```
/**
 * @\file   uart.h
 * @\author Sanju Prakash Kannioth
 * @\brief  This files contains the declarations and header files for
uart transmit and receive on BBG
 * @\date   04/29/2019
 * References : https://github.com/sijpesteijn/BBCLib,
 *
      https://en.wikibooks.org/wiki/Serial_Programming/termios
 *
 */

#ifndef UART_H
#define UART_H

#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <termios.h>
#include <stdlib.h>
#include <stdint.h>

#include "logger.h"
#include "heartbeat.h"

/* Structure to store sensor data and mode of operation to send to remote
request task */
typedef struct {
      float lux;
      float distance;
      float waterLevel;
      int8_t mode;
      int8_t dg_mode;
} communication;

communication comm_rec;

typedef enum {
      uart00 = 0, uart01 = 1, uart02 = 2, uart03 = 3, uart04 = 4, uart05
= 5
} uart;


/* Structure to initialize particular UART on BBG */
typedef struct {
      int fd;
      uart uart_no;
      int baudrate;
}uart_properties;

uart_properties  *uart2;
```

```
/**
--------------------------------------------------------------------------
-------------------
uart_config
--------------------------------------------------------------------------
-------------------
*   This function will configure the specific UART on BBG
*
*   @\param         uart_properties     specifies UART number
*
*   @\return        0                         success
*                            -1                           failure
*
*/
int8_t uart_config(uart_properties *uart);


/**
--------------------------------------------------------------------------
-------------------
uart_close
--------------------------------------------------------------------------
-------------------
*   This function will close the specific UART on BBG
*
*   @\param         uart_properties     specifies UART number
*
*   @\return        0                         success
*
*/
int8_t uart_close(uart_properties *uart);


/**
--------------------------------------------------------------------------
-------------------
uart_send
--------------------------------------------------------------------------
-------------------
*   This function will send specified length of bytes over the UART on
BBG
*
*   @\param         uart_properties     specifies UART number
*                            tx                           byte to be sent
*                            length                       number of bytes
to be sent
*
*   @\return        -1                        Failure
*                             count                       Number of bytes
sent
*
*/
int8_t uart_send(uart_properties *uart, void *tx, int length);
```

```
/**
----------------------------------------------------------------------------
-------------------
uart_receive
----------------------------------------------------------------------------
-------------------
*   This function will receive tiva sensor data over the UART on BBG
*
*   @\param         uart_properties     specifies UART number
*                            rx_r                    byte received
*                            length                        number of bytes
to be received
*
*   @\return        -1                       Failure
*                             count                        Number of bytes
sent
*
*/
int8_t uart_receive(uart_properties *uart, void *rx_r, int length);


/**
----------------------------------------------------------------------------
-------------------
uart_receive_task
----------------------------------------------------------------------------
-------------------
*   This function will receive tiva log data over the UART on BBG
*
*   @\param         uart_properties     specifies UART number
*                            rx_r                    byte received
*                            length                        number of bytes
to be received
*
*   @\return        -1                       Failure
*                             count                        Number of bytes
sent
*
*/
int8_t uart_receive_task(uart_properties *uart, void *rx_r, int length);
#endif
/**
 * @\file   logger.h
 * @\author Sanju Prakash Kannioth
 * @\brief  This files contains the declarations and header files for the
logger
 * @\date   04/29/2019
 *
 */

#ifndef LOGGER_H
#define LOGGER_H
```

```
#include "uart.h"
#include <pthread.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h> // mkdir
#include <mqueue.h>

#define QUEUE_NAME "/msg_queue" // Message queue name
#define MAX_BUFFER_SIZE      200     // Message queue max buffer size


/* Sensor structure that is sent from Tiva */
typedef struct sensor_struct
{
    char task_name[5];
    uint32_t timeStamp;

    float distance;
    float lux;
    uint32_t water;
    int8_t mode;
    int8_t dg_mode;
}sensor_struct;

/* Logger structure that is sent from Tiva */
typedef struct logger_struct
{
      char task_name[5];
      uint32_t timeStamp;

      char log[100];
}logger_struct;

pthread_t logger_thread;

FILE *file_ptr;

mqd_t msg_queue; // Message queue descriptor

extern pthread_mutex_t lock_res; // Mutex


/**
-----------------------------------------------------------------------------
-------------------
time_stamp
-----------------------------------------------------------------------------
-------------------
*   This function will format the timestamp
*
*   @\param
*
*   @\return        timestamp as a string
*
```

```
*/
char *time_stamp();


/**
--------------------------------------------------------------------------
-------------------
logger_thread_callback
--------------------------------------------------------------------------
-------------------
*   This function is the thread callback function for the logger
*
*   @\param         void
*
*   @\return        void
*
*/
void *logger_thread_callback();


/**
--------------------------------------------------------------------------
-------------------
logger_init
--------------------------------------------------------------------------
-------------------
*   This function will initialize the logger
*
*   @\param         void
*
*   @\return        void
*
*/
void logger_init();
#endif


/**
 * @\file   communication.h
 * @\author Sanju Prakash Kannioth
 * @\brief  This files contains the declarations and header files for the
communication interface for tiva and BBG
 * @\date   04/29/2019
 *
 */
#ifndef COMMUNICATION_H
#define COMMUNICATION_H

#include <pthread.h>
#include <stdint.h>

#include "uart.h"
#include "logger.h"
#include "POSIX_timer.h"
```

```
#include "heartbeat.h"


/*
*     GLOBALS
*/
char lux[10];
char distance[10];
char waterLevel[10];
char mode[10];
char dg_mode[10];
char tiva_opstatus[10];
char distance_opstatus[10];
char lux_opstatus[10];
char water_opstatus[10];
char valve_status[10];

pthread_t communication_thread;

uint8_t already_open;
uint8_t already_closed;
uint8_t water_outOfRange;

extern pthread_mutex_t lock_res;


/*
-------------------------------------------------------------------------
-------------------
communication_thread_callback
-------------------------------------------------------------------------
-------------------
*     This is the thread creation callback function for the communication
thread
*
*     @\param                void
*
*     @\return         void
*
*/
void *communication_thread_callback();


/*
-------------------------------------------------------------------------
-------------------
get_lux
-------------------------------------------------------------------------
-------------------
*     This function returns the most updated lux value in string format
*
*     @\param                void
*
*     @\return         string containing most recent lux value
```

```
*
*/
char *get_lux();


/*
-------------------------------------------------------------------------
-------------------
get_distance
-------------------------------------------------------------------------
-------------------
*     This function returns the most updated distance value in string
format
*
*     @\param                void
*
*     @\return         string containing most recent distance value
*
*/
char *get_distance();


/*
-------------------------------------------------------------------------
-------------------
get_waterLevel
-------------------------------------------------------------------------
-------------------
*     This function returns the most updated water level value in string
format
*
*     @\param                void
*
*     @\return         string containing most recent water level value
*
*/
char *get_waterLevel();


/*
-------------------------------------------------------------------------
-------------------
get_mode
-------------------------------------------------------------------------
-------------------
*     This function returns the most updated operating mode in string
format
*
*     @\param                void
*
*     @\return         string containing most updated operating mode
value
*
*/
```

```
char *get_mode();


/*
--------------------------------------------------------------------------
-------------------
get_dgMode
--------------------------------------------------------------------------
-------------------
*     This function returns if the system is in degraded mode in string
format
*
*     @\param                void
*
*     @\return         string containing degraded mode value
*
*/
char *get_dgMode();


/*
--------------------------------------------------------------------------
-------------------
get_opStatus_tiva
--------------------------------------------------------------------------
-------------------
*     This function returns the most updated operation status of the
remote node
*
*     @\param                void
*
*     @\return         string containing the most updated operation
status of the remote node
*
*/
char *get_opStatus_tiva();


/*
--------------------------------------------------------------------------
-------------------
get_opStatus_distance
--------------------------------------------------------------------------
-------------------
*     This function returns the most updated operation status of the
distance sensor
*
*     @\param                void
*
*     @\return         string containing the most updated operation
status of the distance sensor
*
*/
char *get_opStatus_distance();
```

```
/*
-------------------------------------------------------------------------
-------------------
get_opStatus_lux
-------------------------------------------------------------------------
-------------------
*     This function returns the most updated operation status of the lux
sensor
*
*     @\param                void
*
*     @\return         string containing the most updated operation
status of the lux sensor
*
*/
char *get_opStatus_lux();


/*
-------------------------------------------------------------------------
-------------------
get_opStatus_water
-------------------------------------------------------------------------
-------------------
*     This function returns the most updated operation status of the
water level sensor
*
*     @\param                void
*
*     @\return         string containing the most updated operation
status of the water level sensor
*
*/
char *get_opStatus_water();


/*
-------------------------------------------------------------------------
-------------------
get_valveStatus
-------------------------------------------------------------------------
-------------------
*     This function returns the most updated status of the valve
*
*     @\param                void
*
*     @\return         string containing the most updated status of the
valve
*
*/
char *get_valveStatus();
#endif
```

```
/**
 * @\file   server.h
 * @\author Sanju Prakash Kannioth and Steve Antony X
 * @\brief  This files contains the declarations and header files for
server socket
 * @\date   04/29/2019
 *
 */

#ifndef SERVER_H
#define SERVER_H

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdbool.h>

#include "communication.h"

#define PORT_NO 8005

extern pthread_mutex_t lock_res;

pthread_t remote_request_thread;
/**********************************************
       Function for server socket creation
       Parameters : Port number
***********************************************/
int socket_creation_server(int port);


/**************************************************
       Function for remote request thread creation
       Parameters :
***************************************************/
void *remote_request_callback();

#endif
/**
 * @\file   heartbeat.h
 * @\author Sanju Prakash Kannioth
 * @\brief  This files contains the declarations and header files for the
heartbeat
 * @\date   04/29/2019
 *
 */
```

```c
#ifndef HEARTBEAT_H
#define HEARTBEAT_H

#include <pthread.h>
#include "POSIX_timer.h"

#include "communication.h"
#include "uart.h"
#include "logger.h"

#define TIVA_HEART_BEAT_CHECK_PERIOD (5) //5 seconds


/* Global variables for dead/alive status detection of Tiva and sensors
*/
int tiva_active, tiva_active_prev;
int distance_active, distance_active_prev, lux_active, lux_active_prev,
water_active, water_active_prev;

int tiva_dead, distance_dead, lux_dead, water_dead;


timer_t  timer_id_heartbeat;

pthread_t heartbeat_thread;

extern pthread_mutex_t lock_res; // Mutex


/*
----------------------------------------------------------------------
------------------
heartbeat_thread_callback
----------------------------------------------------------------------
------------------
*     This is the thread creation callback function for the heartbeat
thread
*
*     @\param                void
*
*     @\return         void
*
*/
void *heartbeat_thread_callback();


/*
----------------------------------------------------------------------
------------------
beat_timer_handler
----------------------------------------------------------------------
------------------
*     This function is the timer handler for tiva heart beat timer
*
```

```
*     @\param                signal value ( dummy)
*
*     @\return         none
*
*/
void beat_timer_handler(union sigval val);

#endif
/**
 * @\file   log_receiver.h
 * @\author Sanju Prakash Kannioth and Steve Antony X
 * @\brief  This files contains the declarations and header files for
tiva log receiver module
 * @\date   03/30/2019
 *
 */


#ifndef LOG_RECEIVER_H
#define LOG_RECEIVER_H

#include "uart.h"
#include "logger.h"

pthread_t log_receiver_thread;


/**
-----------------------------------------------------------------------------
-------------------
revecive_thread_callback
-----------------------------------------------------------------------------
-------------------
*   This function is the thread callback function for logger messages
coming from Tiva
*
*   @\param         void
*
*   @\return        void
*
*/
void *revecive_thread_callback();

#endif/**
 * @\file   POSIX_timer.h
 * @\author Sanju Prakash Kannioth and Steve Antony X
 * @\brief  This files contains the declarations and header files for
POSIX timer modules
 * @\date   04/29/2019
 *
 */

#ifndef POSIX_Timer_H_
#define POSIX_Timer_H_
```

```
/*****************************************************************
                                   Includes
*****************************************************************/
#include <stdio.h>
#include <pthread.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/stat.h>
#include <errno.h>

enum Status{SUCCESS = 0, ERROR = -1, LUX_ERROR = -2, REMOTE_SOCKET_ERROR
= -3, LOGGER_ERROR = -4,  TEMP_ERROR = -1000, BRIGHT = 1000, DARK = -
1000};

/*****************************************************************
                                   Function Protypes
*****************************************************************/
/*
--------------------------------------------------------------------------
-------------------
kick_timer
--------------------------------------------------------------------------
-------------------
*     This helps in restarting the timer after expiration
*
*     @\param                timer descriptor, timer expiration time in
ns
*
*     @\return         error status
*
*/
int kick_timer(timer_t, int);

/*
--------------------------------------------------------------------------
-------------------
setup_timer_POSIX
--------------------------------------------------------------------------
-------------------
*     This helps in creating the timer
*
*     @\param                timer descriptor, timer handler function
*
*     @\return         error status
*
*/
```

```
int setup_timer_POSIX(timer_t *,void (*handler)(union sigval));

/*
-------------------------------------------------------------------------
-------------------
stop_timer
-------------------------------------------------------------------------
-------------------
*     This helps in deleting the timer
*
*     @\param                timer descriptor
*
*     @\return         error status
*
*/
int stop_timer(timer_t);

/*
-------------------------------------------------------------------------
-------------------
temp_timer_handler
-------------------------------------------------------------------------
-------------------
*     This is the timer handler for temperature timer
*
*     @\param                dummy
*
*     @\return         error status
*
*/
void temp_timer_handler(union sigval);

/*
-------------------------------------------------------------------------
-------------------
lux_timer_handler
-------------------------------------------------------------------------
-------------------
*     This is the timer handler for lux timer
*
*     @\param                dummy
*
*     @\return         error status
*
*/
void lux_timer_handler(union sigval);

/*
-------------------------------------------------------------------------
-------------------
log_timer_handler
-------------------------------------------------------------------------
-------------------
*     This is the timer handler for log timer
```

```
*
*     @\param               dummy
*
*     @\return         error status
*
*/
void log_timer_handler(union sigval);

/****************************************************************
                                  MACROS
*****************************************************************/
#define Delay_NS (2000000000)//2000ms


#endif /* POSIX_Timer_H_ */
/**
 * @\file   POSIX_timer.h
 * @\author Sanju Prakash Kannioth and Steve Antony X
 * @\brief  This files contains the function definitions for POSIX timer
 * @\date   04/29/2019
 *
 */

/*****************************************************************
                                  Includes
*****************************************************************/
#include "POSIX_timer.h"



/*****************************************************************
                             POSIX Timer configuration
*****************************************************************/
int setup_timer_POSIX(timer_t *timer_id,void (*handler)(union sigval))
{
      struct     sigevent sev;
      sev.sigev_notify = SIGEV_THREAD; //Upon timer expiration, invoke 
sigev_notify_function
      sev.sigev_notify_function = handler; //this function will be called 
when timer expires
      sev.sigev_notify_attributes = NULL;
      sev.sigev_value.sival_ptr = &timer_id;


      if(timer_create(CLOCK_REALTIME, &sev, timer_id) != 0) //on success, 
timer id is placed in timer_id
      {
            return ERROR;
      }




    return SUCCESS;
}

/*****************************************************************
                             Start configuration
                 Parameter : delay in nano secs
*****************************************************************/
int kick_timer(timer_t timer_id, int interval_s)
{
   struct itimerspec in;

    in.it_value.tv_sec = interval_s; //sets initial time period
    in.it_value.tv_nsec = 0;
    in.it_interval.tv_sec = interval_s; //sets interval
    in.it_interval.tv_nsec =0;
    //issue the periodic timer request here.
    if( (timer_settime(timer_id, 0, &in, NULL)) != SUCCESS)
    {
      return ERROR;
    }
    return SUCCESS;
}

/*****************************************************************
                             Destroy Timer
*****************************************************************/
int stop_timer(timer_t timer_id)
{
      if( (timer_delete(timer_id)) != SUCCESS)
    {
      printf("Error on delete timer function\n");
      return ERROR;
    }


    return SUCCESS;
}
/**
 * @\file   server.c
 * @\author Sanju Prakash Kannioth and Steve Antony X
 * @\brief  This files contains the function definitions for server
socket
 * @\date   04/29/2019
 *
 */

#include "server.h"
/*****************************************************************
                        Globals
*****************************************************************/
socklen_t clilen;
struct sockaddr_in to_address;

int new_socket, server_socket;
/**********************************************
```

```
       Function for server socket creation
       Parameters : Port number
***********************************************/
int socket_creation_server(int port)
{

        //creating the socket for client

        server_socket = socket(AF_INET,SOCK_STREAM,0);// setting the
client socket
        if(server_socket < 0 ) // enters this loop if port number is not
given as command line argument
        {
                //printing error message when opening client socket
          perror("Error opening server socket\n");
          return -1;
        }


        struct sockaddr_in server_address;

        memset(&server_address,0,sizeof(server_address));

        //assigning values for the server address structure
        server_address.sin_family = AF_INET;
        server_address.sin_port = htons(port); // converting to network
byte order
        server_address.sin_addr.s_addr = INADDR_ANY;



        if(bind(server_socket,(struct
sockaddr*)&server_address,sizeof(server_address))<0)
        {
          perror("Binding failed in the server");
          return -1;
        }

        /*Listening for clients*/
        if(listen(server_socket,10) < 0)
        {
          perror("Error on Listening ");
          return -1;
        }
        else
        {
          printf("\nlistening to remote requests.....\n");
        }

      return 0;

}

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

```
       Function for remote request thread creation
       Parameters : Structure typecasted to void *
***************************************************/
void *remote_request_callback()
{


    char buffer[10];
    char manual = 'm';
    char automatic = 'a';
    char up = 'u';
    char down = 'b';
    char left = 'l';
    char right = 'r';
    char stop = 's';
    char clean = 'o';

    char mode_send[10];
    char waterLevel_send[10];
    char distance_send[10];
    char lux_send[10];
    char dgMode_send[10];
    char tiva_opstatus_send[10];
    char distance_opstatus_send[10];
    char lux_opstatus_send[10];
    char water_opstatus_send[10];
    char valve_status_send[10];

//creating socket for server
    if(socket_creation_server(PORT_NO)== -1)
    {
      perror("Error on socket creation - killed remote request socket");

    }


    while(1)
    {
      new_socket = 0;
      memset(&to_address,0,sizeof(to_address));


      clilen = sizeof(to_address);

    /*accepting client requests*/
      new_socket = accept(server_socket,(struct sockaddr*) &to_address,
&clilen);
      if(new_socket<0)
      {
        perror("Error on accepting client");
      }
      else
      {
        printf("established connection\n");
      }

    /*Forked the request received so as to accept multiple clients*/
      int child_id = 0;
    /*Creating child processes*/
    /*Returns zero to child process if there is successful child
creation*/
      child_id = fork();

    // error on child process
      if(child_id < 0)
      {
        perror("error on creating child\n");
        exit(1);
      }

    //closing the parent
      if (child_id > 0)
      {
        close(new_socket);
        waitpid(0, NULL, WNOHANG);  //Wait for state change of the child
process
      }

      if(child_id == 0)
      {

        memset(buffer,'\0',sizeof(buffer));
        while(recv(new_socket, buffer ,10, 0) > 0)
        {


         // printf("Received request %s - %ld\n",buffer,strlen(buffer));

         if(strcmp(buffer,"display")==0)
         {
          //printf("Received request for display\n");

          strcpy(lux_send, get_lux());
          send(new_socket, lux, 10 , 0);

          strcpy(distance_send, get_distance());
          send(new_socket, distance_send, 10 , 0);

          strcpy(waterLevel_send, get_waterLevel());
          send(new_socket, waterLevel_send, 10, 0);

              strcpy(mode_send, get_mode());
              send(new_socket, mode_send, 10, 0);

              strcpy(dgMode_send, get_dgMode());
              send(new_socket, dgMode_send, 10, 0);

          strcpy(tiva_opstatus_send, get_opStatus_tiva());
```

```
  send(new_socket, tiva_opstatus_send, 10, 0);

  strcpy(distance_opstatus_send, get_opStatus_distance());
  send(new_socket, distance_opstatus_send, 10, 0);

  strcpy(lux_opstatus_send, get_opStatus_lux());
  send(new_socket, lux_opstatus_send, 10, 0);

  strcpy(water_opstatus_send, get_opStatus_water());
  send(new_socket, water_opstatus_send, 10, 0);

  strcpy(valve_status_send, get_valveStatus());
  send(new_socket, valve_status_send, 10, 0);

  }

else if(strcmp(buffer,"manual")==0)
{
  //send m to tiva
  pthread_mutex_lock(&lock_res);
  uart_send(uart2, &manual, sizeof(char));
  pthread_mutex_unlock(&lock_res);
}

else if(strcmp(buffer,"auto")==0)
{
      printf("AUTO\n");
        //send a to tiva
  pthread_mutex_lock(&lock_res);
  uart_send(uart2, &automatic, sizeof(char));
  pthread_mutex_unlock(&lock_res);
}

else if(strcmp(buffer,"up")==0)
{
        //send u to tiva
  pthread_mutex_lock(&lock_res);
  uart_send(uart2, &up, sizeof(char));
  pthread_mutex_unlock(&lock_res);
}

else if(strcmp(buffer,"down")==0)
{
        //send b to tiva
  pthread_mutex_lock(&lock_res);
  uart_send(uart2, &down, sizeof(char));
  pthread_mutex_unlock(&lock_res);
}

else if(strcmp(buffer,"left")==0)
{
        //send l to tiva
  pthread_mutex_lock(&lock_res);
  uart_send(uart2, &left, sizeof(char));
```

```
          pthread_mutex_unlock(&lock_res);
        }

        else if(strcmp(buffer,"right")==0)
        {
                //send r to tiva
          pthread_mutex_lock(&lock_res);
          uart_send(uart2, &right, sizeof(char));
          pthread_mutex_unlock(&lock_res);
        }

        else if(strcmp(buffer,"stop")==0)
        {
                //send s to tiva
          pthread_mutex_lock(&lock_res);
          uart_send(uart2, &stop, sizeof(char));
          pthread_mutex_unlock(&lock_res);
        }

        if(strcmp(buffer, "on") == 0)
        {
          printf("ON RECEIVED\n");
          //send o to tiva
          pthread_mutex_lock(&lock_res);
          uart_send(uart2, &clean, sizeof(char));
          pthread_mutex_unlock(&lock_res);
        }

      }
      close(new_socket);
      exit(0);
    }
  }
}

#include <stdio.h>
#include <pthread.h>
#include <string.h>
#include <sys/types.h>

#include "server.h"
#include "communication.h"
#include "logger.h"
#include "heartbeat.h"
#include "log_receiver.h"


pthread_mutex_t lock_res; // Mutex

int main()
{
      char buffer[MAX_BUFFER_SIZE];

      pthread_attr_t attr;
      pthread_attr_init(&attr);

      logger_init();

      /* Create logger thread */
      if(pthread_create(&logger_thread, &attr, logger_thread_callback,
NULL) != 0)
      {
           memset(buffer,'\0',sizeof(buffer));
          sprintf(buffer,"ERROR CN [%s] Logger thread creation
failed\n",time_stamp());
          mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
           perror("Logger thread creation failed");
      }

      if (pthread_mutex_init(&lock_res, NULL) != 0)
    {
        perror("Mutex init failed\n");
        memset(buffer,'\0',sizeof(buffer));
          sprintf(buffer,"ERROR CN [%s] Mutex init
failed\n",time_stamp());
          mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
        return -1;
    }

      if(pthread_create(&remote_request_thread, &attr,
remote_request_callback, NULL) != 0)
      {
           memset(buffer,'\0',sizeof(buffer));
          sprintf(buffer,"ERROR CN [%s] Remote request thread creation
failed\n",time_stamp());
          mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
           perror("Remote socket thread creation failed");
      }

      if(pthread_create(&communication_thread, &attr,
communication_thread_callback, NULL) != 0)
      {
           memset(buffer,'\0',sizeof(buffer));
          sprintf(buffer,"ERROR CN [%s] Communication thread creation
failed\n",time_stamp());
          mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
           perror("Communication thread creation failed");
      }

      if(pthread_create(&log_receiver_thread, &attr,
revecive_thread_callback, NULL) != 0)
      {
           memset(buffer,'\0',sizeof(buffer));
          sprintf(buffer,"ERROR CN [%s] Logger receive thread creation
failed\n",time_stamp());
          mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
           perror("Receiver logger thread creation failed");
      }
      if(pthread_create(&heartbeat_thread, &attr,
heartbeat_thread_callback, NULL) != 0)
      {
            memset(buffer,'\0',sizeof(buffer));
          sprintf(buffer,"ERROR CN [%s] Heartbeat thread creation
failed\n",time_stamp());
          mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
            perror("Heartbeat thread creation failed");
      }

      memset(buffer,'\0',sizeof(buffer));
    sprintf(buffer,"DEBUG CN [%s] Threads creation
success\n",time_stamp());
    mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);

      printf("Threads created successfully\n");

      pthread_join(communication_thread,NULL);
      pthread_join(logger_thread,NULL);
      pthread_join(log_receiver_thread,NULL);
      pthread_join(heartbeat_thread, NULL);
      pthread_join(remote_request_thread,NULL);

      return 0;
}
/**
 * @\file   heartbeat.c
 * @\author Sanju Prakash Kannioth
 * @\brief  This files contains the definitions for the heartbeat
functions
 * @\date   04/29/2019
 *
 */

#include "heartbeat.h"


/*
--------------------------------------------------------------------------
-------------------
beat_timer_handler
--------------------------------------------------------------------------
-------------------
*     This function is the timer handler for tiva heart beat timer
*
*     @\param                 signal value ( dummy)
*
*     @\return          none
*
*/
void beat_timer_handler(union sigval val)
{
      char buffer[MAX_BUFFER_SIZE];
```

```
if(tiva_active <= tiva_active_prev)
{
      tiva_dead = 1;
      printf("ERROR Tiva dead\n");

      memset(buffer,'\0',sizeof(buffer));
sprintf(buffer,"ERROR CN [%s] Tiva Dead\n", time_stamp());
mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
}
else
{
      tiva_dead = 0;
      printf("INFO Tiva alive\n");

      memset(buffer,'\0',sizeof(buffer));
sprintf(buffer,"DEBUG CN [%s] Tiva Alive\n", time_stamp());
mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
}

if(distance_active <= distance_active_prev)
{
      distance_dead = 1;
      printf("ERROR Ultrasonic dead\n");

      memset(buffer,'\0',sizeof(buffer));
sprintf(buffer,"ERROR CN [%s] Ultrasonic Dead\n", time_stamp());
mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
}
else
{
      distance_dead = 0;

      printf("INFO Ultrasonic alive\n");

      memset(buffer,'\0',sizeof(buffer));
sprintf(buffer,"DEBUG CN [%s] Ultrasonic Alive\n", time_stamp());
mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
}

if(lux_active <= lux_active_prev)
{
      lux_dead = 1;
      printf("ERROR Lux dead\n");

      memset(buffer,'\0',sizeof(buffer));
sprintf(buffer,"ERROR CN [%s] Lux Dead\n", time_stamp());
mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
}
else
{
      lux_dead = 0;

      printf("INFO Lux alive\n");
```

```
           memset(buffer,'\0',sizeof(buffer));
      sprintf(buffer,"DEBUG CN [%s] Lux Alive\n", time_stamp());
      mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
      }

      if((water_active <= water_active_prev) || water_outOfRange)
      {
           water_dead = 1;
           printf("ERROR Water dead\n");

           memset(buffer,'\0',sizeof(buffer));
      sprintf(buffer,"ERROR CN [%s] Water level Dead\n", time_stamp());
      mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
      }
      else
      {
           water_dead = 0;

           printf("INFO Water alive\n");

           memset(buffer,'\0',sizeof(buffer));
      sprintf(buffer,"DEBUG CN [%s] Water level Alive\n", time_stamp());
      mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
      }


      tiva_active_prev = tiva_active;
      distance_active_prev = distance_active;
      lux_active_prev = lux_active;
      water_active_prev = water_active;



      //restarting the heartbeat timer
      if((kick_timer(timer_id_heartbeat, TIVA_HEART_BEAT_CHECK_PERIOD))
== -1)
      {
           memset(buffer,'\0',sizeof(buffer));
          sprintf(buffer,"ERROR CN [%s] Kicking timer for heartbeat
failed\n", time_stamp());
          mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
           perror("Error on kicking timer for heartbeat\n");
      }

}

/*
---------------------------------------------------------------------------
-------------------
heartbeat_thread_callback
---------------------------------------------------------------------------
-------------------
```

```
*     This is the thread creation callback function for the heartbeat
thread
*
*     @\param                void
*
*     @\return          void
*
*/
void *heartbeat_thread_callback()
{
      char buffer[MAX_BUFFER_SIZE];

      memset(buffer,'\0',sizeof(buffer));
    sprintf(buffer,"DEBUG CN [%s] Heartbeat thread active\n",
time_stamp());
    mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
      printf("Inside heartbeat thread\n");


      if((setup_timer_POSIX(&timer_id_heartbeat,beat_timer_handler)) == -
1)
      {
            memset(buffer,'\0',sizeof(buffer));
          sprintf(buffer,"ERROR CN [%s] Creating timer for heartbeat
failed\n", time_stamp());
          mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
            perror("Error on creating timer for heartbeat\n");
      }

      if((kick_timer(timer_id_heartbeat, TIVA_HEART_BEAT_CHECK_PERIOD))
== -1)
      {
            memset(buffer,'\0',sizeof(buffer));
          sprintf(buffer,"ERROR CN [%s] Kicking timer for heartbeat
failed\n", time_stamp());
          mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
            perror("Error on kicking timer for heartbeat\n");
      }

      /* Configure UART2 on BBG */
      uart2  = malloc(sizeof(uart_properties));
      uart2->uart_no = 2;
      uart2->baudrate = B115200;

      uint8_t isOpen2 = uart_config(uart2);

      printf("IS OPEN 2: %d\n",isOpen2);
      char hb ='h';

      if(isOpen2 == 0)
      {
            memset(buffer,'\0',sizeof(buffer));
          sprintf(buffer,"DEBUG CN [%s] UART2 Opened successfully\n",
time_stamp());
```

```
          mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);

            while(1)
            {
                  usleep(1000000); // Sending heartbeat every 1 second

                  pthread_mutex_lock(&lock_res);
                  uart_send(uart2,&hb,sizeof(char));
                  pthread_mutex_unlock(&lock_res);

            }
      }
      uart_close(uart2); // Close UART2
      pthread_cancel(heartbeat_thread);
      return 0;

}
/**
 * @\file   logger.c
 * @\author Sanju Prakash Kannioth
 * @\brief  This files contains the function definitions for the logger
 * @\date   04/29/2019
 *
 */

#include "logger.h"

pthread_mutex_t lock; // Mutex

char time_stam[30];

/**
-------------------------------------------------------------------------
-------------------
time_stamp
-------------------------------------------------------------------------
-------------------
*   This function will format the timestamp
*
*   @\param
*
*   @\return        timestamp as a string
*
*/
char *time_stamp()
{

    memset(time_stam,'\0',30);
      time_t timer;
      timer = time(NULL);
      strftime(time_stam, 26, "%Y-%m-%d %H:%M:%S", localtime(&timer));
      return time_stam;
}
```

```
/**
--------------------------------------------------------------------------
-------------------
logger_init
--------------------------------------------------------------------------
-------------------
*   This function will initialize the logger
*
*   @\param         void
*
*   @\return        void
*
*/
void logger_init()
{
      mq_unlink(QUEUE_NAME);
      file_ptr = fopen("test.log", "w");
      fprintf(file_ptr,"Queue Init\n\n");
      fclose(file_ptr);

    if (pthread_mutex_init(&lock, NULL) != 0)
    {
        perror("Mutex init failed\n");
    }


    struct mq_attr mq_attributes;

    /* Setting the message queue attributes */
    mq_attributes.mq_flags = 0;
    mq_attributes.mq_maxmsg = 5;
    mq_attributes.mq_msgsize = MAX_BUFFER_SIZE;
    mq_attributes.mq_curmsgs = 0;

    msg_queue = mq_open(QUEUE_NAME, O_CREAT | O_RDWR | O_NONBLOCK, 0666,
&mq_attributes);
    perror("MQ FAILED");
    printf("Return value of queue open = %d\n", msg_queue);
}


/**
--------------------------------------------------------------------------
-------------------
logger_thread_callback
--------------------------------------------------------------------------
-------------------
*   This function is the thread callback function for the logger
*
*   @\param         void
*
*   @\return        void
*
```

```
*/
void *logger_thread_callback()
{
      char buffer[MAX_BUFFER_SIZE];

      printf("Inside logger thread\n");

      memset(buffer,'\0',sizeof(buffer));
    sprintf(buffer,"DEBUG CN [%s] Logger thread active\n", time_stamp());
    mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);

      file_ptr = fopen("test.log", "a");
      while(1)
      {

            if(mq_receive(msg_queue, buffer,MAX_BUFFER_SIZE,0) > 0)
            {
                  pthread_mutex_lock(&lock);
                  fprintf(file_ptr,"%s",buffer);
                  fflush(file_ptr);
                  pthread_mutex_unlock(&lock);
            }
      }
      fclose(file_ptr);
      mq_close(msg_queue);
    mq_unlink(QUEUE_NAME);
    pthread_cancel(logger_thread);
}


/**
 * @\file   communication.c
 * @\author Sanju Prakash Kannioth
 * @\brief  This files contains the definitions for the communication
interface for tiva and BBG
 * @\date   04/29/2019
 *
 */
#include "communication.h"

/*
-------------------------------------------------------------------------
-------------------
get_lux
-------------------------------------------------------------------------
-------------------
*     This function returns the most updated lux value in string format
*
*     @\param                void
*
*     @\return          string containing most recent lux value
*
*/
char *get_lux()
```

```
{
      memset(lux,'\0',sizeof(lux));
      sprintf(lux,"%f",comm_rec.lux);
      return lux;
}


/*
--------------------------------------------------------------------------
-------------------
get_distance
--------------------------------------------------------------------------
-------------------
*     This function returns the most updated distance value in string
format
*
*     @\param                void
*
*     @\return         string containing most recent distance value
*
*/
char *get_distance()
{
      memset(distance,'\0',sizeof(distance));
      sprintf(distance,"%f",comm_rec.distance);
      return distance;
}


/*
--------------------------------------------------------------------------
-------------------
get_waterLevel
--------------------------------------------------------------------------
-------------------
*     This function returns the most updated water level value in string
format
*
*     @\param                void
*
*     @\return         string containing most recent water level value
*
*/
char *get_waterLevel()
{
      memset(waterLevel,'\0', sizeof(waterLevel));
      sprintf(waterLevel,"%f", comm_rec.waterLevel);
      return waterLevel;
}


/*
--------------------------------------------------------------------------
-------------------
```

```
get_mode
--------------------------------------------------------------------------
-------------------
*     This function returns the most updated operating mode in string
format
*
*     @\param                void
*
*     @\return          string containing most updated operating mode
value
*
*/
char *get_mode()
{
      memset(mode,'\0', sizeof(mode));
      if(!comm_rec.mode)
            sprintf(mode,"Auto");
      else if(comm_rec.mode)
            sprintf(mode,"Manual");
      return mode;
}

/*
--------------------------------------------------------------------------
-------------------
get_dgMode
--------------------------------------------------------------------------
-------------------
*     This function returns if the system is in degraded mode in string
format
*
*     @\param                void
*
*     @\return          string containing degraded mode value
*
*/
char *get_dgMode()
{
      memset(dg_mode,'\0', sizeof(dg_mode));
      if(!comm_rec.dg_mode)
            sprintf(dg_mode,"Normal");
      else if(comm_rec.dg_mode)
            sprintf(dg_mode,"Degraded");

      return dg_mode;
}

/*
--------------------------------------------------------------------------
-------------------
get_opStatus_tiva
--------------------------------------------------------------------------
-------------------
```

```
*     This function returns the most updated operation status of the
remote node
*
*     @\param                void
*
*     @\return         string containing the most updated operation
status of the remote node
*
*/
char *get_opStatus_tiva()
{
      memset(tiva_opstatus,'\0', sizeof(tiva_opstatus));
      if(!tiva_dead)
            sprintf(tiva_opstatus,"Alive");
      if(tiva_dead)
            sprintf(tiva_opstatus,"Dead");

      return tiva_opstatus;
}


/*
--------------------------------------------------------------------------
-------------------
get_opStatus_distance
--------------------------------------------------------------------------
-------------------
*     This function returns the most updated operation status of the
distance sensor
*
*     @\param                void
*
*     @\return         string containing the most updated operation
status of the distance sensor
*
*/
char *get_opStatus_distance()
{
      memset(distance_opstatus,'\0', sizeof(distance_opstatus));
      if(!distance_dead)
            sprintf(distance_opstatus,"Alive");
      if(distance_dead)
            sprintf(distance_opstatus,"Dead");

      return distance_opstatus;
}


/*
--------------------------------------------------------------------------
-------------------
get_opStatus_lux
--------------------------------------------------------------------------
-------------------
```

```
*     This function returns the most updated operation status of the lux
sensor
*
*     @\param                void
*
*     @\return         string containing the most updated operation
status of the lux sensor
*
*/
char *get_opStatus_lux()
{

      memset(lux_opstatus,'\0', sizeof(lux_opstatus));
      if(!lux_dead)
            sprintf(lux_opstatus,"Alive");
      if(lux_dead)
            sprintf(lux_opstatus,"Dead");

      return lux_opstatus;
}


/*
-------------------------------------------------------------------------
-------------------
get_opStatus_water
-------------------------------------------------------------------------
-------------------
*     This function returns the most updated operation status of the
water level sensor
*
*     @\param                void
*
*     @\return         string containing the most updated operation
status of the water level sensor
*
*/
char *get_opStatus_water()
{
      memset(water_opstatus,'\0', sizeof(water_opstatus));
      if(water_dead || water_outOfRange)
            sprintf(water_opstatus,"Dead");
      else
            sprintf(water_opstatus,"Alive");


      return water_opstatus;
}


/*
-------------------------------------------------------------------------
-------------------
get_valveStatus
```

```
----------------------------------------------------------------------------
-------------------
*     This function returns the most updated status of the valve
*
*     @\param                void
*
*     @\return         string containing the most updated status of the
valve
*
*/
char *get_valveStatus()
{
      memset(water_opstatus,'\0', sizeof(water_opstatus));
      if(already_open)
            sprintf(valve_status,"Open");
      else if(already_closed)
            sprintf(valve_status,"Closed");

      return valve_status;

}



/*
----------------------------------------------------------------------------
-------------------
communication_thread_callback
----------------------------------------------------------------------------
-------------------
*     This is the thread creation callback function for the communication
thread
*
*     @\param                void
*
*     @\return         void
*
*/
void *communication_thread_callback()
{
      char buffer[MAX_BUFFER_SIZE];
      /* Flags to check tiva and sensor heartbeats */
      tiva_active = 0;

      tiva_active_prev = 0;

      distance_active_prev = 0;

      distance_active = 0;

      lux_active_prev = 0;

      lux_active = 0;
      water_active_prev = 0;

    water_active = 0;

    memset(buffer,'\0',sizeof(buffer));
    sprintf(buffer,"DEBUG CN [%s] Communication thread active\n",
time_stamp());
    mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
      printf("Inside communication thread\n");

      struct sensor_struct sensor;

      /* Configure UART4 on BBG */
      uart_properties *uart4 = malloc(sizeof(uart_properties));
      uart4->uart_no = 4;
      uart4->baudrate = B115200;

      uint8_t isOpen4 = uart_config(uart4);

      printf("Open success? %d\n", isOpen4);

      /* Messages to be sent to Tiva for closed loop control */
      char obj_detect = '1';
      char valve_close = '2';
      char valve_open = '3';
      char lux_auto = '4';

      int recv = 0;


      if (isOpen4 == 0) {
            memset(buffer,'\0',sizeof(buffer));
          sprintf(buffer,"DEBUG CN [%s] UART4 Opened successfully\n",
time_stamp());
          mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);

            while(1)
            {
                  recv = uart_receive(uart4,&sensor,
sizeof(sensor_struct)); // Receive sensor data from Tiva on BBG UART4

                  if(recv > 0)
                  {

                        /* Send object detected message to Tiva if object
is detected */
                        if((strcmp(sensor.task_name,"DIST") == 0) &&
comm_rec.distance< 30)
                        {
                              pthread_mutex_lock(&lock_res);
                              uart_send(uart2, &obj_detect, sizeof(char));
                              pthread_mutex_unlock(&lock_res);

                              memset(buffer,'\0',sizeof(buffer));
```

```
                             sprintf(buffer,"WARN CN [%s] Object detected
sent from CN\n", time_stamp());
                             mq_send(msg_queue, buffer, MAX_BUFFER_SIZE,
0);
                       }

                       /* Send lux auto mode on to Tiva if lux is below
threshold */
                       else if((strcmp(sensor.task_name,"LUX") == 0) &&
comm_rec.lux < 10)
                       {

                             pthread_mutex_lock(&lock_res);
                             uart_send(uart2, &lux_auto, sizeof(char));
                             pthread_mutex_unlock(&lock_res);

                             memset(buffer,'\0',sizeof(buffer));
                             sprintf(buffer,"DEBUG CN [%s] Lux auto mode
sent from CN\n", time_stamp());
                             mq_send(msg_queue, buffer, MAX_BUFFER_SIZE,
0);
                       }

                       /* If water level sensor is inactive, or falls out
of range */
                       else if((strcmp(sensor.task_name,"WAT") == 0) &&
comm_rec.waterLevel > 3000)
                       {
                             water_outOfRange = 1;

                             memset(buffer,'\0',sizeof(buffer));
                             sprintf(buffer,"ERROR CN [%s] Water level
out of range\n", time_stamp());
                             mq_send(msg_queue, buffer, MAX_BUFFER_SIZE,
0);
                       }

                       /* Send valve closed to Tiva if water level is
below threshold */
                       else if((strcmp(sensor.task_name,"WAT") == 0) &&
comm_rec.waterLevel < 300 && already_closed == 0)
                       {
                             pthread_mutex_lock(&lock_res);
                             uart_send(uart2, &valve_close,
sizeof(char));
                             pthread_mutex_unlock(&lock_res);

                             memset(buffer,'\0',sizeof(buffer));
                             sprintf(buffer,"DEBUG CN [%s]  Valve close
sent from CN\n", time_stamp());
                             mq_send(msg_queue, buffer, MAX_BUFFER_SIZE,
0);

                             already_closed = 1;
                             already_open = 0;
                             water_outOfRange = 0;
                       }

                       /* Send valve open to Tiva if water level is above
threshold */
                       else if((strcmp(sensor.task_name,"WAT") == 0) &&
comm_rec.waterLevel >= 300 && already_open == 0)
                       {
                             pthread_mutex_lock(&lock_res);
                             uart_send(uart2, &valve_open, sizeof(char));
                             pthread_mutex_unlock(&lock_res);

                             memset(buffer,'\0',sizeof(buffer));
                             sprintf(buffer,"DEBUG CN [%s]  Valve open
sent from CN\n", time_stamp());
                             mq_send(msg_queue, buffer, MAX_BUFFER_SIZE,
0);

                             already_open = 1;
                             already_closed = 0;
                             water_outOfRange = 0;
                       }

                 }

           }
           uart_close(uart4); // Close UART4 file descriptor
           pthread_cancel(communication_thread);
     }
     return 0;
}
/**
 * @\file   log_receiver.c
 * @\author Sanju Prakash Kannioth and Steve Antony X
 * @\brief  This files contains the function definitions for tiva log
receiver module
 * @\date   03/30/2019
 *
 */

#include "log_receiver.h"


/**
-------------------------------------------------------------------------
-------------------
revecive_thread_callback
-------------------------------------------------------------------------
-------------------
*   This function is the thread callback function for logger messages
coming from Tiva
*
*   @\param         void
```

```
*
*   @\return        void
*
*/
void *revecive_thread_callback()
{
      char buffer[MAX_BUFFER_SIZE];


      memset(buffer,'\0',sizeof(buffer));
    sprintf(buffer,"DEBUG CN [%s] Tiva receive logger thread active\n",
time_stamp());
    mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);

      printf("Inside receiver thread\n");

      /* Configure UART1 on BBG */
      uart_properties *uart1  = malloc(sizeof(uart_properties));
      uart1->uart_no = 1;
      uart1->baudrate = B115200;

      uint8_t isOpen1 = uart_config(uart1);

      char log[50];

      if(isOpen1 == 0)
      {
            memset(buffer,'\0',sizeof(buffer));
          sprintf(buffer,"DEBUG CN [%s] UART1 Opened successfully\n",
time_stamp());
          mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);

            while(1)
            {
                  memset(log,'\0', sizeof(log));

                  uart_receive_task(uart1,&log,sizeof(log));

            }
      }
      uart_close(uart1); // Close UART1
      return 0;
}
/*Reference : https://github.com/sijpesteijn/BBCLib */

#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <termios.h>
#include <stdlib.h>
#include <stdint.h>
```

```
typedef enum {
      uart0 = 0, uart1 = 1, uart2 = 2, uart3 = 3, uart4 = 4, uart5 = 5
} uart;


typedef struct {
      int fd;
      uart uart_no;
      int baudrate;
}uart_properties;


uint8_t uart_config(uart_properties *uart)
{
      char path[15] = "/dev/ttyO";
      char uart_no[2];
      sprintf(uart_no,"%d",uart->uart_no);
      strcat(path,uart_no);

      struct termios uart_port;

      uart->fd = open(path, O_RDWR | O_NOCTTY | O_SYNC | O_NONBLOCK);
      if(uart->fd < 0) printf("port failed to open\n");

      uart_port.c_cflag = uart->baudrate | CS8 | CLOCAL | CREAD;
      uart_port.c_iflag&= ~(IGNBRK | ICRNL | INLCR | PARMRK | ISTRIP |
IXON); //= IGNPAR | ICRNL;
      uart_port.c_oflag = 0;
      uart_port.c_lflag = 0;

      uart_port.c_cc[VTIME] = 0;
      uart_port.c_cc[VMIN]  = 1;

      cfsetispeed(&uart_port, B115200);
      cfsetospeed(&uart_port, B115200);

      tcsetattr(uart->fd,TCSAFLUSH,&uart_port);
      printf("Opened\n");
      return 0;
}

int8_t uart_send(uart_properties *uart, char *tx, int length) {
      if (write(uart->fd, tx, length) == -1) {
            printf("Error in write\n");
            return -1;
      }
      printf("Wrote %s to uart %i\n", tx, uart->uart_no);
      return 0;
}


int8_t uart_receive(uart_properties *uart, char *rx, int length) {
      if (read(uart->fd, rx, length) == -1) {
            printf("Error in write\n");
            return -1;
      }

      printf("Read %s from uart %i\n", rx, uart->uart_no);
      return 0;
}

int8_t uart_close(uart_properties *uart) {
      close(uart->fd);
      return 0;
}

int main()
{

      uart_properties *uart  = malloc(sizeof(uart_properties));
      uart->uart_no = uart1;
      uart->baudrate = B115200;

      uart_properties *uart4 = malloc(sizeof(uart_properties));
      uart4->uart_no = 4;
      uart4->baudrate = B115200;

      uint8_t isOpen = uart_config(uart);
      uint8_t isOpen4 = uart_config(uart4);
      int i = 0;
      printf("Open success? %d\n", isOpen);
      if (isOpen == 0) {
            unsigned char receive[30];
            while(1)
            {
            //    printf("Entered while\n");
                  char buf[30];
                  sprintf(buf, "foo %d", ++i);

                  // Send data to uart1
                  if (uart_send(uart4, buf, strlen(buf) + 1) < 0) {
                        printf("Could not send data to uart port");
                        return -1;
                  }


                  usleep(100000);

                  uart_receive(uart4,receive,30);
                         sprintf(buf,"test uart 1");
                  uart_send(uart, buf, strlen(buf) + 1);
                  usleep(100000);
                  // Read data from uart1
                  if (uart_receive(uart, receive, 30)<  0) {
                        printf("Could not read data from uart port");
                        return -1;
                  }
            }
            uart_close(uart);
            uart_close(uart4);
      }
      printf("EOF\n");
      return 0;
}

/**
 * @\file   uart.c
 * @\author Sanju Prakash Kannioth
 * @\brief  This files contains the function definitions for uart
transmit and receive on BBG
 * @\date   04/29/2019
 * References : https://github.com/sijpesteijn/BBCLib,
 *
      https://en.wikibooks.org/wiki/Serial_Programming/termios
 *
 */

#include "uart.h"


char temp[MAX_BUFFER_SIZE];

struct sensor_struct *rx;

/**
-------------------------------------------------------------------------
-------------------
uart_config
-------------------------------------------------------------------------
-------------------
*   This function will configure the specific UART on BBG
*
*   @\param         uart_properties     specifies UART number
*
*   @\return        0                         success
*                            -1                          failure
*
*/

int8_t uart_config(uart_properties *uart)
{
      char path[15] = "/dev/ttyO";
      char uart_no[2];
      sprintf(uart_no,"%d",uart->uart_no);
      strcat(path,uart_no);

      struct termios uart_port;

      uart->fd = open(path, O_RDWR | O_NOCTTY | O_NDELAY);
      if(uart->fd < 0) printf("port failed to open\n");

      if(!isatty(uart->fd)) perror("Not a tty port\n");
      if(tcgetattr(uart->fd, &uart_port) < 0)
      {
            perror("Error getting attributes\n");
            return -1;

      }

      uart_port.c_cflag &= ~(CSIZE | PARENB);
      uart_port.c_cflag |= CLOCAL | CS8;

      uart_port.c_iflag&= ~(IGNBRK | ICRNL | INLCR | PARMRK | ISTRIP |
IXON);
      uart_port.c_oflag = 0;
      uart_port.c_lflag &= ~(ECHO | ECHONL | ICANON | IEXTEN | ISIG);

      fcntl(uart->fd, F_SETFL, 0);

      if(cfsetispeed(&uart_port, B115200) < 0) // Set input baud rate
      {
            perror("Input baud rate invalid\n");
            return -1;
      }

      if(cfsetospeed(&uart_port, B115200) < 0) // Set output baud rate
      {
            perror("Output baud rate invalid\n");
            return -1;
      }

      if(tcsetattr(uart->fd,TCSANOW,&uart_port) < 0) // Change
configurations immediately
      {
            perror("Attribute setting invalid\n");
            return -1;
      }
      printf("Opened\n");
      return 0;
}

/**
--------------------------------------------------------------------------
-------------------
uart_send
--------------------------------------------------------------------------
-------------------
*   This function will send specified length of bytes over the UART on
BBG
*
*   @\param         uart_properties     specifies UART number
*                            tx                            byte to be sent
*                            length                        number of bytes
to be sent
```

```
*
*   @\return        -1                        Failure
*                             count                      Number of bytes
sent
*
*/
int8_t uart_send(uart_properties *uart, void *tx, int length) {
      int count = write(uart->fd, tx, length);
      if (count == -1) {
            printf("Error in write\n");
            return -1;
      }
      return count;
}



/**
-------------------------------------------------------------------------
-------------------
uart_receive
-------------------------------------------------------------------------
-------------------
*   This function will receive tiva sensor data over the UART on BBG
*
*   @\param         uart_properties     specifies UART number
*                            rx_r                   byte received
*                            length                      number of bytes
to be received
*
*   @\return        -1                        Failure
*                             count                      Number of bytes
sent
*
*/
int8_t uart_receive(uart_properties *uart, void *rx_r, int length) {
      int count = 0;
      count = read(uart->fd, rx_r, length);
      if (count  == -1) {
            return -1;
      }

      rx = rx_r;

      if(strcmp(rx->task_name,"DIST") == 0)
      {
            distance_active++;

            comm_rec.distance = rx->distance;

            comm_rec.mode = rx->mode;

      }
      else if(strcmp(rx->task_name,"LUX") == 0)
      {
            lux_active++;

            comm_rec.lux = rx->lux;

            comm_rec.mode = rx->mode;

      }

      else if((strcmp(rx->task_name,"WAT") == 0) && rx->water < 3000)
      {
            water_active++;

            water_outOfRange = 0;

            comm_rec.waterLevel = rx->water;

            comm_rec.mode = rx->mode;
      }

      else if(strcmp(rx->task_name,"BEA") == 0)
      {
            tiva_active++;
            comm_rec.dg_mode = rx->dg_mode;
      }

      return count;
}


/**
-------------------------------------------------------------------------
-------------------
uart_receive_task
-------------------------------------------------------------------------
-------------------
*   This function will receive tiva log data over the UART on BBG
*
*   @\param         uart_properties     specifies UART number
*                            rx_r                   byte received
*                            length                       number of bytes
to be received
*
*   @\return        -1                         Failure
*                             count                       Number of bytes
sent
*
*/
int8_t uart_receive_task(uart_properties *uart, void *rx_r, int length) {

      int count = 0;

      count = read(uart->fd, rx_r, length);
      if(count == -1)
            return -1;

      memset(temp,'\0', sizeof(temp));
      strcpy(temp,rx_r);
      printf("%s",temp);
      if(count > 1)
      {
            mq_send(msg_queue, temp, MAX_BUFFER_SIZE, 0);
      }

      return count;
}


/**
--------------------------------------------------------------------------
-------------------
uart_close
--------------------------------------------------------------------------
-------------------
*   This function will close the specific UART on BBG
*
*   @\param         uart_properties     specifies UART number
*
*   @\return        0                         success
*
*/
int8_t uart_close(uart_properties *uart) {
      close(uart->fd);
      return 0;
}
```

```
/*
 * log.c
 *
 *  Created on: Apr 8, 2019
 *      Author: Steve Antony
 */

/**********************************************
 *        Includes
 **********************************************/
#include "log.h"


/**********************************************
 *        Global definitions
 **********************************************/
//receive log data from other tasks on remote node
char log_data_recv[100];

//structure to be transmitted from remote node to control node
typedef struct
{
    char task[5];
    uint32_t time_stamp;
    float distance;
    float lux;
    uint32_t water;
    int8_t mode_RN;
    int8_t Deg_mode;

}send_sensor_data;


send_sensor_data tx_data;


//to receive sensor values from various tasks to logger tasks
float lux_recv, distance_recv;
uint32_t Water_level_recv;
int8_t beat_recv;

//initiating the queues
void queue_init()
{
    myQueue_light = xQueueCreate(QueueLength, sizeof(float));
    if(myQueue_light == NULL)
    {
        UARTprintf("error on queue creation myQueue_light\n");
    }

    myQueue_ultra = xQueueCreate(QueueLength, sizeof(float));
    if(myQueue_ultra == NULL)
    {
        UARTprintf("error on queue creation myQueue_ultra\n");
    }

    myQueue_water = xQueueCreate(QueueLength, sizeof(uint32_t));
    if(myQueue_water == NULL)
    {
        UARTprintf("error on queue creation myQueue_water\n");
    }

    myQueue_log = xQueueCreate(QueueLength, 100);
    if(myQueue_log == NULL)
    {
        UARTprintf("error on queue creation myQueue_ultra\n");
    }

    myQueue_heartbeat = xQueueCreate(QueueLength, sizeof(int8_t));
    if(myQueue_heartbeat == NULL)
    {
        UARTprintf("error on queue creation myQueue_heartbeat\n");
    }

}

/**********************************************
 *        Logger thread
 **********************************************/
void LogTask(void *pvParameters)
{

    char buffer[50];
    unsigned char *ptr;
    ptr = (uint8_t *) (&tx_data);

    unsigned char *ptr1;
        ptr1 = (uint8_t *) (log_data_recv);




    for(;;)
    {
        //receive lux sensor value lux task
        if(xQueueReceive(myQueue_light, &lux_recv, 0 ) == pdTRUE )
        {
            strcpy(tx_data.task,"LUX");
            tx_data.lux = lux_recv;
            tx_data.time_stamp = xTaskGetTickCount();
            tx_data.mode_RN = mode;
            tx_data.Deg_mode = DEGRADED_MODE_MANUAL;
            UART_send(ptr, sizeof(tx_data));

        }

        //receive ultrasonic sensor value ultrasonic task
        if(xQueueReceive(myQueue_ultra, &distance_recv, 0 ) == pdTRUE )
        {
            strcpy(tx_data.task,"DIST");
            tx_data.distance = distance_recv;
            tx_data.time_stamp = xTaskGetTickCount();
            tx_data.mode_RN = mode;
            tx_data.Deg_mode = DEGRADED_MODE_MANUAL;
            UART_send(ptr, sizeof(tx_data));

        }

        //receive water level sensor value water level task
        if(xQueueReceive(myQueue_water, &Water_level_recv, 0 ) == pdTRUE
)
        {
            strcpy(tx_data.task,"WAT");
            tx_data.water = Water_level_recv;
            tx_data.time_stamp = xTaskGetTickCount();
            tx_data.mode_RN = mode;
            tx_data.Deg_mode = DEGRADED_MODE_MANUAL;

            UART_send(ptr, sizeof(tx_data));

        }

        //receive heartbeat from heartbeat task
        if(xQueueReceive(myQueue_heartbeat, &beat_recv, 0 ) == pdTRUE )
        {
            strcpy(tx_data.task,"BEA");
            tx_data.mode_RN = mode;
            tx_data.Deg_mode = DEGRADED_MODE_MANUAL;
            tx_data.time_stamp = xTaskGetTickCount();
            UART_send(ptr, sizeof(tx_data));

        }

        //receive log data from various tasks
        memset(log_data_recv,'\0',sizeof(log_data_recv));
        if(xQueueReceive(myQueue_log, log_data_recv, 0 ) == pdTRUE )
        {

//            UARTprintf("--> Log  %s\n",log_data_recv);
            if(CN_ACTIVE == pdTRUE)
            {
                UART_send_log(ptr1, strlen(log_data_recv));
            }


        }
    }
}

/*Uart function to sensor data to the control node*/
void UART_send(char* ptr, int len)
{
    while(len != 0)
    {
        UARTCharPut(UART2_BASE, *ptr);
        ptr++;
        len--;
    }
}

/*Uart function to logger data to the control node*/
void UART_send_log(char* ptr, int len)
{
    while(len != 0)
    {
        UARTCharPut(UART3_BASE, *ptr);
        ptr++;
        len--;
    }
}

/* FreeRTOS 8.2 Tiva Demo
 *
 * main.c
 *
 * Steve Antony
 *
 * This is a simple demonstration project of FreeRTOS 8.2 on the Tiva
Launchpad
 * EK-TM4C1294XL.  TivaWare driverlib sourcecode is included.
 */

#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#include "main.h"
#include "drivers/pinout.h"
#include "utils/uartstdio.h"


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

// FreeRTOS includes
#include "FreeRTOSConfig.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
#include "driverlib/gpio.h"
#include "driverlib/inc/hw_memmap.h"
#include "log.h"
```

```
#include "object_detection.h"
#include "uart.h"
#include "interrupt.h"
#include "rom.h"
#include "driverlib/fpu.h"
#include "motor_driver.h"
#include "heartbeat.h"
#include "waterlevel.h"
#include "semphr.h"

/**********************************************
 *               Tasks
 **********************************************/

/********************************************
 *         Actuator task
 * Description : Controls the autonomous movement
 *                of the robot in auto mode
 ********************************************/
void Actuator_motor(void *pvParameters);

/********************************************
 *         ReadUart task
 * Description : This tasks reads the control
 *                data which the Control node sends
 ********************************************/
void ReadUartTask(void *pvParameters);

/**********************************************
 *        Globals
 **********************************************/
// for queues that sends data from different tasks to the logger tasks
QueueHandle_t myQueue_ultra, myQueue_light, myQueue_log, myQueue_water,
myQueue_heartbeat;

//output clock
uint32_t output_clock_rate_hz;

//For object detection notification and heartbeat notification to get
pulses from control node
TaskHandle_t handle_motor,handle_heartbeat;

// flag to start only once based when lux is very low
static uint8_t start_again = 1;

//Flag set when the threads are not created properly
uint8_t STARTUP_FAILED = 0;

/*Sets application mode
 * mode 0  - Auto mode
 * mode 1  - Manual mode
 */
int8_t mode=0; //auto mode on default
```

```
//temporary buffer for logger
char temp_buffer[100];

/*mutex to avoid race condition when many tasks use
 * the same queue for logging
 */

SemaphoreHandle_t xSemaphore;

/**********************************************
 *        Main Function
 **********************************************/
int main(void)
{

    // Initialize system clock to 120 MHz
    output_clock_rate_hz = ROM_SysCtlClockFreqSet(
                               (SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN |
                                SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480),
                               SYSTEM_CLOCK);
    ASSERT(output_clock_rate_hz == SYSTEM_CLOCK);

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

    // Set up the UART which is connected to the virtual COM port
    UARTStdioConfig(0, 115200, SYSTEM_CLOCK);

    //initiating message queues for communication between various tasks
and logger
    queue_init();

    //initiating the semaphore
    xSemaphore = xSemaphoreCreateMutex();

    //configures the uarts UART1, UART2, UART3
    ConfigureUART1();
    ConfigureUART2();
    ConfigureUART3();

    //initiating the motor pins
    init_motor();

    // Create logger task
    if(pdPASS != xTaskCreate(LogTask, (const portCHAR *)"Log",
                   configMINIMAL_STACK_SIZE, NULL, 1, NULL))
   {
       STARTUP_FAILED = pdTRUE;
       LOG_ERROR("Thread creation failed for LogTask\n")
   }


    // Create light task
   if(pdPASS != xTaskCreate(LightTask, (const portCHAR *)"Light",
              configMINIMAL_STACK_SIZE, NULL, 1, NULL))
   {
       STARTUP_FAILED = pdTRUE;
       LOG_ERROR("Thread creation failed for LightTask\n")
   }


    // Create ultrasonic task
    if(pdPASS != xTaskCreate(UtrasonicTask, (const portCHAR
*)"ultrasonic",
                       configMINIMAL_STACK_SIZE, NULL, 1, NULL))
    {
        STARTUP_FAILED = pdTRUE;
        DEGRADED_MODE_MANUAL = 1;
        LOG_ERROR("Thread creation failed for UtrasonicTask\n")
    }

    // Create uart task for reading control data from control node
    if(pdPASS != xTaskCreate(ReadUartTask, (const portCHAR *)"UART",
                          configMINIMAL_STACK_SIZE, NULL, 1, NULL))
    {
        STARTUP_FAILED = pdTRUE;
        LOG_ERROR("Thread creation failed for ReadUartTask\n")
    }

    // Create motor actuator task
    if(pdPASS != xTaskCreate(Actuator_motor, (const portCHAR *)"motion",
                                 configMINIMAL_STACK_SIZE, NULL, 1,
&handle_motor))
    {
        STARTUP_FAILED = pdTRUE;
        LOG_ERROR("Thread creation failed for Actuator_motor\n")
    }

    // Create heartbeat task
    if(pdPASS != xTaskCreate(Control_Node_heartbeat, (const portCHAR
*)"heartbeat",
                                       configMINIMAL_STACK_SIZE, NULL, 1,
&handle_heartbeat))
    {
        STARTUP_FAILED = pdTRUE;
        LOG_ERROR("Thread creation failed for Control_Node_heartbeat\n")
    }

    // Create water level task
    if(pdPASS != xTaskCreate(Water_level, (const portCHAR *)"waterlevel",
                                              configMINIMAL_STACK_SIZE,
NULL, 1, NULL))
    {
        STARTUP_FAILED = pdTRUE;
        LOG_ERROR("Thread creation failed for Water_level\n")
    }
    //Checks if the threads were created successfully
    if(STARTUP_FAILED == pdTRUE)
    {
        LOG_ERROR("Startup test failed in creating tasks\n")
    }


    /*start the schedule*/
    vTaskStartScheduler();


    return 0;

}

/*Task to receive control data from control node*/
void ReadUartTask(void *pvParameters)
{
    for(;;)
        {
            while(UARTCharsAvail(UART1_BASE))
            {
                char c = ROM_UARTCharGet(UART1_BASE);
                UARTprintf("-> %c\n",c);
                if(c == 'h') //heartbeat
                {
                    xTaskNotifyGive(handle_heartbeat);
                }

                else if((c == '1') && (mode == 0) &&
(DEGRADED_MODE_MANUAL == 0))//object detected in auto mode
                {
                    xTaskNotifyGive(handle_motor);
                }

                else if(c == '2')//Water level low
                {
                    close_value();
                }

                else if(c == '3')//Water level high
                {
                    open_value();
                }
                else if(c == '4')//auto start - lux
                {
                    if((start_again == 1) && (mode == 0) &&
(DEGRADED_MODE_MANUAL == 0))
                    {
                        UARTprintf("CN: Auto on of robot\n");
                        LOG_INFO("Auto on of robot\n")
                        forward();
                        start_again = 0;
                    }
```

```
}

else if(c == 'm') //manual mode
{
    UARTprintf("CN: Manual mode\n");
    LOG_INFO("Switched to Manual mode\n")
    mode = 1 ;
    stop();

}
else if(c == 'a') //auto mode
{
    if((DEGRADED_MODE_MANUAL == 0))
    {
        UARTprintf("CN: Auto mode\n");
        LOG_INFO("Switched to Auto mode\n")
        mode = 0 ;
    }

}

else if(c == 'u') //forward
{
    if(mode == 1)
    {
        UARTprintf("CN: forward\n");
        forward();
    }

}
else if(c == 's') //stop
{
   stop();
   start_again = 1;
   UARTprintf("CN: stop\n");
}

else if(c == 'l') //left
{
    if(mode == 1)
    {
        left();
        UARTprintf("CN: left\n");
        vTaskDelay(300/portTICK_PERIOD_MS);
        stop();

    }

}
else if(c == 'r') //right
{
    if(mode == 1)
    {
        right();
```

```
                        UARTprintf("CN: right\n");
                        vTaskDelay(300/portTICK_PERIOD_MS);
                        stop();

                    }

                }
                else if(c == 'b') //back
                {
                    if(mode == 1)
                    {
                       backward();
                       UARTprintf("CN: back\n");

                    }

                }
                else if(c == 'o') //force start from phone
                {
                    if((DEGRADED_MODE_MANUAL == 0))
                    {
                        mode = 0 ;
                        forward();
                        UARTprintf("CN: force turn on\n");
                        LOG_INFO("force turn on from phone\n")

                    }

                }
            }
    }

}

/*Actuator task to control motors when an object is detected*/
void Actuator_motor(void *pvParameters)
{

        for(;;)
        {

            uint32_t ulNotifiedValue = 0;

            ulNotifiedValue  = ulTaskNotifyTake( pdTRUE, 0  );
            if(ulNotifiedValue > 0)
            {

                UARTprintf("Object detected notified\n");
                LOG_INFO("Object detected\n")
                backward();


                 vTaskDelay(1000/portTICK_PERIOD_MS);
                //normal run of motors
                right();

                vTaskDelay(500/portTICK_PERIOD_MS);

                forward();

            }

        }
}


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



/*Uart to transmit sensor data to the control node*/
//Transmit data on PA7
void ConfigureUART2(void)
{
        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);    //Enable GPIO

        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART2);    //Enable
UART0

        ROM_GPIOPinConfigure(GPIO_PA6_U2RX);                //Configure
UART pins
        ROM_GPIOPinConfigure(GPIO_PA7_U2TX);
        ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_6 | GPIO_PIN_7);

        ROM_UARTConfigSetExpClk(UART2_BASE, output_clock_rate_hz, 115200,
                                    (UART_CONFIG_WLEN_8 |
UART_CONFIG_STOP_ONE |
                                                UART_CONFIG_PAR_NONE));

        UARTprintf("configured 2\n");

}
```

```
/*Uart to receive control data from the control node*/
//UART1 recv on PB0
void ConfigureUART1()
{

        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);    //Enable GPIO

        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);    //Enable
UART0

        ROM_GPIOPinConfigure(GPIO_PB0_U1RX);                //Configure
UART pins
        ROM_GPIOPinConfigure(GPIO_PB1_U1TX);
        ROM_GPIOPinTypeUART(GPIO_PORTB_BASE, GPIO_PIN_0 | GPIO_PIN_1);


        ROM_UARTConfigSetExpClk(UART1_BASE, output_clock_rate_hz, 115200,
                                     (UART_CONFIG_WLEN_8 |
UART_CONFIG_STOP_ONE |
                                                 UART_CONFIG_PAR_NONE));

        UARTprintf("configured 1\n");




}

//logger send/*Uart to transmit log data to the control node*/
//UART3 tx on PA5
void ConfigureUART3()
{

        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);    //Enable GPIO

        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART3);    //Enable
UART3

        ROM_GPIOPinConfigure(GPIO_PA5_U3TX);                //Configure
UART pins
        ROM_GPIOPinConfigure(GPIO_PA4_U3RX);
        ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_5 | GPIO_PIN_4);


        ROM_UARTConfigSetExpClk(UART3_BASE, output_clock_rate_hz, 115200,
                                     (UART_CONFIG_WLEN_8 |
UART_CONFIG_STOP_ONE |
                                                 UART_CONFIG_PAR_NONE));

        UARTprintf("configured 3\n");
```

```
}



/*
 * motor_driver.c
 *
 *  Created on: Apr 14, 2019
 *      Author: Steve Antony
 */

#include "motor_driver.h"


void init_motor()
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);

    GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE, GPIO_PIN_0);

GPIOPadConfigSet(GPIO_PORTE_BASE,GPIO_PIN_0,GPIO_STRENGTH_2MA,GPIO_PIN_TY
PE_STD_WPU);

    GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE, GPIO_PIN_1);

GPIOPadConfigSet(GPIO_PORTE_BASE,GPIO_PIN_1,GPIO_STRENGTH_2MA,GPIO_PIN_TY
PE_STD_WPU);

    GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE, GPIO_PIN_2);

GPIOPadConfigSet(GPIO_PORTE_BASE,GPIO_PIN_2,GPIO_STRENGTH_2MA,GPIO_PIN_TY
PE_STD_WPU);


    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
    GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_7);

GPIOPadConfigSet(GPIO_PORTC_BASE,GPIO_PIN_7,GPIO_STRENGTH_2MA,GPIO_PIN_TY
PE_STD_WPU);

}

void stop()
{
    GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_0, 0);
    GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_1, 0);

    GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_2, 0);
    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0);
}

void forward()
{
    GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_0, GPIO_PIN_0);
    GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_1, 0);

    GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_2, GPIO_PIN_2);
    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0);
}

void backward()
{
    GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_0, 0);
    GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_1, GPIO_PIN_1);

    GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_2, 0);
    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, GPIO_PIN_7);
}

void right()
{
    GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_0, GPIO_PIN_0);
    GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_1, 0);

    GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_2, 0);
    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0);
}

void left()
{
    GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_0, 0);
    GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_1, 0);

    GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_2, GPIO_PIN_2);
    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0);
}
/**********************************************
 *               Includes
 **********************************************/
#include "heartbeat.h"

/**********************************************
 *               Globals
 **********************************************/
//timer flag to check the heartbeat after regular intervals
int FLAG_HB = 0;

//flag is set if the control node is active
int CN_ACTIVE = 0;

//for sending heartbeat from Remote node to control node
int8_t BEAT = 1;

//storing pulses to find heartbeat
static uint32_t Pulse = 0, Prev_pulse = 0;

//temporary buffer for logger
char temp_buffer[100];
```

```
/*
-------------------------------------------------------------------------
-------------------
               Control_Node_heartbeat task
-------------------------------------------------------------------------
-------------------
*   This Task sends heartbeat continuosly from remote node to control
node and
*   Checks if the control node is active
*
*/
void Control_Node_heartbeat(void *pvParameters)
{
    UARTprintf("Created heartbeat task\n");
    long x_heartbeat_id = 1019;
   xTimerHandle xTimer_HB;
   xTimer_HB = xTimerCreate("Heart_beat",              // Just a text
name, not used by the kernel.
                             pdMS_TO_TICKS( 1000 ),     // 100ms
                             pdTRUE,                   // The timers will
auto-reload themselves when they expire.
                             ( void * ) x_heartbeat_id,      // Assign
each timer a unique id equal to its array index.
                             vTimerCallback_HB_handler// Each timer calls
the same callback when it expires.
                            );

    if( xTimer_HB == NULL )
    {
       // The timer was not created.
       UARTprintf("Error on HB timer creation\n");
    }

    xTimerStart( xTimer_HB, 0 );
    for(;;)
    {
        uint32_t ulNotifiedValue = 0;


        //notified when heartbeat is received from control node
        ulNotifiedValue  = ulTaskNotifyTake( pdTRUE, 0  );
        if(ulNotifiedValue > 0)
        {

            Pulse++;
        }

        if(FLAG_HB)
        {
            FLAG_HB = pdFALSE;

            //checks if there was a pulse received
            if(Pulse <= Prev_pulse)
            {
               // UARTprintf("Control node dead Pr %d P %d\n",Prev_pulse,
Pulse);
                CN_ACTIVE = pdFALSE;
            }
            else
            {
               // UARTprintf("Control node active Pr %d P
%d\n",Prev_pulse, Pulse);
                CN_ACTIVE = pdTRUE;
            }

            Prev_pulse = Pulse;

            //send pulse from remote node to control node
            xQueueSendToBack( myQueue_heartbeat,( void * ) &BEAT,
QUEUE_TIMEOUT_TICKS ) ;

            //turn off the remote node leds when the control node is
active
            if(CN_ACTIVE)
            {
                GPIOPinWrite(CLP_D1_PORT, CLP_D1_PIN, 0);
                GPIOPinWrite(CLP_D2_PORT, CLP_D2_PIN, 0);
                GPIOPinWrite(CLP_D3_PORT, CLP_D3_PIN, 0);
                GPIOPinWrite(CLP_D4_PORT, CLP_D4_PIN, 0);

            }
            //turn on the remote node leds when the control node is
active
            else
            {
                GPIOPinWrite(CLP_D1_PORT, CLP_D1_PIN, CLP_D1_PIN);
                GPIOPinWrite(CLP_D2_PORT, CLP_D2_PIN, CLP_D2_PIN);
                GPIOPinWrite(CLP_D3_PORT, CLP_D3_PIN, CLP_D3_PIN);
                GPIOPinWrite(CLP_D4_PORT, CLP_D4_PIN, CLP_D4_PIN);

                //move to fail safe mode from degraded mode when the
ultrasonic sensor is dead and control node inactive
                if((DEGRADED_MODE_MANUAL == 1))
                {
                    stop();
                    UARTprintf("System shutdown as no ultrasonic sensor
and no control node - Fail safe\n");
                }
            }


        }

    }

}
```

```
/*Heartbeat Timer handler*/
void vTimerCallback_HB_handler( TimerHandle_t  *pxTimer )
{
    FLAG_HB = pdTRUE;
}

/*
 * water_level.c
 *
 *  Created on: Apr 24, 2019
 *      Author: Steve
 */

/**********************************************
 *        Includes
 **********************************************/
#include "waterlevel.h"

/**********************************************
 *        Globals
 **********************************************/
int FLAG_WL = 0;
static char buffer_log[BUFFER];
char temp_buffer[100];

/*water level task*/
void Water_level(void *pvParameters)
{
    vTaskDelay(1000/portTICK_PERIOD_MS);
    UARTprintf("Water level task\n");
    uint32_t Water_level_data;

    init_valve();

    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);

    GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_3);

    ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);

    ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH0 | ADC_CTL_IE |
                                ADC_CTL_END);

    ADCSequenceEnable(ADC0_BASE, 3);

    ADCIntClear(ADC0_BASE, 3);

    long x_WaterL_id = 1009;
    xTimerHandle xTimer_WL;
    xTimer_WL = xTimerCreate("Waterlevel_timer",               // Just a
text name, not used by the kernel.
                                pdMS_TO_TICKS( 2000 ),     // 1000ms
                                pdTRUE,                    // The timers
will auto-reload themselves when they expire.
                                ( void * ) x_WaterL_id,      // Assign
each timer a unique id equal to its array index.
                                vTimerCallback_WaterLevel_handler// Each
timer calls the same callback when it expires.
                               );



        if( (xTimer_WL == NULL ) )
        {
            // The timer was not created.
            UARTprintf("Error on timer creation - xTimer_WL\n");
        }

    /*start the timer*/
    xTimerStart( xTimer_WL, 0 );


    /*start up test*/
    ADCProcessorTrigger(ADC0_BASE, 3);


    while(!ADCIntStatus(ADC0_BASE, 3, false))
       {
       }
    ADCIntClear(ADC0_BASE, 3);

    ADCSequenceDataGet(ADC0_BASE, 3, &Water_level_data);

    //kill if the startup fails
    if(Water_level_data > 3000)
    {
        UARTprintf("Startup test failed for water level sensor WL %d\n",
Water_level_data);
         LOG_ERROR("Killed water level sensor task - Startup failed\n")
         vTaskDelete( NULL );
    }



    while(1)
    {
        if(FLAG_WL)
        {
            ADCProcessorTrigger(ADC0_BASE, 3);


            while(!ADCIntStatus(ADC0_BASE, 3, false))
               {
               }
            ADCIntClear(ADC0_BASE, 3);
            ADCSequenceDataGet(ADC0_BASE, 3, &Water_level_data);


            if(CN_ACTIVE)
             {
                 xQueueSendToBack( myQueue_water,( void * )
&Water_level_data, QUEUE_TIMEOUT_TICKS ) ;
                 memset(buffer_log,'\0',BUFFER);
                 sprintf(buffer_log,"W %d\n",Water_level_data);
                 LOG_INFO(buffer_log)
             }


            FLAG_WL = pdFALSE;

        }
    }
}
void vTimerCallback_WaterLevel_handler( TimerHandle_t  *pxTimer )
{
    FLAG_WL = pdTRUE;
}

void init_valve()
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK);

    GPIOPinTypeGPIOOutput(GPIO_PORTK_BASE, GPIO_PIN_7);

GPIOPadConfigSet(GPIO_PORTK_BASE,GPIO_PIN_7,GPIO_STRENGTH_2MA,GPIO_PIN_TY
PE_STD_WPU);
}
void open_value()
{
    GPIOPinWrite(GPIO_PORTK_BASE, GPIO_PIN_7, GPIO_PIN_7);
    UARTprintf("CN: Valve opened\n");
    LOG_INFO("Valve opened")
}

void close_value()
{
    GPIOPinWrite(GPIO_PORTK_BASE, GPIO_PIN_7, 0);
    UARTprintf("CN: Valve closed\n");
    LOG_INFO("Valve closed")
}
/*
 * object_detection.c
 *
 *  Created on: Apr 15, 2019
 *      Author: Steve Antony
 */

/**********************************************
 *        Includes
 **********************************************/

#include "object_detection.h"


/**********************************************
 *        GLOBALS
 **********************************************/
//to find the pulse duration
uint32_t start, end;

//conversion complete flag
uint32_t FLAG_UL, conv_complete = 0;

//find the duration of echo on pulse
float time_pulse = 0;

//get the distance
float distance_send;

//for local logger
static char buffer_log[BUFFER];

//flag to indicate if the sensor is dead
uint32_t ULT_DEAD = 0;

//indicate degraded mode
uint32_t DEGRADED_MODE_MANUAL = 0;

//mutex for log
SemaphoreHandle_t xSemaphore;

//local logger buffer
char temp_buffer[100];


void init_ultrasonic_sensor()
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);
    TimerConfigure(TIMER2_BASE, TIMER_CFG_PERIODIC_UP);


    //echo pin
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_3);

//GPIOPadConfigSet(GPIO_PORTF_BASE,GPIO_PIN_3,GPIO_STRENGTH_2MA,GPIO_PIN_
TYPE_STD_WPU);

    GPIOIntEnable(GPIO_PORTF_BASE, GPIO_INT_PIN_3);

    GPIOIntTypeSet(GPIO_PORTF_BASE,GPIO_PIN_3,GPIO_BOTH_EDGES );

    GPIOIntRegister(GPIO_PORTF_BASE,PortFIntHandler);
    GPIOIntClear(GPIO_PORTF_BASE, GPIO_INT_PIN_3);



    //trigger pin
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1);

GPIOPadConfigSet(GPIO_PORTF_BASE,GPIO_PIN_1,GPIO_STRENGTH_2MA,GPIO_PIN_TY
PE_STD_WPU);

    UARTprintf("configured ultrasonic\n");

}


void find_object()
{
    GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_1, 0);
    vTaskDelay(pdMS_TO_TICKS( 1 ));
    GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_1, GPIO_PIN_1);
    vTaskDelay(pdMS_TO_TICKS( 10 ));
    GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_1, 0);
}


void PortFIntHandler()
{


    taskENTER_CRITICAL();
    GPIOIntClear(GPIO_PORTF_BASE, GPIO_INT_PIN_3);


        if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_INT_PIN_3) ==
GPIO_INT_PIN_3)
        {
            HWREG(TIMER2_BASE + TIMER_O_TAV) = 0;
            TimerEnable(TIMER2_BASE,TIMER_A);
            start = TimerValueGet(TIMER2_BASE,TIMER_A);
        }

        else
        {

            end = TimerValueGet(TIMER2_BASE,TIMER_A);
            TimerDisable(TIMER2_BASE,TIMER_A);
            time_pulse = end - start;
            conv_complete = 1;


        }

    taskEXIT_CRITICAL();
```

```
}

void UtrasonicTask(void *pvParameters)
{
    vTaskDelay(1000/portTICK_PERIOD_MS);
    UARTprintf("Created ultrasonic thread\n");
    long x_ultra_id = 1003;
    xTimerHandle xTimer_ult;
    xTimer_ult = xTimerCreate("Timer_ultrasonic",               // Just a
text name, not used by the kernel.
                                pdMS_TO_TICKS( PERIOD_ULTRASONIC ),
                                pdTRUE,                    // The timers
will auto-reload themselves when they expire.
                                ( void * ) x_ultra_id,      // Assign
each timer a unique id equal to its array index.
                                vTimerCallback_Ultra_handler// Each timer
calls the same callback when it expires.
                               );



    if( (xTimer_ult == NULL ) )
    {
        // The timer was not created.
        UARTprintf("Error on timer creation - xTimer_Temp\n");
    }

    else
    {

        /*start the timer*/
         xTimerStart( xTimer_ult, 0 );

         init_ultrasonic_sensor();

         //startup test
         find_object();
         vTaskDelay(500/portTICK_PERIOD_MS);


         if(time_pulse == 0)
         {
             UARTprintf("Startup test failed for ultrasonic sensor\n");
             LOG_ERROR("Startup failed for ULTRASONIC\n")
             DEGRADED_MODE_MANUAL = 1;
             mode = 1;
             vTaskDelete( NULL );
         }



         for(;;)
         {
             if(FLAG_UL == pdTRUE)
             {
                 find_object();

                 if((conv_complete == 1))
                 {
                     distance_send =
(((float)(1.0/(output_clock_rate_hz/1000000))*time_pulse)/58);

                     if(CN_ACTIVE)
                     {
                         xQueueSendToBack( myQueue_ultra,( void * )
&distance_send, QUEUE_TIMEOUT_TICKS ) ;
                         memset(buffer_log,'\0',BUFFER);
                         sprintf(buffer_log,"D %f\n",distance_send);
                         LOG_INFO(buffer_log)
                     }
                     else
                     {
                         if(distance_send < 30)
                         {
                             //when object detected
                             xTaskNotifyGive(handle_motor);


                         }

                     }
                     conv_complete = 0;

                 }
                 else
                 {
                     ULT_DEAD++;
                 }

                 FLAG_UL = pdFALSE;

             }
             if(ULT_DEAD > 5) //switch to degraded mode
             {
                DEGRADED_MODE_MANUAL = 1;
                mode = 1;
                stop();
                UARTprintf("Killed Utrasonic sensor task\n");
                LOG_ERROR("Killed Utrasonic sensor task\n")
                vTaskDelete( NULL );
             }


         }

    }
}
```

```
/**********************************************
 *         Temp timer handler
 **********************************************/
void vTimerCallback_Ultra_handler( TimerHandle_t  *pxTimer )
{
    FLAG_UL = pdTRUE;
}

/*
 * lux.c
 *
 *  Created on: Apr 9, 2019
 *      Author: Steve Antony
 */

/**********************************************
 *        Includes
 **********************************************/
#include <lux.h>

/**********************************************
 *        Globals
 **********************************************/
int FLAG_Light = 0;
struct log_struct_temp log_temp;
static char buffer_log[BUFFER];
char temp_buffer[100];

/*for writing and reading as byte from the registers*/
uint8_t register_data;

/*for storing MSB and LSB of CH0 of lux*/
uint16_t MSB_0;
uint16_t LSB_0;

/*for storing MSB and LSB of CH1 of lux*/
uint16_t MSB_1;
uint16_t LSB_1;

/*16 bit value of CH0 and CH1*/
uint16_t CH0;
uint16_t CH1;
float lux_send;

static uint8_t start_again = 1;


/**********************************************
 *         Temperature thread
 **********************************************/

void LightTask(void *pvParameters)
{
    vTaskDelay(3000/portTICK_PERIOD_MS);
    UARTprintf("Created Light Task\n");

        long x_light_id = 10005;
        xTimerHandle xTimer_light;
        xTimer_light = xTimerCreate("Timer_Light",              // Just a
text name, not used by the kernel.
                                  pdMS_TO_TICKS( TEMP_TIME_PERIOD_MS ),
// 100ms
                                  pdTRUE,                   // The timers
will auto-reload themselves when they expire.
                                   ( void * ) x_light_id,     // Assign
each timer a unique id equal to its array index.
                                  vTimerCallback_Light_handler// Each
timer calls the same callback when it expires.
                                 );

         if( xTimer_light == NULL )
         {
            // The timer was not created.
            UARTprintf("Error on timer creation\n");
         }
         else
         {
            /*Start led timer*/
            xTimerStart( xTimer_light, 0 );
            i2c_setup();
            vTaskDelay(1000/portTICK_PERIOD_MS);

            //Start up tests for lux sensor
            int8_t ret = lux_sensor_setup();
            if(ret == -1)
            {
                UARTprintf("Startup failed for lux\n");
                LOG_ERROR("Killed lux sensor task - Startup failed\n")
                vTaskDelete( NULL );
            }

            for (;;)
            {

                if(FLAG_Light == pdTRUE)
                {
                    FLAG_Light = pdFALSE;
                    read_lux_CH0();
                    read_lux_CH1();

                    lux_send = lux_measurement(CH0,CH1);
                    if(CN_ACTIVE)
                    {
                        xQueueSendToBack( myQueue_light,( void * )
&lux_send, QUEUE_TIMEOUT_TICKS ) ;
                        memset(buffer_log,'\0',BUFFER);
                        sprintf(buffer_log,"L %f\n",lux_send);
                        LOG_INFO(buffer_log)
                    }
                    if((CN_ACTIVE == 0) && (lux_send < 1.0)&&(start_again
== 1))
                    {
                        UARTprintf("CN INACTIVE, lux < 1, start\n");
                        forward();
                        start_again = 0;
                    }


                }
            }
         }
}


void vTimerCallback_Light_handler( TimerHandle_t  *pxTimer )
{
    FLAG_Light = pdTRUE;
}


void i2c_setup()
{
    /*Enabling i2c pheripheral*/
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C2);

    /*Enabling GPIO*/
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);

    /*Configuring I2C SDA GPIO*/
    GPIOPinConfigure(GPIO_PN4_I2C2SDA);

    /*Configuring I2C SCL GPIO*/
    GPIOPinConfigure(GPIO_PN5_I2C2SCL);

    /*Configuring ic2 SCL*/
    GPIOPinTypeI2CSCL(GPIO_PORTN_BASE, GPIO_PIN_5);

    /*Configuring ic2 SDA*/
    GPIOPinTypeI2C(GPIO_PORTN_BASE, GPIO_PIN_4);

    /*wait till specified peripheral is ready
     * returns true if ready*/
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_I2C2));

    /*initiating i2c master*/
    I2CMasterInitExpClk(I2C2_BASE, output_clock_rate_hz, false);
}

/*Configuring lux sensor*/
int8_t lux_sensor_setup()
{
    int flag = 0;
    /*command to write on control register*/
    register_data = 0x03;
    write_byte_i2c2(LIGHT_SENSOR, CONTROL_REGISTER, register_data);

    register_data = 0x00;
    read_byte_i2c2(LIGHT_SENSOR, CONTROL_REGISTER, &register_data);

    //UARTprintf("0x03 --> %x",register_data);
    if((register_data == 0x00))
    {
        return -1;
    }

    /*command to write on TIMING_REGISTER*/
    register_data = 0x12;
    write_byte_i2c2(LIGHT_SENSOR, TIMING_REGISTER, register_data);

    return 0;

}


void read_lux_CH0()
{
    /*command to write on control register*/
    register_data = 0x00;
    read_byte_i2c2(LIGHT_SENSOR, DATA0LOW_REGISTER, &register_data);

    LSB_0 = 0;
    LSB_0 = register_data;

    /*command to write on TIMING_REGISTER*/
    register_data = 0x00;
    read_byte_i2c2(LIGHT_SENSOR, DATA0HIGH_REGISTER, &register_data);

    MSB_0 = 0;
    MSB_0 = register_data;

    /*forming the full 16 bit from MSB and LSB*/
    CH0 = (MSB_0 << 8);
    CH0 |= LSB_0;


}

void read_lux_CH1()
{
    /*command to write on control register*/
    register_data = 0x00;
    read_byte_i2c2(LIGHT_SENSOR, DATA1LOW_REGISTER, &register_data);

    LSB_0 = 0;
    LSB_0 = register_data;
    /*command to write on TIMING_REGISTER*/
    register_data = 0x00;
    read_byte_i2c2(LIGHT_SENSOR, DATA1HIGH_REGISTER, &register_data);

    MSB_1 = 0;
    MSB_1 = register_data;

    /*forming the full 16 bit from MSB and LSB*/
    CH1 = (MSB_1 << 8);
    CH1 |= LSB_1;


}

void read_byte_i2c2(uint8_t slave, uint8_t register_addr, uint8_t * data)
{
    /*select the register to read on slave*/
    I2CMasterSlaveAddrSet(I2C2_BASE, slave, false);

    //command to write to control register
    I2CMasterDataPut(I2C2_BASE, register_addr | WRITE_COMMAND  );

    //Controls the state of the I2C Master , command
    I2CMasterControl(I2C2_BASE, I2C_MASTER_CMD_SINGLE_SEND);

    //Wait until master says it is busy
    while(!I2CMasterBusy(I2C2_BASE));

    //Indicates whether I2C Master is busy
    while(I2CMasterBusy(I2C2_BASE));

    /* reads the data*/
    /*Sets the address that the I2C Master places on the bus*/
     I2CMasterSlaveAddrSet(I2C2_BASE, slave, true);

     I2CMasterControl(I2C2_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);

     //Wait until master says it is busy
     while(!I2CMasterBusy(I2C2_BASE));

     //Indicates whether I2C Master is busy
     while(I2CMasterBusy(I2C2_BASE));

     *data = I2CMasterDataGet(I2C2_BASE);


}


void write_byte_i2c2(uint8_t slave, uint8_t register_addr, uint8_t data)
{
    /*select the register to read on slave*/
    I2CMasterSlaveAddrSet(I2C2_BASE, slave, false);

    //command to write to control register
    I2CMasterDataPut(I2C2_BASE, register_addr | WRITE_COMMAND  );

    //Controls the state of the I2C Master , command
    I2CMasterControl(I2C2_BASE, I2C_MASTER_CMD_SINGLE_SEND);

    //Wait until master says it is busy
    while(!I2CMasterBusy(I2C2_BASE));

    //Indicates whether I2C Master is busy
    while(I2CMasterBusy(I2C2_BASE));

    I2CMasterDataPut(I2C2_BASE, data);


     I2CMasterControl(I2C2_BASE, I2C_MASTER_CMD_SINGLE_SEND);

     //Wait until master says it is busy
     while(!I2CMasterBusy(I2C2_BASE));

     //Indicates whether I2C Master is busy
     while(I2CMasterBusy(I2C2_BASE));


}


/*****************************************************************
                     Getting lux value
*****************************************************************/
float lux_measurement(float CH0, float CH1)
{

    float ratio = (CH1 / CH0);



    //0 < CH1/CH0 â‰¤ 0.50 Sensor Lux = (0.0304 x CH0) â€“ (0.062 x CH0 x
((CH1/CH0)1.4))

    if((ratio <=0.5)&& (ratio > 0))
        return ((0.0304 * CH0) - (0.062 * CH0 * (powf(ratio, 1.4))));

    //0.50 < CH1/CH0 â‰¤ 0.61 Sensor Lux = (0.0224 x CH0) â€“ (0.031 x
CH1)

    else if((ratio  > 0.5)&& (ratio <= 0.61))
        return ((0.0224 * CH0) - (0.031 * CH1));

    //0.61 < CH1/CH0 â‰¤ 0.80 Sensor Lux = (0.0128 x CH0) â€“ (0.0153 x
CH1)
    else if((ratio  > 0.61)&& (ratio <= 0.8))
```

```
        return (0.0128 * CH0) - (0.0153 * CH1);

    //0.80 < CH1/CH0 â‰¤ 1.30 Sensor Lux = (0.00146 x CH0) â€“ (0.00112 x
CH1)
    else if((ratio  > 0.80)&& (ratio <= 1.30))
        return (0.00146 * CH0) - (0.00112 * CH1);

    //CH1/CH0>1.30 Sensor Lux = 0
    else
        return 0;


}
```

```
/*
 * lux.h
 *
 *  Created on: Apr 9, 2019
 *      Author: Steve Antony
 */

#ifndef LUX_H_
#define LUX_H_


/**********************************************
 *        Includes
 **********************************************/
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#include <math.h>

#include "FreeRTOSConfig.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
#include "utils/uartstdio.h"
#include "uart.h"
#include "driverlib/gpio.h"
#include "driverlib/inc/hw_memmap.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "log.h"
#include "i2c.h"

/**********************************************
 *        MACRO
 **********************************************/
#define CONTROL_REGISTER (0X00)
#define TIMING_REGISTER (0X01)
#define THRESHLOWLOW (0x02)
#define THRESHLOWHIGH (0x03)
#define THRESHHIGHLOW (0x04)
#define THRESHHIGHHIGH (0x05)
#define INTERRUPT (0x06)
#define INDICATION_REGISTER (0x0A)
#define DATA0LOW_REGISTER (0X0C)
#define DATA0HIGH_REGISTER (0X0D)
#define DATA1LOW_REGISTER (0X0E)
#define DATA1HIGH_REGISTER (0X0F)

#define WRITE_COMMAND (0x80)


#define QUEUE_TIMEOUT_TICKS (10)
```

```
#define NOTIFY_TAKE_TIMEOUT (500)
#define TEMP_TIME_PERIOD_MS (1000)
#define TEMP_SENSOR_ADDR (0x48)
#define TEMP_REG_OFFSET_ADDR (0x00)
#define LIGHT_SENSOR (0x39)

#define BUFFER (50)

/**********************************************
 *        GLOBALS
 **********************************************/
struct log_struct_temp
{
    char time_stamp[30];
    char temp[40];

};

struct log_struct_led
{
    char time_stamp[30];
    long count;
    char name[10];

};


extern TaskHandle_t handle;

extern uint32_t output_clock_rate_hz;

extern QueueHandle_t myQueue_light;


/**********************************************
 *        Function Prototypes
 **********************************************/
/********************************************
 * Func name :   TempTask
 * Parameters:   none
 * Description : Thread for temperature task
 ********************************************/
void LightTask(void *pvParameters);
/********************************************
 * Func name :   vTimerCallback_Temp_handler
 * Parameters:   none
 * Description : handler for temperature timer
 ********************************************/
void vTimerCallback_Light_handler( TimerHandle_t  *);

/********************************************
 * Func name :   i2c_setup
 * Parameters:   none
 * Description : Configuration of i2c bus
 ********************************************/
void i2c_setup();

/********************************************
 * Func name :   read_lux_CH0
 * Parameters:   none
 * Description : Reads CH0 value of the lux sensor
 ********************************************/
void read_lux_CH0();

/********************************************
 * Func name :   read_lux_CH1
 * Parameters:   none
 * Description : Reads CH1 value of the lux sensor
 ********************************************/
void read_lux_CH1();

/********************************************
 * Func name :   lux_sensor_setup
 * Parameters:   none
 * Description : Wrapper for configuring lux sensor registers
 ********************************************/
int8_t lux_sensor_setup();

/********************************************
 * Func name :   read_byte_i2c2
 * Parameters:   slave address,  register address, address of data
 * Description : Read a byte to any register
 ********************************************/
void read_byte_i2c2(uint8_t slave, uint8_t register_addr, uint8_t *data);

/********************************************
 * Func name :   write_byte_i2c2
 * Parameters:   slave address,  register address, data
 * Description : Write a byte to any register
 ********************************************/
void write_byte_i2c2(uint8_t slave, uint8_t register_addr, uint8_t data);

/********************************************
 * Func name :   lux_measurement
 * Parameters:   CH0 value, CH1 value
 * Description : Calculate lux value based on the channel values
 ********************************************/
float lux_measurement(float , float );

#endif /* LUX_H_ */
/*
 * main.h
 *
 *  Created on: Mar 28, 2015
 *      Author: steve
 */

#ifndef OBJECT_DETECTION_H_
```

```
#define OBJECT_DETECTION_H_

/**********************************************
 *        Includes
 **********************************************/
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>

#include "FreeRTOSConfig.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
#include "timer.h"

#include "driverlib/gpio.h"
#include "driverlib/inc/hw_memmap.h"
#include "driverlib/inc/hw_timer.h"
#include "driverlib/inc/hw_types.h"

#include "driverlib/sysctl.h"
#include "driverlib/rom.h"
#include "utils/uartstdio.h"
#include "driverlib/inc/hw_ints.h"
#include "driverlib/fpu.h"
#include "log.h"
#include "motor_driver.h"
#include "semphr.h"


/**********************************************
 *        Macros
 **********************************************/
#define DETECT_TIME_PERIOD_MS (1000)
#define PERIOD_ULTRASONIC     (530)


/**********************************************
 *        Function prototypes
 **********************************************/
/********************************************
 * Func name :   init_ultrasonic_sensor
 * Parameters:   none
 * Description : Initiates the trigger and echo pins of ultrasonic
sensors
 ********************************************/
void init_ultrasonic_sensor();

/********************************************
 * Func name :   PortFIntHandler
 * Parameters:   none
 * Description : Interrupt handler
 ********************************************/
void PortFIntHandler();

/********************************************
 * Func name :   findobject
 * Parameters:   none
 * Description : makes trigger pin high for 10ms
 ********************************************/
void find_object();

/********************************************
 * Func name :   vTimerCallback_Temp_handler
 * Parameters:   none
 * Description : handler for temperature timer
 ********************************************/
void vTimerCallback_Ultra_handler( TimerHandle_t  *);

/*Ultrasonic task*/
void UtrasonicTask(void *);

/**********************************************
 *        Global declaration
 **********************************************/
extern uint32_t output_clock_rate_hz;
extern QueueHandle_t myQueue_ultra, myQueue_light, myQueue_log;
extern TaskHandle_t handle_motor;



#endif
/*
 * log.h
 *
 *  Created on: Apr 8, 2019
 *      Author: Steve Antony
 */

#ifndef LOG_H_
#define LOG_H_

/**********************************************
 *        Includes
 **********************************************/
#include <lux.h>
#include "FreeRTOS.h"
#include "queue.h"
#include "portmacro.h"
#include "utils/uartstdio.h"
#include "portmacro.h"
#include "time.h"
#include "semphr.h"
/**********************************************
 *        MACRO
 **********************************************/
```

```
#define QueueLength (110)
#define TIMEOUT_TICKS (10)
#define BUFFER (100)


#define LOG_INFO(str) {\
xSemaphoreTake(xSemaphore, 0);\
memset(temp_buffer,'\0',100);\
sprintf(temp_buffer,"INFO RN: [%d] %s",xTaskGetTickCount(),str);\
xQueueSendToBack( myQueue_log,( void * ) temp_buffer, QUEUE_TIMEOUT_TICKS
) ;\
xSemaphoreGive(xSemaphore);\
}

#define LOG_ERROR(str) {\
xSemaphoreTake(xSemaphore, 0);\
memset(temp_buffer,'\0',100);\
sprintf(temp_buffer,"ERROR RN: [%d] %s",xTaskGetTickCount(),str);\
xQueueSendToBack( myQueue_log,( void * ) temp_buffer, QUEUE_TIMEOUT_TICKS
) ;\
xSemaphoreGive(xSemaphore);\
}

#define LOG_WARN(str) {\
xSemaphoreTake(xSemaphore, 0);\
memset(temp_buffer,'\0',100);\
sprintf(temp_buffer,"WARN RN: [%d] %s",xTaskGetTickCount(),str);\
xQueueSendToBack( myQueue_log,( void * ) temp_buffer, QUEUE_TIMEOUT_TICKS
) ;\
xSemaphoreGive(xSemaphore);\
}
/**********************************************
 *        Global declarations
 **********************************************/
extern QueueHandle_t myQueue_light, myQueue_ultra, myQueue_log,
myQueue_water, myQueue_heartbeat;
extern int CN_ACTIVE ;
extern int8_t mode;
extern uint32_t DEGRADED_MODE_MANUAL;
extern SemaphoreHandle_t xSemaphore;

/**********************************************
 *        Function Prototypes
 **********************************************/
/********************************************
 * Func name : queue_init
 * Parameters: none
 * Description : initiates the queues for logger
 */
void queue_init();

/********************************************
 * Func name :   LogTask
 * Parameters:   none
 * Description : Thread for logger task
 ********************************************/
void LogTask(void *pvParameters);


/********************************************
 * Func name :   UART_send
 * Parameters:   Address, length
 * Description : Uart function to send sensor values to the control node
 ********************************************/
void UART_send(char* ptr, int len);

/********************************************
 * Func name :   UART_send_log
 * Parameters:   Address, length
 * Description : Uart function to send log data to the control node
 ********************************************/
void UART_send_log(char* ptr, int len);
#endif /* LOG_H_ */
/*
 * heartbeat.h
 *
 *  Created on: Apr 23, 2019
 *      Author: Steve
 */

#ifndef INC_HEARTBEAT_H_
#define INC_HEARTBEAT_H_

/**********************************************
 *               Includes
 **********************************************/
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>

#include "FreeRTOSConfig.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
#include "timer.h"

#include "driverlib/gpio.h"
#include "driverlib/inc/hw_memmap.h"
#include "driverlib/inc/hw_timer.h"
#include "driverlib/inc/hw_types.h"

#include "driverlib/sysctl.h"
#include "driverlib/rom.h"
#include "utils/uartstdio.h"
#include "driverlib/inc/hw_ints.h"
#include "driverlib/fpu.h"
```

```
#include "driverlib/gpio.h"
#include "drivers/pinout.h"

#include "motor_driver.h"

/**********************************************
 *               Function Prototypes
 **********************************************/
/*Heartbeat task*/
void Control_Node_heartbeat(void *pvParameters);

/*Heartbeat Timer handler*/
void vTimerCallback_HB_handler( TimerHandle_t  *pxTimer );

/**********************************************
 *               Global declarations
 **********************************************/
extern QueueHandle_t myQueue_heartbeat;
extern uint32_t DEGRADED_MODE_MANUAL;

#endif /* INC_HEARTBEAT_H_ */



#ifndef MOTOR_DRIVER_H_
#define MOTOR_DRIVER_H_


/*
 * motor_driver.h
 *
 *  Created on: Apr 14, 2019
 *      Author: Steve Antony
 */

/**********************************************
 *        Includes
 **********************************************/
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>

#include "FreeRTOSConfig.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
#include "timer.h"

#include "driverlib/gpio.h"
#include "driverlib/inc/hw_memmap.h"
#include "driverlib/inc/hw_timer.h"
#include "driverlib/inc/hw_types.h"
```

```
#include "driverlib/sysctl.h"
#include "driverlib/rom.h"
#include "utils/uartstdio.h"
#include "driverlib/inc/hw_ints.h"
#include "driverlib/fpu.h"


/**********************************************
 *        Function Prototypes
 **********************************************/
/*
-------------------------------------------------------------------------
-------------------
init_motor
-------------------------------------------------------------------------
-------------------
*   This functions is used to initiate the motor output pins
*
*   @\param         void
*
*   @\return        void
*
*/
void init_motor();

/*
-------------------------------------------------------------------------
-------------------
stop
-------------------------------------------------------------------------
-------------------
*   This functions stops the motion of robot
*
*   @\param         void
*
*   @\return        void
*
*/
void stop();

/*
-------------------------------------------------------------------------
-------------------
forward
-------------------------------------------------------------------------
-------------------
*   This functions moves the robot forward
*
*   @\param         void
*
*   @\return        void
*
*/
```

```
void forward();

/*
-------------------------------------------------------------------------
-------------------
left
-------------------------------------------------------------------------
-------------------
*   This functions turns the robot left
*
*   @\param         void
*
*   @\return        void
*
*/
void left();

/*
-------------------------------------------------------------------------
-------------------
right
-------------------------------------------------------------------------
-------------------
*   This functions turns the robot right
*
*   @\param         void
*
*   @\return        void
*
*/
void right();

/*
-------------------------------------------------------------------------
-------------------
backward
-------------------------------------------------------------------------
-------------------
*   This functions moves the robot backward
*
*   @\param         void
*
*   @\return        void
*
*/
void backward();




#endif
/*
 * main.h
 *
 *  Created on: Apr 20, 2019
 *      Author: Steve
 */

#ifndef MAIN_H_
#define MAIN_H_

/**********************************************
 *              MACROS
 **********************************************/
// System clock rate, 120 MHz
#define SYSTEM_CLOCK    (120000000U)

#define QUEUE_TIMEOUT_TICKS (10)

/**********************************************
 *              GLOBAL DECLARATION
 **********************************************/
extern uint32_t DEGRADED_MODE_MANUAL;

/**********************************************
 *              Function Prototypes
 **********************************************/
/*
-------------------------------------------------------------------------
-------------------
ConfigureUART2
-------------------------------------------------------------------------
-------------------
*   This configures UART2
*
*   @\param         none
*
*   @\return        none
*
*/
void ConfigureUART2();

/*
-------------------------------------------------------------------------
-------------------
ConfigureUART1
-------------------------------------------------------------------------
-------------------
*   This configures UART1
*
*   @\param         none
*
*   @\return        none
*
*/
void ConfigureUART1();

/*
```

```
-------------------------------------------------------------------------
-------------------
ConfigureUART3
-------------------------------------------------------------------------
-------------------
*   This configures UART3
*
*   @\param         none
*
*   @\return        none
*
*/
void ConfigureUART3();
#endif /* MAIN_H_ */
/*
 * waterlevel.h
 *
 *  Created on: Apr 24, 2019
 *      Author: Steve
 */

#ifndef INC_WATERLEVEL_H_
#define INC_WATERLEVEL_H_


/**********************************************
 *        Includes
 **********************************************/
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>

#include "FreeRTOSConfig.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
#include "timer.h"

#include "driverlib/gpio.h"
#include "driverlib/inc/hw_memmap.h"
#include "driverlib/inc/hw_timer.h"
#include "driverlib/inc/hw_types.h"

#include "driverlib/sysctl.h"
#include "driverlib/rom.h"
#include "utils/uartstdio.h"
#include "driverlib/inc/hw_ints.h"
#include "driverlib/fpu.h"
#include "driverlib/gpio.h"
#include "drivers/pinout.h"
#include "driverlib/adc.h"
#include "log.h"
```

```
/**********************************************
 *        Global declaration
 **********************************************/
extern QueueHandle_t myQueue_water;


/**********************************************
 *        Function prototypes
 **********************************************/
/********************************************
         Water level task
 ********************************************/
void Water_level(void *pvParameters);

/********************************************
      Timer callback for water level task
 ********************************************/
void vTimerCallback_WaterLevel_handler( TimerHandle_t  *pxTimer );

/********************************************
 * Func name :   init_valve
 * Parameters:   none
 * Description : initiates the valve control pin
 ********************************************/
void init_valve();

/********************************************
 * Func name :   close_value
 * Parameters:   none
 * Description : close the water valve
 ********************************************/
void close_value();

/********************************************
 * Func name :   open_value
 * Parameters:   none
 * Description : open the water valve
 ********************************************/
void open_value();


#endif /* INC_WATERLEVEL_H_ */
```