PROJECT 2 - REPORT

By Shreya Chakraborty

1. **Team Members :** Solo Project

2. **Product Description :**

   My Project implements vehicular subsystems like air conditioning, collision detection and parking. The air conditioning subsystem will use temperature sensor, TMP102 for checking the temperature and a fan as an output device. The collision detection will use a SPI based accelerometer sensor to detect collision and displays it on the LCD. The third part is to use a distance sensor to check the distance between the car and nearest obstacle for parking and the buzzer starts beeping with increasing frequency as the obstacle comes closer. The remote node (TIVA board) has all the sensor and output devices connected to it. Whereas the control node (BBG) is used to send commands for specific tasks as a feedback based on sensor values.

3. **Hardware Requirements:**
   1. TMP102 I2C temperature sensor (https://learn.sparkfun.com/tutorials/tmp102-digital-temperature-sensor-hookup-guide/all)
   2. ADXL345 Accelerometer (https://www.sparkfun.com/products/9836).
   3. Distance Sensor (https://www.sparkfun.com/products/13959)
   4. LCD 16x2 hd44780
   5. Buzzer
   6. Motor for fan
   7. Relay
   8. Leds
   9. TIVA EK-TM4C1294XL board
   10. BeagleBone green

4. **Software Requirements:**

   The main 3 software specific implementation of the project are as follows:
   1. Airconditioning Module: The Temp task detects the temperature from the Sensor and sends the data over to the control node periodically, every 2 seconds, the control node compares the data with the thresholds set, if the temperature is greater than the threshold, it sends some form of indication back to the control node which triggers the servo motor with the fan blades attached to it to rotate and it remains this way until the temperature goes below the threshold. This is the soft real time requirement and hence the frequency of data fetch is low.
   2. Collision Detection module: The Accel task will continuously check for collisions based on its x, y, z data. If detected a collision, it will send the collision occurred indication to the Control node. The control node sends a signal back to the remote

node to and the string “***Accident***” gets printed on LCD. This is a hard-real time requirement, hence frequency is 500ms.

3. Safe Parking module: The Distance task will only start checking the distance if it receives the “set reverse gear signal” from the control node. On reception of the signal, it assumes that the car is in reverse gear and hence depending on several distance ranges , it increases the frequency of the red led blink and the buzzer attached to it. On reception on “front gear signal”, the remote node stops collecting distance. This is a real time requirement of the project. Hence it has a frequency of 500ms.
4. In the normal working mode, the logger shows the status as normal in the remote node and control node logger, on failure of 1 or 2 sensors or on failure of the connection between the remote and the control node, the status is shown as degraded. If all 3 sensors fail, then the status is out of service and the program is exited.

5. **Task Description:**

The remote node and the control node are connected using UART/ethernet. All the remote node does is collect the data from the sensor and sends it over to the control node. What to do with the data is up to the control node. The control node offers degraded service in case one or two of the sensors are disconnected with the status led output on the remote node side. The following are the major parts of the program:

1. The SEND and RECEIVE task: There are two tasks for communication between the remote and the control node. The remote node sends all the sensor values periodically to the control node using the SEND task and it receives the commands from the control node using RECEIVE task and same for the control node.
2. SENSOR task: All the sensors work concurrently in the same task called the sensor task but at different frequencies, this is kept this way as the sensor task in general needs to have the highest priority, rather than setting priorities for each task. This task is only for the remote node.
3. LOGGER task: The logger task logs the output messages from all the tasks to the console and file. For both remote node and control node.
4. External Client Handler task: On the control node side, the user can change the threshold values for the sensor tasks. This uses sockets for IPC.
5. Send and receive queue. There are two separate queues corresponding to two tasks for storing data.
6. In case of communication loss, the sensor task itself which checks the connected flag, will use the default given threshold values and work independently

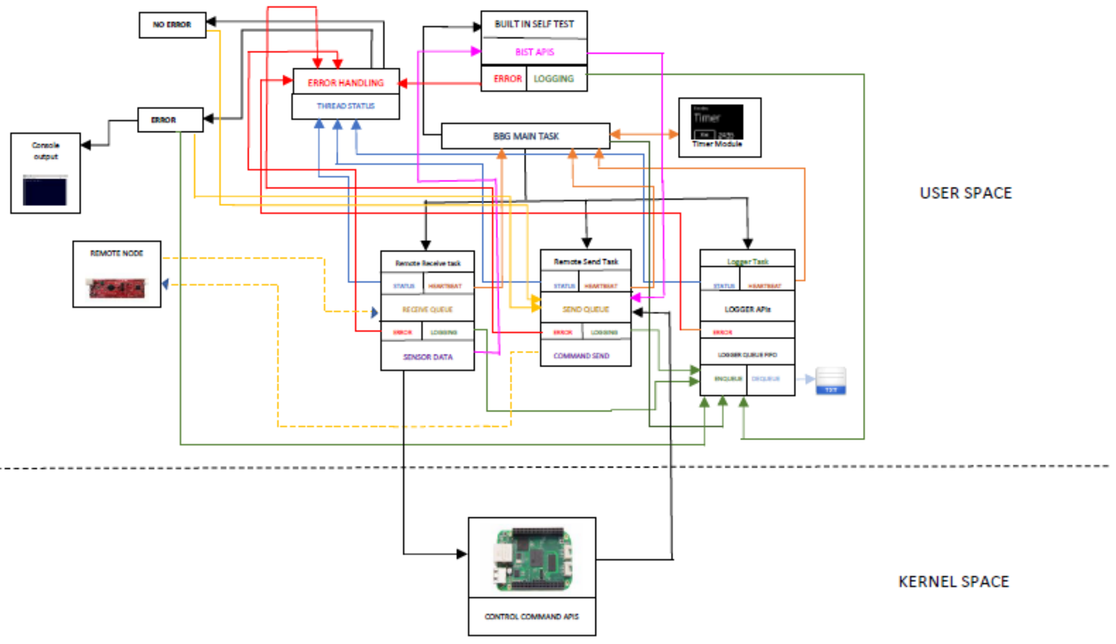
6. **Architecture Description**:

The architecture diagrams for the control node and remote node side is shown as below. The IPC for inter task communication is message queues. Both the nodes convey their messages to a logger task. The method of communication between the two boards is via ethernet. The libraries used on the control node side include the libraries from the Project 1 as well as additional ethernet libraries. The Send and Receive tasks for both

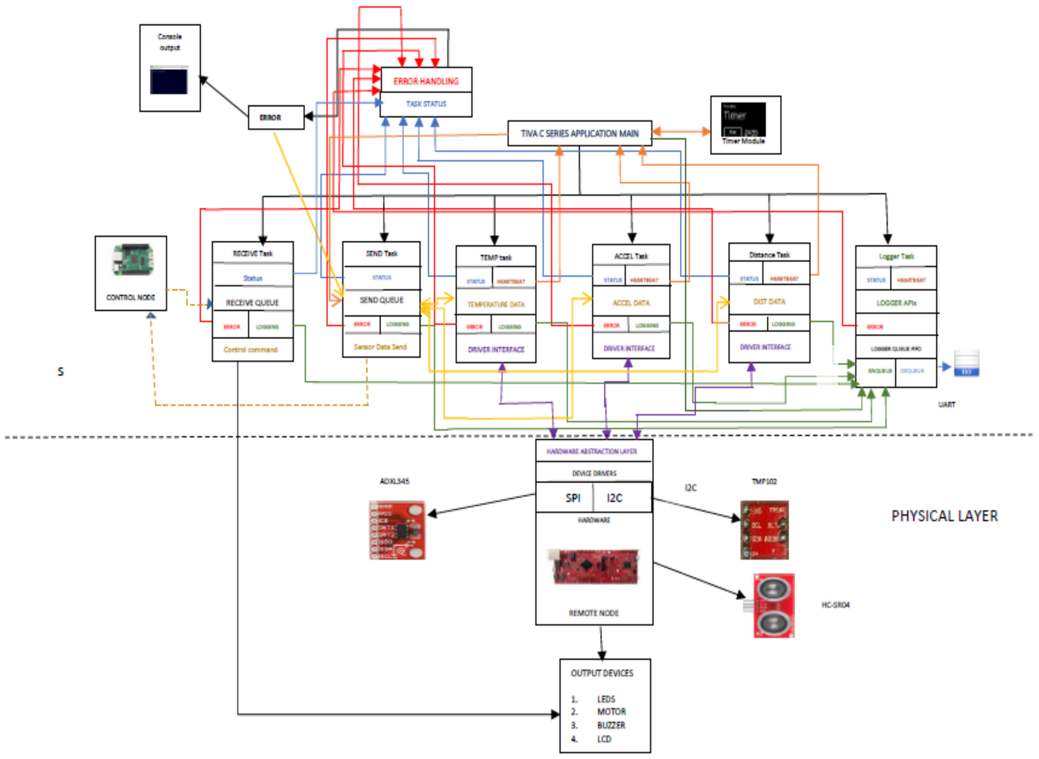
the nodes have their own send and receive queues. The basic APIs for node communication are :
SendDataToControlNode() and ReceiveCmdFromControlNode() on the remote node side and similarly for the control node the APIs are SendCmdToRemoteNode() and ReceiveDataFromRemoteNode(). All the functions mentioned use client-server socket APIs internally.
The remote node will have driver libraries for each sensor. I2C library, Uart library, Gpio library and SPI libraries. In addition it would also have several device driver libraries for specific sensors. The major APIs on the remote side include get_Temp(), get_AccelVal(), get_Distance() and on control the major APIs include Temp_ControlCmd(), Collision_DetectCmd(),Distance_ControlCmd() and so on.

The remote node application spawns 6 threads as given in the diagram. All the thread send their information to the logger task using message queue logger_queue. The data from the sensor tasks are accumulated in the send queue and send over to the control node periodically over ethernet. The commands send by the Control node are accumulated in the receive queue and acted upon by the output devices. The control node only has 3 tasks. 2 for sending and receiving and 1 for logger. They work in similar fashion. The software diagrams are as follows:

The Control Node:



The Remote Node:

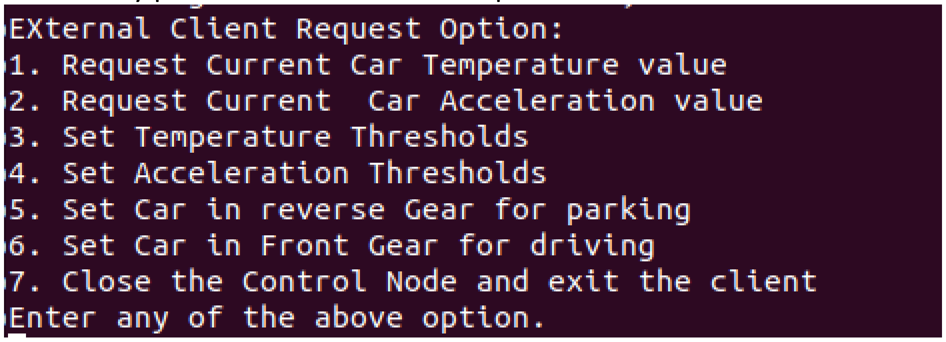


7. Updates From Original Plan

The original plan was to use ethernet to communicate between the two nodes. In fact the program in itself has been implemented in layers. The communication layer uses both UART and Ethernet depending on ifdefs. The ethernet send from the Tiva and receive on the BBG is functional. However, the other part of communication is not functional. There were many peculiar behaviors and errors that have occurred throughout my attempt and I have no exact explanation for it. One time it seems to work and some other time it doesn't. After working on it for 4 days I came to realize that ethernet is not only very tricky to implement but also very unreliable. Nonetheless if I have more time, I will come back to it. As of now, I learned a lot about how it works. The UART is functional and works both ways.

8. Test Plan

My idea for this project was to implement a real time system for a vehicle. Apart from the low level tests which include unit testing, my real application level testing and validation is through the external client. The external client has options to request temperature, distance and acceleration values. It can be used to change gears, put threshold values for the sensors to trigger. My external client acts as a remote control with which you can control all the above mentioned aspects. The external client is situated on the control node side. It is an additional task which sets the thresholds for send tasks to process on the gets the values from the receive tasks. Hence it is sufficiently protected from data corruption.

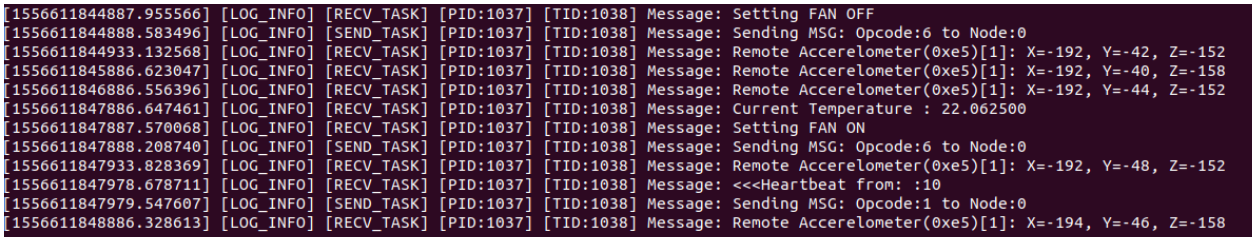


```
EXternal Client Request Option:
1. Request Current Car Temperature value
2. Request Current  Car Acceleration value
3. Set Temperature Thresholds
4. Set Acceleration Thresholds
5. Set Car in reverse Gear for parking
6. Set Car in Front Gear for driving
7. Close the Control Node and exit the client
Enter any of the above option.
```

The steps for test plan are as follows:

1. Option one fetches the latest temperature value from the recv queue. Of BBG.
2. Option 2 fetches the latest acceleration value from the recv queue of BBG
3. In option 3 we set the threshold to compare the temperature value with. On exceeding the threshold the Control node sends the opcode corresponding to “SETTING FAN ON/OF” to control the motor remotely.
4. Option 4 sets the delta value for acceleration. It sends these delta values to the TIVA and tiva itself calculates if the diff between the previous and the current acceleration value is more than the delta then displays the accident .
5. Option 5 and 6 are used to set the car in reverse and front gear. If set in reverse gear, the ultrasonic sensor is activated and it starts beeping depending of the proximity of the obstacle. On setting it to front gear the sensor is deactivated.
6. Last option is used to disconnect the Control node from the remote node. On which the tiva will try a few times to send heartbeat and on no reception of heartbeat ack, it presumes the remote node is dead and starts the independent working.

9. Proof of Execution on Control Node Side:



```
[1556611844887.955566] [LOG_INFO] [RECV_TASK] [PID:1037] [TID:1038] Message: Setting FAN OFF
[1556611844888.583496] [LOG_INFO] [SEND_TASK] [PID:1037] [TID:1038] Message: Sending MSG: Opcode:6 to Node:0
[1556611844933.132568] [LOG_INFO] [RECV_TASK] [PID:1037] [TID:1038] Message: Remote Accerelometer(0xe5)[1]: X=-192, Y=-42, Z=-152
[1556611845886.623047] [LOG_INFO] [RECV_TASK] [PID:1037] [TID:1038] Message: Remote Accerelometer(0xe5)[1]: X=-192, Y=-40, Z=-158
[1556611846886.556396] [LOG_INFO] [RECV_TASK] [PID:1037] [TID:1038] Message: Remote Accerelometer(0xe5)[1]: X=-192, Y=-44, Z=-152
[1556611847886.647461] [LOG_INFO] [RECV_TASK] [PID:1037] [TID:1038] Message: Current Temperature : 22.062500
[1556611847887.570068] [LOG_INFO] [RECV_TASK] [PID:1037] [TID:1038] Message: Setting FAN ON
[1556611847888.208740] [LOG_INFO] [SEND_TASK] [PID:1037] [TID:1038] Message: Sending MSG: Opcode:6 to Node:0
[1556611847933.828369] [LOG_INFO] [RECV_TASK] [PID:1037] [TID:1038] Message: Remote Accerelometer(0xe5)[1]: X=-192, Y=-48, Z=-152
[1556611847978.678711] [LOG_INFO] [RECV_TASK] [PID:1037] [TID:1038] Message: <<<Heartbeat from: :10
[1556611847979.547607] [LOG_INFO] [SEND_TASK] [PID:1037] [TID:1038] Message: Sending MSG: Opcode:1 to Node:0
[1556611848886.328613] [LOG_INFO] [RECV_TASK] [PID:1037] [TID:1038] Message: Remote Accerelometer(0xe5)[1]: X=-194, Y=-46, Z=-158
```

10. Test Results

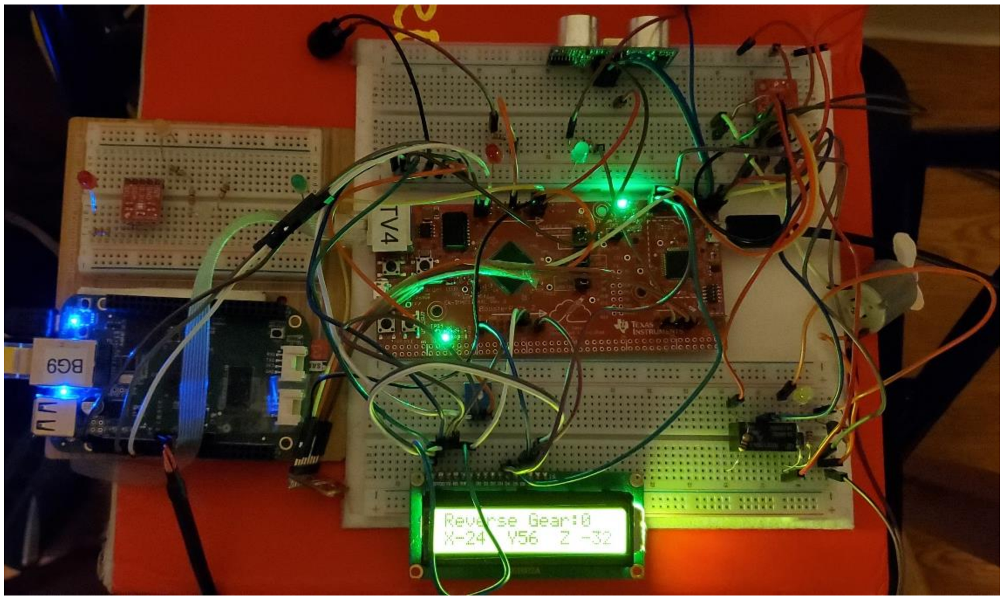
The following Tests were conducted.

1. Testing of connectivity, disconnection and thresholds Using External Client Application.
2. Testing of logger given in the folder test in Linux_BBB of Project 2.
3. Testing of uart loopback.
4. Testing of Network Connectivity for UART. From Tiva to TeraTerm Terminal.

The following are the rest of the tests in a tabular format.

SR NO	TEST	RESULT
1.	Connectivity test	PASS with delay, takes a while to connect, Buzzer on disconnection
2	CRC check test	PASS with exceptions, Initial packets are dropped due to crc fail
3	Distance Sensor Connectivity test	PASS
4.	Accelerometer Sensor Connectivity	Pass with exception, on disconnection resets the program, hardware issue
5.	Dc motor Test	Pass
6.	Temp Sensor Test	Pass with exceptions, Sensor may be damaged
7.	LCD test	Pass with exceptions, seems to behave strange when motor is running due to back emf
8.	Uart test	Pass, tried it with terminal as well as BBG
9.	Socket test	Pass, using external client
10.	Logger Test	Pass, using CMOCKA
11.	SPI test	Pass, to check the working of spi driver
12	I2C test	Pass, working i2c driver
13.	PWM test	Pass, PWM driver for ultrasonic
14	TCP test	Pass, with exception
15	Ethernet test	Pass, with exception
16	Relay test	Pass, tested with led to see if the switch works

11. Project Setup



12. Key Learnings

The key learning of the project2 from my experience are:

1. Interfacing multiple sensors and giving them different priorities might sometimes result in the sensor which the lowest priority to miss out. Initially I had different tasks for each sensor and my entire program had around 7 to 8 tasks, the lowest priority of them was the temperature and that task wasn't allowed to run. Hence I combined all my sensors into one task with the highest priority.
2. Interfaced DC motor in a project for the first time with a switch and learned its inner mechanism.
3. Wrote driver for LDC with 16 pins and almost ran out of pins, hence revised my lcd driver to work with 4 data pins instead of 8.
4. Socket connection between freeRTOS and Linux is tricky and time consuming and has led to erratic behaviors in my case. I learned how the connection happens and what layers are involved in it. Wrote a TCP layer driver and ethernet drivers for the same. Learned about the buffer allocation schemes. The modified the network management layer and learned in great details about the Berkeley sockets APIs.
5. In case of UART task, the uart send from Remote to Control node works fine, but in my case, the other way round was problematic. The Remote recv task was getting preempted by the other higher priority tasks like sensor,

communication send, heartbeat and logger. Hence to resolved this I had to use interrupt based uart.

6. The ADXL345 sensor needs a small delay after each write to its register in case of SPI to function properly. This information was not mentioned anywhere and I had to come up with it myself by trial and error. I am not exactly sure why this happens but I might investigate about it later.

13. Reason For Late Submission

The most difficult part of my project was setting up the communication. I was trying ethernet for a long time and felt like I was very close to getting it work. It would work one time and wouldn't work the next instant. Hence as a back Up option I had to start UART very late. The reason why I did not think of doing UART which is the more common alternative among the two options is because the main aim of this project for me was learning something new and challenging . Since it is a solo project for me, it took more time to figure out from my mistakes and contributed to my late submission.

14. Extra Credits Attempted

1. Control Node Operational status - buzzer
2. SPI based Sensor -> AXDL345 sensor, wrote a spi driver for it
3. Extra Sensor-> Ultrasonic Sensor, wrote a pwm driver for this
4. Non LED output device-> buzzer
5. Real Output device – motor with relay and an led for the switch status.
6. Real Output device --LCD for displaying events, wrote an LCD driver for it.
7. Control Node collection remote node logs after restoration of connection
8. Application Complexity and Design: Ethernet layer and TCP layer functional, Entire project is implemented in layers that are independent of each other.

15. Project Plan

No.	Developments	Date
1.	Sensor Integration	04/13/2019
2.	Communication between Nodes	04/15/2019
3.	Accel task functionality complete	04/17/2019
4.	Temp task functionality complete	04/19/2019
5.	Distance task functionality complete	04/20/2019
6.	Error handling	04/21/2019
7.	LCD drivers and integration	04/23/2019
8.	Led , buzzer functionality complete	04/24/2019
9.	Logging, send receive functionality complete	04/26/2019

10.	Standby task, disconnection handling	04/27/2019
11.	Report and documentation	04/28/2019