Spring 2022
ECEN 5623
Real-Time Embedded Systems

Final Project Report on

Speed Control for Self Driving Cars

Presented by:
Sanish Kharade
Tanmay Kothale
Vishal Raj

Introduction:

In the modern era we are moving towards unmanned vehicles at a rapid speed. Self-driving cars have become one of the most exciting technologies over the past 3 years. These cars are automated and require less or no human intervention. Society of Automotive Engineers (SAE) have classified this automation into 6 levels. These are –
Levels of driving automation

- Level 0 – no automation
- Level 1 – hands on / shared control
- Level 2 – hands off
- Level 3 – eyes off
- Level 4 – mind off
- Level 5 – steering wheel optional

As of March 2022, very few vehicles are operating at level 3 and above. They remain a marginal portion of the market. These self-driving cars have many features and technologies used in them. Amongst many, one of the key features is sign detection. There are so many different signs on roadways like stop signs, traffic lights (red, green, yellow), pedestrian crossing signs, speed limits etc. It is very important that an autonomous car captures these signs, recognizes it, and takes appropriate action in the required amount of time. Basically, this is a case of a real time embedded system where sign detection, identification, and the action, each need to be completed before a certain deadline to prevent accidents and cause harm to other vehicles.

In our project for this course, we have designed and developed such a real time system which emulates a self-driving car, captures, and detects traffic signals and performs the necessary action within a certain deadline. In real world applications we cannot assume ideal cases for any system. Especially if the system is hard real-time and failure to meet the deadlines can lead to fatal accidents. This is true for the braking of any vehicle as well. No vehicle will come to a stop at the exact instant the brake is pressed. The time it takes for a vehicle to stop after applying brakes depends on multiple factors like quality of brakes, weight of the vehicle, speed at which the vehicle was traveling and friction between the tires and the road. The distance the vehicle travels in this time is called the braking distance. We have taken this into account while designing our system. The exact numbers and calculation have been explained later. Our system also deals with obstacles using an ultrasonic sensor for distance measurement.

Our system will consist of a USB camera (C270) which will be connected to the Raspberry Pi. The Raspberry Pi will communicate with the TIVA board over UART. There will be a DC motor and multiple LEDs connected to the TIVA board to demonstrate actuation. The motor can be considered to be emulating a car wheel which needs to stop or change speed depending on the sign detected. The other input to the system is through an ultrasonic sensor input which is used for detecting obstacles for emergency braking. The input of an ultrasonic sensor is going to override the traffic light value any time an object is encountered below the threshold of 35cm.

The camera will keep capturing images at a rate of 2.22 Hz. These images will be processed by an algorithm which will identify the signal. The detected signal data will be transmitted over UART to the TIVA controller which will then take the required action of either stopping the motor or changing its speed. The TIVA controller will be running FreeRTOS since it is highly accurate and can be used to meet hard real-time deadlines.

Functional (Capability) Requirements:

Speed control and braking mechanism is a critical feature of self-driving cars. Here, it is very important that all aforementioned tasks (light-detection, identification, and actuation) have to be completed within a specific deadline. If the system fails to do so, it will result in a crash, maybe a fatal one. Therefore, this system is considered as a 'Hard Real-Time System'.

The system should be capable of:

A. On Raspberry Pi Board:
    1. Capturing the frame using the camera at a required rate.
    2. The captured image should be processed within a certain deadline, and the light-detection algorithm should be able to accurately identify the type of light (Red, Yellow or Green).
    3. The ultrasonic distance sensor should record the correct distance and send accurate data to the microcontroller (stretch goal implemented).
    4. Depending on the type of light (or obstacle) detected, the Raspberry Pi should send a message to the TIVA board over UART.
    5. Wait for acknowledgement from TIVA Board.
    6. Log the WCET and Average ET for all services.

B. On TIVA Board:
    1. The TIVA board should receive data from Raspberry Pi over UART, and send an acknowledgement back.
    2. Based on the value received from Raspberry Pi, the TIVA board should adjust the duty cycle of PWM for motor control.
    3. The adjusted PWM should be directed towards the DC motor to simulate speed control.
    4. Based on the value received from UART, the TIVA board should actuate the corresponding LED.

As mentioned earlier, speed control is a mission-critical feature and hence all services should be completed within a certain deadline. Furthermore, the image processing algorithm should be designed meticulously to specifically identify traffic lights and should not be limited to color detection. The problem with just color detection is that the image processing algorithm may produce false results based on other elements in the surroundings. For example, we don't want speed control to occur when the camera sees a different object (probably a car of Red or Yellow color). Catering to such false data will result in fatal accidents on the road. Hence, the algorithm should only identify a Red/Yellow/Green light when there is a definitive circle of the said color.

Frame capture, processing, light detection, data transmission and reception, and actuation; all these services should complete in a very specific deadline so that there is enough time (and distance) left for the car to gradually come to a full stop.

Functional Design Overview and Diagrams:

[A] Hardware Block Diagram

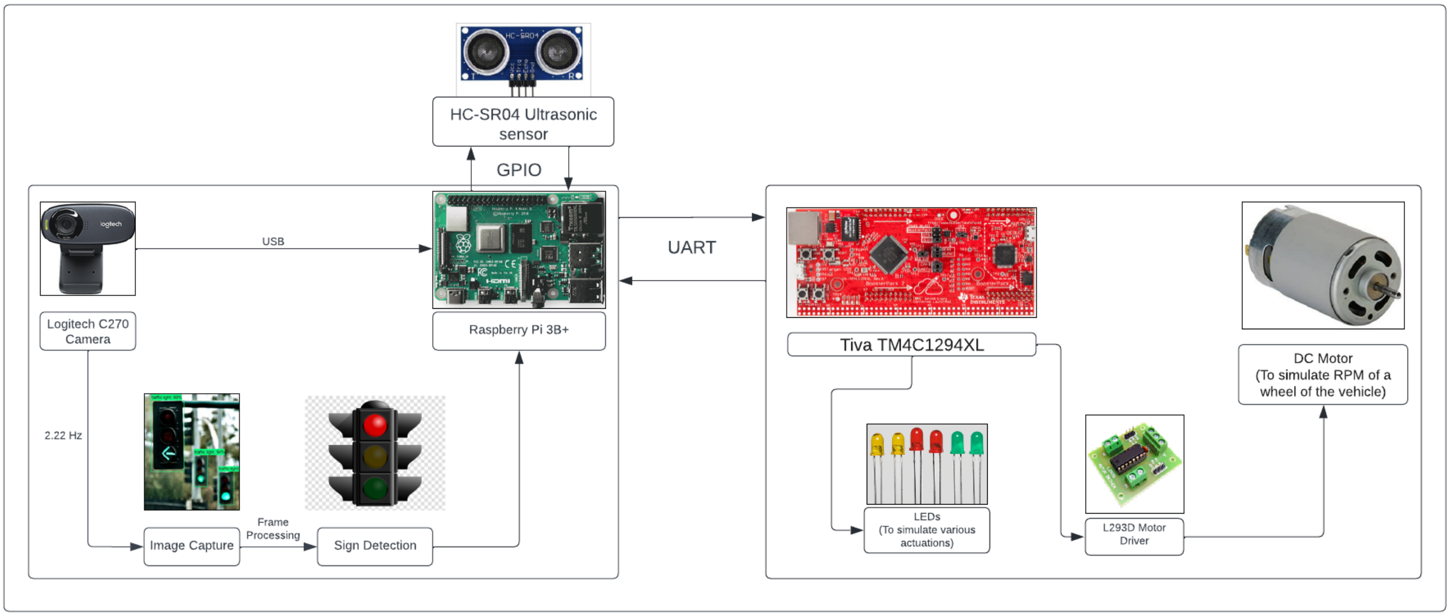


Figure 1: Hardware Block Diagram

Our system uses two boards, namely Raspberry Pi 3B+ and TIVA TM4C1294GXL. All sensing part (frame capture and distance measurement) will occur on Raspberry Pi and all actuation (motor control and LED indications) will occur on TIVA board. For sensors, we are using a Logitech C270 camera and an ultrasonic sensor. These sensors will provide the data 'sensed' from the real world to the Raspberry Pi. The Raspberry Pi will process this data and create a small data packet which will be transferred to the TIVA board over UART communication. Once the TIVA board receives the message, it will generate a PWM signal corresponding to the data received from Raspberry Pi. This PWM signal will then be directed towards a DC motor to simulate speed control. Furthermore, the TIVA board will also actuate a couple of LEDs to indicate the status of the vehicle, whether it is running or not. The Red LED will glow if the car is stationary, and the Green LED will glow if the car is moving.

[B] Software Flow Diagram

Figure 2 depicts the software flow diagram for the proposed system. For the raspberry pi, we are using pthreads to implement a multithreading application and a sequencer which will schedule all the tasks.

First, the software will capture a frame at 2.22 Hz and send that image for processing to the traffic light detection algorithm. If the processing algorithm detects a traffic light, it will immediately exit the processing loop and store the information about traffic light in a data packet.

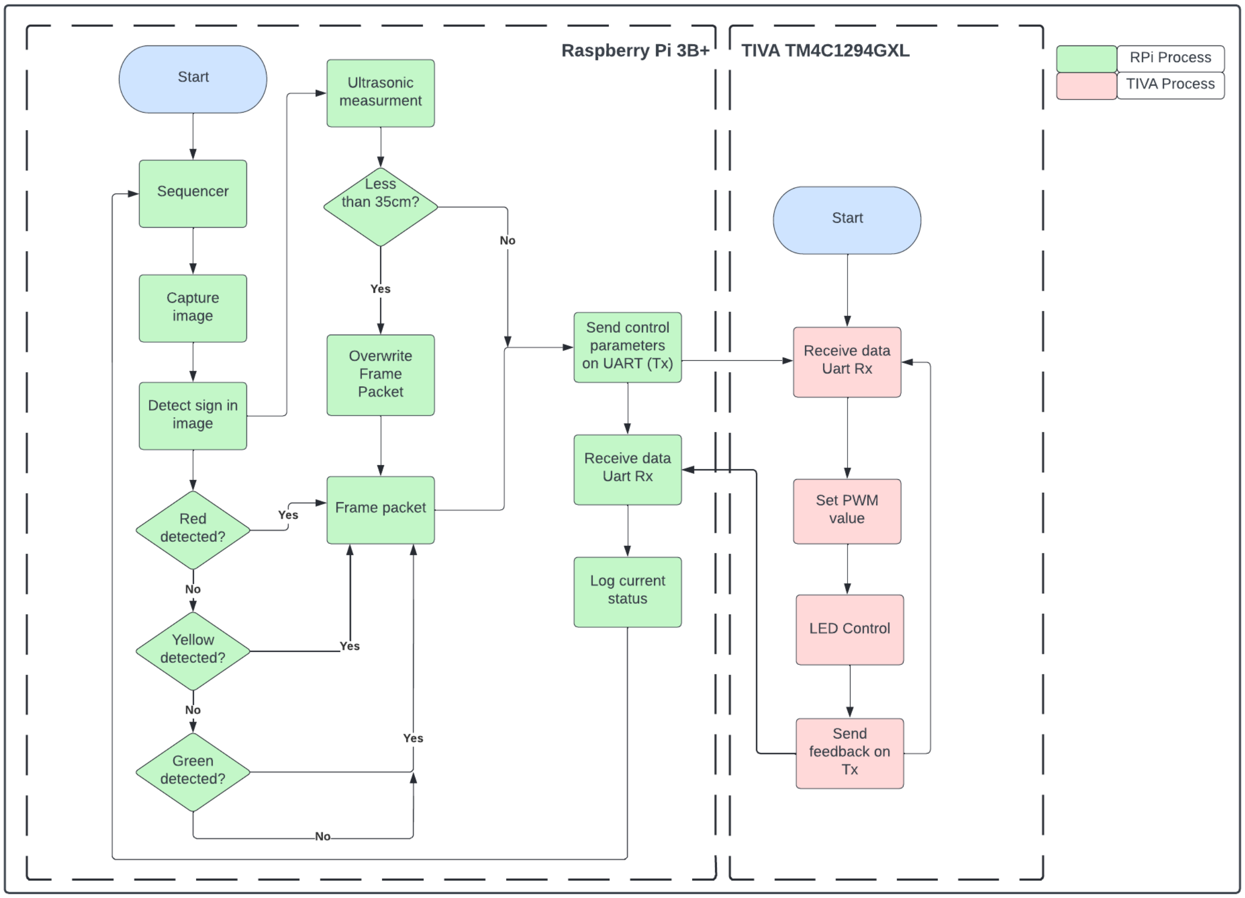


Figure 2: Software Flow Diagram

After that, we take the distance measurement from the ultrasonic sensor. If any obstacle is detected, then it will overwrite the data packet created by the image processing algorithm as a hint to the actuating microcontroller to perform immediate braking. If there is no obstacle detected, the Raspberry Pi will send the traffic light data over to TIVA using UART communication. Once the TIVA board receives the message, it will take corresponding steps to perform actuation. Once the actuation begins, the TIVA board will send an acknowledgement back to the Raspberry Pi board that the actuation has been successfully achieved. After reception of the acknowledgement message, the Raspberry Pi will log all the timings for services. This process repeats indefinitely.

[C] Data Flow Diagram

Figure 3 explains how data flows in our system. Input data is obtained from the 2 sensors - camera and ultrasonic sensor. The frame captured is passed to an image processing algorithm where the traffic light is detected. This algorithm first detects a circle and then detects the color. This data along with ultrasonic data is transferred over UART to the TIVA board. The reason we are doing this is that in a real car, there are multiple microcontrollers with multiple sensors interfaced. These controllers communicate with each other and pass data using the CAN protocol. To simulate such a behavior we have implemented this communication. To simplify stuff, we have used the UART protocol.

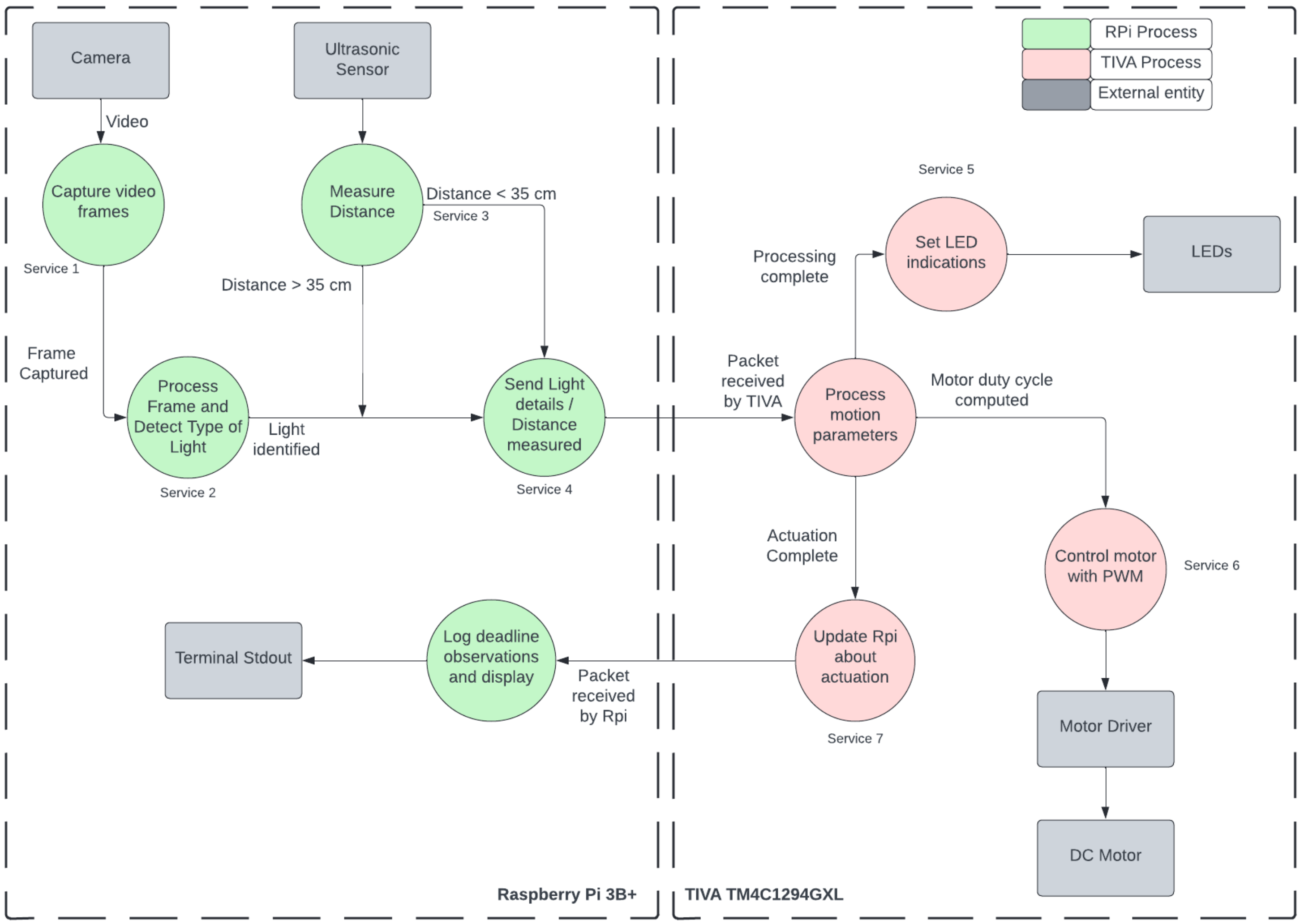


Figure 3: Data Flow Diagram

On the TIVA board, we are controlling the speed of the motor and color of the LEDs based on the data received from the RPI. Once the actuation is initiated the TIVA board sends a feedback message back to the RPI. Based on this message RPI does the time calculation.

Real Time Requirements:

For a real life system like ours, we needed practical data to determine our deadlines. According to the National Safety Council, a lightweight passenger car traveling at 55mph can come to a halt in about 200 feet. For prototyping purposes, we scaled these numbers by a factor of 250 which means that a car going at a speed of 9.83 cm/sec requires 24 cms to stop. The distance of our camera from the signal is 35cm. At the above speed the car would require 3.55 seconds to cover this distance. Out of this time, 2.479 seconds will be spent in braking. Hence we have 1.079 seconds left for our services to execute.

Real-Time Services - We have a total of 7 services each of which are explained below.

Service 1: Frame Capture
This service will capture the frames from the camera using cv::VideoCapture.read() function. This function returns a numeric multidimensional array of the data that is extracted from the frame captured and stores it in a cv::Mat variable. We are also using a flag to check whether the frame was acquired successfully. If the frame acquisition fails, we will halt the system by entering an infinite loop.

Service 2: Processing image and traffic-light detection
This service will perform a color detection and contour detection to check whether a light has been recognized or not. We are capturing the images in BGR format, and converting them into HSV format. In OpenCV, object and color detection is usually done by converting the format of image to HSV. Once the mask of HSV format is created for the captured frame, we then perform contour detection to identify circles present in the image. As soon as the algorithm detects a light, we exit the processing loop and create a data packet which stores the information about the type of light identified. If the algorithm does not detect any traffic light, then it will store a zero value in the data packet.

Service 3: Ultrasonic Distance Measurement
The service executing the ultrasonic sensor functionality implements the ultrasonic sensor driver code for reading distance values calculated in cm scale. The driver code is implemented in such a way that a 10us pulse train is sent from the output trigger pin. These pulses are received on the input echo pin after a certain time and based upon this delay and assuming the speed of sound as 34300cm/sec we calculate the distance. The value of distance is then provided inside our ultrasonic distance measurement service thread using get_distance() API which we have implemented. If the return value of this function is less than or equal to 35cm, the stop signal packet is sent to Tiva overriding the data from camera, as the motor should be stopped irrespective of the traffic light status in case an obstacle is encountered.

Service 4: UART communication RPI
This service is dedicated for transferring data from the Raspberry Pi to the TIVA board. The data to be transferred is obtained from Service 1 and Service 2. We have not used libraries on the RPI end. Since we know that in a linux based environment all devices are treated as files. Thus we use this mechanism coupled with system calls like open, write and read on the /dev/ttyS0 file to transfer the data. On the TIVA end this data is received along with an interrupt generation. Within this interrupt we write the data to the message queue.

Service 5: Speed control of motors
According to the value received through UART from the raspberry pi, the speed of the motor is controlled. For green light a PWM value of 60% is applied to the motor, for yellow light 40% is applied to slow it down and for red light and <30cm udm value 0% is applied for immediate braking.

Service 6: LED color control
Depending on the data obtained from the RPI, we change the color of the LED. This is done by simply sending a GPIO signal to the pin.

Service 7: UART communication TIVA
Once the actuation is initiated on the TIVA end, it sends a success message back to the RPI over UART. This service handles this communication.

Deadline and WCET
If we consider 1 second as the deadline for our services, there is a chance that the system might miss its deadline. It is possible that while the image is being processed, there is another frame available which has a different input compared to the previous one. In this case the processing of this frame will be delayed until after all the services have been completed. This delay could be significant since the image processing service is our heaviest service and takes a lot of time compared to others.

Hence we decided that we should be getting at least 2 sets of inputs (frame and distance) during the 1 second system requirement. Thus we chose a deadline of 450ms for our services. After running the services, we obtained the following worst case execution times

Service	Name	CPU	WCET Ci	Deadline Di (ms)	Period Ti (ms)
S1	Image capture	RPI	18 ms	450	450
S2	Sign detection	RPI	181 ms	450	450
S3	Distance measurement	RPI	112 ms	450	450
S4	UART communication	RPI	10 ms	450	450
S5	Motor Control	TIVA	7 us	450	450
S6	LED Control	TIVA	9 us	450	450
S7	UART communication	TIVA	7 us	450	450

Table 1: WCET, Ti, Di table for all services

Real-Time Analysis:

We performed Cheddar Analysis for both RPI and TIVA.

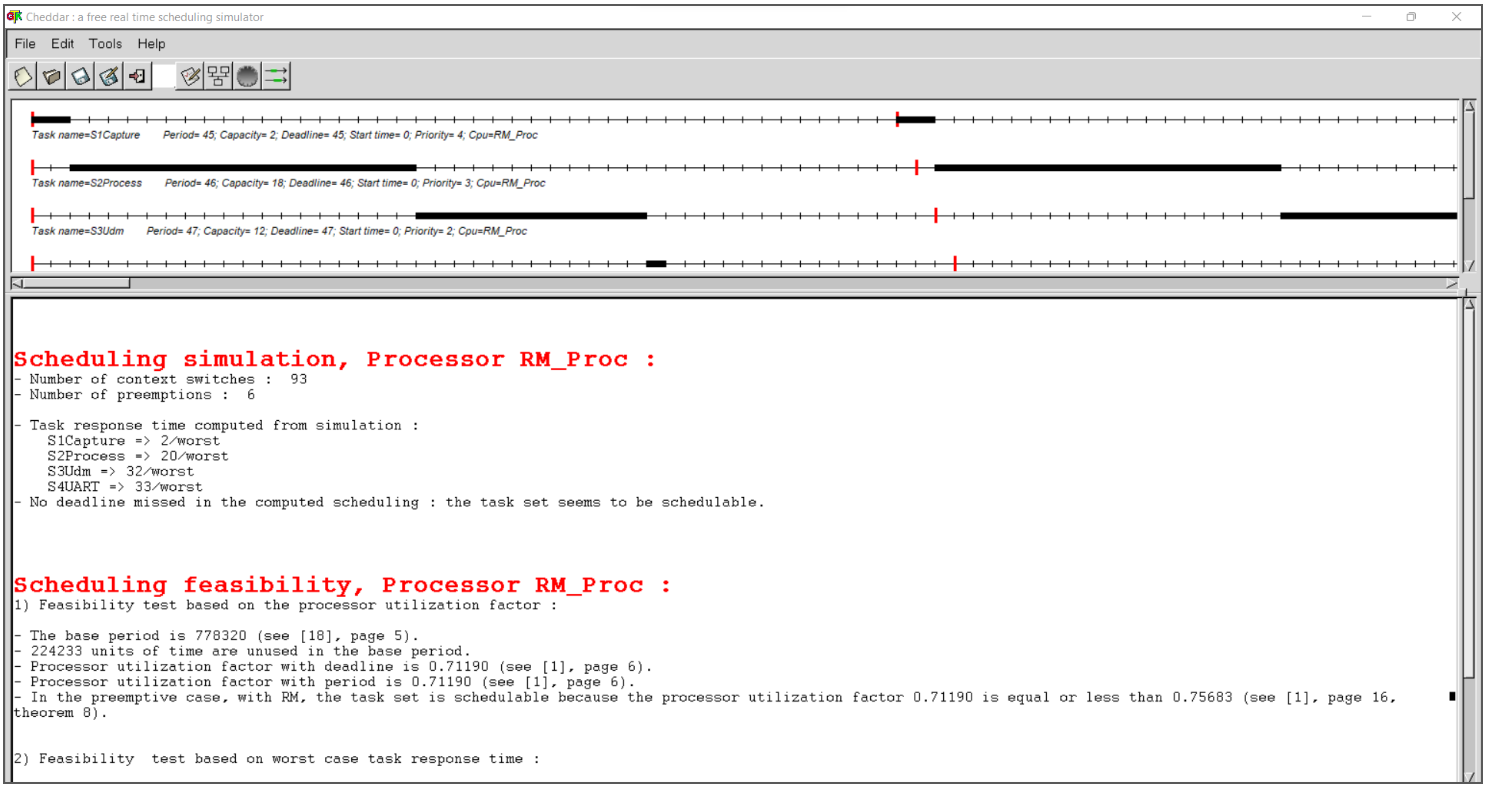


Figure 4: Cheddar analysis for RPI services

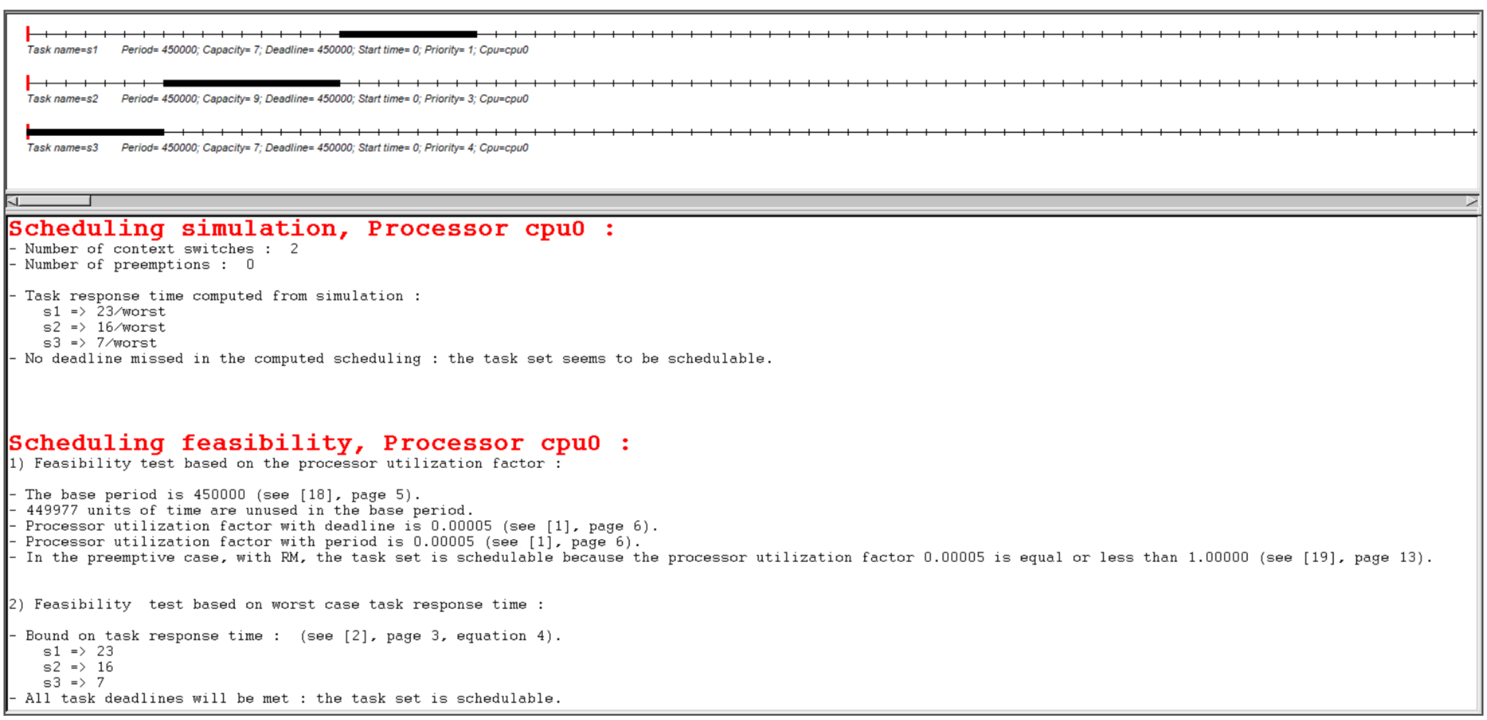


Figure 5: Cheddar analysis for TIVA services

In the case of RPI we observed that for the given service execution times and periods, the CPU utilization comes out to be 0.71 or 71%. This value is below 75.6% RM LUB condition and hence satisfies the RM LUB. The schedulability tests also prove that the service set is schedulable.

In the case of Tiva, the service set is also feasible and much below the RM LUB condition at 0.005%. Such a high utility is proven with the timing diagram which displays a lot of slack time in between the various cycles of task scheduling. The task set is also shown to be schedulable as would be the case due to very low CPU utilization.

CPU Utilization for RPi:

$$U_{RPi} = \frac{C1}{T1} + \frac{C2}{T2} + \frac{C3}{T3} + \frac{C4}{T4}$$

$$U_{RPi} = \frac{18}{450} + \frac{181}{450} + \frac{112}{450} + \frac{10}{450}$$

$$U_{RPi} = \frac{321}{450} = 0.71333$$

RM LUB:

$$U_{RPi} = 4 \times (2^{\frac{1}{4}} - 1) = 0.75683$$

$$U_{TIVA} = 3 \times (2^{\frac{1}{3}} - 1) = 0.77976$$

CPU Utilization for TIVA:

$$U_{TIVA} = \frac{C1}{T1} + \frac{C2}{T2} + \frac{C3}{T3}$$

$$U_{TIVA} = \frac{0.007}{450} + \frac{0.009}{450} + \frac{0.007}{450}$$

$$U_{TIVA} = \frac{0.023}{450} = 0.0000511$$

Figure 6: Manual Calculation of CPU Utilization

Scheduling point completion point test results:
For the above screenshots of cheddar simulations we can see that for both the case of Tiva and Rpi we got the service sets as schedulable for completion point and scheduling point tests conducted on the cheddar. The utilization is also below the RM LUB condition hence the system **satisfies the necessary and sufficient condition**.

Safety margin analysis:
For the case of services on Raspberry pi, the CPU utilization comes at 71%, hence the utilization is under RM LUB condition for 4 services and also has sufficient margin of around 29%. For the various services on Rpi, the WCET is also below the expected values for example 18ms for image capture where we assumed the WCET to be 50ms, hence there is margin available for this service and also other services.

For Tiva, the service utilization is only around 0.005% hence more than sufficient margin is available to add more services or even if there is any jitter in the WCET value of the services on Tiva. The reason for such a high margin on Tiva is that the period for services is assumed to be 450ms which is of the system and is much higher than the Ci value of services in the 6-9 microsecond range.

Proof of Concept with example output and Tests Completed:

```
WCET:
For S1: 18
For S2: 181
For S3: 112
For S4: 10
```

Figure 7: WCET for Tiva services

```
==Service_1 WCET time:6 us==
==Service_1 WCET time:6 us==
==Service_1 WCET time:6 us==
==Service_1 WCET time:6 us==
==Service_1 WCET time:6 us==
==Service_1 WCET time:6 us==
==Service_3 WCET time:7 us==
==Service_1 WCET time:6 us==
==Service_2 WCET time:7 us==
==Service_1 WCET time:6 us==
==Service_1 WCET time:9 us==
==Service_1 WCET time:9 us==
==Service_1 WCET time:9 us==
==Service_1 WCET time:9 us==
==Service_1 WCET time:9 us==
==Service_3 WCET time:7 us==
==Service_3 WCET time:7 us==
```

Figure 8: WCET Tiva services

The above snapshot shows the WCET values obtained for both the boards each running their respective services for a duration of upto 5 minutes. For Tiva, we can see multiple prints of WCET as the corresponding line is printed for a service only when the new WCET is greater than the last WCET. Thus we see multiple print outputs for service 1 and 3 as their WCETs are changing with time and not for service 2.

```
------------------------------------------------------
Red Light!
Distance: 48.674 cm
Service 1: Frame Capture: 0 sec 12 msec
Service 2: Process Image: 0 sec 154 msec
Service 3: Measure Distance: 0 sec 105 msec
Service 4: Send Data: 0 sec 6 msec
Total Time: 0 sec 279 msec
Deadline Met!
------------------------------------------------------
```

Figure 9: Logging result for red light-Test case 1(Red light).

```
------------------------------------------------------
Yellow Light!
Distance: 46.6991 cm
Service 1: Frame Capture: 0 sec 12 msec
Service 2: Process Image: 0 sec 153 msec
Service 3: Measure Distance: 0 sec 105 msec
Service 4: Send Data: 0 sec 7 msec
Total Time: 0 sec 278 msec
Deadline Met!
------------------------------------------------------
```

Figure 10: Logging result for yellow light-Test case 2(Yellow light).

```
-------------------------------------------------------
Green Light!
Distance: 67.8631 cm
Service 1: Frame Capture: 0 sec 9 msec
Service 2: Process Image: 0 sec 159 msec
Service 3: Measure Distance: 0 sec 106 msec
Service 4: Send Data: 0 sec 7 msec
Total Time: 0 sec 281 msec
Deadline Met!
-------------------------------------------------------
```

Figure 11: Logging result for green light-Test case 3 (Green light)

The above 3 screenshots show the valid traffic lights captured case and their corresponding logging threads. The execution times for each task are almost equal for all the three cases. The deadline met criteria is shown based on the weather acknowledgement received from the raspberry pi within the deadline of 450ms. The above figures basically **show the timestamp tracing** results.

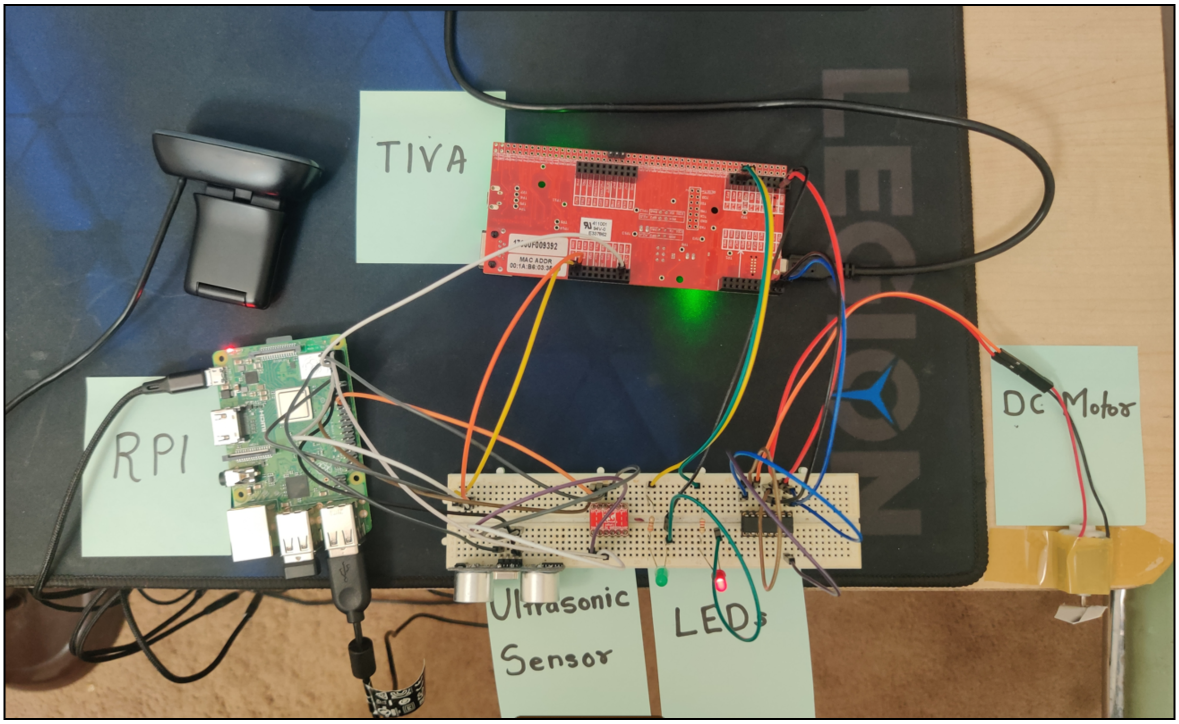
```
-------------------------------------------------------
Red Light!
Distance: 34.5428 cm
Service 1: Frame Capture: 0 sec 9 msec
Service 2: Process Image: 0 sec 131 msec
Service 3: Measure Distance: 0 sec 104 msec
Service 4: Send Data: 0 sec 7 msec
Total Time: 0 sec 252 msec
Deadline Met!
```

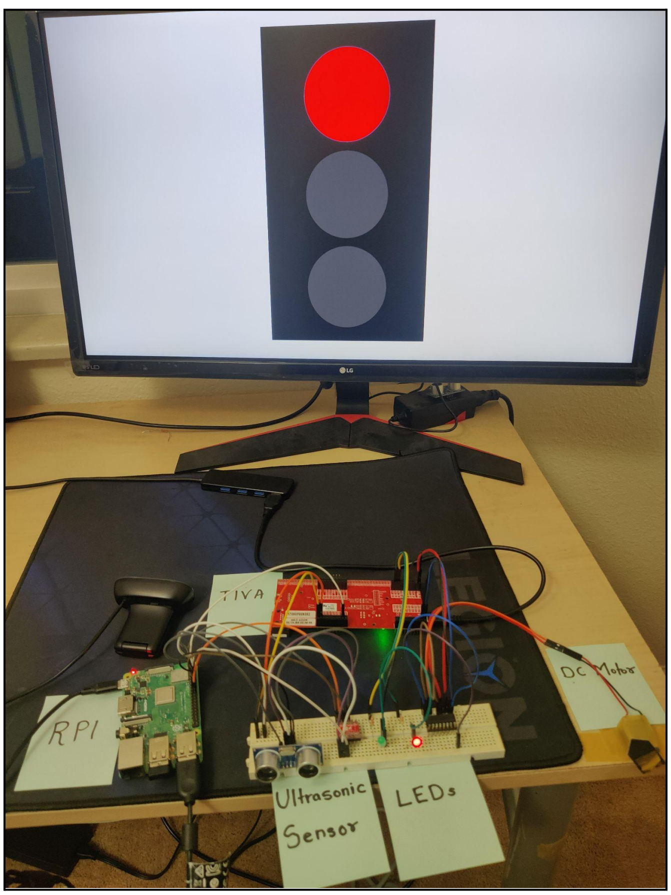
Figure 12: Logging result for obstacle-Test case 4 (UDM input)

The above snapshot shows the test case 4 for UDM input, when the distance is less than 35 cm. Since the obstacle is treated as a red light condition, red light detected is shown. The response received for this input is higher than camera service.

Hardware Setup:

The above snapshots display our project setup consisting of all the components present in the hardware block diagram along with a monitor to display the different traffic lights for the camera to demonstrate project working. As it can be seen in the video the complete system performed according to expectations.





Figures 13, 14: Project implementation setup

Conclusion:

Speed control for self-driving cars is a mission critical system and implementing such a hard real time system was a great opportunity to understand the dynamics of autonomous vehicles and the factors that are considered while designing the system. In our prototype, we successfully implemented a hard real time system utilizing FreeRTOS and Embedded Linux. For image processing purposes, we developed an algorithm which will detect contours on a processed frame which will be masked using Hue, Saturation and Vignette values. In the case of the ultrasonic sensor, the observed response time was much lesser than what we expected initially. Hence, simulating immediate braking was successfully implemented. For the TIVA board, we used FreeRTOS as an operating system and scheduled all the actuation tasks successfully. The dependency of TIVA board's actuation based on input from Raspberry Pi board was a little bit difficult to achieve due to raspberry pi's uart driver library functionality issues. Overall, all deadlines were met for the defined services. After performing feasibility analysis, Rate Monotonic Least Upper Bound for Raspberry Pi would be 0.75 and actual CPU utilization is 0.71. In the case of TIVA, the RM LUB would be 0.77, and the actual CPU utilization is 0.00005. Thus, the developed prototype is feasible using rate monotonic policy. We also performed Cheddar analysis to ensure our calculations matched the simulated output.

Key Learnings:

- Multithreaded application development using Pthreads and FreeRTOS.
- Synchronization of multiple services using semaphores and message queues.
- Basics of image processing for object, color and contour detection.

Challenges:

- High image processing time.
- UART communication between RPI and TIVA.

Future Scope:

- Detection of traffic signals including speed limits and different signs.
- Use of dedicated GPU for lower capture latency.
- Using an improved image processing model utilizing machine learning to improve accuracy in different environments.

References:

[1] Real-Time Traffic Light Signal Recognition System for a Self-driving Car Link

[2] Image contour detection algorithm Link

[3] Traffic Light Detection and Recognition for Autonomous Vehicles Link.

[4] National safety council guidelines for driving Link.

Appendix

[A] Code files for RPI

```
/*
 * @file_name      :   main.cpp
 *
 * @brief          :   RTES Final Project Code
 *
 * @author         :   Sanish Kharade
 *                     Tanmay Kothale
 *                     Vishal Raj
 *
 * @date           :   May 03, 2022
 *
 * @references     :   1. https://github.com/powergee/TrafficLightDetection
 *                     2. http://mercury.pr.erau.edu/~siewerts/cec450/code/sequencer_generic/seqgen.c
 *
 */

// This is necessary for CPU affinity macros in Linux
#define _GNU_SOURCE

/*************************************************************************/
/*                          LIBRARY FILES                                */
/*************************************************************************/
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <sched.h>
#include <sys/sysinfo.h>
#include <time.h>
#include <sys/time.h>
#include <errno.h>
#include <semaphore.h>
#include <signal.h>
#include <iostream>
#include <sstream>
#include <numeric>
#include <opencv2/opencv.hpp>
#include <pigpio.h>
#include "capture.h"
#include "process.h"
#include "uart.h"
```

```
#include "calculate.h"
#include "udm.h"

/*********************************************************************/
/*              NAMESPACE FOR IO OPERATIONS IN CPP                 */
/*********************************************************************/
using namespace std;

/*********************************************************************/
/*               PRIVATE MACROS AND DEFINES                      */
/*********************************************************************/
#define TRUE (1)
#define FALSE (0)
#define USEC_PER_MSEC (1000)
#define NANOSEC_PER_SEC (1000000000)
#define NUM_CPU_CORES (1)
#define NUM_THREADS (7+1)
#define SCHED_POLICY SCHED_FIFO
#define FIXED_DEADLINE (400.00)

void *Sequencer(void *threadp);
void *Service_1(void *threadp);
void *Service_2(void *threadp);
void *Service_3(void *threadp);
void *Service_4(void *threadp);
void *Service_5(void *threadp);

static void print_scheduler(void);

/*********************************************************************/
/*            TYPEDEFS, DATA STRUCTURES & ENUMERATIONS            */
/*********************************************************************/
typedef void* (*worker_t)(void*);
// For sched fifo this ranges from 0-99
typedef uint8_t priority_t;

//for thread callback function
typedef struct
{
   int threadIdx;
   worker_t worker;
} threadParams_t;

//enum for number of services
typedef enum service
{
   SCHEDULER,
```

```
    SERVICE_1,
    SERVICE_2,
    SERVICE_3,
    SERVICE_4,
    SERVICE_5,
    NUM_SERVICES
}service_t;

//to compute wcet and avg execution time
typedef struct service_params_s
{
    uint32_t wcet;
    uint32_t total_ci;
    double avg_ci;

}service_params_t;

/**********************************************************************/
/*                    GLOBAL VARIABLES                    */
/**********************************************************************/
priority_t rt_max_prio, rt_min_prio;            //to set priorities of tasks
pthread_t threads[NUM_THREADS];                 //array of threads
threadParams_t threadParams[NUM_THREADS];       //array to store thread parameters
static struct sched_param rt_param[NUM_THREADS];    //array to store scheduler parameters
pthread_attr_t rt_sched_attr[NUM_THREADS];      //array to store scheduler attributes

sem_t semS1, semS2, semS3, semS4, semS5;        //semaphores for synchronization

struct timeval start_time_val;                  //to set scheduler frequency

//following structure instances are used for logging purposes
struct timespec start_s1 = {0,0}, start_s2 = {0,0}, start_s3 = {0,0}, start_s4 = {0,0}, start_total = {0,0},
            end_s1 = {0,0}  , end_s2 = {0,0}  , end_s3 = {0,0}  , end_s4 = {0,0}  , end_total = {0,0},
            delta_s1 = {0,0}, delta_s2 = {0,0}, delta_s3 = {0,0}, delta_s4 = {0,0}, delta_total = {0,0};

//array of service parameter structure to store wcet and acet
service_params_t service_times[NUM_SERVICES];

//global object used to capture frames
cv::VideoCapture cap(0);

//to find contours in a given area of a frame
int min_area, max_area;

//to calculate deadlines and distance
float distance_cm = 0, computed_deadline = 0, time_to_stop_sec = 0;
double distance_udm;
```

```
uint32_t ctr=0;

/************************************************************************************
 * @brief   :   Function to print the scheduler properties
 *
 * @param   :   none
 *
 * @return  :   void
 *
 ************************************************************************************/
static void print_scheduler(void)
{
   int schedType;

   schedType = sched_getscheduler(getpid());

   switch(schedType)
   {
       case SCHED_FIFO:
           printf("Pthread Policy is SCHED_FIFO\n");
           break;
       case SCHED_OTHER:
           printf("Pthread Policy is SCHED_OTHER\n"); exit(-1);
         break;
       case SCHED_RR:
           printf("Pthread Policy is SCHED_RR\n"); exit(-1);
           break;
       default:
           printf("Pthread Policy is UNKNOWN\n"); exit(-1);
   }
}

/************************************************************************************
 * @brief   :   Function to get the priority of the service
 *
 * @param   :   service_t   -  service who's priority is to be returned
 *
 * @return  :   priority_t  -  priority of service
 *
 ************************************************************************************/
priority_t get_priority(service_t t)
{
    switch (t)
    {
        case SCHEDULER:
            return rt_max_prio;
            break;
```

```
        case SERVICE_1:
            return rt_max_prio - 1;
            break;

        case SERVICE_2:
            return rt_max_prio - 2;
            break;

        case SERVICE_3:
            return rt_max_prio - 3;
            break;

        case SERVICE_4:
            return rt_max_prio - 4;
            break;

        case SERVICE_5:
            return rt_max_prio - 5;
            break;

        default:
            printf("ERROR: No such task\n");
            return -1;
            break;
    }
}

/***************************************************************************************
 * @brief   :   Function that assigns a worker function to each thread
 *
 * @param   :   service_t t -  service who's worker needs to be assigned
 *
 * @return  :   worker_t    -  worker function for a particular service
 *
 ***************************************************************************************/
worker_t get_worker(service_t t)
{
    switch (t)
    {
        case SCHEDULER:
            return Sequencer;
            break;

        case SERVICE_1:
            return Service_1;
            break;
```

```
        case SERVICE_2:
            return Service_2;
            break;

        case SERVICE_3:
            return Service_3;
            break;

        case SERVICE_4:
            return Service_4;
            break;

        case SERVICE_5:
            return Service_5;
            break;

        default:
            cout << "Error: No such task found." << endl;
            return 0;
            break;
    }
}

/***********************************************************************************
 * @brief   :   configures the scheduler to execute all services
 *
 * @param   :   none
 *
 * @return  :   none
 *
 ***********************************************************************************/
void configure_service_scheduler(void)
{
    cpu_set_t allcpuset;
    int i;
    pid_t mainpid;
    int rc, scope;

    struct sched_param main_param;
    pthread_attr_t main_attr;

    // Work related to CPU and cores
    cout << "System has " << get_nprocs_conf() << " processors configured and " << get_nprocs() << "
available." << endl;

    CPU_ZERO(&allcpuset);
```

```
    for(i=0; i < NUM_CPU_CORES; i++)
        CPU_SET(i, &allcpuset);

    cout << "Using CPUS = "<< CPU_COUNT(&allcpuset) << " from total available." << endl;

    // Work related to services
    rt_max_prio = sched_get_priority_max(SCHED_FIFO);
    rt_min_prio = sched_get_priority_min(SCHED_FIFO);

    // Record the main application's process ID
    mainpid=getpid();

    rc=sched_getparam(mainpid, &main_param);
    main_param.sched_priority=rt_max_prio;
    rc=sched_setscheduler(getpid(), SCHED_POLICY, &main_param);
    if(rc < 0)
    {
        perror("main_param");
    }

    print_scheduler();

    pthread_attr_getscope(&main_attr, &scope);

    if(scope == PTHREAD_SCOPE_SYSTEM)
      cout << "PTHREAD SCOPE SYSTEM" << endl;
    else if (scope == PTHREAD_SCOPE_PROCESS)
      cout << "PTHREAD SCOPE PROCESS" << endl;
    else
      cout << "PTHREAD SCOPE UNKNOWN" << endl;

    cout << "rt_max_prio= " << rt_max_prio << endl;
    cout << "rt_min_prio= " << rt_min_prio << endl;
}

/*************************************************************************************
 * @brief   :   schedules all the services
 *
 * @param   :   threadp - Thread parameters
 *
 * @return  :   none
 *
 *************************************************************************************/
void *Sequencer(void *threadp)
{
    struct timeval current_time_val;
```

```
    struct timespec delay_time = {0,10000000}; // delay for 10msec, Frequency = 100 Hz

    struct timespec remaining_time;
    double current_time;
    double residual;
    int rc, delay_cnt=0;
    unsigned long long seqCnt=0;
    threadParams_t *threadParams = (threadParams_t *)threadp;

    gettimeofday(&current_time_val, (struct timezone *)0);

    cout << "Sequencer thread @ sec= " << (int)(current_time_val.tv_sec-start_time_val.tv_sec)
      << ", msec= " << (int)current_time_val.tv_usec/USEC_PER_MSEC << endl;

    do
    {
        delay_cnt=0; residual=0.0;

        do
        {
            rc=nanosleep(&delay_time, &remaining_time);

            if(rc == EINTR)
            {
                residual = remaining_time.tv_sec + ((double)remaining_time.tv_nsec /
(double)NANOSEC_PER_SEC);

                if(residual > 0.0)
                    cout << "residual= " << residual << ", sec=" << (int)remaining_time.tv_sec << ", nsec=" <<
(int)remaining_time.tv_nsec << endl;

                delay_cnt++;
            }
            else if(rc < 0)
            {
                perror("Sequencer nanosleep");
                exit(-1);
            }

        } while((residual > 0.0) && (delay_cnt < 100));

        seqCnt++;
        gettimeofday(&current_time_val, (struct timezone *)0);

        if(delay_cnt > 1)
            cout << "Sequencer looping delay " << delay_cnt << endl;
```

```
        //post the first service every 450 msec = 2.22 Hz
        if((seqCnt % 45) == 0)
        {
            sem_post(&semS1);
        }

    } while(1);

    pthread_exit((void *)0);
}

/*************************************************************************************
 * @brief   :   Captures the frame
 *
 * @param   :   threadp - Thread parameters
 *
 * @return  :   none
 *
**************************************************************************************/
void *Service_1(void *threadp)
{

    while(1)
    {
        sem_wait(&semS1);                              //wait for sequencer to post the semaphore

        clock_gettime(CLOCK_REALTIME, &start_total);    //record start time of total execution
        clock_gettime(CLOCK_REALTIME, &start_s1);       //record start time for this service
        capture_frame(cap);                             //capture the frame
        clock_gettime(CLOCK_REALTIME, &end_s1);         //record end time for this service

        sem_post(&semS2);                              //post semaphore for next service
    }

    pthread_exit((void *)0);
}

/*************************************************************************************
 * @brief   :   Performs UART transmission and reception
 *
 * @param   :   none
 *
 * @return  :   none
 *
**************************************************************************************/
void test(void)
{
```

```
    int fd;
    int rv = 0;
    char s = getColor();

    /*Override Image processing algorithm*/
    if(distance_udm < 35.0)
        s = RED;

    char r;

    fd = open("/dev/ttyS0", O_RDWR);
    if (fd == -1)
        perror("error: open");

    rv = write(fd, &s, 1);
    if (rv == -1)
        perror("error: write");

    rv = read(fd, &r, 1);
    if (rv == -1)
        perror("error: read");

    close(fd);

}

/*************************************************************************************
 * @brief   :   Processes the image and checks for traffic light
 *
 * @param   :   threadp - Thread parameters
 *
 * @return  :   none
 *
 *************************************************************************************/
void *Service_2(void *threadp)
{

    while(1)
    {
        sem_wait(&semS2);

        clock_gettime(CLOCK_REALTIME, &start_s2);
        process_image(min_area, max_area);
        clock_gettime(CLOCK_REALTIME, &end_s2);

        sem_post(&semS3);
    }
```

```
    pthread_exit((void *)0);
}

/*****************************************************************************************
 * @brief   :   Measures distance from Ultrasonic sensor
 *
 * @param   :   threadp - Thread parameters
 *
 * @return  :   none
 *
*****************************************************************************************/
void *Service_3(void *threadp)
{

    while(1)
    {
        sem_wait(&semS3);

        //changes by Tanmay
        clock_gettime(CLOCK_REALTIME, &start_s3);

        /*UDM Test*/
        distance_udm = get_distance();

        clock_gettime(CLOCK_REALTIME, &end_s3);

        sem_post(&semS4);
    }

    pthread_exit((void *)0);
}

/*****************************************************************************************
 * @brief   :   Sends the data packet to TIVA using UART
 *
 * @param   :   threadp - Thread parameters
 *
 * @return  :   none
 *
*****************************************************************************************/
void *Service_4(void *threadp)
{
    char c;
    while(1)
    {
        sem_wait(&semS4);
```

```
        clock_gettime(CLOCK_REALTIME, &start_s4);
        test();
        clock_gettime(CLOCK_REALTIME, &end_s4);

        clock_gettime(CLOCK_REALTIME, &end_total);

        sem_post(&semS5);
    }

    pthread_exit((void *)0);
}

/************************************************************************************
 * @brief   :   Logs the timings for all services
 *
 * @param   :   threadp - Thread parameters
 *
 * @return  :   none
 *
 ************************************************************************************/
void *Service_5(void *threadp)
{
    static uint32_t total_time_msec;
    char c;

    uint32_t s1_msec=0, s2_msec=0, s3_msec=0, s4_msec=0;

    while(1)
    {
        sem_wait(&semS5);

        ctr++;

        delta_t(&end_s1, &start_s1, &delta_s1);

        delta_t(&end_s2, &start_s2, &delta_s2);

        delta_t(&end_s3, &start_s3, &delta_s3);

        delta_t(&end_s4, &start_s4, &delta_s4);

        delta_t(&end_total, &start_total, &delta_total);

        s1_msec = delta_s1.tv_nsec/NSEC_PER_MSEC;
        s2_msec = delta_s2.tv_nsec/NSEC_PER_MSEC;
        s3_msec = delta_s3.tv_nsec/NSEC_PER_MSEC;
```

```
s4_msec = delta_s4.tv_nsec/NSEC_PER_MSEC;


service_times[SERVICE_1].total_ci += s1_msec;
service_times[SERVICE_2].total_ci += s2_msec;
service_times[SERVICE_3].total_ci += s3_msec;
service_times[SERVICE_4].total_ci += s4_msec;

if(s1_msec > service_times[SERVICE_1].wcet)
{
   service_times[SERVICE_1].wcet = s1_msec;
}

if(s2_msec > service_times[SERVICE_2].wcet)
{
   service_times[SERVICE_2].wcet = s2_msec;
}

if(s3_msec > service_times[SERVICE_3].wcet)
{
   service_times[SERVICE_3].wcet = s3_msec;
}

if(s4_msec > service_times[SERVICE_4].wcet)
{
   service_times[SERVICE_4].wcet = s4_msec;
}

cout << "Distance: " << distance_udm << " cm" << endl;

cout << "Service 1: Frame Capture: " << delta_s1.tv_sec << " sec "
    << s1_msec << " msec"
    << endl;

cout << "Service 2: Process Image: " << delta_s2.tv_sec << " sec "
    << s2_msec << " msec"
    << endl;

cout << "Service 3: Measure Distance: " << delta_s3.tv_sec << " sec "
    << s3_msec << " msec"
    << endl;

cout << "Service 4: Send Data: " << delta_s4.tv_sec << " sec "
    << s4_msec << " msec"
    << endl;

cout << "Total Time: " << delta_total.tv_sec << " sec "
```

```
            << delta_total.tv_nsec/NSEC_PER_MSEC << " msec"
            << endl;

        total_time_msec = (uint32_t)((delta_total.tv_sec*1000) + (delta_total.tv_nsec/NSEC_PER_MSEC));

        if (total_time_msec < FIXED_DEADLINE)
        {
            cout << "Deadline Met!" << endl;
        }
        else
        {
            cout << "Deadline missed!" << endl;
        }

        cout << "-----------------------------------------------------" << endl;

    }

    pthread_exit((void *)0);
}

/*****************************************************************************************
 * @brief   :   Configure all services as threads and assign a worker function
 *
 * @param   :   none
 *
 * @return  :   none
 *
*****************************************************************************************/
static void configure_services(void)
{
    int i, rc;
    cpu_set_t threadcpu;

    for(i=0; i < NUM_SERVICES; i++)
    {

        CPU_ZERO(&threadcpu);
        CPU_SET(3, &threadcpu);

        rc=pthread_attr_init(&rt_sched_attr[i]);
        rc=pthread_attr_setinheritsched(&rt_sched_attr[i], PTHREAD_EXPLICIT_SCHED);
        rc=pthread_attr_setschedpolicy(&rt_sched_attr[i], SCHED_FIFO);


        rt_param[i].sched_priority = get_priority((service_t)i);
        pthread_attr_setschedparam(&rt_sched_attr[i], &rt_param[i]);
```

```
        threadParams[i].threadIdx = i;
        threadParams[i].worker = get_worker((service_t)i);

        rc=pthread_create(&threads[i],              // pointer to thread descriptor
                  &rt_sched_attr[i],          // use specific attributes
                  threadParams[i].worker,     // thread function entry point
                  (void *)&(threadParams[i])  // parameters to pass in
                  );
        if(rc < 0)
           printf("pthread_create for service %d\n", i);
        else
           printf("pthread_create successful for service %d\n", i);

    }

    printf("Service threads will run on %d CPU cores\n", CPU_COUNT(&threadcpu));

}

/*****************************************************************************************
 * @brief   :   Initialize all semaphores for synchronization
 *
 * @param   :   none
 *
 * @return  :   none
 *
 *****************************************************************************************/
void semaphores_init()
{
    if (sem_init (&semS1, 0, 0)) { printf ("Failed to initialize S1 semaphore\n"); exit (-1); }
    if (sem_init (&semS2, 0, 0)) { printf ("Failed to initialize S2 semaphore\n"); exit (-1); }
    if (sem_init (&semS3, 0, 0)) { printf ("Failed to initialize S3 semaphore\n"); exit (-1); }
    if (sem_init (&semS4, 0, 0)) { printf ("Failed to initialize S4 semaphore\n"); exit (-1); }
    if (sem_init (&semS5, 0, 0)) { printf ("Failed to initialize S5 semaphore\n"); exit (-1); }

}

/*****************************************************************************************
 * @brief   :   initialize the camera
 *
 * @param   :   none
 *
 * @return  :   none
 *
 *****************************************************************************************/
void camera_init()
```

```
{
    //cv::VideoCapture cap(0);
    double width = cap.get(cv::CAP_PROP_FRAME_WIDTH);
    double height = cap.get(cv::CAP_PROP_FRAME_HEIGHT);
    cv::namedWindow("Result");

    //int min_area, max_area;
    cv::createTrackbar("Minimum Area", "Result", &min_area, 100000);
    cv::createTrackbar("Maximum Area", "Result", &max_area, 100000);
    cv::setTrackbarPos("Minimum Area", "Result", 1000);
    cv::setTrackbarPos("Maximum Area", "Result", 100000);
}

/*************************************************************************************
 * @brief   :   handles signal (in this case, SIGTERM)
 *
 * @param   :   sig_no - macro defined for a specific signal
 *
 * @return  :   none
 *
 *************************************************************************************/
void sighandler(int sig_no)
{

    cout<<"SIGTERM detected!"<<endl;

    gpioTerminate();

    cout << "Average execution times: " << endl;

    cout << "For S1: " << service_times[SERVICE_1].total_ci/ctr << endl;
    cout << "For S2: " << service_times[SERVICE_2].total_ci/ctr << endl;
    cout << "For S3: " << service_times[SERVICE_3].total_ci/ctr << endl;
    cout << "For S4: " << service_times[SERVICE_4].total_ci/ctr << endl;


    cout << "\nWorst Case Execution Time: " << endl;

    cout << "For S1: " << service_times[SERVICE_1].wcet << endl;
    cout << "For S2: " << service_times[SERVICE_2].wcet << endl;
    cout << "For S3: " << service_times[SERVICE_3].wcet << endl;
    cout << "For S4: " << service_times[SERVICE_4].wcet << endl;

    exit(0);
}

/*************************************************************************************
```

```
 * @brief   :   initializes ultrasonic sensor
 *
 * @param   :   none
 *
 * @return  :   none
 *
************************************************************************************/
void udm_init()
{
        if(gpioInitialise() < 0){
            cout << "pigpio initialisation failed" << endl;
            signal(SIGINT, sighandler);
            exit(1);
        }
        else{
            signal(SIGINT, sighandler);

            cout << "pigpio initialisation ok" << endl;

            init_udm();
        }
}

/************************************************************************************
 * @brief   :   application entry point
 *
 * @param   :   argc - number of command line arguments
 *              argv - command line argument as string
 *
 * @return  :   zero
 *
************************************************************************************/
int main(int argc, char** argv)
{
    if (argc < 2)
    {
        cout << "Please provide distance. Do sudo ./main <distance_in_cm>" << endl;
        cout << "Insufficient arguments. Exiting..." << endl;
        exit (EXIT_FAILURE);
    }
    else
    {
        sscanf(argv[1], "%f", &distance_cm);
    }

    cout << "----------------------------------------------------------------------" << endl;
```

```
    cout << "Assuming red light is detected " << distance_cm << " cm from Car's current position:" << endl;
    cout << "Current speed of car: " << CURRENT_SPEED_OF_CAR << " cm/sec" << endl;
    time_to_stop_sec = time_to_stop_in_sec (CURRENT_SPEED_OF_CAR, distance_cm);
    cout << "Total time required for car to come to a full Stop: " << time_to_stop_sec << " seconds" << endl;

    computed_deadline = compute_deadline_to_complete_tasks (distance_cm);
    cout << "Computed deadline for all tasks: " << computed_deadline/2 << " seconds" << endl;
    cout << "---------------------------------------------------------------------" << endl;

    int i;

    camera_init();

    udm_init();

    configure_service_scheduler();

    configure_services();

    //should never come here
    for(i=0;i<NUM_SERVICES;i++)
    {
        pthread_join(threads[i], NULL);
    }

   return 0;
}



/*
 * @file_name       :   capture.cpp
 *
 * @brief           :   RTES Final Project Code
 *
 * @author          :   Sanish Kharade
 *                      Tanmay Kothale
 *                      Vishal Raj
 *
 * @date            :   May 03, 2022
 *
 * @references      :   1. https://github.com/powergee/TrafficLightDetection
 *                      2. http://mercury.pr.erau.edu/~siewerts/cec450/code/sequencer_generic/seqgen.c
 *
 */

/**********************************************************************/
```

```
/*                      LIBRARY FILES                         */
/*****************************************************************************/
#include <iostream>
#include <numeric>
#include <opencv2/opencv.hpp>
#include <time.h>
#include "process.h"
#include "capture.h"

/*****************************************************************************/
/*              NAMESPACE FOR IO OPERATIONS IN CPP                   */
/*****************************************************************************/
using namespace std;

/*****************************************************************************/
/*                EXTERN GLOBAL VARIABLES                         */
/*****************************************************************************/
cv::Mat frame;
bool frame_flag = false;

/*see documentation in capture.h*/
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)
      {
         //printf("nanosec greater at stop than start\n");
         delta_t->tv_sec = 0;
         delta_t->tv_nsec = dt_nsec;
      }

      else if(dt_nsec > NSEC_PER_SEC)
      {
            //printf("nanosec overflow\n");
      delta_t->tv_sec = 1;
      delta_t->tv_nsec = dt_nsec-NSEC_PER_SEC;
      }

      else // dt_nsec < 0 means stop is earlier than start
      {
```

```
            printf("stop is earlier than start\n");
            return(ERROR);
        }
    }

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

      //printf("dt more than 1 second\n");
        if(dt_nsec >= 0 && dt_nsec < NSEC_PER_SEC)
        {
            //printf("nanosec greater at stop than start\n");
            delta_t->tv_sec = dt_sec;
            delta_t->tv_nsec = dt_nsec;
        }

        else if(dt_nsec > NSEC_PER_SEC)
        {
            //printf("nanosec overflow\n");
            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
        {
          //printf("nanosec roll over\n");
          delta_t->tv_sec = dt_sec-1;
          delta_t->tv_nsec = NSEC_PER_SEC + dt_nsec;
        }
    }

    return(OK);
}

/*see documentation in capture.h*/
void capture_frame(cv::VideoCapture cap)
{

   frame_flag = cap.read(frame);
   if (!frame_flag)
   {
      cout << "Frame capture failed! Entering infinite loop...." << endl;
      while(1);
   }
}
```

```
/*
 * @file_name      :   process.cpp
 *
 * @brief          :   RTES Final Project Code
 *
 * @author         :   Sanish Kharade
 *                     Tanmay Kothale
 *                     Vishal Raj
 *
 * @date           :   May 03, 2022
 *
 * @references     :   1. https://github.com/powergee/TrafficLightDetection
 *
 */

/*******************************************************************/
/*                      LIBRARY FILES                              */
/*******************************************************************/
#include <iostream>
#include <numeric>
#include <opencv2/opencv.hpp>
#include "process.h"
#include "capture.h"

/*******************************************************************/
/*                      GLOBAL VARIABLES                           */
/*******************************************************************/
color_t color = NONE;

/*******************************************************************/
/*              NAMESPACE FOR IO OPERATIONS IN CPP                 */
/*******************************************************************/
using namespace std;

/*see documentation in process.h*/
cv::Mat mask_img(cv::Mat& frame, int h, int error, int s_min, int v_min)
{
   cv::Mat hsv_img;
   cv::cvtColor(frame, hsv_img, cv::COLOR_BGR2HSV);
   int lowH = (h-error >= 0) ? h-error : h-error+180;
   int highH = (h+error <= 180) ? h+error : h+error-180;

   std::vector<cv::Mat> channels;
   cv::split(hsv_img, channels);
```

```
    if (lowH < highH)
        cv::bitwise_and(lowH <= channels[0], channels[0] <= highH, channels[0]);
    else
        cv::bitwise_or(lowH <= channels[0], channels[0] <= highH, channels[0]);

    channels[1] = channels[1] > s_min;
    channels[2] = channels[2] > v_min;

    cv::Mat mask = channels[0];
    for (int i = 1; i < 3; ++i)
        cv::bitwise_and(channels[i], mask, mask);

    cv::Mat grey = cv::Mat::zeros(mask.rows, mask.cols, CV_8U);
    for (int row = 0; row < mask.rows; ++row)
    {
        for (int col = 0; col < mask.cols; ++col)
        {
            auto v1 = channels[0].data[row*mask.cols + col];
            auto v2 = channels[1].data[row*mask.cols + col];
            auto v3 = channels[2].data[row*mask.cols + col];
            grey.data[row*mask.cols + col] = (v1 && v2 && v3 ? 255 : 0);
        }
    }

    return grey;
}

/*see documentation in process.h*/
bool isConvex(Contour& c, double area)
{
    Contour hull_cntr;
    cv::convexHull(c, hull_cntr);
    double hull_area = cv::moments(hull_cntr).m00;
    return abs(hull_area - area) / area <= 0.1;
}

/*see documentation in process.h*/
Shape labelPolygon(Contour& c, double area)
{
    double peri = cv::arcLength(c, true);
    Contour approx;
    cv::approxPolyDP(c, approx, 0.02*peri, true);

    if (approx.size() > 7 && isConvex(c, area))
        return Shape::Circle;

    return Shape::Undefined;
```

```
}

/*see documentation in process.h*/
std::vector<Contour> findShapes(Shape shapeToFind, cv::Mat& grey, int min_area, int max_area)
{
   std::vector<Contour> contours;
   cv::findContours(grey, contours, cv::RETR_LIST, cv::CHAIN_APPROX_SIMPLE);
   std::vector<Contour> found;

   for (auto c : contours)
   {
      cv::Moments m;
      m = cv::moments(c);

      if (m.m00 != 0 && min_area <= m.m00 && m.m00 <= max_area)
      {
         Shape shape = labelPolygon(c, m.m00);
         if (shape == shapeToFind)
            found.push_back(c);
      }
   }

   return found;
}

/*see documentation in process.h*/
void setColor(int c)
{

   switch(c)
   {
      case 1:
         color = RED;
         break;

      case 2:
         color = YELLOW;
         break;

      case 3:
         color = GREEN;
         break;

      default:
         color = NONE;
         break;
   }
```

```
}

/*see documentation in process.h*/
color_t getColor(void)
{
   return color;
}

/*see documentation in process.h*/
void process_image(int min_area, int max_area)
{
    cv::Mat redMasked = mask_img(frame, 0, 15, 180, 128);
    cv::Mat yellowMasked = mask_img(frame, 30, 15, 120, 60);
    cv::Mat greenMasked = mask_img(frame, 60, 15, 90, 60);

    const static std::string captions[] = { "Red Light!", "Yellow Light!", "Green Light!"};
    const static cv::Scalar colors[] = { cv::Scalar(0, 0, 255), cv::Scalar(131, 232, 252), cv::Scalar(0, 255, 0)};

    std::vector<Contour> found[] = {
        findShapes(Shape::Circle, redMasked, min_area, max_area),
        findShapes(Shape::Circle, yellowMasked, min_area, max_area),
        findShapes(Shape::Circle, greenMasked, min_area, max_area),
    };

    int i;
    for ( i= 0; i < 3; ++i)
    {
        if (!found[i].empty())
        {
            cout << captions[i] << endl;
            setColor(i+1);
            break;
        }
    }
   if(i == 3)
   {
      setColor(0);
   }
}



/*
 * @file_name      :   calculate.cpp
 *
 * @brief          :   RTES Final Project Code
 *
```

```
 * @author       :  Sanish Kharade
 *                 Tanmay Kothale
 *                 Vishal Raj
 *
 * @date         :  May 03, 2022
 *
 */

/*LIBRARY FILES*/
#include "calculate.h"

/*see documentation in calculate.h*/
float time_to_stop_in_sec(float speed, float distance)
{
	float deadline;

	deadline = distance/speed;

	return deadline;
}

/*see documentation in calculate.h*/
float compute_deadline_to_complete_tasks (float distance)
{
	float deadline, time_to_stop, deadline_for_tasks;

	time_to_stop = time_to_stop_in_sec(CURRENT_SPEED_OF_CAR, DISTANCE_REQD_TO_STOP);

	deadline = time_to_stop_in_sec(CURRENT_SPEED_OF_CAR, distance);

	deadline_for_tasks = deadline - time_to_stop;

	return deadline_for_tasks;
}




/*
 * @file_name     :  udm.cpp
 *
 * @brief         :  RTES Final Project Code
 *
 * @author        :  Sanish Kharade
 *                 Tanmay Kothale
 *                 Vishal Raj
```

```
 *
 * @date          :   May 03, 2022
 *
 * @references              :       https://github.com/undqurek/RaspberryPI_UltrasonicRangeFinder
 */

/*********************************************************************/
/*                       LIBRARY FILES                              */
/*********************************************************************/
#include <pigpio.h>
#include <iostream>
#include <time.h>
#include "udm.h"
#include <signal.h>
#include <stdlib.h>
#include <sys/time.h>
#include <unistd.h>

/*********************************************************************/
/*                PRIVATE MACROS AND DEFINES                        */
/*********************************************************************/
#define TRIG  23
#define ECHO           24
#define speed  17150

/*********************************************************************/
/*              NAMESPACE FOR IO OPERATIONS IN CPP                  */
/*********************************************************************/
using namespace std;

//to get current time
struct timeval tv;

double get_instant()
{
        gettimeofday(&tv, NULL);

   return (double)tv.tv_sec + (double)tv.tv_usec * 0.000001;
}

/*see documentation in udm.h*/
void init_udm()
{
   gpioSetMode(TRIG, PI_OUTPUT);
   gpioSetMode(ECHO, PI_INPUT);
}
```

```
/***************************************************************************************
 * @brief   :   creates a delay
 *
 * @param   :   value  -        time for which delay is requested
 *              limit   -        max limit is 1 msec
 *
 * @return  :   true on success, false on failure
 *
 ***************************************************************************************/
bool delay(int value, int limit = 1000000)
{
   for(int i = 0; gpioRead(ECHO) == value; ++i)
   {
      if(i >= limit)
         return false;
   }

   return true;
}

/*see documentation in udm.h*/
double get_distance()
{
   gpioWrite(TRIG, 0);
   usleep(100000);
   gpioWrite(TRIG, 1);
   usleep(10);
   gpioWrite(TRIG, 0);

   if(delay(0))
   {
      double start_pulse = get_instant();

      if(delay(1))
      {
         double end_pulse = get_instant();

         double time = end_pulse - start_pulse;
         double distance = time * speed;

         return distance;
      }
   }

   return 0.0 / 0.0;
}
```

```
/*
 * @file_name      :   capture.h
 *
 * @brief          :   RTES Final Project Code
 *
 * @author         :   Sanish Kharade
 *                     Tanmay Kothale
 *                     Vishal Raj
 *
 * @date           :   May 03, 2022
 *
 */

#ifndef _CAPTURE_H_
#define _CAPTURE_H_

/*************************************************************************/
/*                        LIBRARY FILES                                  */
/*************************************************************************/
#include <iostream>
#include <numeric>
#include <opencv2/opencv.hpp>
#include <time.h>
#include "process.h"

/*************************************************************************/
/*                  PRIVATE MACROS AND DEFINES                           */
/*************************************************************************/
#define ERROR (-1)
#define OK (0)
#define NSEC_PER_SEC (1000000000)
#define NSEC_PER_MSEC (1000000)
#define NSEC_PER_MICROSEC (1000)
#define USEC_PER_MSEC (1000)
#define NANOSEC_PER_SEC (1000000000)
#define TRUE (1)
#define FALSE (0)

/*************************************************************************/
/*                  EXTERN GLOBAL VARIABLES                              */
/*************************************************************************/
//defined in capture.cpp
```

```
extern cv::Mat frame;
extern bool frame_flag;

/*****************************************************************************************
 * @brief   :   captures a frame from camera
 *
 * @param   :   cv::VideoCapture cap - global object instance used to capture frame
 *
 * @return  :   none
 *
 *****************************************************************************************/
void capture_frame(cv::VideoCapture cap);

/*****************************************************************************************
 * @brief   :   calculates absolute difference between two timespec structures
 *
 * @param   :   stop     - end time
 *                  start     - start time
 *                  delta_t - structure in which absolute difference will be stored
 *
 * @return  :   0 on success, -1 on failure
 *
 *****************************************************************************************/
int delta_t(struct timespec *stop, struct timespec *start, struct timespec *delta_t);

#endif /*_CAPTURE_H_*/




/*
 * @file_name       :   process.h
 *
 * @brief           :   RTES Final Project Code
 *
 * @author          :   Sanish Kharade
 *                  Tanmay Kothale
 *                  Vishal Raj
 *
 * @date            :   May 03, 2022
 *
 */
#ifndef _PROCESS_H_
#define _PROCESS_H_

/*************************************************************************/
/*                      LIBRARY FILES                          */
```

```
/**********************************************************************/
#include <iostream>
#include <numeric>
#include <opencv2/opencv.hpp>
#include "capture.h"

/*ENUMERATION OF DESIRED SHAPES*/
enum Shape { Circle, Undefined };
/*TYPEDEF FOR TEMPLATE*/
typedef std::vector<cv::Point> Contour;

typedef enum
{
   NONE,
   RED,
   YELLOW,
   GREEN,

}color_t;


/***********************************************************************************
 * @brief   :   masks the image from BGR to HSV values
 *
 * @param   :   frame  -       captured frame to be masked
 *              h      -       Hue value
 *              error  -       error margin acceptable
 *              s_min  -       minimum value for saturation
 *              v_min  -       minimum value for vignette
 *
 *
 * @return  :   masked image
 *
 ***********************************************************************************/
cv::Mat mask_img(cv::Mat& frame, int h, int error, int s_min, int v_min);

/***********************************************************************************
 * @brief   :   performs contour detection to identify various shapes (circles)
 *
 * @param   :   shapeToFind  -       shape that needs to be detected
 *              grey         -       masked image
 *              min_area     -       lower limit of area to look for
 *              max_area     -       upper limit of area to look for
 *
 * @return  :   identified shape
 *
 ***********************************************************************************/
```

```
std::vector<Contour> findShapes(Shape shapeToFind, cv::Mat& grey, int min_area, int max_area);

/***************************************************************************************
 * @brief   :   Lables the polygon identified in the contour detection
 *
 * @param   :   c       -       contour
 *              area    -       area to look for
 *
 * @return  :   identified shape
 *
 ***************************************************************************************/
Shape labelPolygon(Contour& c, double area);

/***************************************************************************************
 * @brief   :   identifies circles (convex contours)
 *
 * @param   :   c       -       contour
 *              area    -       area to look for
 *
 * @return  :   true if circle is detected, false otherwise
 *
 ***************************************************************************************/
bool isConvex(Contour& c, double area);

/***************************************************************************************
 * @brief   :   processes the frame captured
 *
 * @param   :   min_area    -       lower limit of area to look for
 *              max_area    -       upper limit of area to look for
 *
 * @return  :   none
 *
 ***************************************************************************************/
void process_image(int min_area, int max_area);

/***************************************************************************************
 * @brief   :   sets the color once identified
 *
 * @param   :   c       -       macro of color identified
 *
 * @return  :   none
 *
 ***************************************************************************************/
void setColor(int c);

/***************************************************************************************
 * @brief   :   determines which color was identified and returns it to calling function
```

```
 *
 * @param   :   none
 *
 * @return  :   color identified
 *
*******************************************************************************************/
color_t getColor(void);
#endif /*_PROCESS_H_*/





/*
 * @file_name       :   calculate.h
 *
 * @brief           :   RTES Final Project Code
 *
 * @author          :   Sanish Kharade
 *                      Tanmay Kothale
 *                      Vishal Raj
 *
 * @date            :   May 03, 2022
 *
 */
#ifndef _CALCULATE_H_
#define _CALCULATE_H_

/*****************************************************************************/
/*                          LIBRARY FILES                                    */
/*****************************************************************************/
#include <stdio.h>
#include <stdint.h>
#include <math.h>
#include "process.h"
#include "capture.h"

/*****************************************************************************/
/*                  PRIVATE MACROS AND DEFINES                               */
/*****************************************************************************/
#define CURRENT_SPEED_OF_CAR        (9.834880)
#define TOTAL_TIME_REQD_TO_STOP (2.479339)
#define DISTANCE_REQD_TO_STOP       (24.38400)

/*******************************************************************************************
 * @brief   :   computes the time required for a moving vehicle to come to a complete
 *                                      stop
```

```
 *
 * @param   :   speed           -          speed at which vehicle is traveling
 *                              distance          -          distance at which traffic light is detected
 *
 * @return  :   deadline        -          time required to complete all computations
 *
******************************************************************************************/
float time_to_stop_in_sec(float speed, float distance);

/******************************************************************************************
 * @brief   :   computes the time required to complete all tasks before coming to a
 *                              full stop
 *
 * @param   :   distance                          -          distance at which traffic light is detected
 *
 * @return  :   deadline_for_tasks      -          time required to complete all computations
 *
******************************************************************************************/
float compute_deadline_to_complete_tasks (float distance);

#endif /*_CALCULATE_H_*/




/*
 * @file_name       :   udm.h
 *
 * @brief           :   RTES Final Project Code
 *
 * @author          :   Sanish Kharade
 *                  Tanmay Kothale
 *                  Vishal Raj
 *
 * @date            :   May 03, 2022
 *
 */

#ifndef _UDM_H_
#define _UDM_H_

//function prototypes
/******************************************************************************************
 * @brief   :   calculates distance using ultrasonic sensor
 *
 * @param   :   none
 *
```

```
 * @return  :   distance          -          calculated distance
 *
****************************************************************************************/
double get_distance();

/***************************************************************************************
 * @brief   :   initialize ultrasonic function
 *
 * @param   :   none
 *
 * @return  :   none
****************************************************************************************/
void init_udm();


#endif /*_UDM_H_*/




MAKEFILE

CC=g++
CFLAGS = -O0 -Wall -Werror
LDFLAGS =
CPPLIBS= -L/usr/lib -lopencv_core -lopencv_flann -lopencv_video -lrt -lpthread -lpigpio

OBJFILES = process.o capture.o uart.o calculate.o udm.o main.o

TARGET = main

all: $(TARGET)

$(TARGET): $(OBJFILES)
	$(CC) $(CFLAGS) -o $(TARGET) $(OBJFILES) $(LDFLAGS) `pkg-config --libs opencv` $(CPPLIBS)

clean:
	rm -f $(OBJFILES) $(TARGET) *~
```

[B] Code files for TIVA

```
/******************************************************************************************
****************************
* File Name    : main.c
```

```
* Project      : RTES Final Project FreeRTOS service implementation.
* Version      : 1.0.
* Description  : Contains the source code for creation of multiple tasks and usage of various synchronization
*              mechanisms between them for RTES project services.
* Author       : Vishal Raj & Sanish Kharade, referred from
https://github.com/akobyl/TM4C129_FreeRTOS_Demo,
*              TivaWare_C_Series-2.2.0.295 SDK examples.
* Creation Date: 4.30.22
****************************************************************************************
****************************/
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include "main.h"
#include "drivers/pinout.h"
#include "utils/uartstdio.h"


// TivaWare includes
#include "driverlib/sysctl.h"
#include "driverlib/debug.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"


// For UART
#include "driverlib/uart.h"
#include "driverlib/gpio.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"


// FreeRTOS includes
#include "FreeRTOSConfig.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "driverlib/pin_map.h"

// For PWM
#include "driverlib/pwm.h"

// For Timer
//#include "inc/tm4c1294ncpdt.h"
#include "inc/hw_types.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"
```

```
#define SERIAL_TASK_PRIORITY    (1)
#define LED_TASK_PRIORITY       (1)
#define MOTOR_TASK_PRIORITY     (1)


#define SERIAL_TASK_STACK_SIZE    (configMINIMAL_STACK_SIZE)
#define LED_TASK_STACK_SIZE       (configMINIMAL_STACK_SIZE)
#define MOTOR_TASK_STACK_SIZE     (configMINIMAL_STACK_SIZE)
#define SCALING_FACTOR            100000//1000 //for 1ms timer
#define TIMER_FREQ                2500
#define CLOCK_FREQ                120000000
#define TIMER_LOAD_VALUE          (CLOCK_FREQ/SCALING_FACTOR)

#define SPEED_FOR_RED       0
#define SPEED_FOR_YELLOW    4
#define SPEED_FOR_GREEN     6

uint32_t ui32Period,pom = 0;


//Task and function prototypes
void LEDTask(void *pvParameters);
void SerialTask(void *pvParameters);
void MOTORTask(void *pvParameters);
void pwm_control(void);
void run_motor_speed(int level);
static void calculate_wcet(uint32_t start, uint32_t end,uint32_t *WCET,uint8_t id);


uint32_t g_ui32SysClock;
xQueueHandle g_pMyQueue;
xQueueHandle g_pMotorQueue;
// For PWM
uint32_t g_ui32PWMIncrement;

typedef enum
{
   NONE = 0,
   RED,
   YELLOW,
   GREEN
}color_t;


color_t color = NONE;
uint8_t current_color = 0;
```

```
char cMessage;
uint8_t speed = 5;

/*******************************************************************************
******
* Name          : UART3IntHandler
* Description   : UART interrupt handler.
* Parameters    : None
* RETURN        : None
*******************************************************************************
*****/
void UART3IntHandler(void)
{
   uint32_t ui32Status;

   // Get the interrrupt status.
   ui32Status = MAP_UARTIntStatus(UART3_BASE, true);

   // Clear the asserted interrupts.
   MAP_UARTIntClear(UART3_BASE, ui32Status);

   // Loop while there are characters in the receive FIFO.
   char c = 0;
   while(MAP_UARTCharsAvail(UART3_BASE))
   {
      // Read the next character from the UART and write it back to the UART.
      c =  MAP_UARTCharGetNonBlocking(UART3_BASE);
      current_color = c;

      if(xQueueSend(g_pMotorQueue, &current_color, portMAX_DELAY) != pdPASS)
      {
         UARTprintf("\nQueue full. This should never happen.\n");
         while(1)
         {
         }
      }
   }
}

/*******************************************************************************
******
* Name          : UARTSend
* Description   : function for sending data through UART to raspberry pi.
* Parameters    : @param pui8Buffer Buffer reference
*               @param ui32Count size of buffer to be send
* RETURN        : None
*******************************************************************************
```

```
*****/
void UARTSend(const uint8_t *pui8Buffer, uint32_t ui32Count)
{
   // Loop while there are more characters to send.
   while(ui32Count--)
   {
      // Write the next character to the UART.
      MAP_UARTCharPutNonBlocking(UART3_BASE, *pui8Buffer++);
   }
}

/*****************************************************************************************
******
* Name          : UART_init
* Description   : function for initializing the UART module for inter board communication.
* Parameters    : None
* RETURN        : None
*****************************************************************************************
*****/
void UART_init(void)
{
   // Enable the peripherals used by this example.
   MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART3);
   MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

   // Enable processor interrupts.
   MAP_IntMasterEnable();

   // Set GPIO A0 and A1 as UART pins.
   MAP_GPIOPinConfigure(GPIO_PA4_U3RX);
   MAP_GPIOPinConfigure(GPIO_PA5_U3TX);
   MAP_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_4 | GPIO_PIN_5);

   // Configure the UART for 115,200, 8-N-1 operation.
   MAP_UARTConfigSetExpClk(UART3_BASE, g_ui32SysClock, 9600,
                     (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                      UART_CONFIG_PAR_NONE));

   // Enable the UART interrupt.
   MAP_IntEnable(INT_UART3);
   MAP_UARTIntEnable(UART3_BASE, UART_INT_RX | UART_INT_RT);
}

/*****************************************************************************************
******
* Name          : ConfigureUART
* Description   : function for initializing the UART module for logging and profiling services
```

```
* Parameters    : None
* RETURN        : None
*****************************************************************************************
*****/
void ConfigureUART(void)
{
   // Enable the GPIO Peripheral used by the UART.
   MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

   // Enable UART0.
   MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

   // Configure GPIO Pins for UART mode.
   MAP_GPIOPinConfigure(GPIO_PA0_U0RX);
   MAP_GPIOPinConfigure(GPIO_PA1_U0TX);
   MAP_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

   // Initialize the UART for console I/O.
   UARTStdioConfig(0, 115200, g_ui32SysClock);
}

/****************************************************************************************
******
* Name          : PWM_init
* Description   : function for initializing the PWM module.
* Parameters    : None
* RETURN        : None
*****************************************************************************************
*****/
void PWM_init(){

   uint32_t ui32PWMClockRate;

   // The PWM peripheral must be enabled for use.
   MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);

   // Enable the GPIO port that is used for the PWM output.
   MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

   // Configure the GPIO pad for PWM function on pins PF2 and PF3.
   MAP_GPIOPinConfigure(GPIO_PF2_M0PWM2);
   MAP_GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_2);

   // Set the PWM clock to be SysClk / 8.
   MAP_PWMClockSet(PWM0_BASE, PWM_SYSCLK_DIV_8);

   //PWM generator period.
```

```
    //120Mhz ,15Mhz PWM
    ui32PWMClockRate = g_ui32SysClock / 8;

    //6Khz increment
    g_ui32PWMIncrement = ((ui32PWMClockRate / 250) / 10); //10% increment

    // Configure PWM2 to count up/down without synchronization.
    MAP_PWMGenConfigure(PWM0_BASE, PWM_GEN_1,
                    PWM_GEN_MODE_UP_DOWN | PWM_GEN_MODE_NO_SYNC);

    // Set the PWM period to 250Hz.
    MAP_PWMGenPeriodSet(PWM0_BASE, PWM_GEN_1, (ui32PWMClockRate / 250));


    // Set PWM2 to a duty cycle of 60%.
    MAP_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_2,
                    g_ui32PWMIncrement * 6); //start at 60%


    // Enable the PWM Out Bit 2 (PF2)
    MAP_PWMOutputState(PWM0_BASE, PWM_OUT_2_BIT, true);


    // Enable the PWM generator block.
    MAP_PWMGenEnable(PWM0_BASE, PWM_GEN_1);
}

/*******************************************************************************************
******
* Name          : vHWTimerInit
* Description   : Function for initializing TimerA0 module.
* Parameters    : None
* RETURN        : None
*******************************************************************************************
*****/
void vHWTimerInit(void){

    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);

    ui32Period = (g_ui32SysClock / SCALING_FACTOR);
    TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period);

    IntEnable(INT_TIMER0A);
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    IntMasterEnable();
```

```
    TimerEnable(TIMER0_BASE, TIMER_A);

}

/*******************************************************************************************
******
* Name          : calculate_wcet
* Description   : Function for computing the WCET for each service base on start time, end time
*               and function id.
* Parameters    : @param start - start time of service.
*               @param end - end time of service execution.
*               @param WCET - current WCET for the service.
*               @param id - service id.
* RETURN        : None
********************************************************************************************
*****/
void calculate_wcet(uint32_t start, uint32_t end,uint32_t *WCET,uint8_t id){

    uint32_t Exec_t = 0;

    Exec_t = (end - start + TIMER_LOAD_VALUE) % TIMER_LOAD_VALUE;

    if(Exec_t > *WCET){

        *WCET = Exec_t;
        taskENTER_CRITICAL();

        //For x1 frequency.
        UARTprintf("==Service_%d WCET time:%d us==\n", id, (Exec_t*8)/1000);

        taskEXIT_CRITICAL();
    }

}

/*******************************************************************************************
******
* Name          : main
* Description   : entry point function of initialize all modules and create threads.
* Parameters    : None
* RETURN        : exit status of program.
********************************************************************************************
*****/
int main(void)
{
    // Initialize system clock to 120 MHz
    //uint32_t output_clock_rate_hz;
```

```
g_ui32SysClock = ROM_SysCtlClockFreqSet(
                        (SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN |
                         SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480),
                        SYSTEM_CLOCK);
ASSERT(g_ui32SysClock == SYSTEM_CLOCK);

UART_init();
g_pMyQueue = xQueueCreate(5, sizeof(color_t));
g_pMotorQueue = xQueueCreate(2, sizeof(uint32_t));

//Enable clock
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);

//Check if h/w access enabled
while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPION))
{
}

//Set output
GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_2);
GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_3);

GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_2, GPIO_PIN_2);
GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_3, GPIO_PIN_3);


vHWTimerInit();

PWM_init();

// Initialize the GPIO pins for the Launchpad
PinoutSet(false, false);

UARTStdioConfig(0, 57600, SYSTEM_CLOCK);
// Create tasks
xTaskCreate(LEDTask, (const portCHAR *)"LEDs",
            LED_TASK_STACK_SIZE, NULL, LED_TASK_PRIORITY, NULL);

xTaskCreate(SerialTask, (const portCHAR *)"Serial",
           SERIAL_TASK_STACK_SIZE, NULL, SERIAL_TASK_PRIORITY, NULL);

xTaskCreate(MOTORTask, (const portCHAR *)"Motor",
            MOTOR_TASK_STACK_SIZE, NULL, MOTOR_TASK_PRIORITY, NULL );

vTaskStartScheduler();

// Code should never reach this point
```

```
    return 0;
}

/*****************************************************************************************
******
* Name          : Timer0AIntHandler
* Description   : Interrupt handler for Timer0A.
* Parameters    : None
* RETURN        : None
*****************************************************************************************
*****/
void Timer0AIntHandler(void)
{
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

    pom++;

}

/*****************************************************************************************
******
* Name          : MOTORTask
* Description   : The motor control service to vary motor speed according to UART input.
* Parameters    : @param pvParameters - thread parameter.
* RETURN        : None
*****************************************************************************************
*****/
void MOTORTask(void *pvParameters){

    for(;;){

        pwm_control();

    }

}

/*****************************************************************************************
******
* Name          : pwm_control
* Description   : The function implementing motor speed control according to UART input.
* Parameters    : None
* RETURN        : None
*****************************************************************************************
*****/
void pwm_control(void){
```

```
    uint8_t queueData = 0;
    TickType_t xStartTime,xEndTime;
    static uint32_t WCET = 0;

    if(xQueueReceive(g_pMotorQueue, &queueData, portMAX_DELAY) == pdPASS){

        /*Start time log*/
        xStartTime = TimerValueGet(TIMER0_BASE, TIMER_A);;

        if(queueData == RED)
            run_motor_speed(SPEED_FOR_RED);
        else if(queueData == YELLOW)
            run_motor_speed(SPEED_FOR_YELLOW);
        else if(queueData == GREEN)
            run_motor_speed(SPEED_FOR_GREEN);
        else
            run_motor_speed(SPEED_FOR_GREEN);

        /*End time log*/
        xEndTime = TimerValueGet(TIMER0_BASE, TIMER_A);;

        calculate_wcet(xStartTime,xEndTime, &WCET, 3);
    }

    //Send data to Rpi
    if(xQueueSend(g_pMyQueue, &queueData, portMAX_DELAY) != pdPASS)
    {
        // Error. The queue should never be full. If so print the
        // error message on UART and wait for ever.

        while(1)
        {
        }
    }

}

/*******************************************************************************************
******
* Name          : run_motor_speed
* Description   : Sets the PWM value for motor according to required level.
* Parameters    : @param level - motor speed from 0 to 10 i.e 0 to 100%.
* RETURN        : None
*******************************************************************************************
*****/
void run_motor_speed(int level)
{
```

```
    MAP_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_2, g_ui32PWMIncrement * level);
}


/*****************************************************************************************
******
* Name          : LEDTask
* Description   : The task to control green and red led status according to UART input.
* Parameters    : @param pvParameters - thread parameter.
* RETURN        : None
*****************************************************************************************
*****/
void LEDTask(void *pvParameters)
{
    TickType_t xStartTime,xEndTime;
    uint32_t WCET = 0;

    for (;;)
    {
        /*Start time log*/
        xStartTime = TimerValueGet(TIMER0_BASE, TIMER_A);

        switch(color)
        {
            case NONE:
                LEDWrite(0x0F, 0x01);
                break;

            case RED:
                LEDWrite(0x0F, 0x02);

                GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_3, GPIO_PIN_3);

                GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_2, 0x00);

                break;

            case YELLOW:
                LEDWrite(0x0F, 0x04);
                break;

            case GREEN:
                LEDWrite(0x0F, 0x08);

                GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_2, GPIO_PIN_2);

                GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_3, 0x00);
```

```
            break;

        default:
            LEDWrite(0x0F, 0x00);
            break;

    }

    /*End time log*/
    xEndTime = TimerValueGet(TIMER0_BASE, TIMER_A);


    calculate_wcet(xStartTime,xEndTime, &WCET, 1);

  }
}

/*****************************************************************************************
******
* Name          : SerialTask
* Description   : The task to send the acknowladgement back to Rpi after motor and LED actutation
*               is complete.
* Parameters    : @param pvParameters - thread parameter.
* RETURN        : None
******************************************************************************************
*****/
void SerialTask(void *pvParameters)
{
  // Set up the UART which is connected to the virtual COM port

  TickType_t xStartTime,xEndTime;
  uint32_t WCET = 0;

  for (;;)
  {

    if(xQueueReceive(g_pMyQueue, &color, portMAX_DELAY) == pdPASS)
    {
      /*Start time log*/
      xStartTime = TimerValueGet(TIMER0_BASE, TIMER_A);;

      UARTSend((uint8_t *)(&color), 1);

      /*End time log*/
      xEndTime = TimerValueGet(TIMER0_BASE, TIMER_A);;

      calculate_wcet(xStartTime,xEndTime, &WCET, 2);
```

```
        }

    }
}

/*  ASSERT() Error function
 *
 *  failed ASSERTS() from driverlib/debug.h are executed in this function
 */
void __error__(char *pcFilename, uint32_t ui32Line)
{
    // Place a breakpoint here to capture errors until logging routine is finished
    while (1)
    {
    }
}
//[EOF]

/*****************************************************************************************
*****************************
* File Name    : main.h
* Project      : RTES Final Project FreeRTOS service implementation.
* Version      : 1.0.
* Description  : Header file for main.c
* Author       : Vishal Raj & Sanish Kharade, referred from
https://github.com/akobyl/TM4C129_FreeRTOS_Demo,
*               TivaWare_C_Series-2.2.0.295 SDK examples.
* Creation Date: 4.30.22
******************************************************************************************
****************************/

#ifndef MAIN_H_
#define MAIN_H_

// System clock rate, 120 MHz
#define SYSTEM_CLOCK    120000000U


#endif /* MAIN_H_ */
```