University of Colorado
Boulder

ECEN – 5623 Real-Time Embedded Systems

Final Project

Pi Parking System

Team Members

Krishna Suhagiya

Unmesh Phaterpekar

Table of Contents

Introduction

Our project, named the "Pi Parking System," employs real-time embedded system principles using a Raspberry Pi 4+ as the central processing unit. This system integrates three key hardware components: a Logitech camera, motorized wheels, and an ultrasonic sensor. We have designed three distinct tasks within the system: the Camera Task, the Motor Task, and the Sensor Task, all orchestrated to enable the system to operate in two modes—forward and reverse.

In the forward mode, the ultrasonic sensor actively monitors for obstacles ahead of the bot. Concurrently, the camera remains inactive, serving as a sort of infotainment blackout. The bot's wheels propel it forward unless an obstacle is detected, at which point the system engages the brakes to halt the bot. A pushbutton then triggers the reverse mode, wherein the bot reverses direction; the wheels turn backward, and the camera feed becomes active, displaying on the video display screen. The webcam integrates with the system via the "Cheese" software.

To manage the execution frequency of these tasks, we utilize a Sequencer function. This function governs a thread that runs at a predefined frequency, leveraging the nanosleep function to achieve precise timing. The thread checks the return value of nanosleep to account for any interruptions and adjust its sleep duration, accordingly, ensuring the timing remains accurate. As time progresses, the sequencer counts cycles with the seqCnt variable and releases semaphores for the respective tasks at specific frequencies, effectively prioritizing each service within the system.

This project is a collaborative effort between two individuals, with responsibilities divided as follows:

- Motor Control Task - Unmesh
- Camera Task - Krishna
- Sensor Task - Jointly handled by Unmesh and Krishna
- Scheduler Task – Collaboratively managed by Unmesh and Krishna

For this project on Linux, we will implement the SCHED_FIFO scheduling policy, which is a real-time policy that assigns tasks with a first-in, first-out methodology. Task priorities are distributed in accordance with the operation frequency of each task, strictly following the Rate Monotonic Policy. This policy dictates that tasks with shorter operation periods receive higher priority.

To manage the precise timing of task execution, a specialized scheduler has been developed. This scheduler functions by signalling the appropriate semaphores at predetermined times, coordinating the sequence of task activations. Each task is initially blocked by its designated semaphore. At the appropriate execution time, the scheduler unlocks these semaphores, thereby permitting the tasks to proceed. This mechanism ensures that high-priority tasks are given precedence, enhancing the system's responsiveness and predictability.

Functional/Capability Requirements

1. **Motor Driver Integration**: The motor driver ensures bidirectional control of the wheel, enabling smooth transitions between forward and reverse movements upon button presses, aligning with the set gear and responding within approximately 127 milliseconds.
2. **Movement Smoothness**: Control mechanisms are implemented to ensure that changes in direction are executed smoothly, avoiding sudden jerks, and promoting stable motion, crucial for both safety and mechanical integrity.
3. **Real-Time Camera Activation**: A camera system activates during reverse gear usage, providing a real-time video feed with a minimum frame rate of 15Hz to facilitate safe reverse parking operations.
4. **Camera Field of View**: The camera covers all necessary angles for comprehensive visibility, automatically adjusting based on the bot's manoeuvring to avoid blind spots and enhance navigation safety.
5. **Ultrasonic Sensor Functionality**: The bot is equipped with an ultrasonic sensor that updates proximity measurements every 23 milliseconds, enabling dynamic adjustments based on the bot's speed and environmental obstacle density.
6. **Obstacle Detection Response**: The sensor system promptly halts the bot when detecting obstacles within a 7 cm range in forward gear, effectively preventing potential collisions and ensuring user safety.
7. **Sensor Data Integration**: Sensor data are seamlessly integrated into the bot’s navigation algorithms, allowing for real-time adjustments to the bot's trajectory and speed based on immediate environmental feedback.
8. **Feedback and Monitoring**: Feedback mechanisms within the motor and sensor systems are included to monitor and confirm the accuracy of operations, ensuring that movements and halts are executed as intended and are error-free.
9. **User Interface and Alerts**: A user interface is developed that displays real-time camera feeds and sensor data, including alerts for obstacle proximity and system status, providing the user with intuitive and immediate information for decision-making.
10. **System Reliability and Safety**: Robust safety features and redundancy in sensor and motor operations are implemented to handle unexpected scenarios, prevent system failures, and ensure the bot’s reliability and durability in various operational environments.

Functional Hardware Design Overview

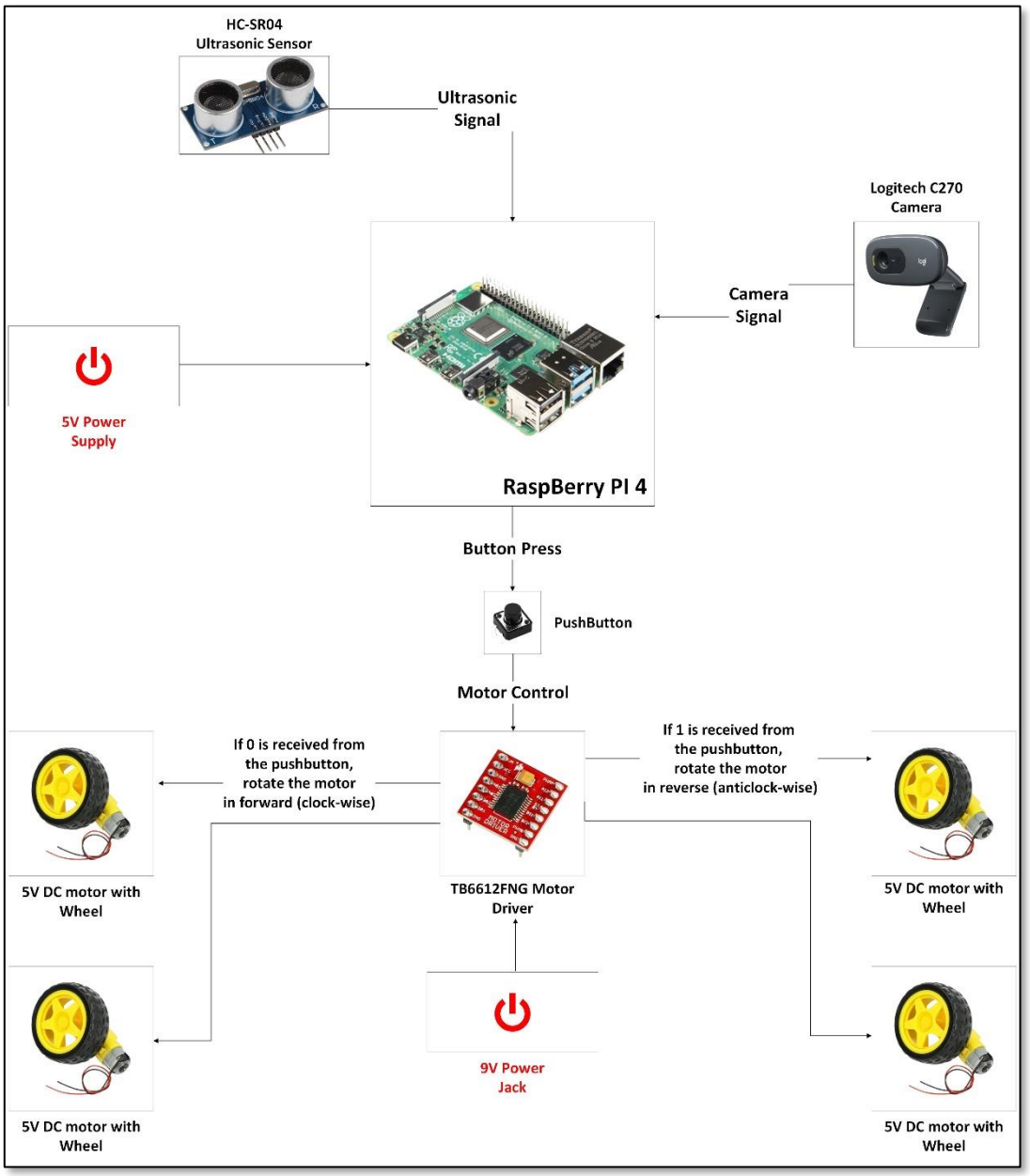

Figure 1 - Hardware Block Diagram

The figure presents our bot which uses a Raspberry Pi 4 as the central controller. Inputs to the Raspberry Pi include signals from an HC-SR04 ultrasonic sensor and a Logitech C270 camera. Additionally, a pushbutton interface with the Raspberry Pi to control motor direction via a TB6612FNG motor driver. Depending on the button input, the motor driver commands two 5V DC motors to rotate either clockwise or anticlockwise. The Raspberry Pi, sensor, and camera are powered by a 5V power supply, while the motors and motor driver receive power from a 9V power source, ensuring sufficient power for operation.

Flowchart

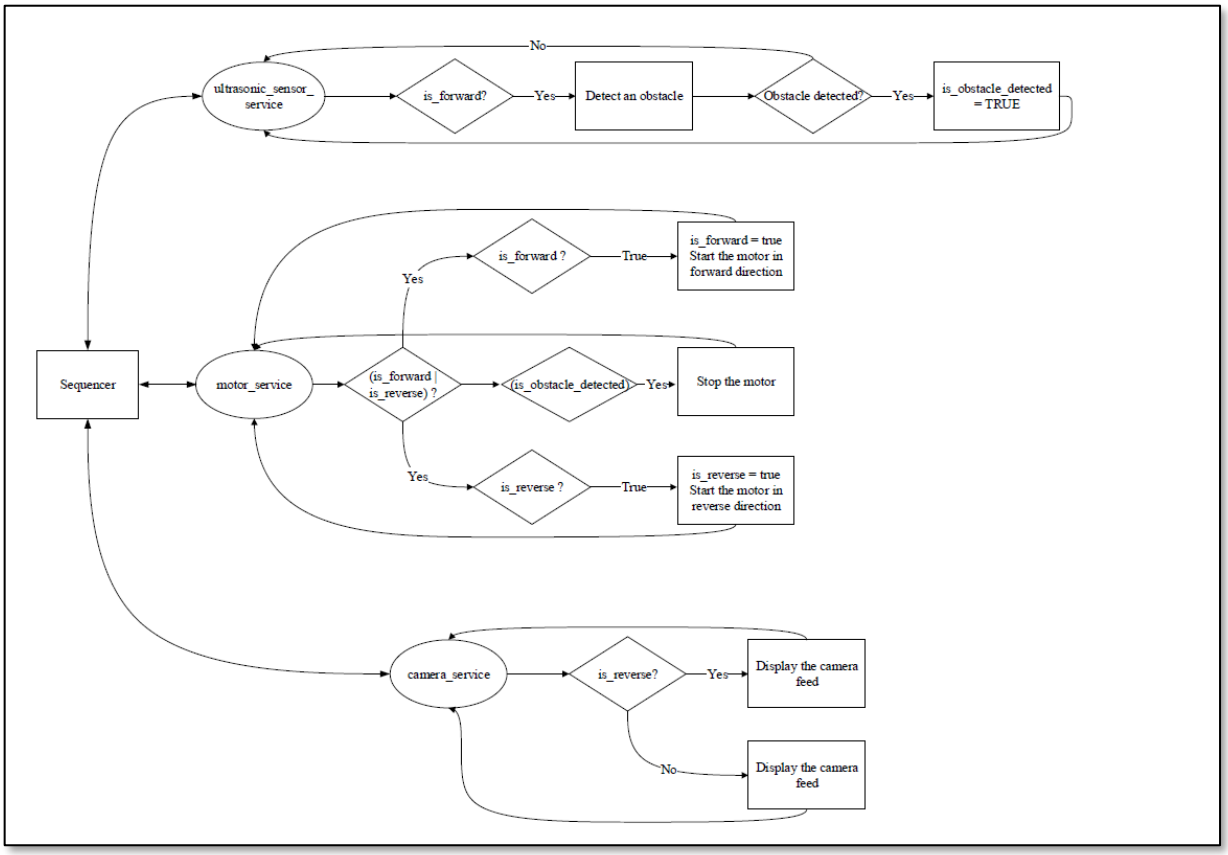

Figure 2 – Flow chart

We have total of four services in our system: Sequencer service, Ultrasonic sensor service, Motor service and camera service.
The sequencer is used to sequence the services. The frequencies of all the services were calculated from the deadlines and the sequencer is set to run at the LCM of all the service frequencies.
The motor task monitors the GPIO status from the push button, if it is 0, it sets the is_forward flag to TRUE. The is_reverse flag is set if the GPIO status is 1.
The gear flags are passed onto the other services i.e., motor service and ultrasonic sensor service.
The ultrasonic sensor service checks if the is_forward flag is set. If it is set, the service checks for the obstacles within the threshold limit i.e., in our case 7 cm. If the obstacle is detected, the is_obstacle detected flag is set. The motor service further checks the status of this flag and stops the motor if required.
The camera service checks the is_reverse flag. If it is set, the live camera feed is displayed on the window. Otherwise, a black buffer is displayed. The black buffer can be replaced by any infotainment application in the non-reverse gear.

In conclusion, our system adeptly integrates four key services—Sequencer, Ultrasonic Sensor, Motor, and Camera—ensuring coordinated and efficient operations. This setup enhances safety and responsiveness while providing adaptive visual feedback. It exemplifies a robust framework with potential for future enhancements, such as infotainment features.

Control Flow Diagram

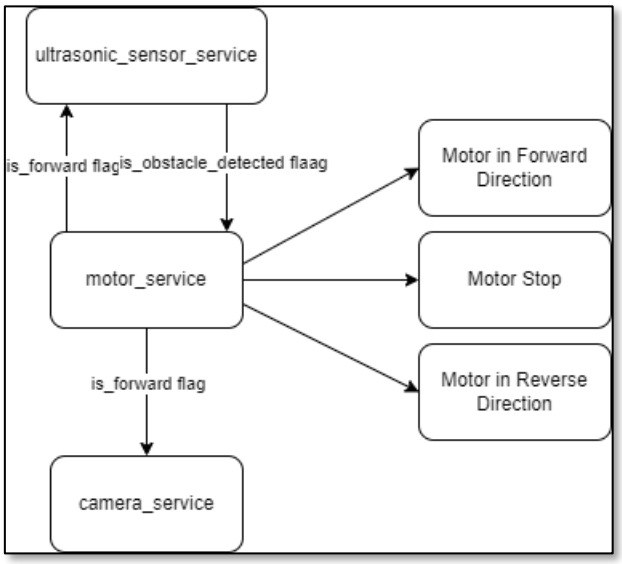

Figure 3 - CFD

The motor service functions as the central control unit, dictating the system's behavior. It guides the bot's movement—either forward or in reverse—based on the status of the is_forward, is_reverse, and is_obstacle_detected flags. Meanwhile, the sensor and camera services adapt their operations in response to these flag statuses.

Real-time Requirements

- The bot swiftly transitions to reverse direction and displays the camera feed upon receiving a reverse signal.
- The ultrasonic sensor monitors proximity to objects and signals the motor system to stop if an object is detected within a specified range.
- The motor system reacts promptly to commands, efficiently executing instructions to halt, advance, or reverse.

Services	Time Period (Ti) (ms)	WCET (Ci) (ms)	Deadline (Di) (ms)	Frequency (Hz)
Camera Service	66	15	66	15
Motor Service	127	1	127	8
Ultrasonic Sensor Service	173	72	173	6

Sensor Service:

We came up with the following operating framework for our ultrasonic sensor system after looking over the HC-SR04 datasheet. The ultrasonic pulse emitter on the system is set to fire on a regular basis. The following data is used to determine the timing: The sensor's maximum detection range is 4 meters, and sound travels at a speed of 340 meters per second.

• The ultrasonic pulse's round-trip time (out and back) is roughly 2×4/340 seconds, or 23.4 milliseconds. This is the input latency of the ultrasonic sensor to send a signal once an obstacle has been detected. To stop the trigger signal from becoming an echo signal, the datasheet suggests using a measurement cycle longer than 60 milliseconds.

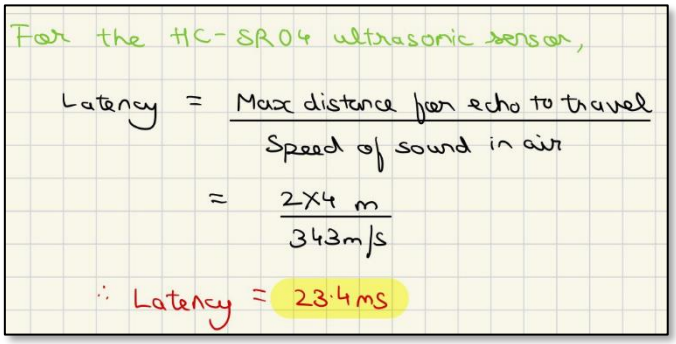

Figure 4 – Ultrasonic Input Latency

The sensor task has an input latency of ~23ms to detect the obstacle as calculated in the Fig 4 above.

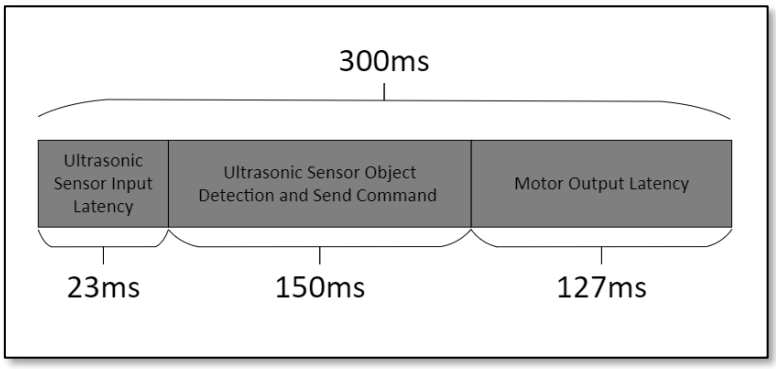

Figure 5 – Timing Diagram

As seen in Fig 5, the total time for the Sensor detecting an obstacle and sending the command to the motor task till the motor coming to a complete stop is coming out to be 300 ms which we have taken from the Fuzzy Logic Controller on Automated Car Braking System Reference Paper. In the paper as the velocity is 10 m/s, the time required for the vehicle to come to a complete stop is 3s. But since, we have calculated the velocity of our motor to be around 1.02 m/s, we scaled down the deadline by a factor of 10 so we got a deadline of 300 ms. The calculations are shown in the Fig 6 given below.

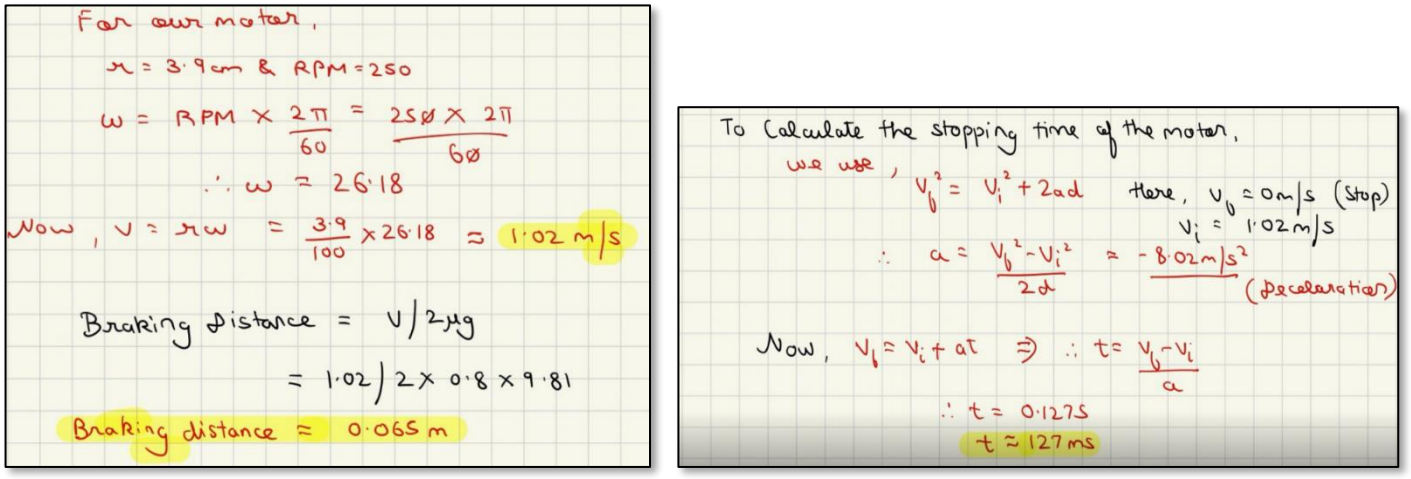

Figure 6 – Motor Calculations

The HC-SR04 ultrasonic sensor measures distance by emitting and receiving ultrasound waves. It requires a 10-microsecond trigger pulse to start. Once triggered, it emits eight pulses at 40 kHz. The main latency comes from the time ultrasound waves take to travel to an object and return, which varies with distance. The echo's travel time is significant as sound travels at about 343 meters per second. Post-reception, the sensor quickly processes the signal. The overall input latency primarily depends on the echo's travel time, with minimal electronic delay. The sensor also needs about 60 milliseconds between measurements to reset.

Motor Service:

This task uses a pushbutton to detect the gear status and then uses the proper steps to drive the bot forward, backward, or stop altogether.
The camera feed will either be shown or obstacle detection using an ultrasonic sensor will be initiated based on the direction given to the bot. This task's execution duration of 127 milliseconds is optimized to match the update rate of the sensor inputs.
The reference for these calculations is taken from a research paper published by Dinesh Manickam named "Designing and Controlling the Speed of Single Phase Induction Motor using Raspberry pi System".

This task is structured around two core functions that respond to incoming signals:

1. **Forward Movement and Obstacle Detection**:

- **Signal Initiation**: Upon receiving a forward command, the task activates the motor to begin advancing.
- **Obstacle Monitoring**: Simultaneously, the ultrasonic sensor checks for any obstacles ahead. If an impediment is identified, the task can either modulate the motor's speed or stop the motor entirely, depending on the situation.
- **Speed Control**: The motor's velocity is regulated through adjustments to the PWM duty cycle, allowing for dynamic responses to the surroundings and input from the ultrasonic sensor.

2. **Reverse Movement**:

- **Signal Initiation**: The task initiates reverse movement of the motor upon receiving a reverse signal.
- **Direction Control**: This directional change is facilitated by toggling the input pins on the motor driver, effectively reversing the motor's operation.

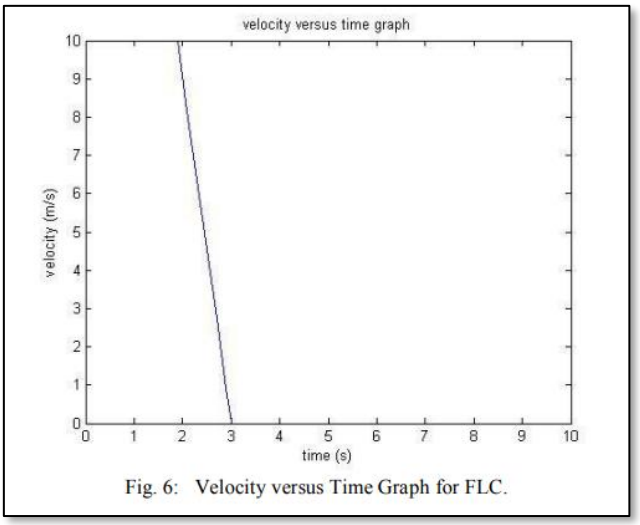

Fig. 6: Velocity versus Time Graph for FLC.

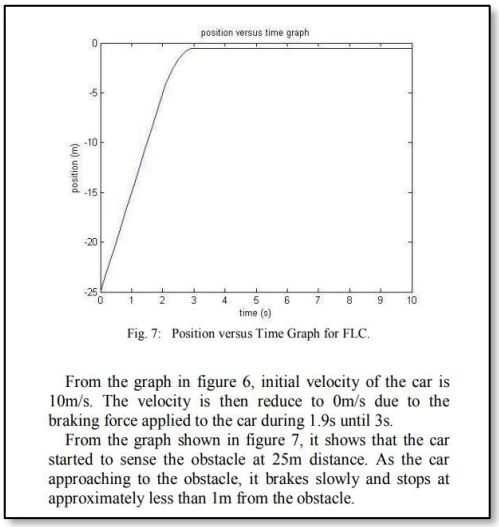

Fig. 7: Position versus Time Graph for FLC.

From the graph in figure 6, initial velocity of the car is 10m/s. The velocity is then reduce to 0m/s due to the braking force applied to the car during 1.9s until 3s.

From the graph shown in figure 7, it shows that the car started to sense the obstacle at 25m distance. As the car approaching to the obstacle, it brakes slowly and stops at approximately less than 1m from the obstacle.

- **Velocity versus Time Graph (Fig. 6)**: This graph displays the velocity of a car over time. Initially, the car is traveling at 10 m/s. Between 1.9 seconds and 3 seconds, a braking force is applied, which reduces the car's velocity to 0 m/s sharply.
- **Position versus Time Graph (Fig. 7)**: This graph illustrates the car's position relative to an obstacle over time. The car detects an obstacle at 25 meters. As it approaches, it begins braking gradually, decreasing its velocity and finally stopping less than 1 meter from the obstacle.

These graphs are used to demonstrate the effectiveness of the FLC in controlling the car's speed and stopping distance in response to obstacles, highlighting the system's responsiveness and precision in safety-critical scenarios.

Camera Task:

The robot will begin traveling in the other direction as soon as it detects the reverse signal, and the camera feed will be displayed. Initially we had used the measured value from The National Highway Traffic Safety Administration (NHTSA) FMVSS 111 for Rear Visibility. The reference for this can be found here.
We initially calculated a response time of around 2 seconds, which we then reduced to 500 milliseconds. However, operating the camera at 2Hz resulted in noticeable lag, which is suboptimal for a standard car rear camera. To address this, we consulted an article from the EE Times on How to implement automotive smart rear-view cameras which recommended a minimum frame rate of 25Hz. Considering the processing constraints of the Raspberry Pi 4, which also handled multiple concurrent tasks, maintaining a 25Hz rate proved unfeasible. Consequently, we set the camera frame rate to 15Hz, ensuring stable performance for all running services.

So, in the figure 7 below, we see a blank screen as the bot is in forward motion. Nothing will be displayed here on the camera here as we have put a

Mat blackframe = Mat::zeros(Size(640, 480), CV_8UC3); command here. For future implementation, a person can create a new service to show an Infotainment System.

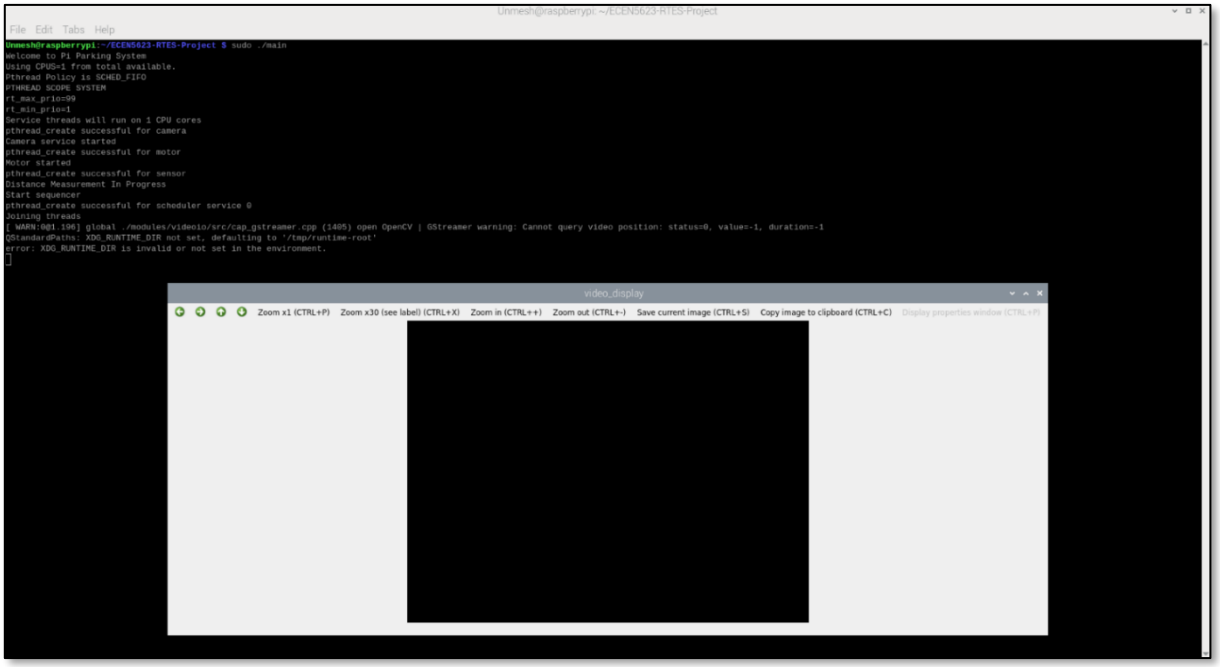

Figure 7 – Forward Camera

Now, once the vehicle has been put into reverse, the camera feed will immediately turn on and display the feed at 15Hz as shown in Fig 8. We are using the “cheese” command to run the camera.

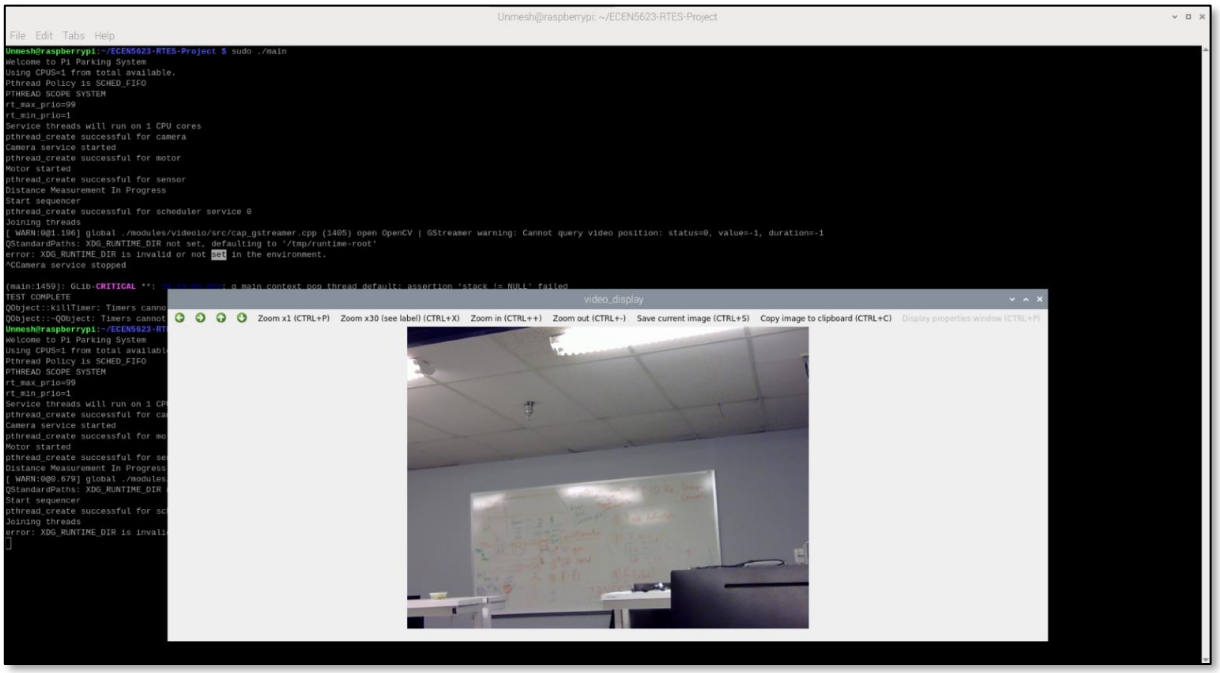

Figure 8 – Reverse Camera

WCET Analysis and CPU Utilization

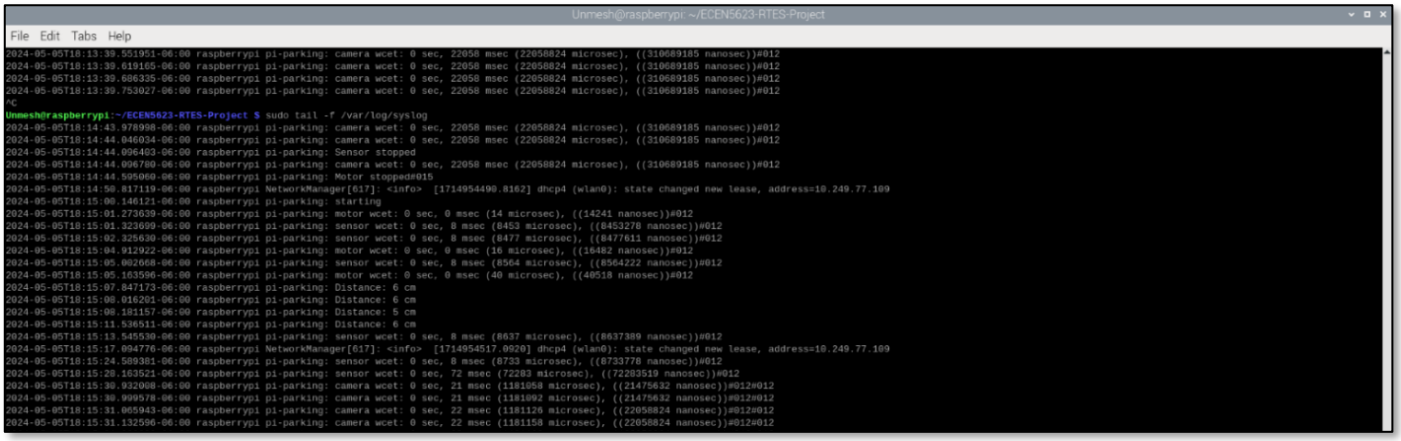

Figure 9 – WCET Calculations

To calculate the Worst-Case Execution Time (WCET), we measure the execution duration of each of the three services. We execute each thread multiple times and consistently record the time taken for each execution. The minimum execution time recorded across these runs is then designated as the WCET for each service. This approach helps in identifying the shortest possible execution time under the most favourable conditions, which is considered the WCET.

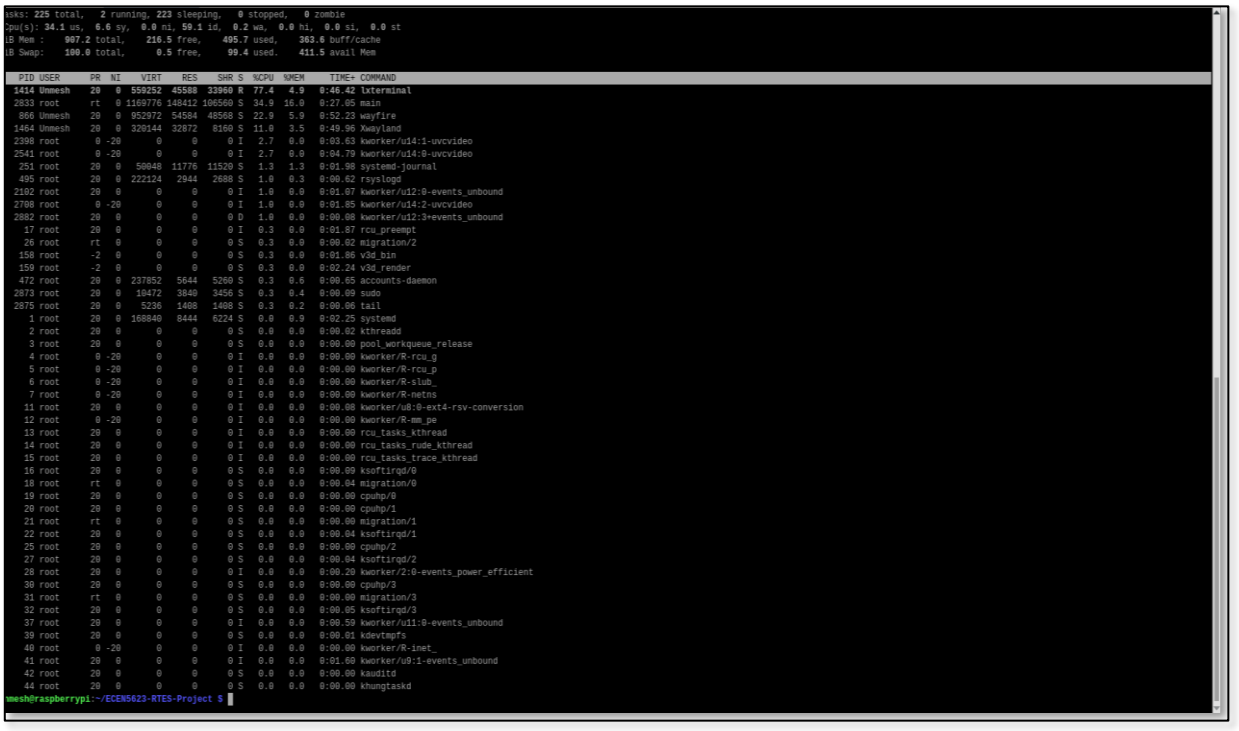

Figure 10 – CPU Utilization

That's great to hear! Achieving your target CPU utilization rate of 75% suggests you've successfully optimized the system for efficient performance without overloading it. This balance is crucial for maintaining stability and responsiveness. If you have any more goals or need further adjustments, feel free to ask for advice or share more details!

Rate Monotonic LUB and Cheddar Analysis

Implementing the RM LUB Formula, which is as follows, we get the Utilization to be less than the LUB which indicates that the service set is feasible.

$$U = \sum_{i=1}^{m} (Ci / Ti) \leq m(2^{\frac{1}{m}} - 1)$$

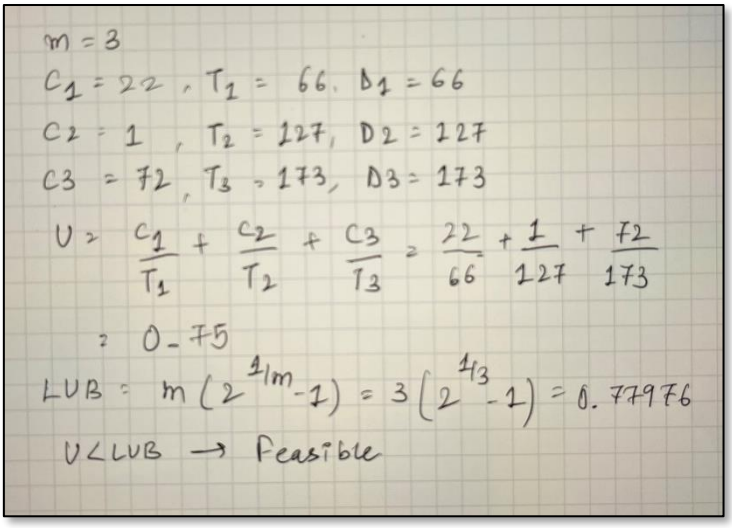

Figure 11 – LUB Calculation

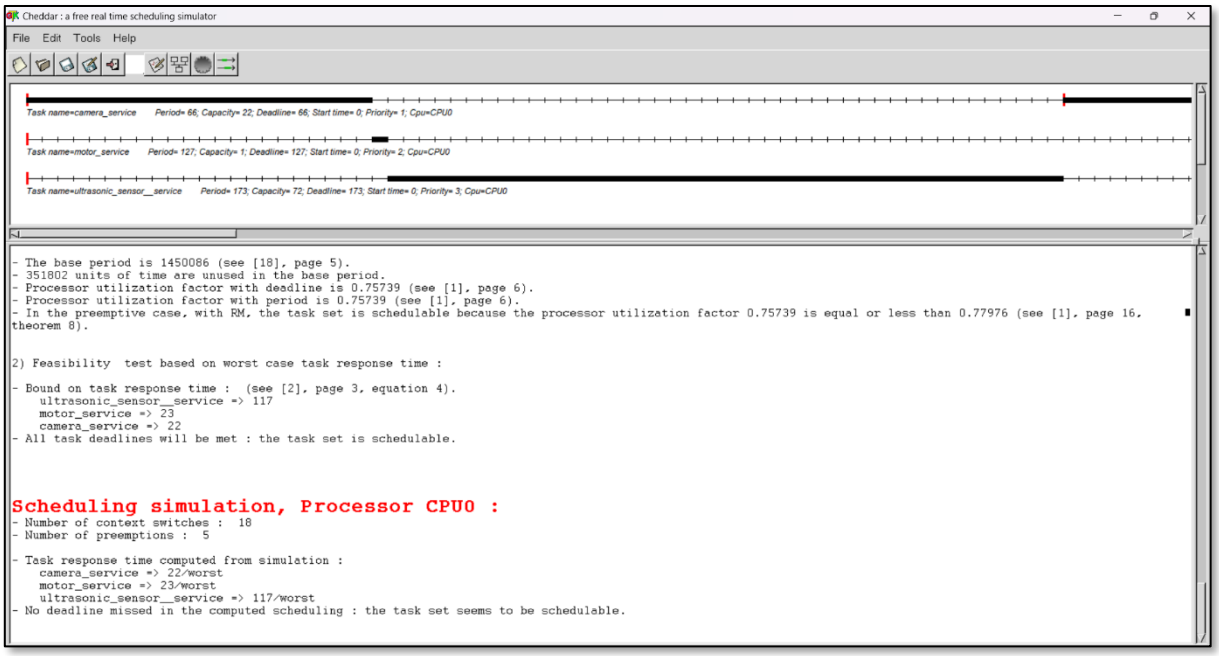

Figure 12 – Cheddar Analysis

So, from the cheddar output and the RM LUB calculation, we can see that the given set of tasks is schedulable.

Proof-of-Concept Analysis

➢ Hardware Interfaced:

- Raspberry Pi 4 features a Broadcom BCM2711 quad-core Cortex-A72 processor, offering options of 2GB, 4GB, or 8GB LPDDR4 RAM. Supports dual-band Wi-Fi, Bluetooth 5.0, dual 4K micro-HDMI, Gigabit Ethernet, USB 3.0, and extensive GPIO pins for hardware projects.
- Logitech C270 Webcam
- 5V DC TT Gearbox Motor
- TB6612FNG Motor Driver
- HC-SR04 Ultrasonic Sensor

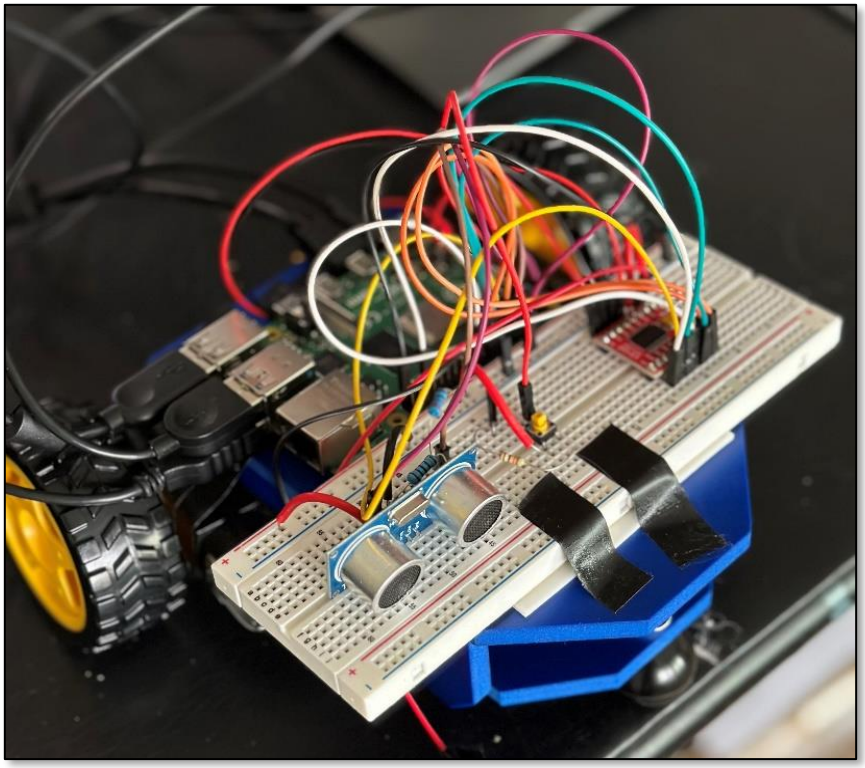

Figure 13 – Overall Hardware

➢ Kernel Level Implementation Used

1. POSIX Threads
2. Semaphores
3. WiringPi
4. Syslog
5. Scrot
6. OPENCV 4.6.0

Link to the Demo Video: IMG_1382.MOV

Verification & Validation Plans

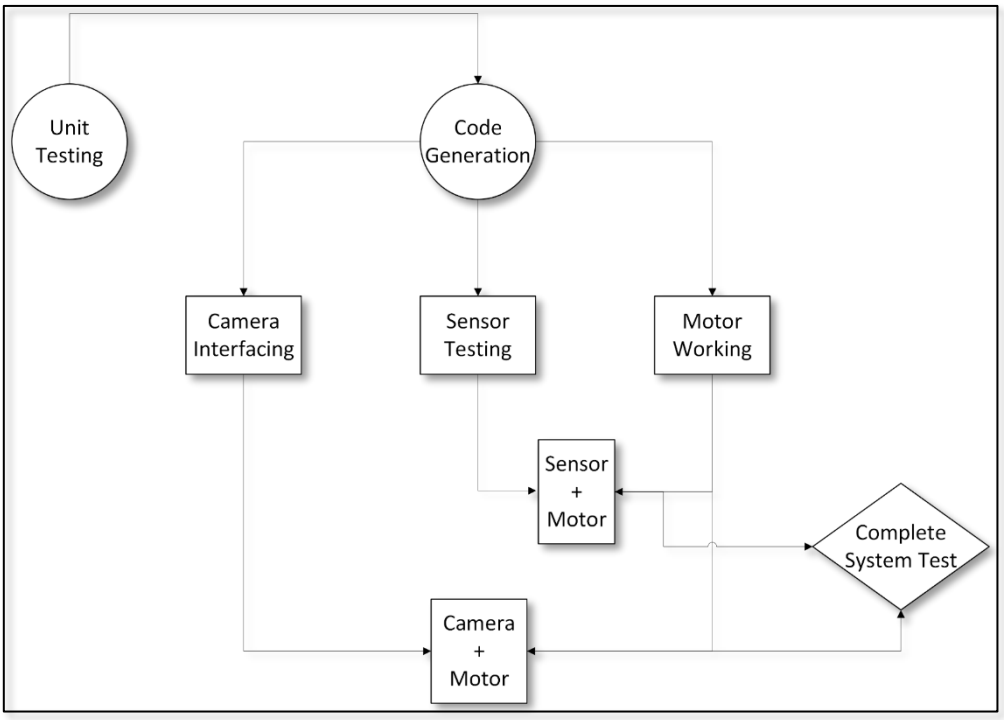

Figure 14 – Testing and Validation

The Fig 14 is a flowchart diagram that outlines a structured process for testing and integrating various components of the system. Here's an analysis of each part:

- **Unit Testing**: This is the initial phase where individual units or components of the system are tested in isolation to ensure they function correctly.
- **Code Generation**: This step involves writing or generating the necessary code that will control the system or facilitate the interaction between different components.
- **Camera Interfacing**: This is a specific test for integrating the camera into the system.
- **Sensor Testing**: Like camera interfacing, this phase tests the sensor involved in the system. It's about verifying that the sensor works as expected and can communicate their data to the system.
- **Motor Working**: This checks the operation of the motors in the system, ensuring they respond correctly to the control signals from the system.
- **Sensor + Motor**: This step involves testing the sensors and motors together, probably to check the coordination between sensory inputs and motor outputs.
- **Camera + Motor**: Similarly, in the sensor and motor integration, this phase tests the interaction between the camera and the motors, possibly focusing on tasks like object tracking and motor response based on visual inputs.
- **Complete System Test**: The final stage, represented by a diamond shape, suggests a decision point or a critical test that encompasses the entire system working together. This would validate the full integration of all components and their interoperation under various conditions.

References

1. https://www.youngwonks.com/blog/Raspberry-Pi-4-Pinout
2. https://www.digikey.com/en/articles/product-detection-and-ranging-using-ultrasonic-sensors#:~:text=These%20ultrasonic%20sensors%20have%20a,ultrasonic%20frequency%20of%20320%20kHz
3. https://www.sparkfun.com/products/14451
4. https://thepihut.com/blogs/raspberry-pi-tutorials/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi
5. https://www.adafruit.com/product/3777#technical-details
6. https://ijisrt.com/wp-content/uploads/2017/05/Automatic-Braking-System-Using-Ultrasonic-Sensor.pdf
7. https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf
8. https://ieeexplore.ieee.org/document/8281927
9. https://journals.sagepub.com/doi/abs/10.1177/09544070211009076
10. https://www.researchgate.net/profile/Nor-Maniha-Ghani/publication/224113164_Fuzzy_Logic_Controller_on_Automated_Car_Braking_System/links/5714401708aeff315ba35cac/Fuzzy-Logic-Controller-on-Automated-Car-Braking-System.pdf

Appendix

main.cpp

```
/**
 * Sam Siewert, December 2017
 * Modified by Krishna Suhagiya and Unmesh Phaterpekar
 *
 * @file    main.cpp
 * @brief   This file contains the main function and scheduler.
 * @date    30th April 2024
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

#include <pthread.h>
#include <sched.h>
#include <semaphore.h>
#include <signal.h>

#include <syslog.h>

#include <errno.h>

#include "capture.h"
#include "motor.h"
#include "ultrasonic_sensor.h"

#define USEC_PER_MSEC (1000)
#define NANOSEC_PER_SEC (1000000000)
#define NUM_CPU_CORES (1)

#define NUM_THREADS (3+1)

bool abortS=FALSE, abortS1=FALSE, abortS2=FALSE, abortS3=FALSE;
struct timeval start_time_val;

typedef struct
{
    int threadIdx;
    unsigned long long sequencePeriods;
} threadParams_t;

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);
   }
}

void intHandler(int arg)
{
    // Abort the sequencer itself
    abortS=TRUE; abortS1=TRUE; abortS2=TRUE; abortS3=TRUE;
}

void *sequencer(void *threadp)
{
    struct timeval current_time_val;
    struct timespec delay_time = {0,8333333}; // delay for 8.33 msec, 120 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);

    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) printf("residual=%lf, sec=%d, nsec=%d\n",
residual, (int)remaining_time.tv_sec, (int)remaining_time.tv_nsec);

                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);
        //syslog(LOG_INFO, "Sequencer cycle %llu @ sec=%d, msec=%d\n", seqCnt,
(int)(current_time_val.tv_sec-start_time_val.tv_sec),
(int)current_time_val.tv_usec/USEC_PER_MSEC);


        if(delay_cnt > 1) printf("Sequencer looping delay %d\n", delay_cnt);

        // Release each service at a sub-rate of the generic sequencer rate

        // Camera service = RT_MAX-1    @ 15 Hz
        if((seqCnt % 8) == 0) sem_post(&sem_camera);

        // Motor service = RT_MAX-2 @ 8 Hz
        if((seqCnt % 15) == 0) sem_post(&sem_motor);

        // Ultrasonic service = RT_MAX-3    @ 6 Hz
        if((seqCnt % 20) == 0) sem_post(&sem_ultrasonic);

    } while(!abortS);

    sem_post(&sem_camera); sem_post(&sem_motor); sem_post(&sem_ultrasonic);
    abortS1=TRUE; abortS2=TRUE; abortS3=TRUE;

    pthread_exit((void *)0);
}

int i, rc, scope;
cpu_set_t threadcpu;
pthread_t threads[NUM_THREADS];
```

```
threadParams_t threadParams[NUM_THREADS];
pthread_attr_t rt_sched_attr[NUM_THREADS];
int rt_max_prio, rt_min_prio;
struct sched_param rt_param[NUM_THREADS];
struct sched_param main_param;
pthread_attr_t main_attr;
pid_t mainpid;
bool is_obstacle_detected;

int main( int argc, char *argv[] )
{
    cpu_set_t allcpuset;

    printf("Welcome to Pi Parking System\r\n");

    setup_gpio();
    setup_ultasonic_sensor();
    openlog("pi-parking", 0, LOG_USER);
    syslog(LOG_INFO, "starting");

    /* Stop program with Ctrl+C */
    struct sigaction act;
    act.sa_handler = intHandler;
    sigaction(SIGINT, &act, NULL);

    CPU_ZERO(&allcpuset);

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

    printf("Using CPUS=%d from total available.\r\n", CPU_COUNT(&allcpuset));


    // initialize the sequencer semaphores
    //
    if (sem_init (&sem_camera, 0, 0)) { printf ("Failed to initialize
sem_camera\r\n"); exit (-1); }
    if (sem_init (&sem_motor, 0, 0)) { printf ("Failed to initialize
sem_motor\r\n"); exit (-1); }
    if (sem_init (&sem_ultrasonic, 0, 0)) { printf ("Failed to initialize
sem_ultrasonic\r\n"); exit (-1); }

    mainpid=getpid();

    rt_max_prio = sched_get_priority_max(SCHED_FIFO);
    rt_min_prio = sched_get_priority_min(SCHED_FIFO);

    rc=sched_getparam(mainpid, &main_param);
```

```
    main_param.sched_priority=rt_max_prio;
    rc=sched_setscheduler(getpid(), SCHED_FIFO, &main_param);
    if(rc < 0) perror("main_param");
    print_scheduler();

    pthread_attr_getscope(&main_attr, &scope);

    if(scope == PTHREAD_SCOPE_SYSTEM)
      printf("PTHREAD SCOPE SYSTEM\r\n");
    else if (scope == PTHREAD_SCOPE_PROCESS)
      printf("PTHREAD SCOPE PROCESS\r\n");
    else
      printf("PTHREAD SCOPE UNKNOWN\r\n");

    printf("rt_max_prio=%d\r\n", rt_max_prio);
    printf("rt_min_prio=%d\r\n", rt_min_prio);

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

      CPU_ZERO(&threadcpu);
      CPU_SET(0, &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);
      rc=pthread_attr_setaffinity_np(&rt_sched_attr[i], sizeof(cpu_set_t),
&threadcpu);

      rt_param[i].sched_priority=rt_max_prio-i;
      pthread_attr_setschedparam(&rt_sched_attr[i], &rt_param[i]);

      threadParams[i].threadIdx=i;
    }

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

    // camera_service = RT_MAX-1 @ 15 Hz
    //
    rt_param[1].sched_priority=rt_max_prio-1;
    pthread_attr_setschedparam(&rt_sched_attr[1], &rt_param[1]);
    rc=pthread_create(&threads[1],               // pointer to thread descriptor
                      &rt_sched_attr[1],         // use specific attributes
                      //(void *)0,               // default attributes
                      camera_service,                 // thread function entry
point
                      (void *)&(threadParams[1]) // parameters to pass in
                     );
```

```
    if(rc < 0)
        perror("pthread_create for camera\r\n");
    else
        printf("pthread_create successful for camera\r\n");

    // motor_service = RT_MAX-2 @ 8 Hz
    //
    rt_param[2].sched_priority=rt_max_prio-2;
    pthread_attr_setschedparam(&rt_sched_attr[2], &rt_param[2]);
    rc=pthread_create(&threads[2], &rt_sched_attr[2], motor_service, (void
*)&(threadParams[2]));
    if(rc < 0)
        perror("pthread_create for motor\r\n");
    else
        printf("pthread_create successful for motor\r\n");

    // ultrasonic_sensor_service = RT_MAX-3 @ 6 Hz
    //
    rt_param[3].sched_priority=rt_max_prio-3;
    pthread_attr_setschedparam(&rt_sched_attr[3], &rt_param[3]);
    rc=pthread_create(&threads[3], &rt_sched_attr[3], ultrasonic_sensor_service,
(void *)&(threadParams[3]));
    if(rc < 0)
        perror("pthread_create for sensor failed\r\n");
    else
        printf("pthread_create successful for sensor\r\n");

    // Wait for service threads to initialize and await release by sequencer.
    usleep(1000000);

    // Create Sequencer thread
    printf("Start sequencer\n");

    // Sequencer = RT_MAX   @ 120 Hz
    //
    rt_param[0].sched_priority=rt_max_prio;
    pthread_attr_setschedparam(&rt_sched_attr[0], &rt_param[0]);
    rc=pthread_create(&threads[0], &rt_sched_attr[0], sequencer, (void
*)&(threadParams[0]));
    if(rc < 0)
        perror("pthread_create for scheduler service 0");
    else
        printf("pthread_create successful for scheduler service 0\n");

   printf("Joining threads \r\n");


   for(i=0;i<NUM_THREADS;i++)
```

```
        pthread_join(threads[i], NULL);

    printf("TEST COMPLETE\n");
    return 0;
}
```

Capture.h

```
/*****************************************************************************
 * Copyright (C) 2024 by Krishna Suhagiya and Unmesh Phaterpekar
 *
 * Redistribution, modification or use of this software in source or binary
 * forms is permitted as long as the files maintain this copyright. Users are
 * permitted to modify this and use it to learn about the field of embedded
 * software. Krishna Suhagiya, Unmesh Phaterpekar and the University of Colorado
are not liable for
 * any misuse of this material.
 * ***************************************************************************/

/**
 * @file    capture.h
 * @brief   This file contains definition of various functions developed for motor
functionality
 * @date    2nd May 2024
 *
 */

#ifndef _CAPTURE_H
#define _CAPTURE_H


#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>

extern sem_t sem_camera;

/**
 * @brief Camera service to display the black screen/camera feed on the display
 */
void *camera_service(void *threadp);

#endif
```

Capture.cpp

```
/*
 *
 *  Example by Sam Siewert
 *  Modified by Krishna Suhagiya and Unmesh Phaterpekar
 *
 *  Created for OpenCV 4.x for Jetson Nano 2g, based upon
 *  https://docs.opencv.org/4.1.1
 *
 *  Tested with JetPack 4.6 which installs OpenCV 4.1.1
 *  (https://developer.nvidia.com/embedded/jetpack)
 *
 *  Based upon earlier simpler-capture examples created
 *  for OpenCV 2.x and 3.x (C and C++ mixed API) which show
 *  how to use OpenCV instead of lower level V4L2 API for the
 *  Linux UVC driver.
 *
 *  Verify your hardware and OS configuration with:
 *  1) lsusb
 *  2) ls -l /dev/video*
 *  3) dmesg | grep UVC
 *
 *  Note that OpenCV 4.x only supports the C++ API
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <iostream>

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>

#include <syslog.h>
#include <semaphore.h>
#include <pthread.h>

#include "capture.h"
#include "motor.h"
#include "time_stamp.h"

using namespace cv;
using namespace std;

extern bool is_forward;
extern bool is_reverse;
extern bool abortS1;

#define SYSTEM_ERROR (-1)
```

```
sem_t sem_camera;

void *camera_service(void *threadp)
{
    struct timespec start = {0,0};
    struct timespec stop = {0,0};
    static struct timespec wcet = {0,0};
    struct timespec time_taken = {0,0};
    printf("Camera service started\r\n");
    VideoCapture cam0(0);
    namedWindow("video_display");
    char winInput;

    if (!cam0.isOpened())
    {
        exit(SYSTEM_ERROR);
    }

    cam0.set(CAP_PROP_FRAME_WIDTH, 640);
    cam0.set(CAP_PROP_FRAME_HEIGHT, 480);

    Mat frame;
    Mat blackframe = Mat::zeros(Size(640, 480), CV_8UC3);

    while (!abortS1)
    {
        sem_wait(&sem_camera);
        if (is_reverse)
        {
            clock_gettime(CLOCK_REALTIME, &start);
            cam0.read(frame);
            imshow("video_display", frame);
            clock_gettime( CLOCK_REALTIME, &stop);
            delta_t(&stop, &start, &time_taken);
            if(check_wcet(&time_taken, &wcet))
            {
                syslog(LOG_INFO, "camera wcet: %lu sec, %lu msec (%lu microsec),
((%lu nanosec))\n\n", wcet.tv_sec, (wcet.tv_nsec / NSEC_PER_MSEC), (wcet.tv_nsec /
NSEC_PER_MICROSEC),wcet.tv_nsec);
            }
        }
        else
        {
            imshow("video_display", blackframe);
        }

        winInput = waitKey(10);
```

```
    }

    destroyWindow("video_display");
    printf("Camera service stopped\n");
    pthread_exit(NULL);
}
```

Motor.h

```
/*****************************************************************************
 * Copyright (C) 2024 by Krishna Suhagiya and Unmesh Phaterpekar
 *
 * Redistribution, modification or use of this software in source or binary
 * forms is permitted as long as the files maintain this copyright. Users are
 * permitted to modify this and use it to learn about the field of embedded
 * software. Krishna Suhagiya, Unmesh Phaterpekar and the University of Colorado
are not liable for
 * any misuse of this material.
 * ***************************************************************************/

/**
 * @file    motor.h
 * @brief   This file contains declaration of various functions developed for
motor functionality
 * @date    1st May 2024
 *
 */

#include <stdio.h>
#include <stdint.h>
#include <wiringPi.h>

extern bool is_obstacle_detected;
extern sem_t sem_motor;

/*
 * @brief Function to setup GPIOs for motor
 */
void setup_gpio();

/*
 * @brief Function to trigger the specified motor to move at the specified speed
in the specific direction
 */
void control_motor(int motor, int speed, int direction);

/*
```

```
 * @brief Motor service to move the motor in the direction based on the gear
status/sensor status
 */
void *motor_service(void *threadp);
```

Motor.cpp

```
/*****************************************************************************
 * Copyright (C) 2024 by Krishna Suhagiya and Unmesh Phaterpekar
 *
 * Redistribution, modification or use of this software in source or binary
 * forms is permitted as long as the files maintain this copyright. Users are
 * permitted to modify this and use it to learn about the field of embedded
 * software. Krishna Suhagiya, Unmesh Phaterpekar and the University of Colorado
are not liable for
 * any misuse of this material.
 * ***************************************************************************/

/**
 * @file    motor.cpp
 * @brief   This file contains definition of various functions developed for motor
functionality
 * @date    1st May 2024
 *
 */

#include <stdio.h>
#include <syslog.h>
#include <wiringPi.h>
#include <semaphore.h>
#include <pthread.h>

#include "motor.h"
#include "time_stamp.h"

sem_t sem_motor;
bool is_forward = true;
bool is_reverse = false;
extern bool abortS2;

// GPIO pin definitions
#define MOTOR_PWM_A 1  // PWM for Motor A (GPIO 18)
#define MOTOR_IN1_A 4  // Direction IN1 for Motor A (GPIO 23)
#define MOTOR_IN2_A 5  // Direction IN2 for Motor A (GPIO 24)
#define MOTOR_PWM_B 23 // PWM for Motor B (GPIO 13)
#define MOTOR_IN1_B 3  // Direction IN1 for Motor B (GPIO 22, WiringPi pin 3)
#define MOTOR_IN2_B 2  // Direction IN2 for Motor B (GPIO 27, WiringPi pin 2)
#define STBY_PIN 6     // Standby pin (GPIO 25)
```

```
#define BUTTON_PIN 7   // Button pin (GPIO 4, WiringPi pin 7)

// Initialize GPIO pins
void setup_gpio() {
    wiringPiSetup();
    pinMode(BUTTON_PIN, INPUT);  // Set button pin as input
    pullUpDnControl(BUTTON_PIN, PUD_UP);  // Enable pull-up resistor
    pinMode(MOTOR_PWM_A, PWM_OUTPUT);
    pinMode(MOTOR_IN1_A, OUTPUT);
    pinMode(MOTOR_IN2_A, OUTPUT);
    pinMode(MOTOR_PWM_B, PWM_OUTPUT);
    pinMode(MOTOR_IN1_B, OUTPUT);
    pinMode(MOTOR_IN2_B, OUTPUT);
    pinMode(STBY_PIN, OUTPUT);
    digitalWrite(STBY_PIN, HIGH);  // Take motor driver out of standby mode
}

// Control motor speed and direction
void control_motor(int motor, int speed, int direction) {
    if (motor == 1) {  // Motor A
        pwmWrite(MOTOR_PWM_A, speed);
        digitalWrite(MOTOR_IN1_A, direction ? HIGH : LOW);
        digitalWrite(MOTOR_IN2_A, direction ? LOW : HIGH);
    } else if (motor == 2) {  // Motor B
        pwmWrite(MOTOR_PWM_B, speed);
        digitalWrite(MOTOR_IN1_B, direction ? HIGH : LOW);
        digitalWrite(MOTOR_IN2_B, direction ? LOW : HIGH);
    }
}

void *motor_service(void *threadp)
{
    struct timespec start = {0,0};
    struct timespec stop = {0,0};
    static struct timespec wcet = {0,0};
    struct timespec time_taken = {0,0};
    int button_state = 0;
    printf("Motor started\r\n");

    while(!abortS2)
    {
        sem_wait(&sem_motor);
    clock_gettime(CLOCK_REALTIME, &start);
        button_state = digitalRead(BUTTON_PIN);  // Read button state
        if (button_state == 1) {  // Button is pressed
        is_forward = !is_forward;  // Toggle forward state
        is_reverse = !is_reverse;  // Toggle reverse state
    }
```

```
    if(is_forward | is_reverse)
    {
        if(is_forward)
        {
            // Test Motor A
            control_motor(1, 512, 1);  // Half speed forward
            // Test Motor B
            control_motor(2, 512, 1);  // Half speed forward
        }
        else if(is_reverse)
        {
            control_motor(1, 512, 0);  // Half speed backward
            control_motor(2, 512, 0);  // Half speed backward
        }
    }

    if(is_obstacle_detected)
    {
        control_motor(1, 0, 0);    // Stop Motor A
        control_motor(2, 0, 0);    // Stop Motor B
    }
    clock_gettime( CLOCK_REALTIME, &stop);
    delta_t(&stop, &start, &time_taken);
    if(check_wcet(&time_taken, &wcet))
    {
        syslog(LOG_INFO, "motor wcet: %lu sec, %lu msec (%lu microsec), ((%lu
nanosec))\n\n", wcet.tv_sec, (wcet.tv_nsec / NSEC_PER_MSEC), (wcet.tv_nsec /
NSEC_PER_MICROSEC),wcet.tv_nsec);
    }
    }


    delay(500);

    control_motor(1, 0, 0);    // Stop Motor A

    control_motor(2, 0, 0);    // Stop Motor B

    syslog(LOG_INFO, "Motor stopped\r\n");

    pthread_exit(NULL);
}
```

Timestamp.h

```
/*****************************************************************************
 * Copyright (C) 2024 by Krishna Suhagiya and Unmesh Phaterpekar
 *
 * Redistribution, modification or use of this software in source or binary
 * forms is permitted as long as the files maintain this copyright. Users are
 * permitted to modify this and use it to learn about the field of embedded
 * software. Krishna Suhagiya, Unmesh Phaterpekar and the University of Colorado
are not liable for
 * any misuse of this material.
 * ***************************************************************************/

/**
 * @file    time_stamp.h
 * @brief   This file contains declaration of various functions developed for time
profiling
 * @date    3rd May 2024
 *
 */

#include <stdio.h>
#include <stdint.h>
#include <sys/time.h>

#define MSEC_PER_SEC (1000)
#define NSEC_PER_SEC (1000000000)
#define NSEC_PER_MSEC (1000000)
#define NSEC_PER_MICROSEC (1000)

/*
 * @brief Function to measure the difference between the two timestamps
 */
int delta_t(struct timespec *stop, struct timespec *start, struct timespec
*delta_t);

/*
 * @brief Function to check if the current time observed is more than WCET
 */
bool check_wcet(struct timespec *time_taken, struct timespec *wcet);
```

Timestamp.cpp

```
/*****************************************************************************
 * Copyright (C) 2024 by Krishna Suhagiya and Unmesh Phaterpekar
 *
 * Redistribution, modification or use of this software in source or binary
 * forms is permitted as long as the files maintain this copyright. Users are
 * permitted to modify this and use it to learn about the field of embedded
 * software. Krishna Suhagiya, Unmesh Phaterpekar and the University of Colorado
are not liable for
 * any misuse of this material.
 * ***************************************************************************/

/**
 * @file    time_stamp.cpp
 * @brief   This file contains definition of various functions developed for time
profiling
 * @date    3rd May 2024
 *
 */

#include "time_stamp.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;

  if(dt_sec >= 0)
  {
    if(dt_nsec >= 0)
    {
      delta_t->tv_sec=dt_sec;
      delta_t->tv_nsec=dt_nsec;
    }
    else
    {
      delta_t->tv_sec=dt_sec-1;
      delta_t->tv_nsec=NSEC_PER_SEC+dt_nsec;
    }
  }
  else
  {
    if(dt_nsec >= 0)
    {
      delta_t->tv_sec=dt_sec;
      delta_t->tv_nsec=dt_nsec;
    }
```

```
    else
    {
      delta_t->tv_sec=dt_sec-1;
      delta_t->tv_nsec=NSEC_PER_SEC+dt_nsec;
    }
  }

  return(1);
}

bool check_wcet(struct timespec *time_taken, struct timespec *wcet)
{
  if(((double)((double)time_taken->tv_sec * MSEC_PER_SEC) +
(double)((double)time_taken->tv_nsec / NSEC_PER_MSEC)) > ((double)((double)wcet-
>tv_sec * MSEC_PER_SEC) + (double)((double)wcet->tv_nsec / NSEC_PER_MSEC)))
  {
    wcet->tv_sec = time_taken->tv_sec;
    wcet->tv_nsec = time_taken->tv_nsec;
    return true;
  }
  return false;
}
```

Ultrasonic_sensor.h

```
/*****************************************************************************
 * Copyright (C) 2024 by Krishna Suhagiya and Unmesh Phaterpekar
 *
 * Redistribution, modification or use of this software in source or binary
 * forms is permitted as long as the files maintain this copyright. Users are
 * permitted to modify this and use it to learn about the field of embedded
 * software. Krishna Suhagiya, Unmesh Phaterpekar and the University of Colorado
are not liable for
 * any misuse of this material.
 * ***************************************************************************/

/**
 * @file    ultrasonic_sensor.cpp
 * @brief   This file contains declaration of various functions developed for
ultrasonic sensor functionality
 * @date    2nd May 2024
 *
 */

#include <stdio.h>
#include <wiringPi.h>

extern sem_t sem_ultrasonic;
```

```
/*
 * @brief Function to setup the ultrasonic sensor
 */
void setup_ultasonic_sensor();

/*
 * @brief The ultrasonic sensor service to detect the obstacles and set the
is_obstacle_detected flag
 */
void *ultrasonic_sensor_service(void *threadp);
```

Ultrasonic_sensor.cpp

```
/*****************************************************************************
 * Copyright (C) 2024 by Krishna Suhagiya and Unmesh Phaterpekar
 *
 * Redistribution, modification or use of this software in source or binary
 * forms is permitted as long as the files maintain this copyright. Users are
 * permitted to modify this and use it to learn about the field of embedded
 * software. Krishna Suhagiya, Unmesh Phaterpekar and the University of Colorado
are not liable for
 * any misuse of this material.
 * ***************************************************************************/

/**
 * @file    ultrasonic_sensor.cpp
 * @brief   This file contains definition of various functions developed for
ultrasonic sensor functionality
 * @date    2nd May 2024
 *
 */

#include <stdio.h>
#include <syslog.h>
#include <wiringPi.h>
#include <semaphore.h>
#include <pthread.h>

#include "motor.h"
#include "time_stamp.h"

// Define GPIO pins for Trigger and Echo pins
#define TRIG 15
#define ECHO 16
#define DISTANCE_THRESHOLD 7

sem_t sem_ultrasonic;
extern bool is_forward;
```

```
extern bool is_reverse;
extern bool abortS3;

void setup_ultasonic_sensor() {
    wiringPiSetup();
    pinMode(TRIG, OUTPUT);
    pinMode(ECHO, INPUT);

    // Ensure the trigger pin is low
    digitalWrite(TRIG, LOW);
    delay(30);
}

void *ultrasonic_sensor_service(void *threadp) {
    struct timespec start = {0,0};
    struct timespec stop = {0,0};
    static struct timespec wcet = {0,0};
    struct timespec time_taken = {0,0};
    printf("Distance Measurement In Progress\n");

    while (!abortS3) {
        sem_wait(&sem_ultrasonic);
        if(is_forward == true)
        {
            clock_gettime(CLOCK_REALTIME, &start);
            struct timeval detection_start, detection_end;
            long travel_time, distance;

            // Triggering the sensor for 10 microseconds
            digitalWrite(TRIG, HIGH);
            delayMicroseconds(10);
            digitalWrite(TRIG, LOW);

            // Wait for the echo start
            while (digitalRead(ECHO) == LOW);

            // Record time of signal return
            gettimeofday(&detection_start, NULL);
            while (digitalRead(ECHO) == HIGH);
            gettimeofday(&detection_end, NULL);

            // Calculate the distance
            travel_time = (detection_end.tv_sec - detection_start.tv_sec) *
1000000L + detection_end.tv_usec - detection_start.tv_usec;
            distance = travel_time / 58;
            if(distance < DISTANCE_THRESHOLD)
            {
                syslog(LOG_INFO, "Distance: %d cm\n", distance);
```

```
                is_obstacle_detected = true;
            }
            else
            {
                is_obstacle_detected = false;
            }
                clock_gettime( CLOCK_REALTIME, &stop);
                delta_t(&stop, &start, &time_taken);
                if(check_wcet(&time_taken, &wcet))
                {
                syslog(LOG_INFO, "sensor wcet: %lu sec, %lu msec (%lu microsec),
((%lu nanosec))\n\n", wcet.tv_sec, (wcet.tv_nsec / NSEC_PER_MSEC), (wcet.tv_nsec /
NSEC_PER_MICROSEC),wcet.tv_nsec);
                }
        }
    }

    syslog(LOG_INFO, "Sensor stopped\n");

    pthread_exit(NULL);
}
```