University of Colorado
Boulder

ECEN 5623 Real-Time Embedded Systems
Force Sensitive Intelligent Skateboard

Under the guidance of:

Prof. Timothy L. Scherr

By:

Hardik Senjaliya

Sarthak Jain

Vatsal Sheth

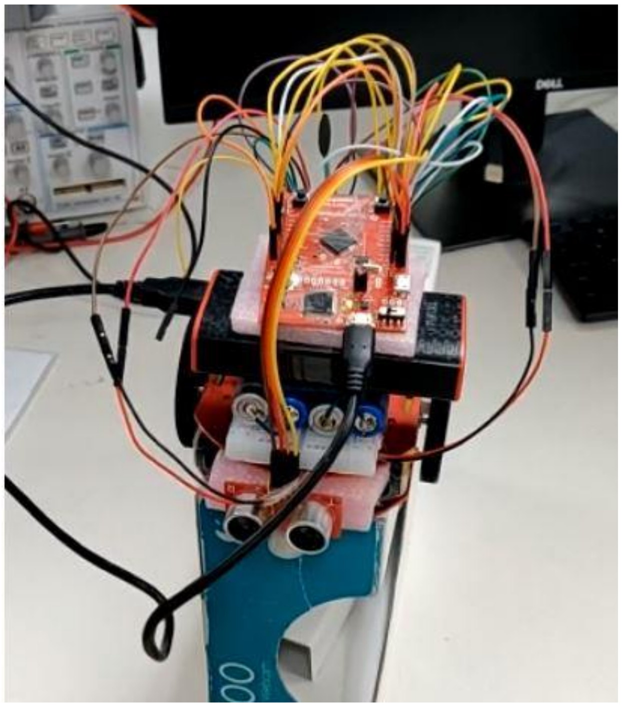
Dated 05/04/2019

Contents

I. Introduction

This is a project for the course of Real-Time Embedded Systems (ECEN 5623), spring of 2019. We have attempted to design a “smart skateboard” of sorts, in a further attempt to make man’s life easier, specially for those who do not know how to ride a skateboard, but still wish to do so anyway.

The objective of this particular project is to design a real-time system which schedules a set of three services, each working in conjunction with the other to produce a real-time application relying on human inputs and possible external obstacles to deliver a pseudo-intelligent skateboard.



The end product will be a skateboard, with two force sensitive sensors on it, each below the foot of a potential skateboarder. The sensors would each drive a motor, and as an extra precaution/failsafe, the skateboard will sport an ultrasonic sensor at the front to detect any incoming obstacles. In order to turn in a particular direction (for eg. left), some extra pressure needs to be applied to the left foot. The left foot’s sensor would detect this increase in pressure and would increase the RPMs of the right motor being driven by it to turn in that direction. Similarly, to turn right, some extra pressure would be applied to the right sensor, which would drive the left motor’s RPMs, increasing the speed of the left wheel and resulting in a right turn. As a failsafe precaution, the ultrasonic sensor mounted on the front of the board would also take readings at a particular period and would override the pressure sensors to slow the motors and bring the board to a gradual stop.

This entire project has been implemented using the Rate Monotonic Scheduling Policy, reasons for which has been elaborated later in this report.

II. Functional Description

A. Design Specifications:

We are implementing a hard-real-time scheduler to manage a set of three services using the rate monotonic scheduling policy. We have chosen this scheduling policy in keeping with the design considerations as supplied in [1] and keeping in mind our implementation of this product. While designing the scheduler for this product, we had the choice of using either a processor-heavy implementation, in order to make the schedule as responsive as possible, or going with a relaxed implementation in which we would keep a processor margin, for the rare case when an interrupt from the pressure sensor takes control of the processor, not allowing the other tasks to run as per priority.

Fortunately, the consideration for memory or I/O does not arise in this case, as the only memory required for each task will be overwritten each time the tasks are run, and I/O latency has been provided for by use of the Rate Monotonic protocol as per [2]. Task specific are as provided below:

a. Task 1: This task shall constantly poll the ADC for readings from the pressure sensors and will notify the other tasks in case the ADC output crosses a certain threshold/range. We have estimated the period of requesting ADC data as **60 milliseconds**, based on the time taken by ADC to process the range provided to it by the pressure sensors, and convert into a digital range. This digital range will drive the motor speeds by notifying task 2.
b. Task 2: This task shall drive the motors based on notifications provided to it by Task 1. This task has the responsibility to increase or decrease the motors' speed by the range of values provided to it by the pressure sensor task. Giving this task a request period of less than 5 milliseconds seems redundant, as the data would be same within that period. Therefore, we decided to give it a period of **60 milliseconds** as well.
c. Task 3: This task shall periodically instruct the ultrasonic sensor to send a pulse, and based on the readings from it, will either do nothing, or if it detects an object in front, will gradually bring the motors to a stop based on the distance it receives. We have given this task a request period of **100 milliseconds**, giving it sufficient time to receive a reply pulse as well as take a decision and notify the motors' task.

Keeping paper [2] in mind, we have also plotted a Cheddar simulation which helps us determine three key design steps:

I. Measured the system to find out whether a performance enhancement is needed. This was determined by the % processor utilization, which as shown below is close to the RM LUB boundary, leading us to the conclusion further enhancement is not required.
II. Measured any performance bottlenecks, which would only occur in the case of notifications from Task 2. Sufficient margin has been kept for the same.
III. Iterate constantly to check if any parameters have changed.

Keeping paper [3] in mind, we have designed our project keeping these constraints in mind:

I. Minimum performance criteria: We require a certain response time, above which design of the product would result in late responses subsequent miss of deadlines. Our current provision of 5, 5 and 10 milliseconds would at first design, seem sufficient to prevent missing of any deadlines.

II. Dependability constraints: The product needs to be dependable, in the sense that if the ultrasonic sensor notifies the motor task to slow down, the motor task should respond reliably. This can only be confirmed by testing and is on our checklist.

III. Cost constraints: The product cannot be of high economic value. Keeping this in mind, the only cost incurred is that of the pressure sensors, motor driver and ultrasonic sensor.

B. Individual Contributions:

Our team consists of Vatsal Sheth, Hardik Senjaliya and Sarthak Jain.

a. Vatsal Sheth – Vatsal will be working on integrating the pressure sensors, sampling their analog data via use of the ADC, and decide the range of values for a particular set of commands. This will be implemented in a single task, which monitors both pressure sensors and notifies the motors' task to actuate the motors accordingly. Additionally, Vatsal will be working on task creation and program backbone, ie. size of variables to be used, task priorities, task stacks, etc.

b. Hardik Senjaliya – Hardik will be working on actuating the motors based on notifications from the motors' task and the ultrasonic sensor's task. Based on these notifications, the motors will be instructed to either speed up, slow down, reverse or stop completely. Additionally, Hardik will be working on the hardware design, the chassis integration with the TIVA, motor drivers and the motors themselves.

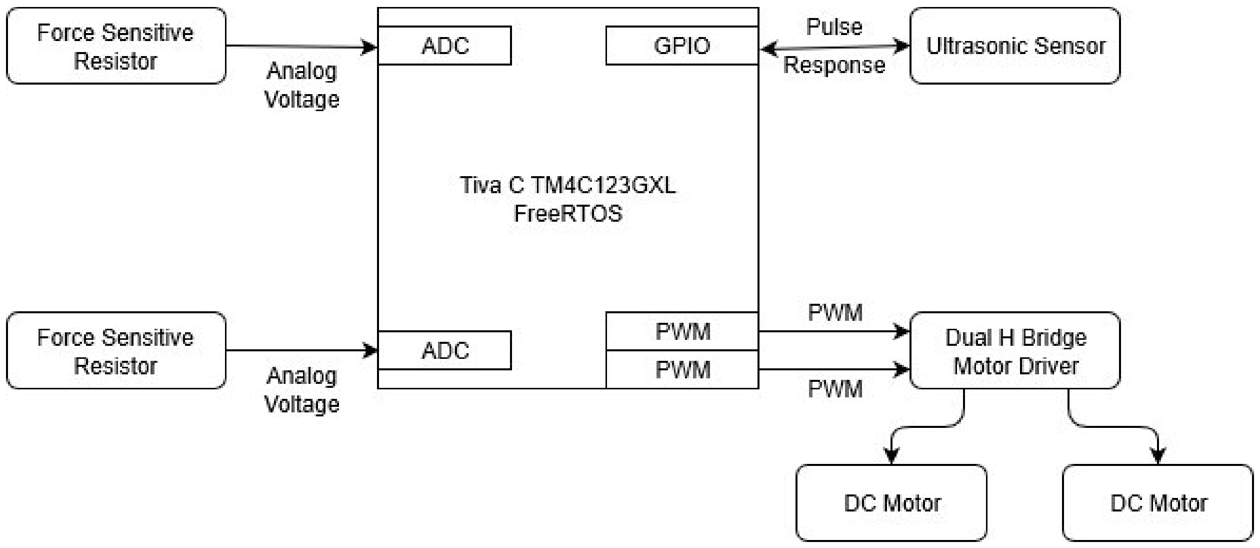
c. Sarthak Jain – Sarthak will be working on the ultrasonic sensor task. Periodically, a pulse will be sent forth to check for any obstacle in front, and based on the time of response, distance from the board to the object will be computed and a decision will be sent to the motors' task. Additionally, Sarthak will be working on the scheduler, and timing of each task's release and servicing.

III. Block Diagram and Software Flow

The scope of our project is to do the following things:

- Measure the pressure on each of the force sensitive resistor to get the rider input for direction control. This pressure difference is relative that is, if equal pressure is applied on both the sensor then it should move ahead.
- Continuously at regular interval check for any obstacle ahead and to avoid collision. This is implemented in two ways. If the object is far enough then, it reduces the speed and then eventually stops. And if obstacle detected is near then to apply brakes immediately.
- Motor Driver control needs to manage the speed and relative turn to apply. Turn is implemented using moving one motor and stopping other one. Forward and breaking is applied using two level speed control, one with 80% duty cycle and other with 50% duty cycle PWM signal.

A. Block Diagram

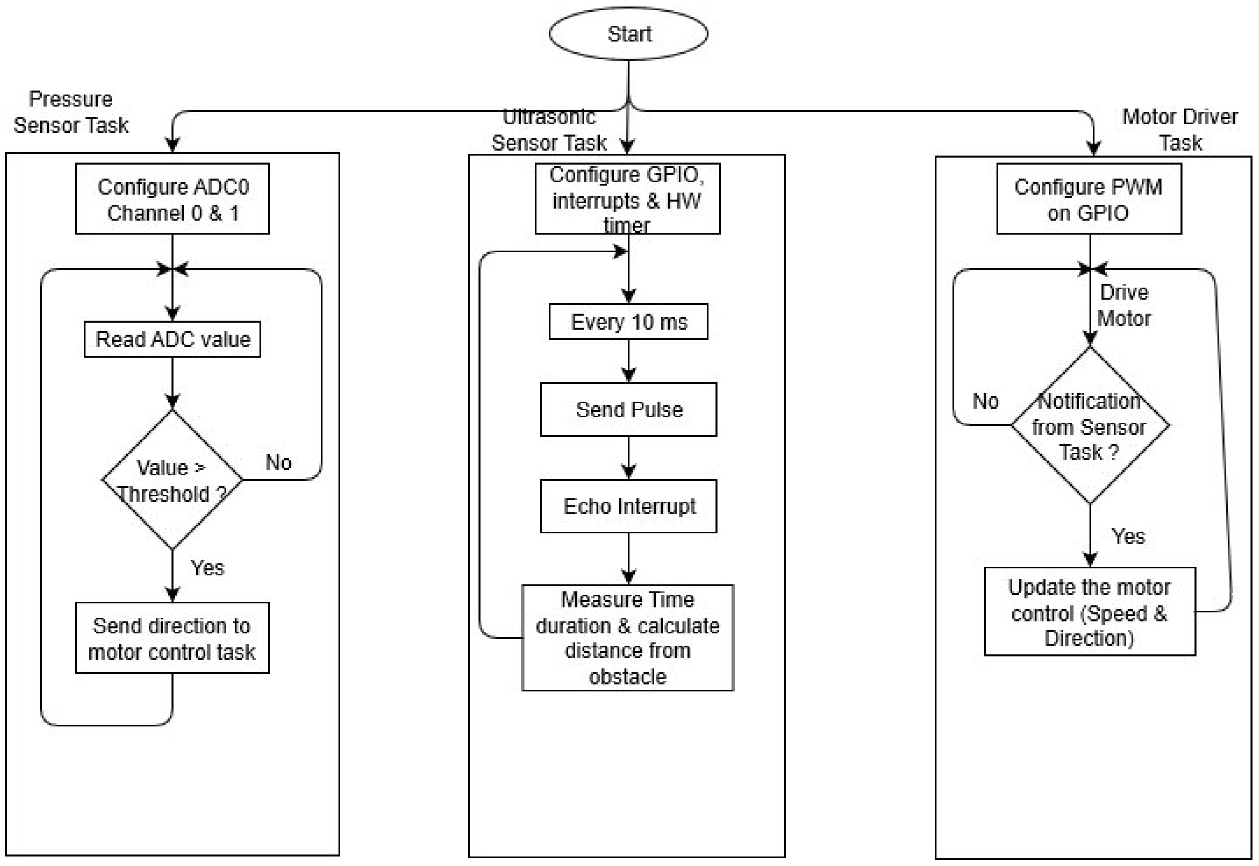


- FreeRTOS was used with Tiva C 123 GXL Development board. FreeRTOS is a real time operating system for embedded device which is distributed under MIT license. It is relatively simple and small, which is from the fact that the entire implementation is distributed between just three C files. This makes it very easy to port between various hardware platform. Few of the critical functions are implemented in assembly which makes that part architecture specific. FreeRTOS kernel is preemptible based on priority which makes it a first choice for real-time application.
- Force Sensitive Resistor is a kind of flex resistor, which changes it resistance based on the pressure applied on it. Part used in our prototype varies its resistance from few MΩ for no load to 3 KΩ for max load of 1 Kg which it can detect. Voltage Divider circuit is used where one resistor of 3 KΩ and this sensor as a second resistor is used. Change in pressure results in change in resistance

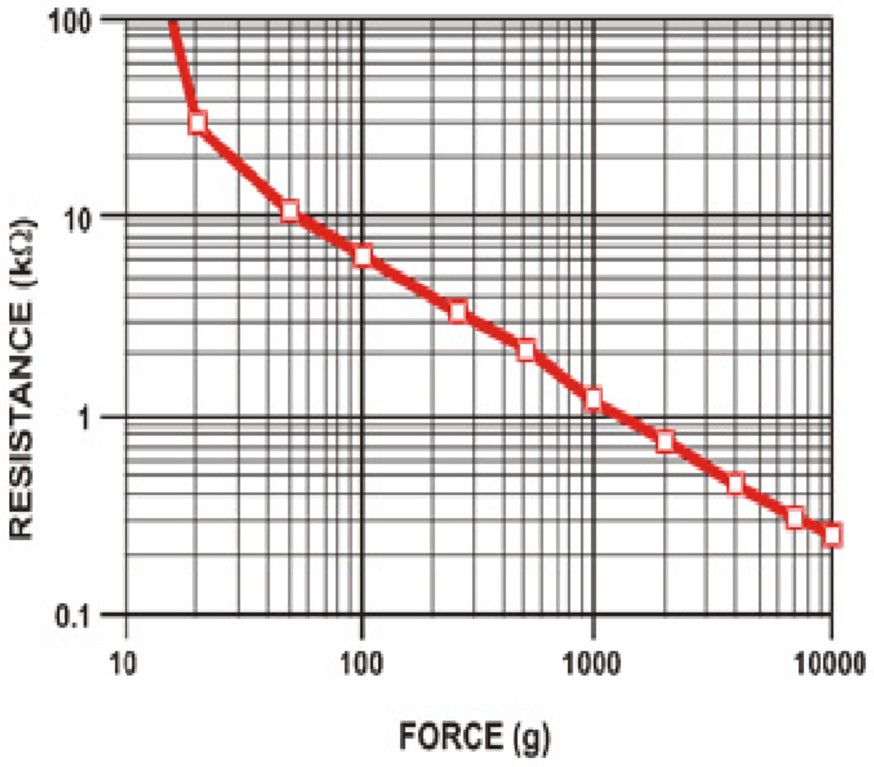
which in turn will vary the voltage at the common junction which can be used to measure the change in physical pressure applied on it.

- ADC in Tiva C series micro-controller is implemented based on programmable hardware sequencer so that ADC operations can be made independent of processor interventions. There are 4 possible programmable modes for this sequencer. Sequencer 0 can take at most generate eight steps of converted data, similarly sequencer 1 and 2 can generate at most 4 and sequencer 3 can at most generate 1 conversion data and then they write to FIFO. Various interrupts are programmable which can then be used to generate interrupt and read data from this FIFO. In our implementation we have used sequencer 1 where only 2 steps are used. 1st step is configured to work with single ended mode on channel 0 which is physically connected to left side force resistor sensor. 2nd step configured to work with single ended mode on channel 1 which is physically connected to right side force resistor sensor.
- PWM generator in Tiva C series micro-controller is implemented with hardware timer which is physically mapped to specific GPIO through memory mapped IOs. One hardware PWM generator is configured to generate PWM signal of specific frequency and duty cycle. This is then provided to motor driver module. GPIO are used to control the motor driver input pins, which decides it direction of rotation.
- GPIO is used to send trigger pulse and another GPIO is polled continuously to check for echo pulse. Difference between both of them is measured using Hardware Timer and is used to calculate the distance.
- Two DC Motor are used, each of which has following specifications:
    1. Voltage: 3V – 6V
    2. No load current: 200 mA
    3. Stall current: 3A @ 6V
    4. Gearbox Ratio: 48:1
    5. Wheel speed: 65 RPM @ 3V unloaded

B. Software Flowchart

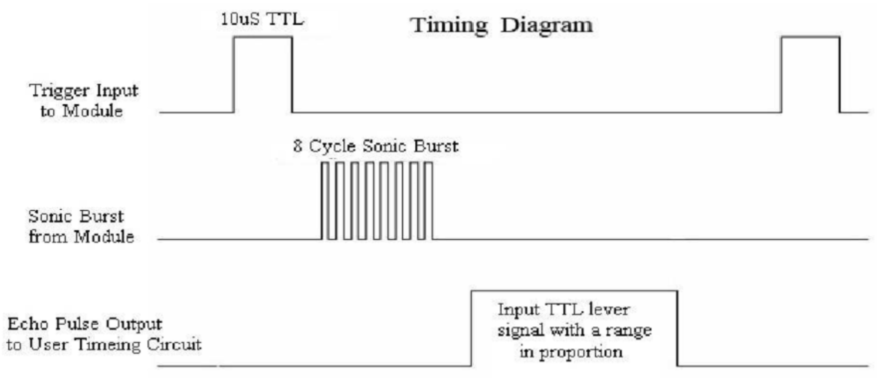

Start
Pressure Sensor Task
Ultrasonic Sensor Task
Motor Driver Task
Configure ADC0 Channel 0 & 1
Read ADC value
Value > Threshold ?
No
Yes
Send direction to motor control task
Configure GPIO, interrupts & HW timer
Every 10 ms
Send Pulse
Echo Interrupt
Measure Time duration & calculate distance from obstacle
Configure PWM on GPIO
Drive Motor
Notification from Sensor Task ?
No
Yes
Update the motor control (Speed & Direction)

- Hardware Timer is used to schedule three tasks. This timer has a timeout of 10 ms and it is used to post semaphores for all three tasks to schedule it using Rate Monotonic Policy. RM policy is used compared to dynamic policies because there is no such specific need in application, and it introduces run time complexities.
- Force sensitive task gets its semaphore posted by scheduler every 60 ms that is, its request period is 60 ms. After that it triggers one-shot ADC sample conversion on sequencer 1 and keeps polling interrupt flag to check for completion of conversion. Based on the ADC value decision is made on whether force is above certain threshold to write in motor driver task queue with appropriate direction and speed control command. Below is the graph of force applied and resistance.



Typical Force Curve

- Ultrasonic Sensor task gets it semaphore posted every 100 ms that is, its request period 100 ms. After that it sends a pulse to sensor and starts a hardware timer. Then it keeps on polling for the echo pulse on the GPIO. On getting that it measures the time difference and calculates the distance and takes a decision and passes that message to motor driver task using queue. If no obstacle is detected than it passes the message to keep moving forward to motor task. Only when obstacle is detected it passes the message to stop in front of the queue which will act as the highest priority message. Below is the timing diagram for trigger and echo pulse from datasheet.


10uS TTL
Timing Diagram
Trigger Input to Module
8 Cycle Sonic Burst
Sonic Burst from Module
Echo Pulse Output to User Timeing Circuit
Input TTL lever signal with a range in proportion

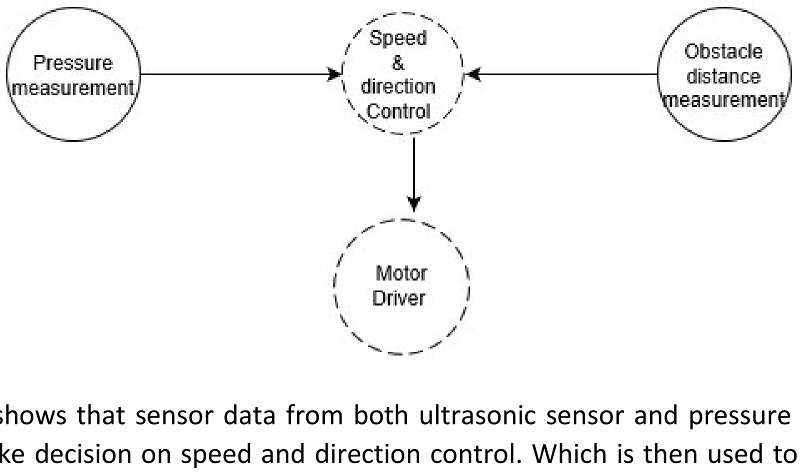
Timing Diagram

- Motor driver task need to be scheduled only if there are any updates from sensor. Fastest sensor task is pressure sensor task with request period of 60ms therefore, motor driver task is also scheduled every 60ms by scheduler by posting the semaphore. After that it reads from the queue and executes direction and speed control by manipulating PWM and GPIO which drive IN1 and IN2 signals of motor driver. In case it receives a stop message from ultrasonic task then it resets the queue to flush it so as to avoid any erroneous steps after stopping because of previous data in queue. Below is the table to control motor direction and speed from datasheet of H-bridge driver IC:

Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

Hardware – Software Control Function

C. Data Flow Diagram



Data Flow Diagram shows that sensor data from both ultrasonic sensor and pressure sensor is passed through queue to take decision on speed and direction control. Which is then used to drive the motor appropriately.

IV. Capabilities and Requirements

A. Functional Capabilities:

- *Force Sensitive Sensor interface*
  Depending on force this sensor changes its resistance from 1 M Ohm (No force) to 2.5 K Ohm (Max Force). Two ADC channels are used to get force value from both the sensors every 10 ms. Based on the values and set thresholds, this task will decide the direction of turn if required and issue and appropriate command to motor control task.

- *Ultrasonic Sensor interface*
  Short pulse is sent on one of the pins of this sensor and time between this pulse and GPIO interrupt on echo pin is used to measure the distance of obstacle. This task is scheduled every 100 ms to look for obstacle. In case an obstacle is detected, it issues breaking command to motor control task.

- *DC Motor Control using H bridge motor driver*
  Motor control using H bridge gives the capability to change the direction of rotation of motor. By rotating one wheel forward and other one backwards gives the capability to make sharp turns. That is it reduces turning radius.

- *DC Motor Speed Control using PWM*
  Speed control is achieved by changing the duty cycle of PWM signal. On time of PWM signal will determine the average current delivered which will determine the speed of rotation.

- *Task synchronization, Message passing, Notification and Tasks scheduler*
  Task scheduler will be implemented using hardware timer. Period of this timer will be HCF of request period of all tasks which gives the scheduler running at lowest frequency. At appropriate time this handler will post semaphores for respective tasks. Each task will be waiting for their respective semaphores. This mechanism will achieve scheduling task based on RM policy. Message passing is implemented through task notification where payload will carry the appropriate command for motor control task.

B. Completeness:

All the three tasks are executed as per RM policy. They are independent in their execution, so RM policy can be applied on it. Motor control task depends on other two tasks for data if there are any changes to be made otherwise it is independent. Whenever this task gets notification then, it needs to execute some register write which is its worst-case execution time. But since these writes are negligible in execution time, they can be ignored for execution time calculation. And thus, this system satisfies all the assumptions made in RM policy.

C. Real-Time Requirements

- *Pressure Sensor Task*

  The Pressure Sensor Task needs to acquire ADC sensor readings every 60 ms by triggering two channels of the ADC in One-Shot mode. The requirement of it is not actually for 60 ms and can be kept a lot lower. However, for reasons not completely understood, if the pressure sensor task is given a request period of less than 60 ms, the scheduler begins to fail. It is our understanding that the pressure sensor task begins producing data faster than the motor task can process it as well as respond to the other task sending it data. Another reason for selecting 60 ms is that the fastest human response time for touch is 150 ms, and the pressure sensor task can easily sample the data well before that.
- *Ultrasonic Sensor Task*

  Ultrasonic Sensor Task should acquire distance from obstacle if any, every 100ms by measuring echo of sent pulse. The task does so by sending a pulse and then holding the line high for 10 µs. It then polls the echo pin for a response, and then multiplies the time at which echo is received by the speed of sound, giving the distance at which, the object is located from the ultrasonic sensor.
- *Motor driver Task*

  Motor driver task should have a request period equal to at least that of any one of the other two services. Giving it a request period less than both the other tasks makes no sense, as it would anyway have no functionality to perform before receiving any inputs from one of the other two tasks.

D. Worst Case Execution Time Requirements

- *The Worst-Case Execution Time Requirement* is such that the WCET of each task should be less than its respective request period.
- *The WCET* should also be such that it is in range to satisfy the schedule's feasibility as per the Rate Monotonic scheduling policy.

V. Functional Description

Pressure sensors are polled every 10 ms and ADC will be used in one shot conversion mode to fetch the data. Based on the readings from two sensors decision is made on direction change. If required, then it will notify the motor control task about that. Similarly, ultrasonic sensor fetches data every 100 ms and in case an obstacle is detected it will notify the motor control task about the rate at which it needs to break and stop ultimately. Motor control task on other hand maintains the direction and speed at last set value, till it receives notification from either of the other tasks and reacts accordingly.

A. Task 1

This task shall constantly poll the ADC for readings from the pressure sensors and will notify the other tasks in case the ADC output crosses a certain threshold/range. The execution time has been estimated at **5 milliseconds**, with a request period of **60 milliseconds**. The runtime was selected at 3 milliseconds by estimating the total ADC time, which include sampling time + conversion time. Sampling time depends upon ADC clock frequency and the mode. Conversion time depends upon hardware averaging and various long sampling modes and high-speed conversion settings, etc. Therefore, 3 milliseconds are a sufficient overestimation for ADC time. Actual time will depend on the configurations mentioned above and will be updated in the final report.

B. Task 2

This task shall drive the motors based on notifications provided to it by Task 1. This task has the responsibility to increase or decrease the motors' speed by the range of values provided to it by the pressure sensor task. Giving this task a request period of less than 60 milliseconds seems redundant, as the data would be same within that period. Therefore, we decided to give it a period of **60 milliseconds** as well. The capacity has been estimated at **3 milliseconds**, which seems sufficient. This time has been decided keeping in mind that this task has only two actions based on notifications. If a notification to increase speed comes, the task shall change the PWM duty cycle by writing to the timer's registers. If a notification to reverse direction comes, the task shall toggle the input pins to the motor driver on the motor driver. For this simple writing to registers or toggling of pins, 1 millisecond seems to be a sufficient overestimation.

C. Task 3

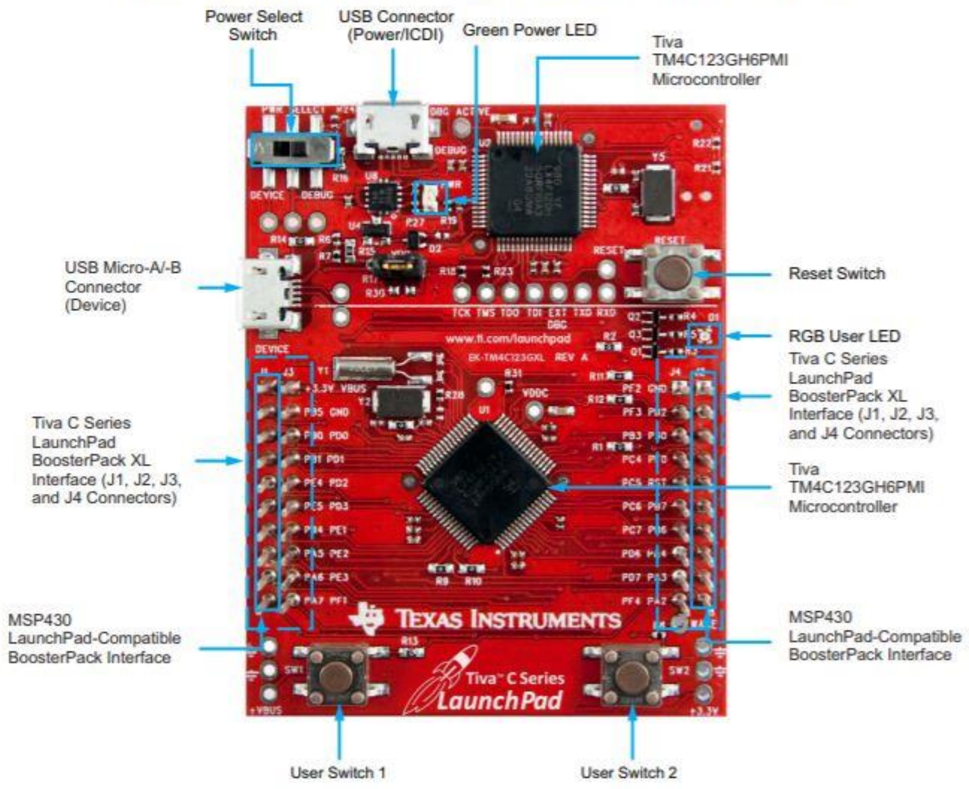
This task shall periodically instruct the ultrasonic sensor to send a pulse, and based on the readings from it, will either do nothing, or if it detects an object in front, will gradually bring the motors to a stop based on the distance it receives. We have given this task a request period of **100 milliseconds**, giving it sufficient time to receive a reply pulse as well as take a decision and notify the motors' task. The execution time has been estimated at **25 milliseconds**. This calculation can be showed as follows:

Maximum detectable distance by ultrasonic sensor = 4m.
Speed of sound = 340 m/s.
Time taken for pulse and received echo = 2 * (4/340) ~= 23.4 ms.
Considering I/O latency, estimate WCET = 25 milliseconds.

VI. Resources and Analysis

A. Platform Resources

Figure 1-1. Tiva C Series TM4C123G LaunchPad Evaluation Board


Power Select Switch
USB Connector (Power/ICDI)
Green Power LED
Tiva TM4C123GH6PMI Microcontroller
USB Micro-A/-B Connector (Device)
Reset Switch
RGB User LED
Tiva C Series LaunchPad BoosterPack XL Interface (J1, J2, J3, and J4 Connectors)
Tiva C Series LaunchPad BoosterPack XL Interface (J1, J2, J3, and J4 Connectors)
Tiva TM4C123GH6PMI Microcontroller
MSP430 LaunchPad-Compatible BoosterPack Interface
MSP430 LaunchPad-Compatible BoosterPack Interface
User Switch 1
User Switch 2

The TIVA TM4C123G was chosen primarily because of these mentioned features:

- An ARM Cortex M-4 64-pin 80 MHz processor
- On- board USB ICDI
- 16 MHz main oscillator crystal
- 32 KHz real time clock crystal
- 3.3 V regulator
- ADC operation without power intervention

B. Analysis

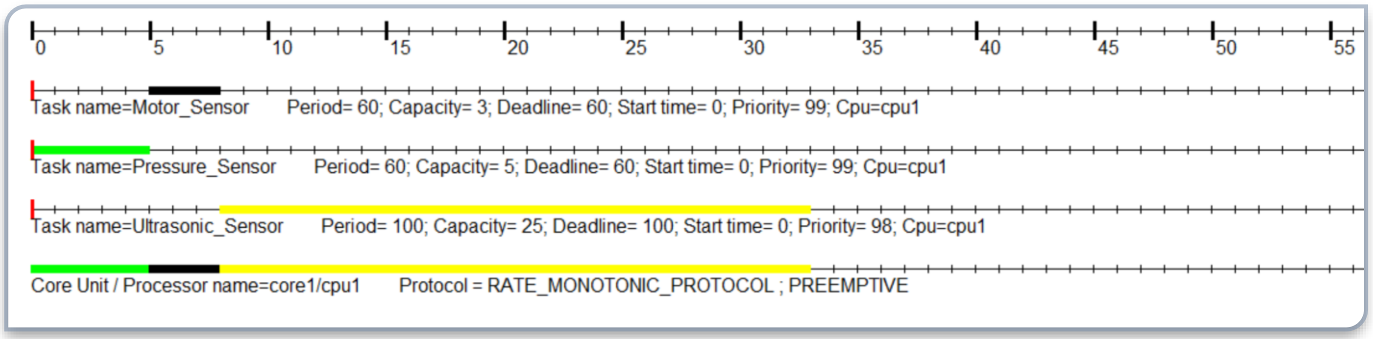
- *High Risk Area of Synchronization*

  The highest risk area of this project was the synchronization between the three tasks and making sure one doesn't overuse resources and disallow other services from having their own share of CPU time. Keeping this in mind, we carefully arrived at the current values of deadline, request period and computation time. Each is such that sufficient margin is provided for the service to run its course and allow the schedule to remain feasible. On a similar note, we have also decided to use the Rate Monotonic Scheduling policy, as a backup of sorts. In case the schedule begins to fail due to overuse of resources by one particular service, Rate Monotonic allow the service set to fail gracefully. This was observed during testing as well. If the Deadline for a particular task, say the pressure sensing task is reduced to half of its current value, it is obvious the schedule will fail, but the manner in which this failure occurs is worth noting. The task with least priority, the ultrasonic sensing task, begins to fail, but the rest of the services execute within their time periods and in keeping with proper operation. This is in keeping with the principle of Rate Monotonic policy which neglects the task with lowest priority in favor of the higher priority tasks. This is another reason we decided to go with the fixed priority policy instead of dynamic policies like Earliest Deadline First or Least Laxity First. In case of failure with either of the latter policies, analysis and debugging would have been that much more difficult.
- *Cheddar analysis and margins for safety*

  The hand-drawn Excel sheet was not quite feasible, as the LCM of periods in our case comes to be 300 ms.

Task No.	Task Name (T)	Request Period (T_i)	Execution Time WCET (C_i)	Deadline (D_i)
1	Pressure sensor T	60 ms	5 ms	60 ms
2	Motor driver T	60 ms	3 ms	60 ms
3	Ultrasonic sensor T	100 ms	25 ms	100 ms

Task Period, Execution Time and Deadline



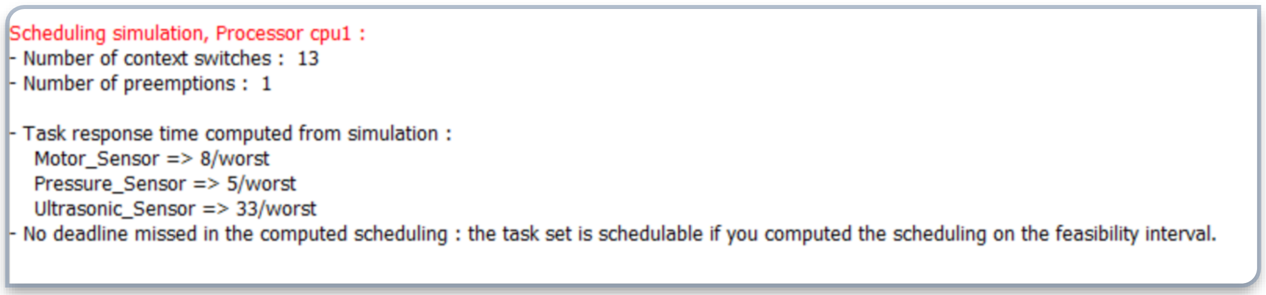
Task name=Motor_Sensor Period= 60; Capacity= 3; Deadline= 60; Start time= 0; Priority= 99; Cpu=cpu1

Task name=Pressure_Sensor Period= 60; Capacity= 5; Deadline= 60; Start time= 0; Priority= 99; Cpu=cpu1

Task name=Ultrasonic_Sensor Period= 100; Capacity= 25; Deadline= 100; Start time= 0; Priority= 98; Cpu=cpu1

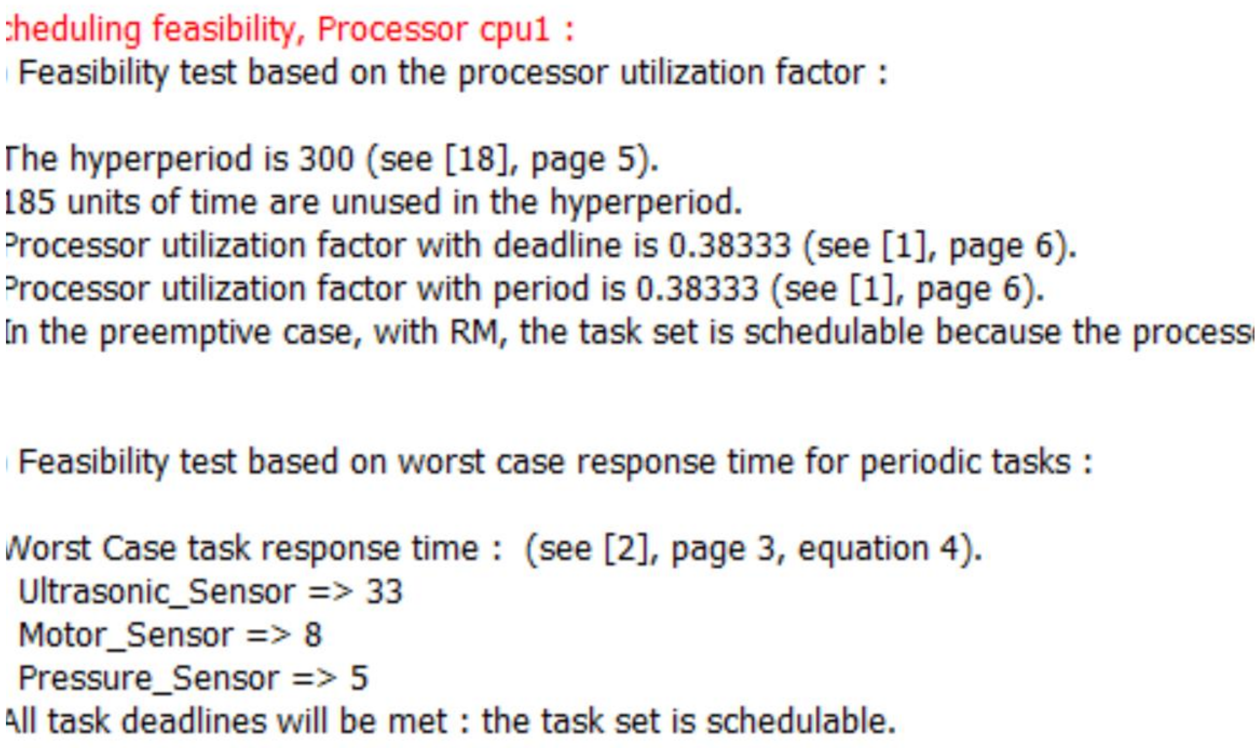
Core Unit / Processor name=core1/cpu1 Protocol = RATE_MONOTONIC_PROTOCOL ; PREEMPTIVE

Cheddar plot of tasks



```
Scheduling simulation, Processor cpu1 :
- Number of context switches :  13
- Number of preemptions :  1

- Task response time computed from simulation :
   Motor_Sensor => 8/worst
   Pressure_Sensor => 5/worst
   Ultrasonic_Sensor => 33/worst
- No deadline missed in the computed scheduling : the task set is schedulable if you computed the scheduling on the feasibility interval.
```



```
:heduling feasibility, Processor cpu1 :
 Feasibility test based on the processor utilization factor :

The hyperperiod is 300 (see [18], page 5).
185 units of time are unused in the hyperperiod.
Processor utilization factor with deadline is 0.38333 (see [1], page 6).
Processor utilization factor with period is 0.38333 (see [1], page 6).
In the preemptive case, with RM, the task set is schedulable because the process


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

Worst Case task response time :  (see [2], page 3, equation 4).
 Ultrasonic_Sensor => 33
 Motor_Sensor => 8
 Pressure_Sensor => 5
All task deadlines will be met : the task set is schedulable.
```

Details of simulation using RM policy, along with % CPU utilization

We ran Cheddar simulations on our proposed and examined values of Deadline, computation time and request periods. We found the processor utilization to be around 38.33%, which leaves us a lot of margin in case of failure of one the services. The calculated CPU utilization is less than the RM LUB of 0.78.

Calculated CPU utilization = $\Sigma (C_i/T_i)$ = (3/60 + 5/60 + 25/100) = 0.05 + 0.0833 + 0.4 = 0.533

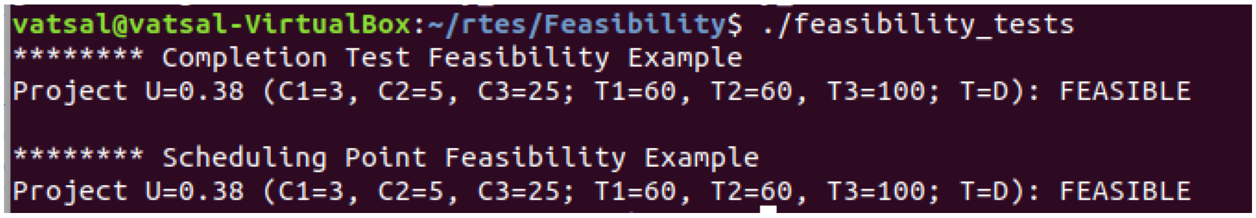
RM LUB = $m(2^{1/m} - 1) = 3(1.26 - 1) = 0.78$.

Calculated CPU utilization < RM LUB.

Rate Monotonic proclaims this set of services feasible.

C. Time Stamps and Tests

Time-stamp tracing is implemented during development phase. Here each task's execution time was measured as time difference between task getting its semaphore and its execution completion. Count of this execution time is maintained over an iteration of 50. And the max value is displayed at last which is taken as the WCET value. In final code all time-stamp measurements and UART prints are removed so as not to affect execution time during run-time.



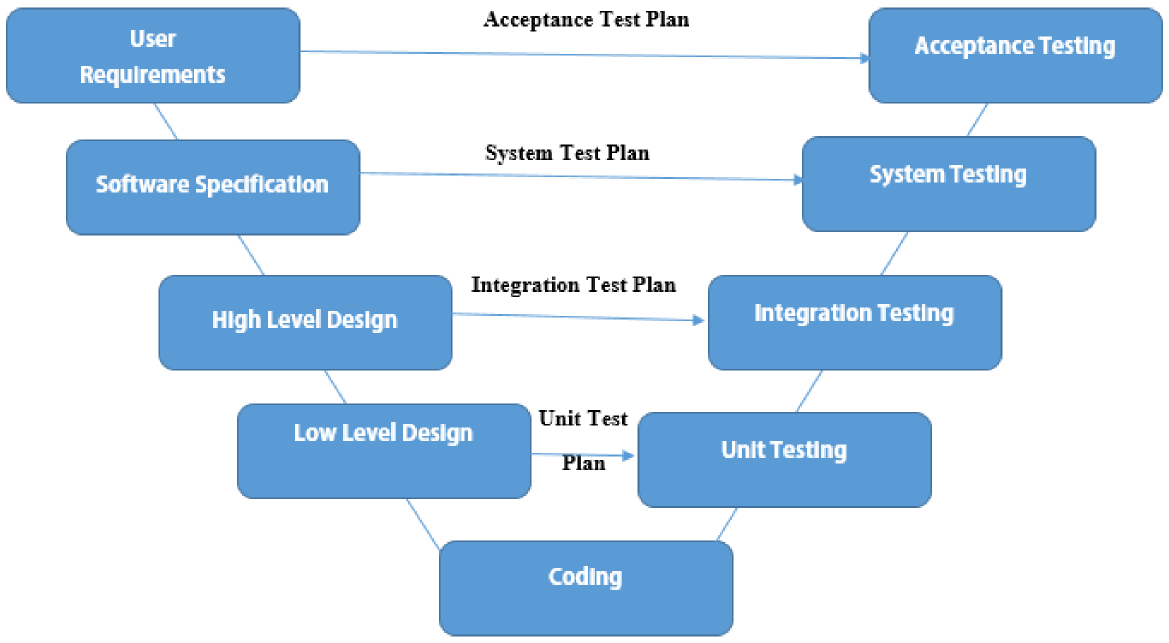
```
vatsal@vatsal-VirtualBox:~/rtes/Feasibility$ ./feasibility_tests
******** Completion Test Feasibility Example
Project U=0.38 (C1=3, C2=5, C3=25; T1=60, T2=60, T3=100; T=D): FEASIBLE

******** Scheduling Point Feasibility Example
Project U=0.38 (C1=3, C2=5, C3=25; T1=60, T2=60, T3=100; T=D): FEASIBLE
```

Scheduling point and completion test is run on feasibility test code from Exercise 2 by Prof Sam Siewert with values from our D C T Table. Our scheduler values passed both Scheduling and Completion feasibility tests. Our scheduler passed both necessary and sufficient tests.

VII. Validation Plan

- Force sensitive service tested separately using ROM APIs, and ADC values were checked with maximum and minimum range of sensors.
- Ultrasonic sensor range of detection was tested to be upto 3m.
- Motors were tested with PWM separately.
- Integrated parts were tested, with slight variations to proposed values of WCET, request period and deadline.



V-Model

In keeping with the Validation model, we have followed above, the user requirements were recognized first. In this case, the requirements were to design a set of services to handle the ultrasonic sensor which would detect obstacles, a motor driver service to drive the motors and a pressure sensor task to sample ADC data and give a reading of whether or not the pressure sensors have been depressed.

The software architecture was to be decided, wherein we chose to go with the Rate Monotonic policy and a timer handler which would act a service scheduler. We chose the FreeRTOS environment, as it is quite convenient to create tasks, schedule them and notify each other using the inbuilt APIs.

The High- and low-level designs came next, wherein we decided the hardware components to be used, namely the TIVA TM4X123G, the ultrasonic sensor HC-SR04, the motor driver Dual TB6612FNG and the actual motors themselves.

Code for the above components was written, and unit testing was performed. For the purpose of testing, separate scripts were written to check each and every valid case of the sensors to verify their complete functionality.

The system was completely integrated for the purpose of testing. A few issues we faced during integration and system testing were the scheduling of all three services together. At times, the ultrasonic sensor would not respond at all, while at others, the motors would take control of all system resources and not allow any of the other services to regain control. Fine-tuning of the services and their request periods solved this problem.

VIII. Conclusion

This project has helped us learn a lot of new concepts, some of which are how to design a hard-real-time scheduler, interface of cameras and the use of FreeRTOS. The bot was eventually able to control the speed and usage of motors based on two other services, detect upto 4m in front of it, and accordingly make decisions. All of this, within the scope of a real-time scheduler so as to prevent overrun of any service. It was an extremely enriching experience, and one which taught us how to schedule our own projects.

We would like to thank Prof. Tim Scherr for his guidance and for giving us the opportunity to design a hard-real-time scheduler by ourselves. We would also like to thank the TAs for their constant support and patient handling of our doubts.

IX. References

1. http://www.cse.unt.edu/~rakl/KAH08.pdf
2. https://www.sciencedirect.com/science/article/pii/B9780123749574000153
3. http://www.electronique-mixte.fr/wp-content/uploads/2015/03/LES-SYSTEMES-TEMPS-REELS-Conception-des-applications-embarqu%C3%A9s-distribu%C3%A9s.pdf
4. http://www.ti.com/lit/ug/spmu296/spmu296.pdf
5. http://robotsforroboticists.com/systems-engineering/
6. Ultrasonic Ranging Module HC - SR04 Datasheet: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf
7. FSR 400 Data Sheet: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/2010-10-26-DataSheet-FSR400-Layout2.pdf
8. Thosiba TB6612FNG Datasheet: https://www.sparkfun.com/datasheets/Robotics/TB6612FNG.pdf