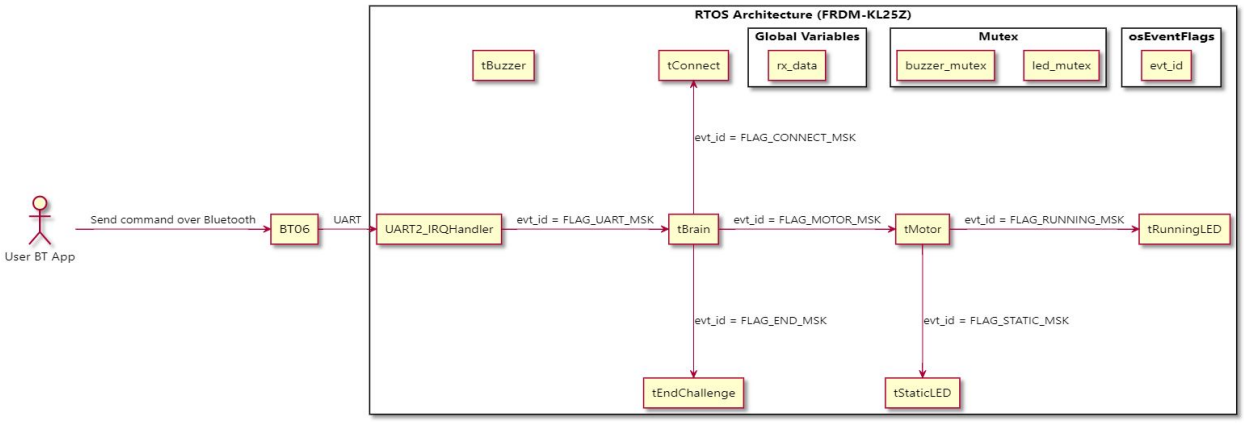
CG2271 Mini Project Report

RTOS ARCHITECTURE



The overall RTOS architecture on the FRDM-KL25Z along with a few other components is shown in the above UML diagram. The arrows between them indicate the means by which inter-process communication is achieved.

Components

1. BT App
    a. Used to send commands based on the user's requirements.
2. BT06 Module
    a. Receives the command sent over by the user through the BT application.
3. FRDM KL25Z
    a. Used to implement the RTOS architecture to accomplish the various requirements.

Global Variables

1. rx_data
    a. This is an 8-bit volatile variable which is used to store the command sent over by the user.
    b. This variable is then used by the tBrain thread to interpret and execute the command sent over by the user.

Mutex

1. led_mutex
    a. Since, tConnect, tStaticLED and tRunningLED threads all use the same set of LED strips, led_mutex is used to make sure that only one of these threads has access to the LED strips at a specific point of time (prevent race conditions). So that the LED's work in the required way.
    b. led_mutex is purely to guard against tConnect (prevent race conditions), **not** for tStaticLed and tRunningLed to be against each other (flags took care of that).
    c. Alternatively, tConnect can be set to have higher priority instead of having a led_mutex.

2. buzzer_mutex
    a. Since, tConnect, tBuzzer and tEndChallenge threads all use the same buzzer to play the required tone, buzzer_mutex is used to make sure that only one of these threads have access to the buzzer at a specific point of time (avoid race conditions). So that the buzzer plays the various tones in the required way.

osEventFlags

1. evt_id
    a. This osEventFlag evt_id is used for inter-process communication. Various threads can be put into the READY/BLOCKED state based on the value set for the evt_id flag.
    b. The following table lists the different values set to the flag.

Flag Mask Name	Value	Thread
FLAG_UART_MSK	0x00000001U	tBrain
FLAG_RUNNING_MSK	0x00000002U	tRunningLED
FLAG_STATIC_MSK	0x00000020U	tStaticLED
FLAG_MOTOR_MSK	0x00000004U	tMotor
FLAG_CONNECT_MSK	0x00000008U	tConnect
FLAG_END_MSK	0x00000010U	tEndChallenge

    c. Only the thread corresponding to the flag set gets to go into the READY state, the rest of the threads are put into the BLOCKED state. The osScheduler then puts the thread in the READY state into the RUNNING state.

IRQ Handler

1. UART2_IRQHandler
    a. It first receives the command packet sent over by BT06 module through serial communication (UART).
    b. Then it sets the evt_id as FLAG_UART_MSK so that the tBrain thread could go from the BLOCKED → READY, and eventually from the READY → RUNNING state. The command is then handled by the tBrain thread.

Threads

1. tBrain
    a. It uses the rx_data value sent over by the user to decide which thread has to be run to in order to accomplish the command.
    b. Then, evt_id flag is set according to the following table in order to choose the thread that has to be run.

rx_data	Type of Command / Event	Thread
0x00 to 0x08	Movement Command	tMotor
0x09	End Challenge Command	tEndChallenge
0x0A	Bluetooth Connection established	tConnect

2. tRunningLED
    a. It is used to control the LED strips when the bot is moving.
    b. The front 8 green LED's light up sequentially and the rear 8 red LED's flash continuously at the rate of 500 ms ON and 500 ms OFF.

3. tStaticLED
    a. It is used to control the LED strips when the bot is static.
    b. The front 8 green LED's are all lighted up and the rear 8 red LED's flash continuously at the rate of 250 ms ON and 250 ms OFF.

4. tMotor
    a. It calls the required function to move the robot as per the user's command.
    b. It also sets the evt_id flag according to the following table to run either tRunningLED thread (or) tStaticLED thread based on the user's command.

rx_data	Thread
0X00	tStaticLED
0X01 to 0X08	tRunningLED

    c. Since the access to the common resource is handled by letting them wait for separate values of the evt_id. Issues regarding concurrent resource access are avoided.

5. tConnect
    a. It blinks the LED twice and plays a unique tone to indicate that the Bluetooth connection is established.

6. tEndChallenge
    a. It plays a unique tone to indicate that the challenge has ended.

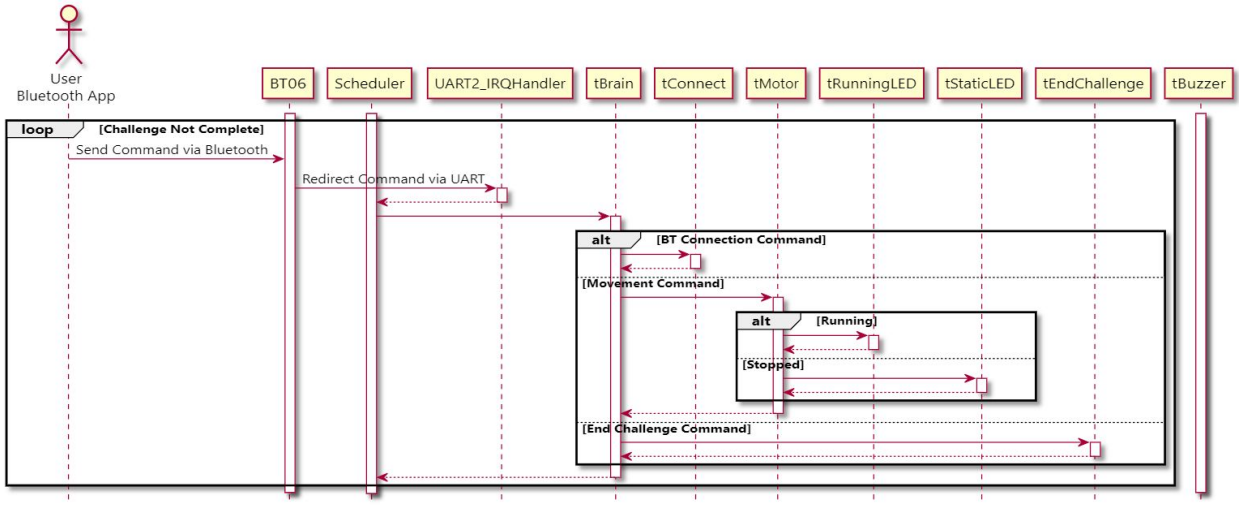
7. tBuzzer
    a. It plays the song that has to be played continuously as per the requirements of the project.

Design Considerations

- Event flags were used instead of thread flags as it allows more flexibility and also simplifies the code.
- Message queues weren't used as the current usage of event flags can already meet the requirements and introducing it could have increased overhead and affected the performance of our robot.
- Semaphores were not needed as mutexes were sufficient, we simply just wanted mutual exclusion of the buzzer and LED resource.
- Similarly, queues weren't used to store the received data from the IRQ_Handler as we felt that our current implementation is already responsive enough, introducing a buffer such as a circular queue could have introduced significant overheads affecting our robot's performance.
- All threads have the same priority levels, the event flags and mutex were sufficient to meet the project requirements. We also avoid starvation and priority inversion problems that may arise if we use priority.

Sequence Diagram

The following sequence diagram indicates the flow of control on how the various tasks are run and how tasks communicate with each other based on the previous architecture.



Done by Group 33:

DANIEL TAN WEE LIANG (e0310018@u.nus.edu)
GANAPATHY SANATH BALAJI (e0311091@u.nus.edu)
CHU JUI HUNG @JEFFERSON CHU (e0309385@u.nus.edu)