Group 40

Chan Xu Hui A0199793B, Joel Ng Yi Xian A0206134U, Joseph Abraham A0199472M

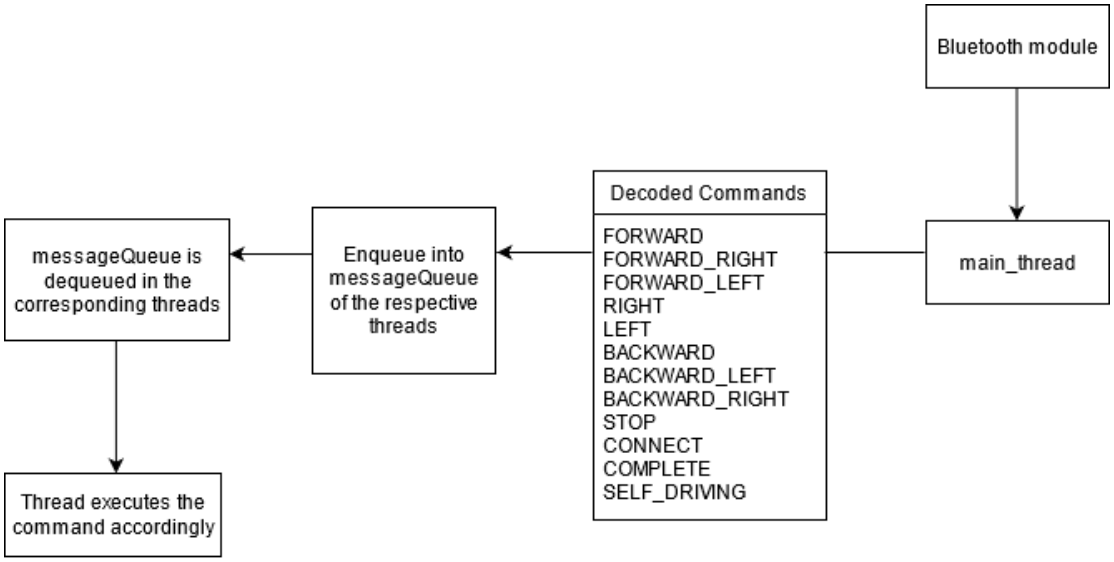
RTOS Architecture Report

Introduction

This project requires us to design, develop and build a robotic car that runs based on a Real Time Operating System (RTOS). The project has certain key requirements, which can be summarised in the following:

1) Using an android application to control the robot's movements and modes, of which there are 2 (remote-controlled, self-driving). This is accomplished with UART Serial Communications through a Bluetooth connection between the robot's Bluetooth module and the android phone's in-built Bluetooth module.
2) Having the buzzer, front and back LEDs change their behaviour in accordance to the robot's movements and modes.

RTOS Architecture



To start off, our RTOS receives data from the Bluetooth module sent from a compatible device. This data is then stored in our custom-made queue *rx_q*.

We have a total of 6 threads:

1. *main_thread:* dequeues the data from *rx_q* throughout its lifetime, then decodes it and enqueues commands using message queues *moveMessage, toneMessage,*

ledRedMessage and *ledGreenMessage* for *move_thread, tone_thread, red_led_thread* and *green_led_thread* respectively.

2. *move_thread:* controls the motor and movement of the robot.
3. *tone_thread:* plays the correct music according to the robot's status – *my_r_state*.
4. *red_led_thread:* controls the logic of the red LEDs
5. *green_led_thread:* controls the logic of the green LEDs
6. *ultrasonic_thread:* controls the ultrasonic sensor used in our self-driving mode. It is in charge of dispatching trigger pulses for activation of the sensor.

Our architecture minimizes the use of global variables with the use of messages to communicate and synchronize between threads.

Remote-Controlled Mode

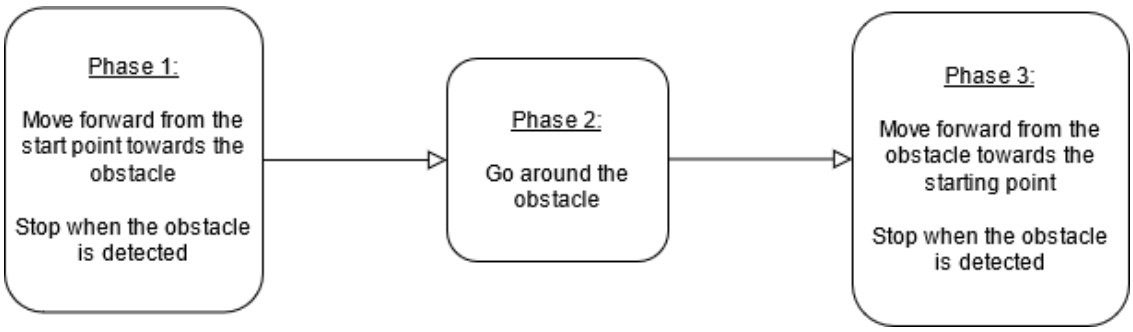
The robot is by default, in the remote-controlled mode. Upon receiving data from the Bluetooth module, *main_thread* decodes the command using switch statements and can parse the following commands: CONNECT, FORWARD, LEFT, RIGHT, BACKWARD, FORWARD_LEFT, FORWARD_RIGHT, BACKWARD_LEFT, BACKWARD_RIGHT, COMPLETE and places the respective instructions into the message queues of *move_thread, tone_thread, red_led_thread* and *green_led_thread*.

Self-Driving Mode

Upon receiving the SELF_DRIVE command, *main_thread* places the appropriate instruction into the message queue of *move_thread*, where *move_thread* takes over the self-driving process.

The robot will undergo change in the following states: the robot's state *r_state*, ultrasonic *us_state*, self-driving *sd_state*.

The self-driving challenge is divided into 3 phases:



In this mode, *move_thread* changes the *sd_state* of the robot from *IDLE* to *PHASE_ONE*, signifying the start of the robot's self-driving process, and dispatches commands to *tone_thread, led_red_thread, led_green_thread* for every movement change in each phase.

The *ultrasonic_thread* monitors the *sd_state* of the robot and activates the ultrasonic sensor whenever *sd_state* is in *PHASE_ONE* or *PHASE_THREE*. Upon sending the trigger signal, *us_state* changes from *ECHO_IDLE* to *ECHO_TRIGGERED*.

To measure the length of the echo signal received, we used the *TPM0* module, as well as 5 global variables: *timerCounter, countDistance, startTime, endTime* and *lastMod*.

We use *PORTA_IRQHandler* to detect the echo signal of the ultrasonic sensor. Upon detection of the rising edge of the echo signal, *us_state* changes from *ECHO_TRIGGERED* to *ECHO_MEASURE*, and the *TPM0_CNT* value is captured into *startTime*, As our *TPM0* module is shared with our buzzer PWM, *TPM0_CNT* is controlled by the *mod* value of *tone_thread*. Hence, to obtain the right measurements, we have to continuously monitor *TPM0_MOD* value before the overflow of *TPM0_CNT*.

During the measurement of the echo signal, *TPM0_IRQHandler* monitors for any timer counter overflows. If there are any overflows detected when *us_state* is in *ECHO_MEASURE* state, we add the difference between *lastMod* and *startTime* to *timerCounter*, and resets *startTime* and *lastMod* to the new *TPM0_CNT* and *TPM0_MOD* values respectively.

Upon the detection of the falling edge of the echo signal, *us_state* changes from *ECHO_MEASURE* to *ECHO_FINISHED*, and the *TPM0_CNT* value is captured into *endTime*. The difference between *endTime* and *startTime* is added into the *timerCounter*. *PORTA_IRQHandler* will then calculate the distance detected by the ultrasonic sensor, and if the distance detected is less than 30cm, a message will be sent to the *selfDriveMessage* queue monitored by *move_thread* to signal the detection of an obstacle.

Upon receiving the message from *PORTA_IRQHandler*, *move_thread* changes the *sd_state* from *PHASE_ONE* to *PHASE_TWO*, and executes the go-around procedure.

With the completion of the go-around procedure, *move_thread* changes the *sd_state* from *PHASE_TWO* to *PHASE_THREE*, activating the ultrasonic sensor to detect the final obstacle for completion of the challenge.

Throughout the challenge, *r_state* of the robot will change according to the movement commands issued by *move_thread*, as per the require

Upon completion, *sd_state* will be changed from *PHASE_THREE* back to *IDLE* to prevent the triggering of the ultrasonic sensor, and the robot is now ready for the self-driving mode to be called again.

Hardware Design

The main hardware components in our robot include: 1 FRDM-KL25Z board, 4 DC gear motors powering a wheel each, 2 DVR8833 Dual Motor Driver Carriers, 1 BT06 Module, 1 7805 Regulator, 1 Buzzer and 1 Ultrasonic sensor. In addition, many LEDs, resistors and power sources are used.

We organized our robot into 2 layers: the top and bottom layers. The bottom layer holds the main hardware components. The top layer holds the power sources, which include 2 battery holders and 1 powerbank. It also provides some protection to the wiring and connections in the bottom layer, should the robot topple.

In addition, we taped bits of rubber bands to our wheels with duct tape to improve traction, so as to attain more consistency in the robot's turns (which is especially important for calibrating the robots' movements in preparation for the self-driving challenge) and also to aid the robot in moving up ramps.