University of Colorado Boulder

REAL TIME ROAD SIGN DETECTION

BY

SIDDHANT JAJOO
SATYA MEHTA
AYUSH DHOOT

UNDER THE GUIDANCE OF:

PROFESSOR TIMOTHY SCHERR,
UNIVERSITY OF COLORADO BOULDER

5/4/2019

TABLE OF CONTENTS

INTRODUCTION

The Project is aimed at designing a bot that captures certain symbols and subsequently carries out certain actions based on the symbols. The project design includes motors, ultrasonic sensor and Pi-Camera Module V2. The board used for processing is the Raspberry Pi.

The Camera would search for this sign in its frame:

When the Camera detects the **YIELD** sign, the motors stop. In case of an obstacle in the path of the bot, the care is taken that it does not crash onto the object. For this purpose, an ultrasonic sensor is attached onto the bot which would constantly detect for any obstacles in its path. If an obstacle is detected, it will immediately stop and resume its operation if the object is removed from its path.

The Scheduling policy that would be used for this project on Linux is SCHED_FIFO. The priorities are assigned on the frequency of operation of different tasks, thus following the Rate Monotonic Policy. In order to determine which task runs at a particular point of time, a scheduler is designed which posts the semaphores for the respective tasks. All the tasks are blocked on their respective semaphores which would be released by the scheduler.

This project is being done in a group of 3 people. Different tasks are being designed by different individuals:

1. Motor Task - Satya Mehta.
2. Camera Task - Ayush Dhoot.
3. IR Sensor Task - Siddhant Jajoo.
4. Scheduler Task – Satya Mehta, Siddhant Jajoo, Ayush Dhoot.

The Ultrasonic Sensor Task

This task will check any object, if detected in its range as and when deployed by the scheduler. If an object is detected, the task will raise the obstacle flag which would be checked in the motor task. If no object is detected the flag is set low.

The Camera Task

This task will check any signs - Left, Right and Forward, if detected in its range as and when deployed by the scheduler. If any sign is detected, the task will raise the corresponding motor direction flag which would be checked in the motor task. If no object is detected the flags are set low.

The Motor Task

This task will check for all the flags set by the other two tasks and take the corresponding actions to move the bot forward, right, left or stop.

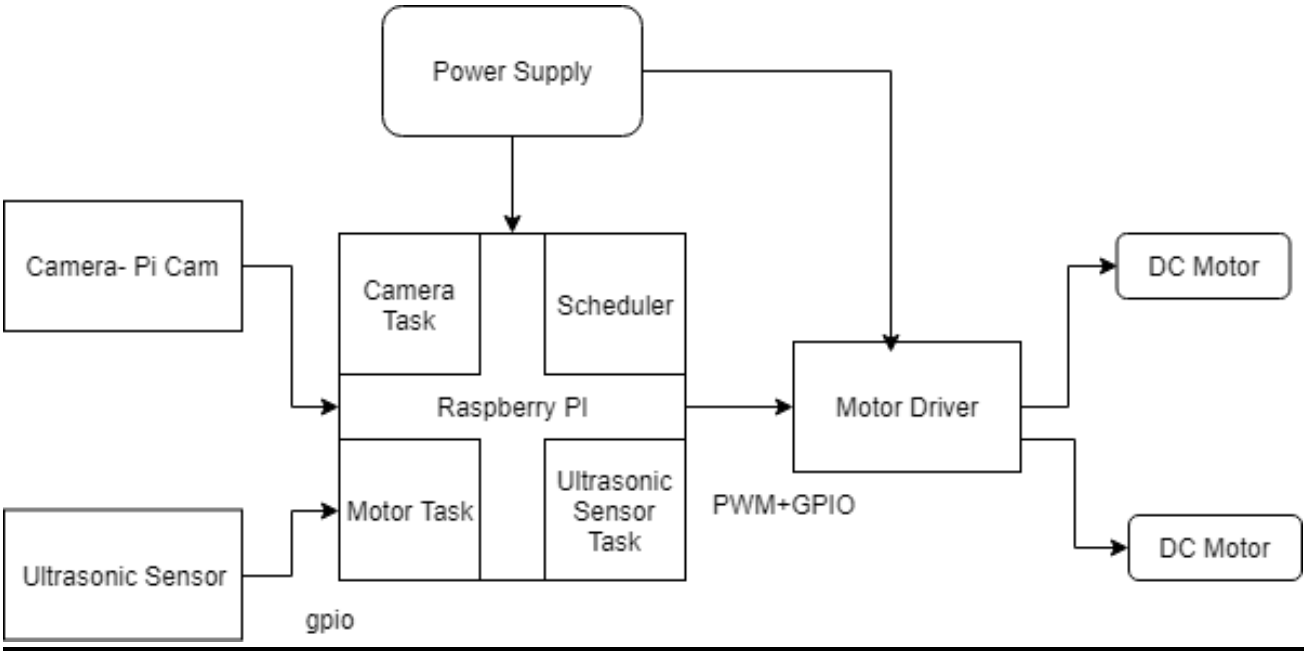
The Scheduler Task

This task will periodically be invoked to check which task should be deployed next. The different tasks would be deployed by releasing the semaphores based on the frequencies of the different services.

This Project implements the following features and concepts:

1. Multithreading using POSIX Threads.
2. Rate Monotonic Priority Implementation
3. Synchronization Primitives.

BLOCK DIAGRAM



In the above block diagram, the various hardware used in the project are shown. The project uses Raspberry Pi as the main processing unit and the central node for other interfaces. The bot requires DC motors to move which is interfaced using a motor driver IC which controls the speed of the bot by using the PWM signals generated. The motors are powered using a external 9V battery. For the sign detection service, there is a PiCam interfaced to the RPI which captures frames from a live feed to detect signs using image detection algorithms. The last hardware used is an ultrasonic sensor which returns the total distance between the bot and any obstacle present in the face of the sensor.

FUNCTIONAL REQUIREMENTS

1. Raspberry Pi Camera Interface.
   In order to detect signs, the system needs to be interfaced with the Raspberry Pi Camera. As soon as the sign is detected, the motor needs to be switched off. The Camera functionality is implemented in a different thread with the help of a scheduler. The Camera thread basically acts as the sensor thread.
2. Ultrasonic sensor interface
   The system is interfaced with an ultrasonic sensor in order to detect any obstacle in its path. As soon as an obstacle is detected, the motor is switched off. This functionality is implemented in a different thread as compared to the scheduler. This thread acts as a sensor thread.
3. DC Motors interface
   The DC motors are used to react to the flags set in the Camera and Ultrasonic threads. The DC motors are controlled in a separate task. The DC motors virtually feed of the values from the ultrasonic and camera task thus making it an actuator task.
4. The system is scheduled using the Rate Monotonic Policy. The highest frequency task would have the highest priority.
5. The scheduler posts semaphores of the services at predetermined frequency.

CAPABILITY REQUIREMENTS

For any sign detection bot, it is required to process images to detect signs. Hence, it is required to train the processor using machine vision algorithm. This bot is trained for the YIELD sign using Haar Cascade classifiers before the system can be operated. The bot can detect signs like left, right, yield, stop, etc to move in the direction. Hence, the camera service is required to continuously compare the captured frame with the pre given .xml file. More the signs, more will be the required computation time for the camera services. For our requirement, the bot is trained to stop at YIELD sign and if there is no sign detected the bot moves. The bot also has Ultrasonic obstacle detection feature which stops abruptly when any obstacle is detected. Hence, when no obstacle is detected the motor should stop.

Capability Requirements:

- The system should run based on the sign detected.
- The motor should actuate based on the inputs from the sensor.
- The ultrasonic sensor should detect the obstacle and set the flag for the motor service.
- The system should stop when YEILD sign is detected.
- The bot should stop on detection of the obstacle.
- The bot should stop on detection of both YIELD and obstacle.

REAL TIME REQUIREMENTS

- Scheduler should run at the desired frequency which is 200 Hz for our application.
- Scheduler should schedule the individual services at their desired frequency.
- As per our schedule requirements, the Motor service should run at 40Hz, Ultrasonic service should run at 4Hz and the camera service should run at 4Hz.
- When two tasks are posted at the same then the higher priority task preempts lower one.
- Priorities are assigned based on Rate Monotonic Policy with Motor has the highest priority, Ultrasonic sensor as the second highest priority and the Camera has the lowest priority.
- RM policy was chosen as opposed to any dynamic deadline policy due to the fact that all services would be independent of its execution of each other. Suppose if 1 of the service fails to meet its deadline, there would not be any cascading effect on the other tasks. The other higher priority tasks would preempt the lower ones to carry out their functionality. In addition to this, debugging is easier for a deterministic schedule.
- Each task in the system is critical for the functioning of the bot. Failure of one task results in the failure of the entire system. The system is an implementation of hard real-time system as it is a prototype of any autonomous vehicle which requires instant decisions to avoid collisions. The time periods and deadlines have been selected after multiple iterations.

FUNCTIONAL DESIGN OVERVIEW

RASPBERRY PI 3B +

[*]

PLATFORM RESOURCES

- Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- 1GB LPDDR2 SDRAM
- CSI camera port for connecting a Raspberry Pi camera
- 4 USB 2.0 ports
- WiringPi GPIO libraries.
- Extended 40-pin GPIO header
- WIFI support.

There are a few reasons which made us choose RPI for the project.
As far as real-time operating systems are concerned there were two options - FreeRTOS and Linux OS.

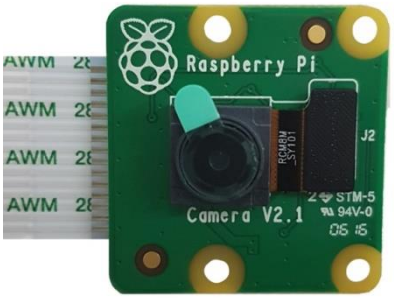
- The project requires interfacing camera to capture images and hence it was difficult to analyze how complex it would be on FreeRTOS. Also, the resources on TIVA board to process the images are quite limited. Therefore the only option left was Linux.
- It is easier to mount a small board like Rpi than mounting a large board like Jetson on the bot.
- Pi Camera can represent an image with more pixels as compared to the logitech camera.
- RPI has inbuilt GPIO libraries which helps to directly control the pins whereas Jetson would have required to call more system calls to operate on GPIO files systems in linux.
- The Latest RPI model has WIFI support and for this application which is a moving bot it was helpful for us to control it remotely using VNC server.

PINOUT

PINS (P9 HEADER)	FUNCTIONALITY
1	GND
2	GND
3	VCC (3.3 V)
11	RX (UART FOR TIVA)
12 (GPIO 60)	CS (NRF24L01+)
13	TX (UART FOR TIVA)
14 (GPIO 50)	IRQ (NRF24L01+)
16 (GPIO 51)	CE (NRF24L01+)
18	MOSI (NRF24L01+)
21	MISO (NRF24L01+)
22	SCLK (NRF24L01+)

SENSORS AND ACTUATORS

RASPBERRY PI CAMERA V2.1



The PI Camera is used to detect the signs in the path of the autonomous bot.

- Sony IMX219 8-megapixel sensor.
- Supports 1080p30, 720p60 and VGA90 video modes as well as still capture.
- Simply attaching ribbon cable to the CSI port of RPi.
- No InfraRed (NoIR) filter on the lens which makes it perfect for doing Infrared photography and taking pictures in low light (twilight) environments.

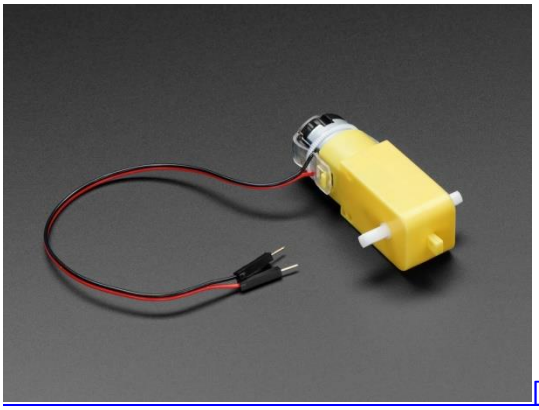
ULTRASONIC SENSOR (HC-SR04)

[*]

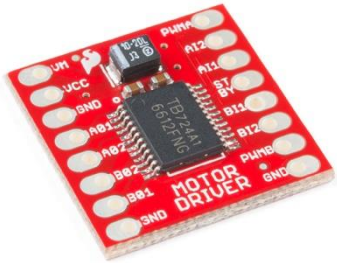
The Ultrasonic Sensor is used to detect an obstacle in the path of the autonomous bot.

- Working voltage: 3-5.5VDC.
- Static current: less than 2 mA.
- Induction angle: not more than 15 degrees.
- Detection range: 2-400 cm.
- Accuracy: 3mm.

DC MOTOR AND MOTOR DRIVER



[*]

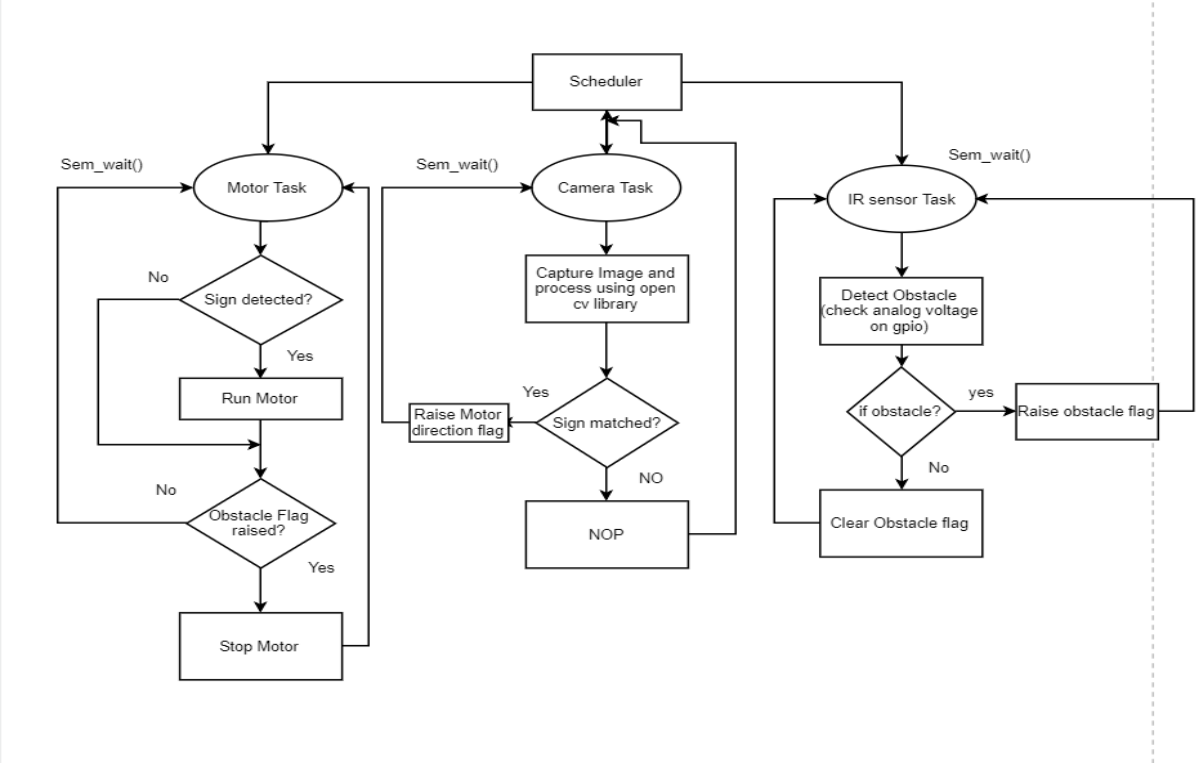


[*]

The DC Motor is used to drive the bot forward. The Motor Driver is used to regulate the current to the DC motor with the help of PWM.

- TT DC Gearbox Motor with a gear ratio of 1:48
- Rated Voltage : 3VDC up to 6VDC

FLOWCHART

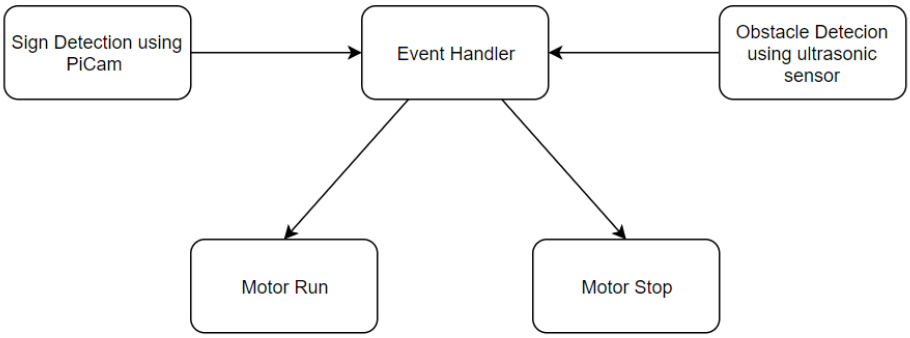


The above flowchart shows the software implementation for the project. It gives us the base on which the whole code is written over.

The flowchart above consists of 3 threads each named Motor, Camera and Ultrasonic Sensor Task. These tasks are executed with the help of semaphores that are posted in the scheduler. The Camera and the Ultrasonic Sensors act as the sensor tasks and based on certain values detect if there is a sign or obstacle in the bot's path respectively. If the values cross the threshold, the respective threads raise a corresponding flag. These flags are checked at the determined schedule in the Motor Task which acts as the actuator task. Depending on the value of the flag, the motor is switched on or off.

CONTROL FLOW DIAGRAM

The flag from the sensor and the camera task is sent to the event handler which checks the flags and gives the motor instruction to run or stop accordingly.



We have implemented flags for each events. When any sign is detected the flag is raised. The event handler is implemented in the motor service as its a lone actuator service. Similarly, the obstacle detection service raises a flag when an obstacle is detected. Motors are driven based on the events flag raised.

REAL TIME ANALYSIS AND DESIGN

DCT CALCULATION AND RATE MONOTONIC ANALYSIS

TASK	PRIORITY	DEADLINE	WCET	TIME PERIOD	FREQUENCY
Motor	Highest	33.33	1	33.33	30
Ultrasonic Sensor	Medium	250	130	250	4
Pi Camera	Lowest	500	150	500	2

PRIORTIES

The above table shows us the Priority, Deadline, WCET, Time Period and Frequency of the services in this project. Since we are following the Rate Monotonic Policy Schedule, the service which has the highest frequency would have the highest priority. The frequency of a particular service depends upon the application. In this case, the motor service is the one which feeds on the values obtained from the camera and the ultrasonic sensors. So, it is imperative to have the motor service to check the values obtained from the other two tasks frequently as this motor task switches the motor on or off. Thus, the motor task has the highest priority with the highest frequency or lowest time period/deadline. In case of Rate Monotonic Policy, Time period is equal to the Deadline. The Ultrasonic sensor and the Pi Camera have been assigned as the second and the third highest priority. The reason for this is that there can be human intervention or obstacle at any point of time in the path of the bot. This would logically result in having the bot to check for any obstacles in its path more frequently than the camera task thus resulting in having its priority higher than the camera task.

So the final priority order is as follows:

- Motor Task = RT_MAX - 1.
- Ultrasonic Task = RT_MAX - 2.
- Camera Task = RT_MAX - 3.

NOTE: The Scheduler task will always have the highest priority which is RT_MAX.

WORST CASE EXECUTION TIME (WCET)

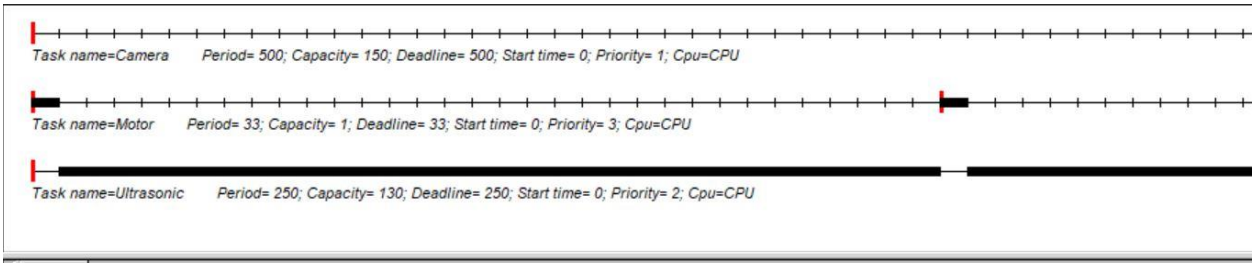
The Motor Service has the lowest WCET (10-20 microseconds) as it just drives the GPIO pins depending on the flags set in the Motor and the Camera task.

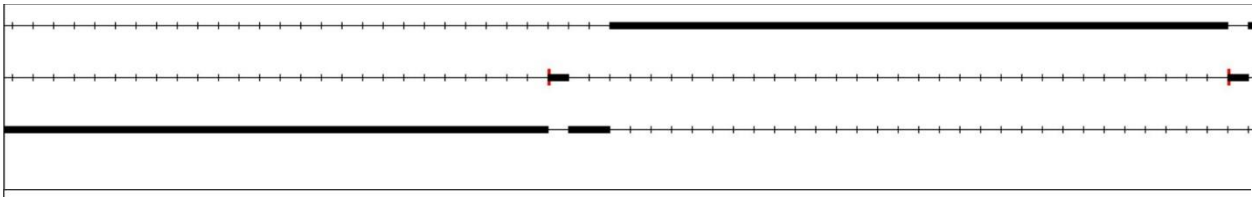
The Ultrasonic Sensor typically takes around 10 – 20 microseconds of time if an obstacle is detected. If no obstacle is detected in its range the there is no low pulse on the echo pin. The ultrasonic sensor module then takes about 130 milliseconds to complete its functionality and return garbage value indicating out of range condition.

The camera takes the highest time of all to complete the processing. The WCET for the Camera task is taken as around 150 milliseconds.

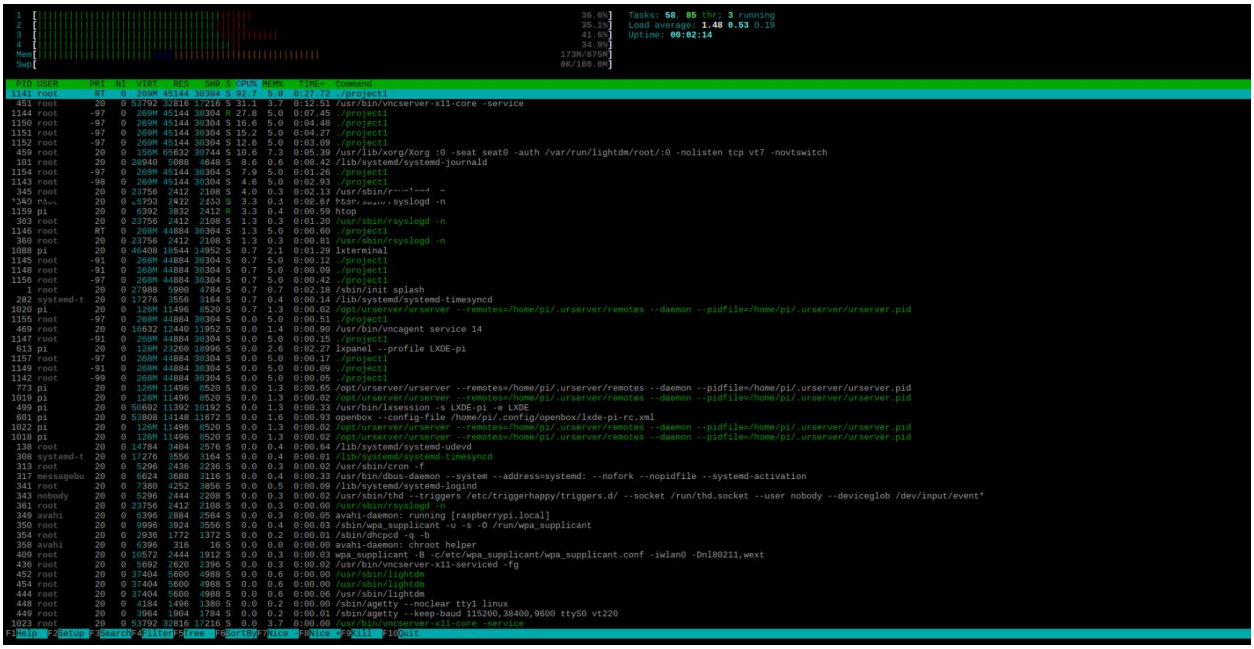
TIMING DIAGRAMS

The image below is the Cheddar image of the schedule with the values of DCT specified in the table above. It can be seen that the camera task appears at the last after the Ultrasonic task has finished its execution since the camera task has the lowest priority. It can be observed here that the Utilization factor obtained here is around 85%.



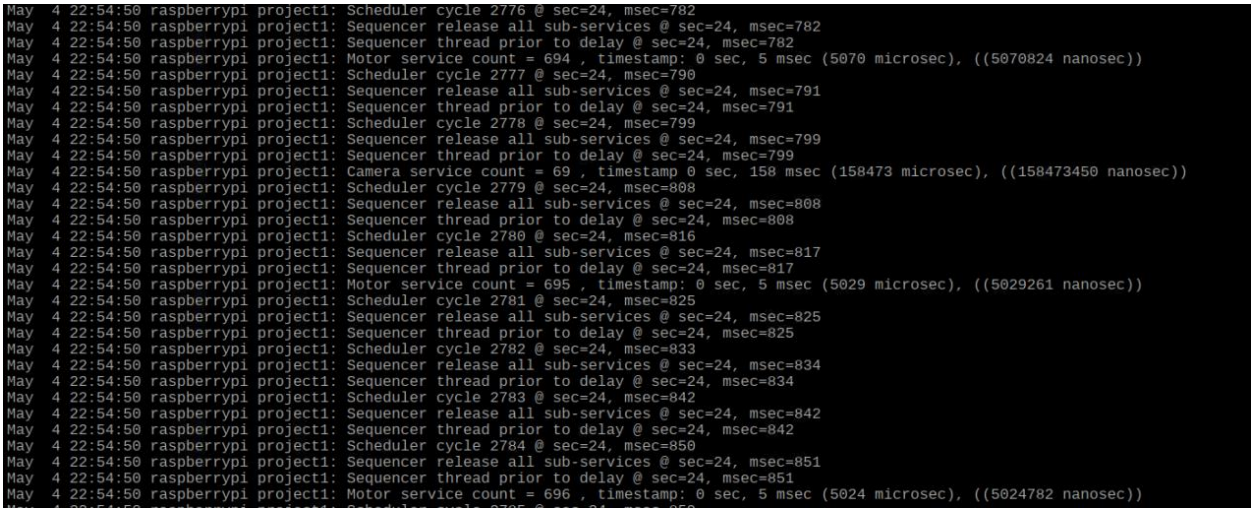


```
Scheduling feasibility, Processor CPU :
1) Feasibility test based on the processor utilization factor :

- The base period is 16500 (see [18], page 5).
- 2470 units of time are unused in the base period.
- Processor utilization factor with deadline is 0.85030 (see [1], page 6).
- Processor utilization factor with period is 0.85030 (see [1], page 6).
- In the preemptive case, with RM, we can not prove that the task set is schedulable because the processor utilization factor 0.85030 is
more than 0.77976 (see [1], page 16, theorem 8).


2) Feasibility  test based on worst case task response time :

- Bound on task response time :  (see [2], page 3, equation 4).
    Camera => 423
    Ultrasonic => 135
    Motor => 1
- All task deadlines will be met : the task set is schedulable.
```

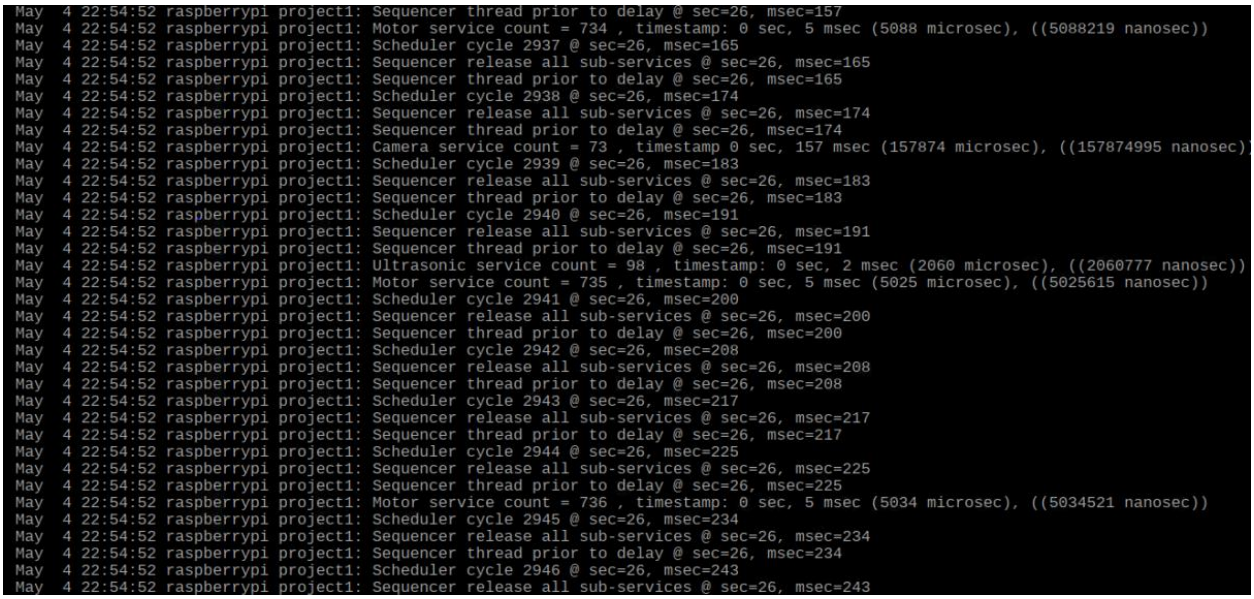


The above image shows the CPU loading due to the code that has been run on RPI. The light blue row at the top shows the CPU loading value as 92%. It can be observed that the there is a difference of about 7% between the utilization factor calculated and the one above. This might be due to the background processes or the linux kernel too.

PROOF OF CONCEPT SCREENSHOTS



Syslog screenshot



Syslog screenshot

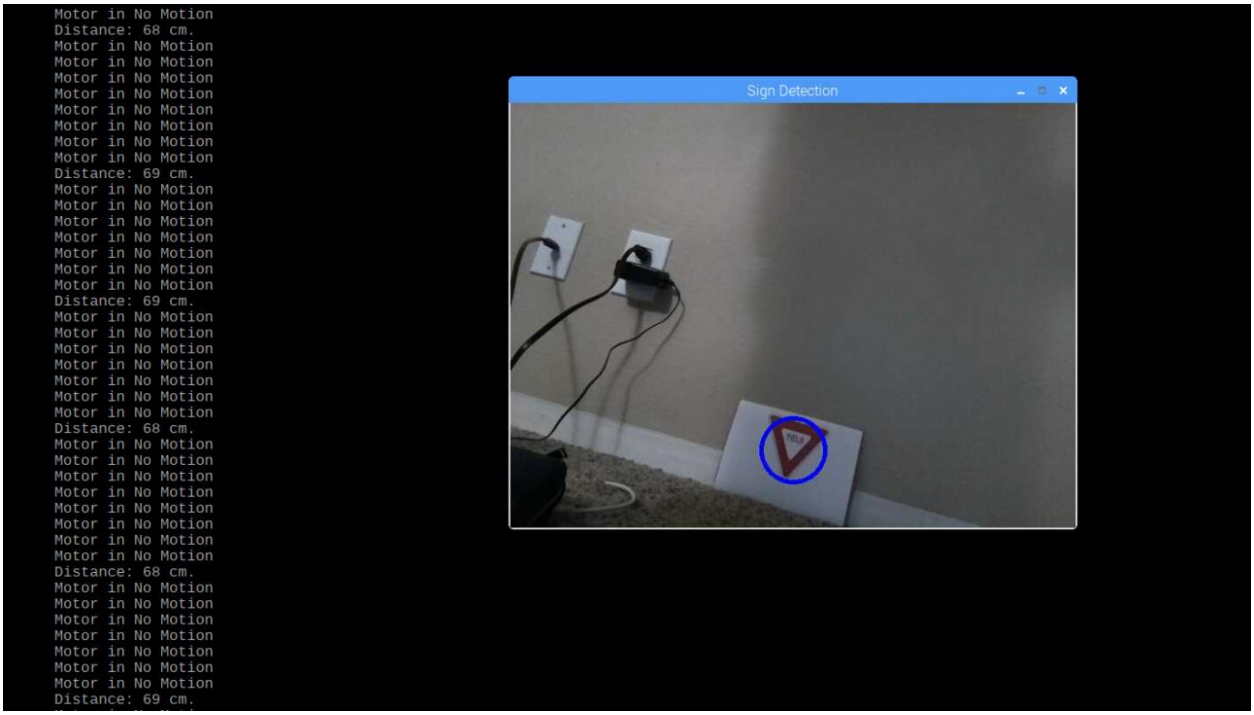
Test Cases:

```
Motor in Forward Motion
Motor in Forward Motion
Motor in Forward Motion
Motor in Forward Motion
Distance: 88 cm.
Motor in Forward Motion
Motor in Forward Motion
Motor in Forward Motion
Motor in Forward Motion
Motor in Forward Motion
Motor in Forward Motion
Motor in Forward Motion
Motor in Forward Motion
Motor in Forward Motion
Motor in No Motion
Motor in No Motion
Distance: 2316 cm.
Motor in No Motion
Motor in No Motion
Motor in No Motion
Motor in No Motion
Motor in No Motion
Motor in No Motion
Motor in No Motion
```

Forward motion

```
otor in No Motion
otor in No Motion
otor in No Motion
istance: 3 cm.
otor in No Motion
otor in No Motion
otor in No Motion
otor in No Motion
otor in No Motion
otor in No Motion
otor in No Motion
otor in No Motion
istance: 3 cm.
otor in No Motion
otor in No Motion
otor in No Motion
otor in No Motion
otor in No Motion
otor in No Motion
otor in No Motion
istance: 3 cm.
otor in No Motion
otor in No Motion
otor in No Motion
otor in No Motion
otor in No Motion
otor in No Motion
otor in No Motion
istance: 5 cm.
otor in No Motion
otor in No Motion
otor in No Motion
otor in No Motion
otor in No Motion
otor in No Motion
otor in No Motion
istance: 10 cm.
otor in No Motion
```

Ultrasonic Detect



Sign Detection

The working of the entire code and its implementation was demonstrated in class. Syslog Screenshots have been attached as proof of concept code.

The Demo Video can be found over here:
https://www.youtube.com/watch?v=5pxvPxIWPSY&feature=youtu.be

FAIL SAFE METHODS

The system has been designed to be fail safe. For example if the camera says to move forward and the ultrasonic sensor detects an obstacle and wants to stop it will stop. The same is valid if the camera detects the yield sign and the ultrasonic sensor does not detect any obstacle.

VERIFICATION PLANS

TASK	EXPECTED DATE	Completion Date
Raspberry Pi Setup	4/26/2019	04/25/2019
Camera - Interfacing	4/27/2019	4/25/2019
Setting up Motors with Motor Driver IC	4/28/2019	4/28/2019
Ultrasonic Sensor – Interfacing	4/28/2019	4/28/2019
Assembling the bot and interfaces	4/29/2019	4/30/2019
Tasks and Scheduler Creation	4/30/2019	5/1/2019
Logging functionality	5/1/2019	5/1/2019
Running Tests and Analysis	5/2/2019	5/3/2019
Testing of final application	5/3/2019	5/3/2019

TEST PLAN AND RESULTS

Tests	Result	Validation
Interface PiCam with RPi	Pass	Successfully interfaced PiCam
PiCam capturing frames from a live feed	Pass	PiCam captures single frames from the feed
Camera detecting signs (Yield, Stop) using Haar Cascade classifiers	Pass	Camera detects the signs
Camera service WCET calculation	Pass	WCET for camera is 150ms
Interface Ultrasonic sensor	Pass	Successfully interfaced the sensor
Detecting an obstacle	Pass	Obstacle detected and distance returned
Obstacle service WCET calculation	Pass	WCET for sensor is 130ms
Interface Motor and Motor Driver IC	Pass	Sucessfully interfaced two motors along with a motor driver IC
Motor working	Pass	Motor stops and runs forward with varying PWM
Motor service WCET calculation	Pass	WCET for motor is 1ms
Integrating all above interfaces with scheduler	Pass	All modules are interfaced and working as per the schedule provided
Final Application	Pass	Working as designed

CHALLENGES FACED

- The schedule failed when we tried to detect more than one sign. This was because the processor was taking a lot of time to process a single sign. If the processor speed was more, it could have been done.
- WCET of ultrasonic sensor was longer than expected. It was almost comparable to the WCET of Camera task. Thus it was getting difficult to schedule the 3 services.
- If we increase frequency of camera service, the schedule was no longer feasible.

CONCLUSION

The project was implemented successfully with all the major requirements completed. The real time and functional requirements were taken care of. The Schedule was feasible and was executed with utilization factor of 85%. All the test cases have been passed.

The project included only one sign detection because of hardware constraints and the processor speed. If the processor speed is increased, a lot more signs could have been added and a fully function autonomous bot could be made. The future scope of this project can be in the form of unmanned vehicle or indoor mapping.

This project has helped us to learn new concepts and perfect already known concepts such as OpenCv, multithreading, semaphores, scheduling policy, DCT calculation and analysis and Haar Cascade.

The Demo Video can be found over here:
https://www.youtube.com/watch?v=5pxvPxIWPSY&feature=youtu.be

REFERENCES

- https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf
- https://github.com/sparkfun/HC-SR04_UltrasonicSensor/blob/master/Firmware/HC-SR04_UltrasonicSensorExample/HC-SR04_UltrasonicSensorExample.ino
- http://www.cse.unt.edu/~rakl/KAH08.pdf
- https://users.ece.cmu.edu/~koopman/des_s99/real_time/
- https://github.com/opencv/opencv/blob/master/samples/cpp/facedetect.cpp
- http://wiringpi.com/
- https://autottblog.wordpress.com/programming-the-car/opencv/

APPENDICES

1. Code
2. Project Validation and Simulation.pdf