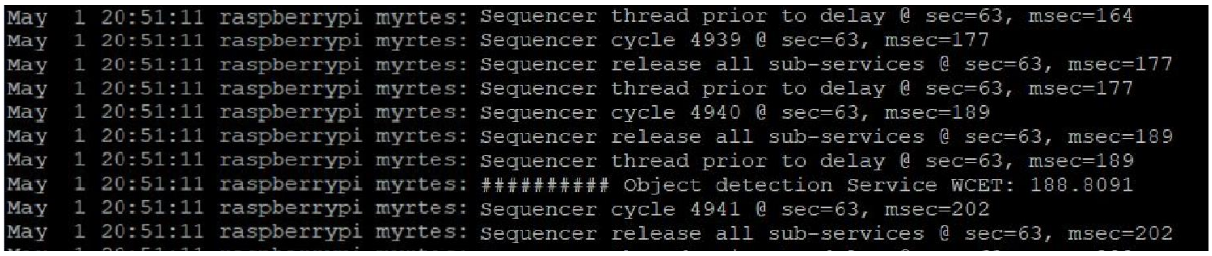
❑ TIMESTAMPING USING SYSLOG

- ➢ The below screenshots show the time stamping for calculation of WCET.
- ➢ WCET of the Motor task is 0.95ms ≅ 1ms where as the WCET of the object detection task is 180ms.



```
May  1 20:51:11 raspberrypi myrtes: Sequencer release all sub-services @ sec=30, msec=711
May  1 20:51:11 raspberrypi myrtes: Sequencer thread prior to delay @ sec=30, msec=711
May  1 20:51:11 raspberrypi myrtes: Sequencer cycle 2300 @ sec=30, msec=724
May  1 20:51:11 raspberrypi myrtes: Sequencer release all sub-services @ sec=30, msec=724
May  1 20:51:11 raspberrypi myrtes: Sequencer thread prior to delay @ sec=30, msec=724
May  1 20:51:11 raspberrypi myrtes: object detection Service WCET: 1454.0422
May  1 20:51:11 raspberrypi myrtes: In the motor task
May  1 20:51:11 raspberrypi myrtes: motor service WCET: 0.9580
May  1 20:51:11 raspberrypi myrtes: Sequencer cycle 2301 @ sec=30, msec=737
May  1 20:51:11 raspberrypi myrtes: Sequencer release all sub-services @ sec=30, msec=737
May  1 20:51:11 raspberrypi myrtes: Sequencer thread prior to delay @ sec=30, msec=737
May  1 20:51:11 raspberrypi myrtes: Sequencer cycle 2302 @ sec=30, msec=750
May  1 20:51:11 raspberrypi myrtes: Sequencer release all sub-services @ sec=30, msec=750
May  1 20:51:11 raspberrypi myrtes: Sequencer thread prior to delay @ sec=30, msec=750
```

Fig. Motor Service WCET

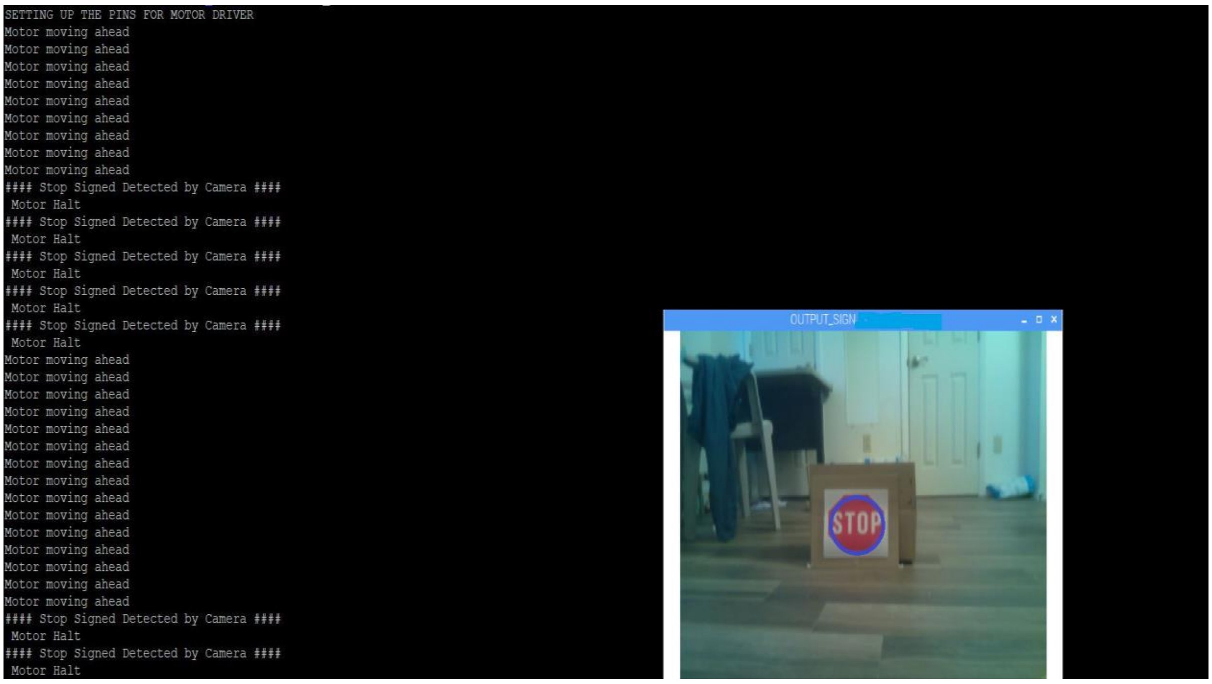


```
May  1 20:51:11 raspberrypi myrtes: Sequencer thread prior to delay @ sec=63, msec=164
May  1 20:51:11 raspberrypi myrtes: Sequencer cycle 4939 @ sec=63, msec=177
May  1 20:51:11 raspberrypi myrtes: Sequencer release all sub-services @ sec=63, msec=177
May  1 20:51:11 raspberrypi myrtes: Sequencer thread prior to delay @ sec=63, msec=177
May  1 20:51:11 raspberrypi myrtes: Sequencer cycle 4940 @ sec=63, msec=189
May  1 20:51:11 raspberrypi myrtes: Sequencer release all sub-services @ sec=63, msec=189
May  1 20:51:11 raspberrypi myrtes: Sequencer thread prior to delay @ sec=63, msec=189
May  1 20:51:11 raspberrypi myrtes: ########## Object detection Service WCET: 188.8091
May  1 20:51:11 raspberrypi myrtes: Sequencer cycle 4941 @ sec=63, msec=202
May  1 20:51:11 raspberrypi myrtes: Sequencer release all sub-services @ sec=63, msec=202
```

Fig. Object Detection Service WCET

❑ <u>Tests Undertaken:</u>

- ➢ The below output shows the output of the project testing. The left side of the image shows the motion of the motor while object detection service detects the STOP sign.
- ➢ It output reads as follows"
- ➢ "Stop Sign Detected by Camera #### Motor Halt"
- ➢ The pre-trained XML classifiers detect the stop sign and draw a circle around it once the stop sign is detected.
- ➢ The motion of the motor again changes and the car starts moving in the forward direction.



```
SETTING UP THE PINS FOR MOTOR DRIVER
Motor moving ahead
Motor moving ahead
Motor moving ahead
Motor moving ahead
Motor moving ahead
Motor moving ahead
Motor moving ahead
Motor moving ahead
Motor moving ahead
#### Stop Signed Detected by Camera ####
 Motor Halt
#### Stop Signed Detected by Camera ####
 Motor Halt
#### Stop Signed Detected by Camera ####
 Motor Halt
#### Stop Signed Detected by Camera ####
 Motor Halt
#### Stop Signed Detected by Camera ####
 Motor Halt
Motor moving ahead
Motor moving ahead
Motor moving ahead
Motor moving ahead
Motor moving ahead
Motor moving ahead
Motor moving ahead
Motor moving ahead
Motor moving ahead
Motor moving ahead
Motor moving ahead
Motor moving ahead
Motor moving ahead
Motor moving ahead
Motor moving ahead
#### Stop Signed Detected by Camera ####
 Motor Halt
#### Stop Signed Detected by Camera ####
 Motor Halt
```

Fig. Working Test Case

❑ Test Objectives & Substantiation

Sr. No	Test	Status	Substantiation
1.	Installing OpenCV on Raspberry Pi 3.	Done	Successfully Installed OpenCV 4 on Raspberry Pi.
2.	Installing RaspiCam Library and Research on Wiring Pi Library Implementation.	Done	Installed RaspiCam Library for using C++ camera interfacing functions & WiringPi library use for GPIO Configuration.
3.	Writing Motor Service code and Motor Driver & DC Motors Interfacing with Raspberry Pi.	Done	Interfaced TB6612FNG Motor driver and DC motors with R Pi and wrote a Motor service thread.
4.	Configuring R-Pi Camera in the Raspian OS and implementing R-Pi Camera Functions to capture Images in C++	Done	Enabled the Camera Configurations in Raspian OS of R-Pi and implemented functionalities in the code to capture & store images.
5.	Implementing Haar Cascade Classifier to detect the road stop signs in Camera Object Detection service.	Done	The camera detected the stop sign using the XML stored file and set a flag in Camera service.
6.	Calculating WCET for the Motor Task and Object Detection service.	Done	Calculated the WCET for both the tasks using timestamps in code.
7.	Integrating the two tasks with Scheduler.	Done	Verified the working of both the services handled by the schedular
8.	Cheddar Analysis to verify the feasibility of the whole system.	Done	Verified the feasibility of the tasks by using Cheddar Software
9.	Code Optimization & Testing.	Done	Validated the functioning of the code after integration of the whole system & tested the code for desired output.

❑ PROJECT MANAGEMENT SCHEDULE AND TIMELINE

Task	**Expected Completion Date**	**Actual Completion Date**
Raspicam Library & OpenCV Installation	04/24/2020	04/24/2020
Raspberry Pi camera Interfacing & Verification of Basic Functions	04/25/2020	04/25/2020
Implemented OpenCV functions using Haar Cascade Classifier to detect Stop Sign	04/26/2020	04/28/2020
Inclusion of Timestamps in code and WCET Calculation	04/27/2020	04/29/2020
Implementing Scheduler & Integrating two tasks	04/27/2020	04/29/2020
Cheddar Analysis for the system and scrutinizing real-time requirements of the systems	04/28/2020	04/30/2020
Project Functionality Verification and analysing loopholes based of Professor’s Proposal Feedback	04/29/2020	04/30/2020
Code Optimization and Testing the system workability	04/30/2020	05/01/2020

IX. Conclusion

- ➢ Thus we were able to successfully design the ‘Autonomous Car Braking System’ which met the Deadline requirements for real world scenarios.
- ➢ Along with Professor Tim Scherr’s guidance, we were able to comprehend the real-time concepts and develop the mindset required to design a hard real time system.
- ➢ We learnt about developing a scheduler and integrating the heavy computation tasks like the Object detection task and meeting the real-time requirements of the system.
- ➢ We learnt and speculated in depth for factors to be considered for Real-time System Design like Multithreading, Context switching, Scheduling Policies, Cheddar Analysis
- ➢ Furthermore, we got an opportunity to get hands-on Object Detection Algorithms and concepts like Haar Cascade Classifier, Camera Interfacing and usage of OpenCV.
- ➢ We would like to thank Professor Tim Scherr for providing us with precious knowledge and guidance because of which, we were able to design this Hard-real time embedded system.
- ➢ We would also like to thank the TAs for clarifying our doubts while designing our project.

X. Challenges

- ➢ In the earlier stages of analyzing the project hardware requirements, we faced the challenge of choosing between Jetson Nano and Raspberry Pi 3 since both had their Perks as well as drawbacks.
- ➢ Raspberry Pi 3 had the advantage of having a WiringPi library while in Jetson Nano, interfacing motor driver was a tedious process since there was any predefined patch or library to enable the GPIO pins
- ➢ On the other hand, we had OpenCV already installed on Jetson Nano while we knew that it would take some time and a lengthy process to installm OpenCV on Raspberry Pi.
- ➢ Therefore, we made a tradeoff between time and compatibility and selected Raspberry Pi.
- ➢ Earlier, the scheduling was failed and we were trying to debug but weren't able to comprehend the reason for it. But then , after reducing a few print statements we were able to solve this problem.

XI. Formal References

1. **https://www.geeksforgeeks.org/opencv-c-program-face-detection/**
2. **http://wiringpi.com/reference/software-pwm-library/**
3. **https://www.learnopencv.com/install-opencv-4-on-raspberry-pi/**
4. **https://linuxize.com/post/how-to-install-opencv-on-raspberry-pi/**
5. **https://github.com/cfizette/road-sign-cascades/tree/master/Stop%20Signs/StopSign_HAAR**
6. **https://learn.sparkfun.com/tutorials/activity-guide-for-sparkfun-tinker-kit/circuit-10-motor-basics**
7. **'Real Time Embedded Components and Systems' book by Sam Siewert**

XII. Appendices with results, code and supporting material

1) Code is attached in the zip file
2) Demo Video can be found at
https://drive.google.com/drive/u/1/folders/1TXawl9LmJtcWMQW482g-hBJ4qp8fD2wn