Team 07: Wang Zihao (A0204706M), Zhuang Mengjin (A0204716L), Wu Nan (A0206048L)

RTOS Architecture Report

Hardware Design

The robotic car contains 8 green LEDs at the front, 8 red LEDs at the rear, 1 DRV8833 Dual Motor Driver Carrier, 1 Piezo Buzzer, 1 BT06 Module, 1 7805 Regulator and 1 FRDM KL25Z Board. The BT06 Module is for bluetooth connection with the Android phone, which is powered by GP 9V Battery. The DRV8833 Dual Motor Driver Carrier is to control the four motors, which is powered by 6 × 1.5V Duracell Batteries. The FRDM KL25Z Board is powered by the power bank (Figure 1 & 2). Four different PWMs will be sent to the motors via the DRV8833 Dual Motor Driver Carrier to control the movement of the robotic car respectively.

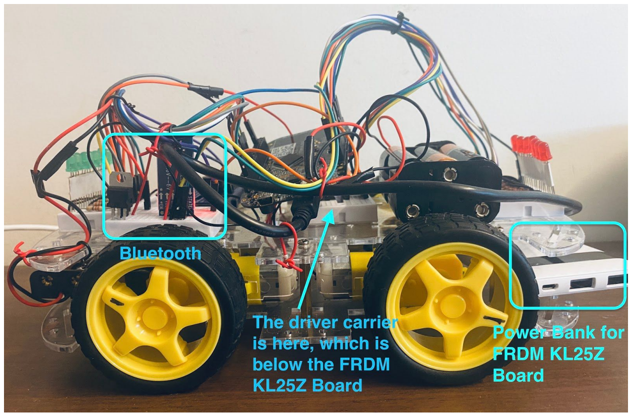

Bluetooth
The driver carrier is here, which is below the FRDM KL25Z Board
Power Bank for FRDM KL25Z Board

Figure 1 Side View of the Robotic Car

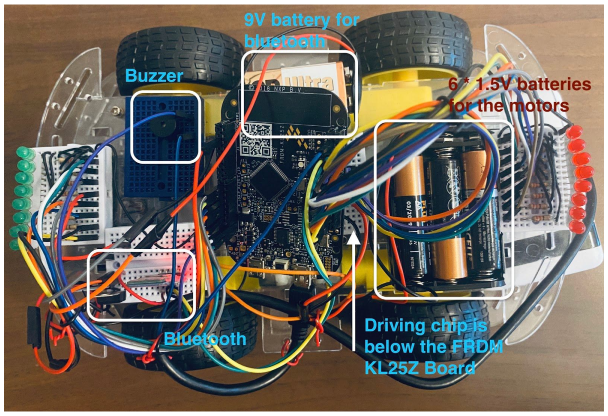

9V battery for bluetooth
Buzzer
6 * 1.5V batteries for the motors
Bluetooth
Driving chip is below the FRDM KL25Z Board

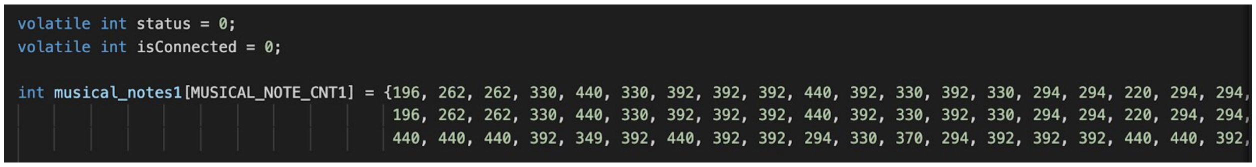
Figure 2 Top View of the Robotic Car

Software Design

Overall, the program uses a **multithreaded environment** to achieve concurrency and uses the concept of **message queue** for basic communication between threads.

Global Variables

The global variable `status` is used to process the command sent from the Android phone and change the status of the robotic card correspondingly. It is updated in UART2_IRQHandler and used in the command_thread(). The global variable `isConnected` is used to identify whether the bluetooth connection is established or not. The musical_note1 array is used to store the musical notes that will be used in the function audio() to display the Doraemon theme song.

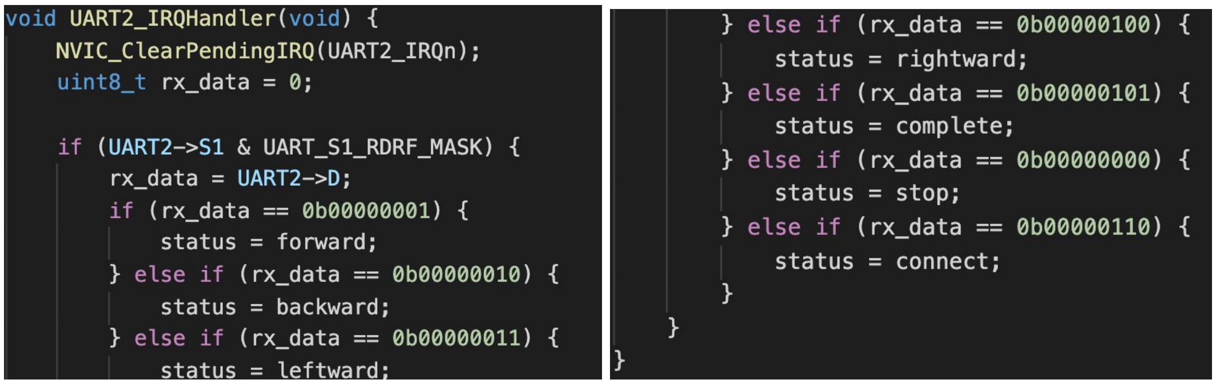


```c
volatile int status = 0;
volatile int isConnected = 0;

int musical_notes1[MUSICAL_NOTE_CNT1] = {196, 262, 262, 330, 440, 330, 392, 392, 392, 440, 392, 330, 392, 330, 294, 294, 220, 294, 294,
                                          196, 262, 262, 330, 440, 330, 392, 392, 392, 440, 392, 330, 392, 330, 294, 294, 220, 294, 294,
                                          440, 440, 440, 392, 349, 392, 440, 392, 392, 294, 330, 370, 294, 392, 392, 392, 440, 440, 392,
```

Figure 3 Global Variables

UART2_IRQHandler

The serial data coming from the BT06 device will be captured through the use of UART2_IRQHandler() (Figure 4). According to the serial data that are sent, respective `status` will be updated. When serial data = 0 (i.e. no serial data has been sent), `status` will be updated as stop. When serial data = 1, `status` will be updated as forward. When serial data = 2, `status` will be updated as backward. When serial data = 3, `status` will be updated as leftward. When serial data = 4, `status` will be updated as rightward. When serial data = 5, `status` will be updated as complete. When serial data = 6, `status` will be updated as connect.



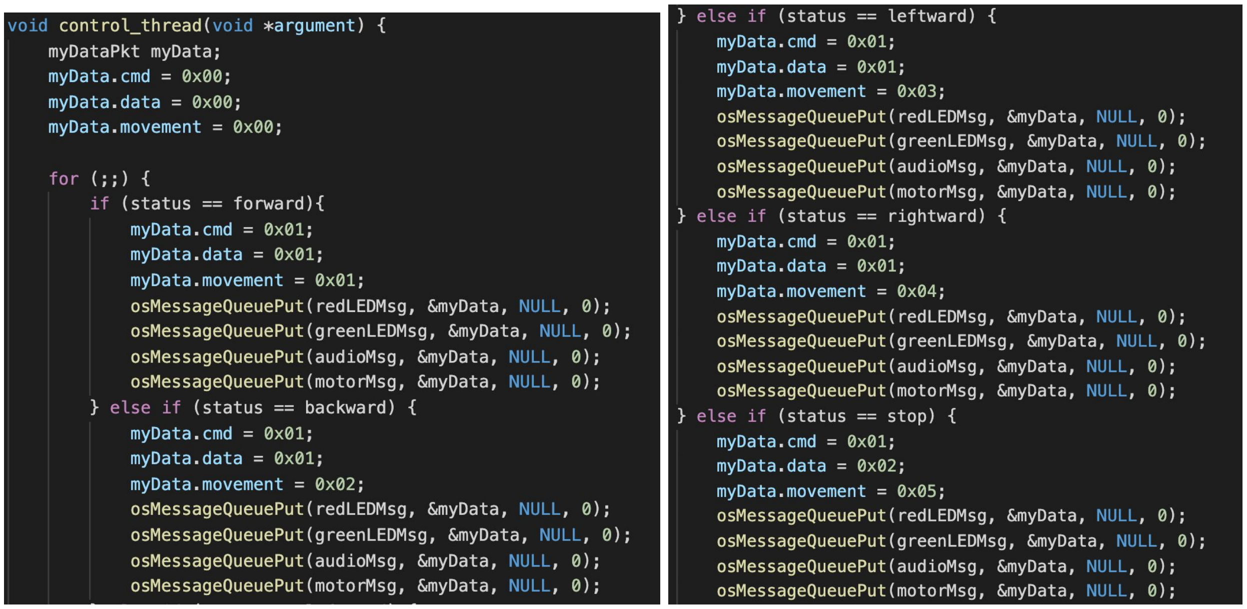
```
void UART2_IRQHandler(void) {
    NVIC_ClearPendingIRQ(UART2_IRQn);
    uint8_t rx_data = 0;

    if (UART2->S1 & UART_S1_RDRF_MASK) {
        rx_data = UART2->D;
        if (rx_data == 0b00000001) {
            status = forward;
        } else if (rx_data == 0b00000010) {
            status = backward;
        } else if (rx_data == 0b00000011) {
            status = leftward;
```

```
        } else if (rx_data == 0b00000100) {
            status = rightward;
        } else if (rx_data == 0b00000101) {
            status = complete;
        } else if (rx_data == 0b00000000) {
            status = stop;
        } else if (rx_data == 0b00000110) {
            status = connect;
        }
    }
}
```

Figure 4 UART2_IRQHandler

<u>Control Thread</u>

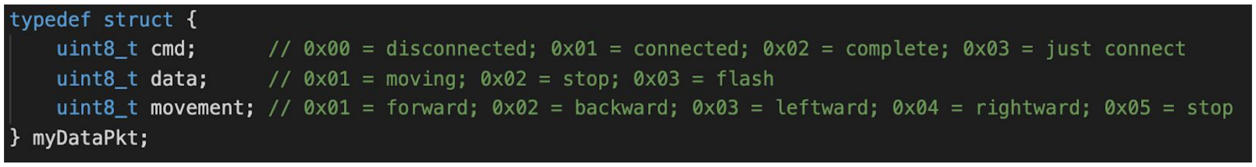
This thread is deemed as the brain of the program. It is used to process the command and send the message correspondingly based on the global variable `status` via the use of message queue (Figure 5). Other threads will receive the message sent and determine whether there is a need to execute corresponding functions or not. A structure is created to contain the message (FIgure 6).



```
void control_thread(void *argument) {
    myDataPkt myData;
    myData.cmd = 0x00;
    myData.data = 0x00;
    myData.movement = 0x00;

    for (;;) {
        if (status == forward){
            myData.cmd = 0x01;
            myData.data = 0x01;
            myData.movement = 0x01;
            osMessageQueuePut(redLEDMsg, &myData, NULL, 0);
            osMessageQueuePut(greenLEDMsg, &myData, NULL, 0);
            osMessageQueuePut(audioMsg, &myData, NULL, 0);
            osMessageQueuePut(motorMsg, &myData, NULL, 0);
        } else if (status == backward) {
            myData.cmd = 0x01;
            myData.data = 0x01;
            myData.movement = 0x02;
            osMessageQueuePut(redLEDMsg, &myData, NULL, 0);
            osMessageQueuePut(greenLEDMsg, &myData, NULL, 0);
            osMessageQueuePut(audioMsg, &myData, NULL, 0);
            osMessageQueuePut(motorMsg, &myData, NULL, 0);
```

```
} else if (status == leftward) {
    myData.cmd = 0x01;
    myData.data = 0x01;
    myData.movement = 0x03;
    osMessageQueuePut(redLEDMsg, &myData, NULL, 0);
    osMessageQueuePut(greenLEDMsg, &myData, NULL, 0);
    osMessageQueuePut(audioMsg, &myData, NULL, 0);
    osMessageQueuePut(motorMsg, &myData, NULL, 0);
} else if (status == rightward) {
    myData.cmd = 0x01;
    myData.data = 0x01;
    myData.movement = 0x04;
    osMessageQueuePut(redLEDMsg, &myData, NULL, 0);
    osMessageQueuePut(greenLEDMsg, &myData, NULL, 0);
    osMessageQueuePut(audioMsg, &myData, NULL, 0);
    osMessageQueuePut(motorMsg, &myData, NULL, 0);
} else if (status == stop) {
    myData.cmd = 0x01;
    myData.data = 0x02;
    myData.movement = 0x05;
    osMessageQueuePut(redLEDMsg, &myData, NULL, 0);
    osMessageQueuePut(greenLEDMsg, &myData, NULL, 0);
    osMessageQueuePut(audioMsg, &myData, NULL, 0);
    osMessageQueuePut(motorMsg, &myData, NULL, 0);
```

Figure 5 Part of Control Thread

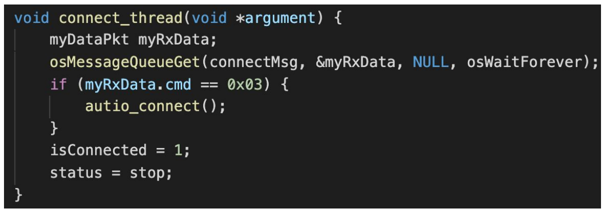


```
typedef struct {
    uint8_t cmd;      // 0x00 = disconnected; 0x01 = connected; 0x02 = complete; 0x03 = just connect
    uint8_t data;     // 0x01 = moving; 0x02 = stop; 0x03 = flash
    uint8_t movement; // 0x01 = forward; 0x02 = backward; 0x03 = leftward; 0x04 = rightward; 0x05 = stop
} myDataPkt;
```

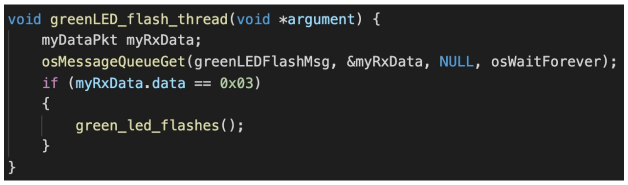
Figure 6 Structure of Data Package

<u>Connect Thread & Green LED Flash Thread</u>

When the message received matches the expected value, the connect_thread() will call the function audio_connect() to display the unique tone sequence to indicate successful connection, and set the value of the global variable `isConnected` to 0 to indicate successful connection (Figure 7). Moreover, the greenLED_flash_thread() will call the function green_led_flashes() to make all green LEDs flash two times (Figure 8).



```
void connect_thread(void *argument) {
    myDataPkt myRxData;
    osMessageQueueGet(connectMsg, &myRxData, NULL, osWaitForever);
    if (myRxData.cmd == 0x03) {
        autio_connect();
    }
    isConnected = 1;
    status = stop;
}
```



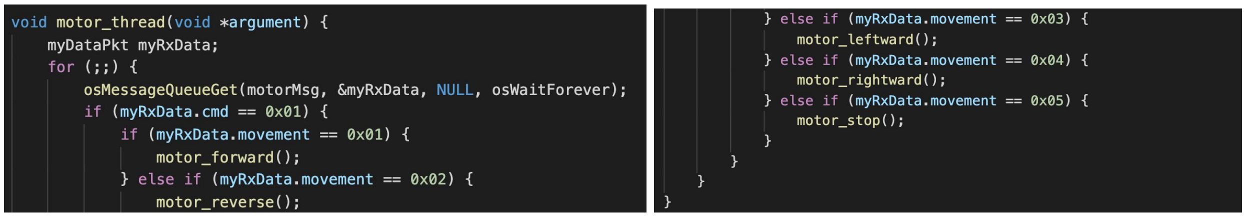
```
void greenLED_flash_thread(void *argument) {
    myDataPkt myRxData;
    osMessageQueueGet(greenLEDFlashMsg, &myRxData, NULL, osWaitForever);
    if (myRxData.data == 0x03)
    {
        green_led_flashes();
    }
}
```

Figure 7 Connect Thread

Figure 8 Green LED Flash Thread

<u>Motor Thread</u>

When the message received matches the expected value, the motor_thread() will call different functions accordingly to control the movement of the robotic car (Figure 9).

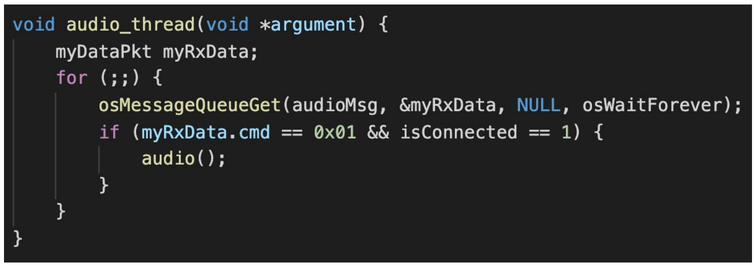


```
void motor_thread(void *argument) {
    myDataPkt myRxData;
    for (;;) {
        osMessageQueueGet(motorMsg, &myRxData, NULL, osWaitForever);
        if (myRxData.cmd == 0x01) {
            if (myRxData.movement == 0x01) {
                motor_forward();
            } else if (myRxData.movement == 0x02) {
                motor_reverse();
            } else if (myRxData.movement == 0x03) {
                motor_leftward();
            } else if (myRxData.movement == 0x04) {
                motor_rightward();
            } else if (myRxData.movement == 0x05) {
                motor_stop();
            }
        }
    }
}
```

Figure 9 Motor Thread

<u>Audio Thread</u>

When the message received matches the expected value, the audio_thread() will call the function audio() to display the Doraemon theme song (Figure 10).

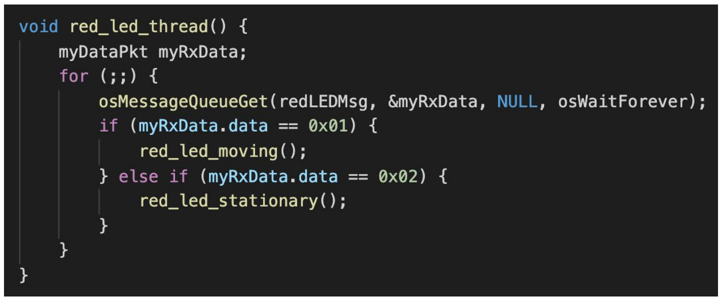


```
void audio_thread(void *argument) {
    myDataPkt myRxData;
    for (;;) {
        osMessageQueueGet(audioMsg, &myRxData, NULL, osWaitForever);
        if (myRxData.cmd == 0x01 && isConnected == 1) {
            audio();
        }
    }
}
```

Figure 10 Audio Thread

<u>Red LED Thread</u>

When the message received matches the expected value, the red_led_thread() will call different functions accordingly to control the performance of red LEDs (Figure 11).

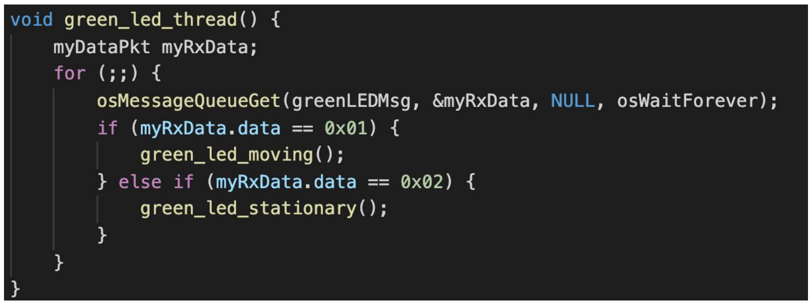


```
void red_led_thread() {
    myDataPkt myRxData;
    for (;;) {
        osMessageQueueGet(redLEDMsg, &myRxData, NULL, osWaitForever);
        if (myRxData.data == 0x01) {
            red_led_moving();
        } else if (myRxData.data == 0x02) {
            red_led_stationary();
        }
    }
}
```

Figure 11 Red LED Thread

<u>Green LED Thread</u>

When the message received matches the expected value, the green_led_thread() will call different functions accordingly to control the performance of green LEDs (Figure 12).

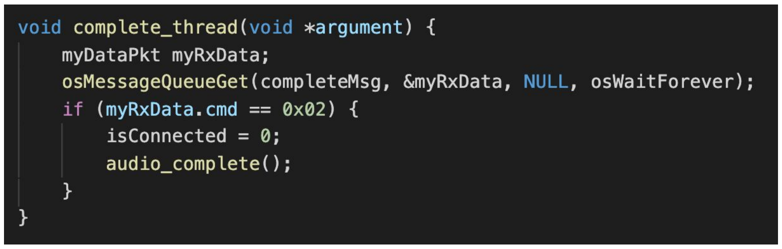


```
void green_led_thread() {
    myDataPkt myRxData;
    for (;;) {
        osMessageQueueGet(greenLEDMsg, &myRxData, NULL, osWaitForever);
        if (myRxData.data == 0x01) {
            green_led_moving();
        } else if (myRxData.data == 0x02) {
            green_led_stationary();
        }
    }
}
```

Figure 12 Green LED Thread

<u>Complete Thread</u>

When the message received matches the expected value, the complete_thread() will call the function audio_complete() to display the unique tone sequence to indicate that the challenge run is completed, and set the value of the global variable `isConnected` to be 0 to indicate that the challenge run is completed (Figure 13).



```
void complete_thread(void *argument) {
    myDataPkt myRxData;
    osMessageQueueGet(completeMsg, &myRxData, NULL, osWaitForever);
    if (myRxData.cmd == 0x02) {
        isConnected = 0;
        audio_complete();
    }
}
```

Figure 13 Complete Thread