CIE 408 | Stage 3 Report
Autonomous Robot

BY:

Enji Wael 202101008
Hala Mohamed 202100987
Mai Mohamed 202000746
Yousef Said 202100637
Ahmed Farid 202000625
Abdulrhman Ewiah 201800747

Table of Contents:

I. Introduction:

Autonomous robots are engineered to operate across different environments without requiring constant human input. These systems use a range of advanced sensors and actuators to function independently, gathering data from their surroundings and responding accordingly. The collected sensor data is processed by an embedded system powered by a Real-Time Operating System (RTOS), allowing the robot to react swiftly and accurately to environmental changes.

This rapid response capability makes autonomous robots essential in scenarios where immediate decision-making is crucial. In the realm of robotics, Artificial Neural Networks (ANNs) combined with Tiva C Series microcontrollers offer a powerful technological foundation for enhancing autonomous navigation. This report explores key research studies that demonstrate the application of ANNs in robotic systems, emphasizing how the Tiva C microcontrollers contribute to real-time data processing and efficient control.

II. Literature Review:

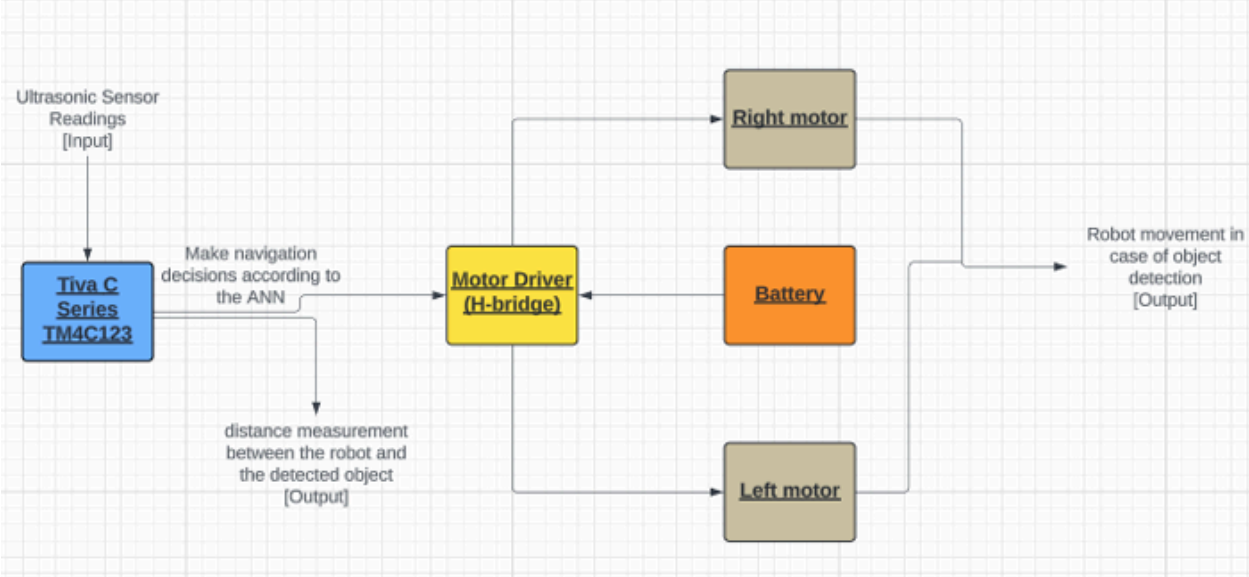
Artificial Neural Networks (ANNs) have significantly transformed how autonomous robots interpret data and make navigation decisions. The study titled *"Autonomous Robot Control by Neural Networks"* highlights the role of ANNs in enabling robots to perform actions with greater precision and adaptability. It compares ANN-based systems to earlier methods like fuzzy logic, showing that neural networks are more efficient in processing real-time sensor inputs. The researchers tested their approach using a robot platform controlled via an Android-based remote system, successfully demonstrating the ANN's effectiveness in navigation and obstacle avoidance.

Another study, *"An Autonomous Robot: Using ANN to Navigate in a Static Path,"* focuses on robotic navigation in environments filled with fixed obstacles. It emphasizes training the ANN using sensory input, allowing the robot to recognize and respond to barriers. The training, performed on an Arduino platform, proved that ANNs significantly improve autonomous path-following while minimizing collisions.

The paper *"Development of Embedded Devices in Real-Time Autonomous Robots"* discusses the importance of embedded systems in supporting real-time robotic perception. The authors introduced YARTOS, a real-time microkernel running on a Motorola Coldfire microcontroller, which efficiently schedules and manages real-time tasks. Benchmark tests validated its performance in handling complex robotic operations.

Lastly, the study *"Autonomous Navigation of Mobile Robots in Factory Environment"* explores robot navigation in unstructured factory settings. To keep costs low and efficiency high, it uses minimal sensors alongside neural networks, SLAM (Simultaneous Localization and Mapping), and visual processing. These technologies enable the robot to identify markers and avoid obstacles autonomously, illustrating the growing integration of AI in industrial robotics.

III. Functional Block Diagram:



The system's block diagram is organized into several functional components:

1. **Ultrasonic Sensor [Input]:**
   The ultrasonic sensor collects distance measurements from the robot's surroundings. These readings are sent to the Tiva C Series TM4C123 microcontroller for further processing.

2. **Tiva C Series TM4C123 [Microcontroller]:**
   This microcontroller serves as the brain of the robot, processing the incoming data using an Artificial Neural Network

(ANN). When an object is detected, it calculates the distance and makes navigation decisions accordingly. It utilizes FreeRTOS to handle multitasking and ensure the robot responds in real-time.

3. **Dual H-Bridge Motor Driver:**
   This component manages the speed and direction of the robot's DC motors using PWM (Pulse Width Modulation). By adjusting the PWM duty cycle, it effectively regulates the current to the motors, allowing precise control of movement.

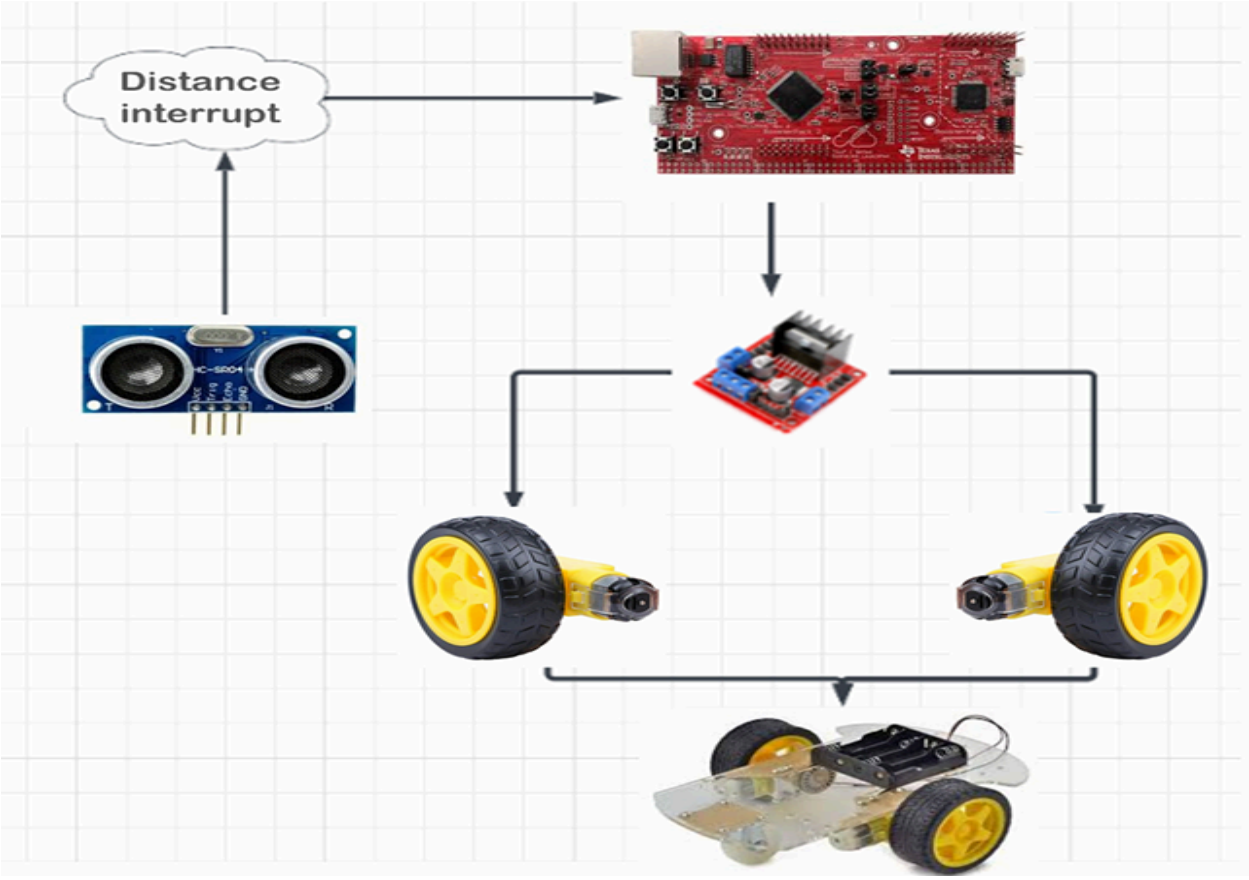
4. **DC Motors:**
   These motors receive control signals from the H-Bridge and adjust their speed and rotation direction. Based on sensor input, they drive the robot forward or redirect it to avoid obstacles.

5. **Battery:**
   The battery supplies power to the H-Bridge module and, by extension, to the entire motor system.

IV. SchematicDiagram:

The project operates based on the flow shown in the schematic diagram.

First, the ultrasonic sensor continuously measures the distance between the robot and nearby objects. This distance data is fed into the Artificial Neural Network (ANN) running on the Tiva C microcontroller. Based on this input, the ANN determines whether the robot should change its direction. The microcontroller then sends this decision to the H-bridge driver, which controls the wheels accordingly—either maintaining the current path or adjusting it to avoid obstacles.



V. ComponentsList:

Below is a summary of the hardware components used in the project, including their quantities, prices, and sources:

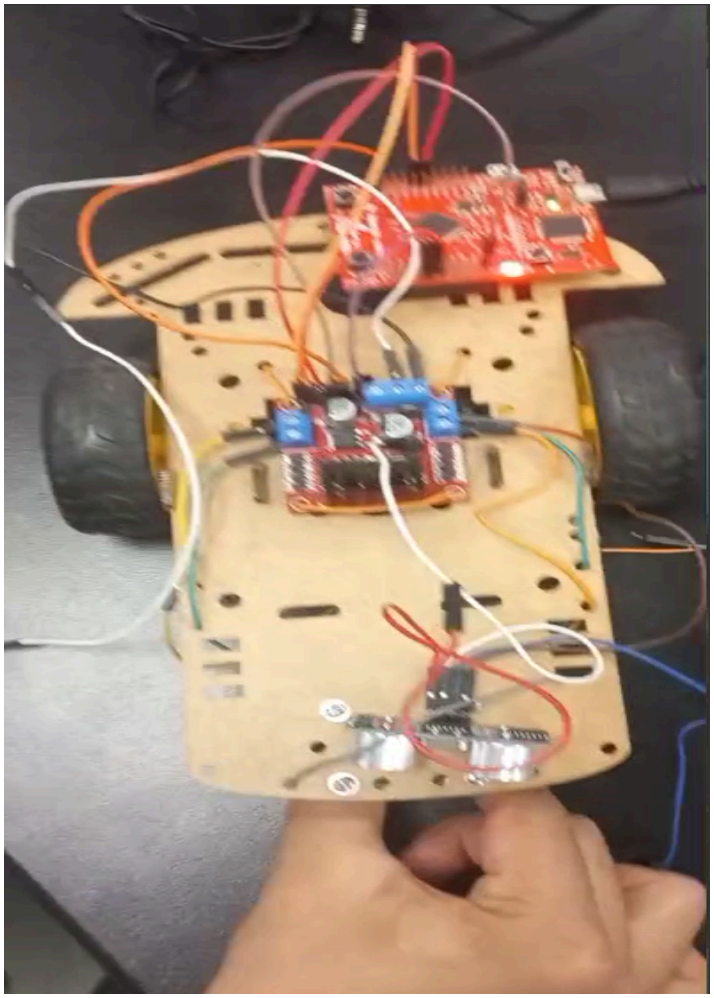
No.	Component	Quantity	Price (EGP)	Link
1	Tiva C Series TM4C123G LaunchPad	1	3500.00	Link
2	2-Wheel Drive Robot Car Chassis Kit	1	355.00	Link

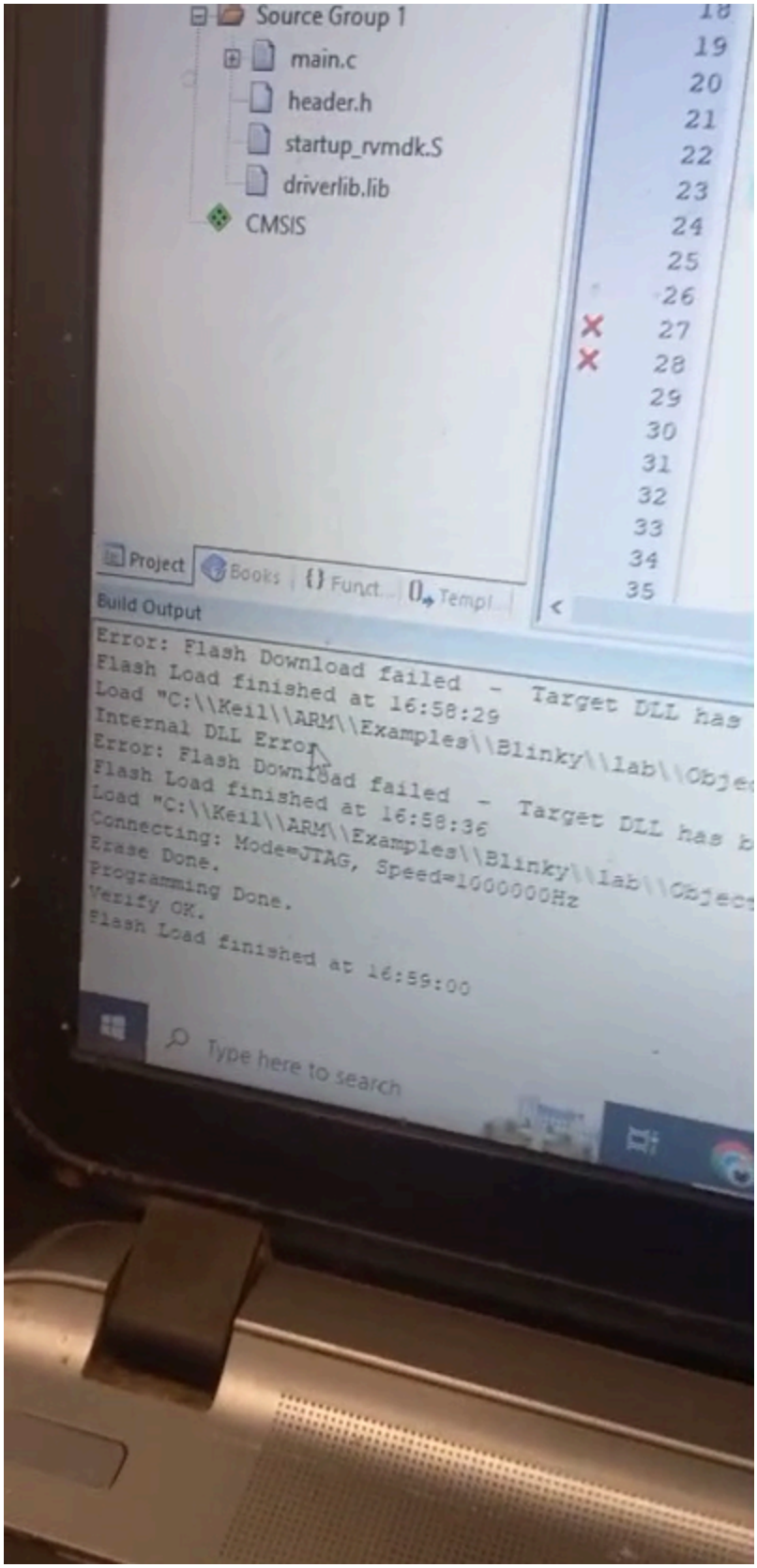
3	Dual H-Bridge Motor Driver (L298N Module)	1	95.00	Link
4	Ultrasonic Distance Sensor Module	1	45.00	Link
5	3.7V Lithium-Ion Battery (1200 mAh)	3	180.00	Link
6	Jumper Wires (Male-to-Female, 10cm)	20	20.00	Link
7	Breadboard (830 tie-points)	1	40.00	Link

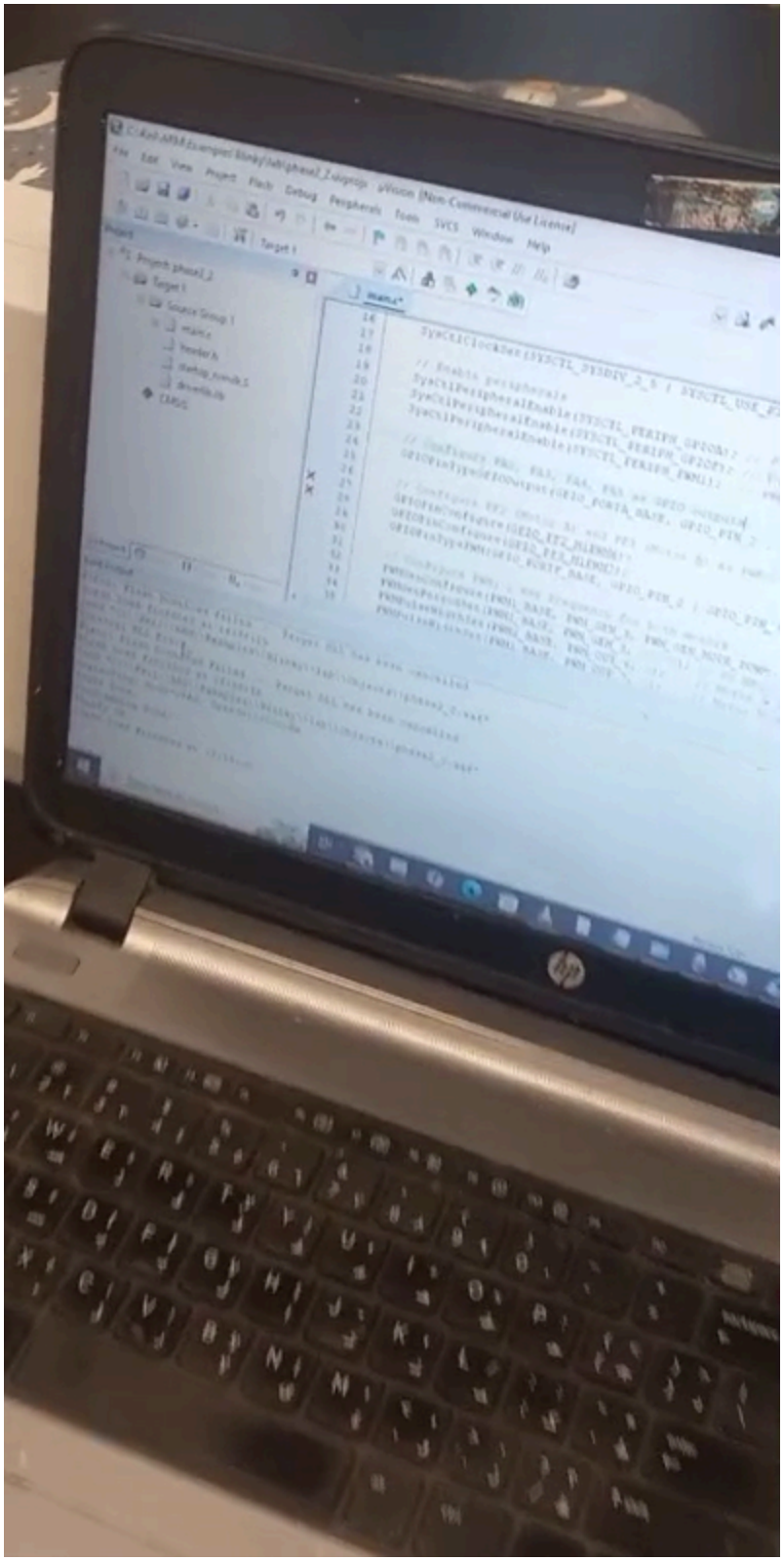
Total Cost: 4235.00 EGP

VI. AssembleHardware:

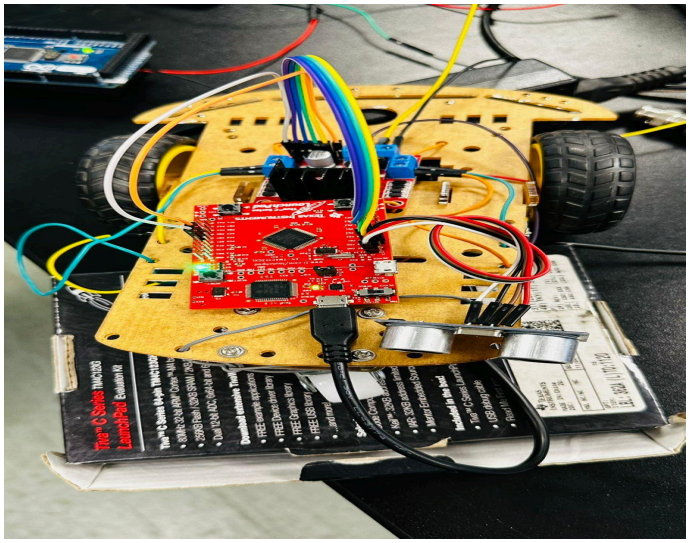
In this phase, we assembled the hardware components and developed the corresponding code for each module. The functionality of the motion sensor, motor control via the H-Bridge, and the ultrasonic sensor were all successfully implemented using the Tiva C microcontroller. The system's responses—such as LED indicators and motor movements—are clearly demonstrated in the attached videos, which provide a full visual explanation of each stage.

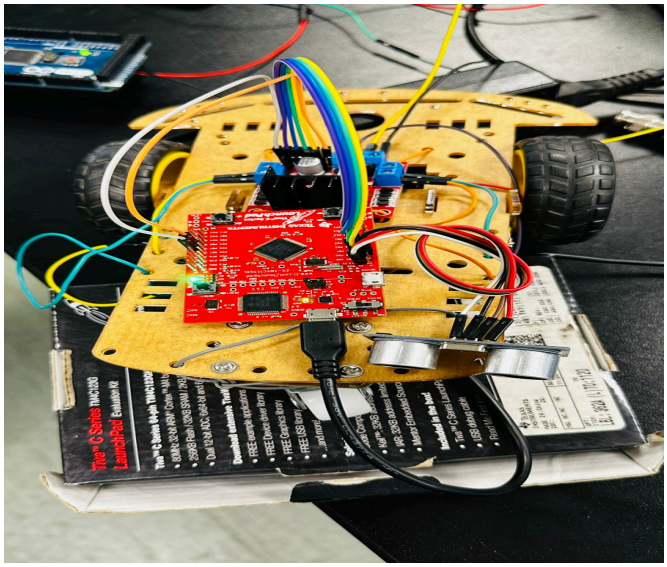



Source Group 1
main.c
header.h
startup_rvmdk.S
driverlib.lib
CMSIS
Project
Books
Funct...
Templ...
Build Output
Error: Flash Download failed - Target DLL has
Flash Load finished at 16:58:29
Load "C:\\Keil\\ARM\\Examples\\Blinky\\lab\\Obje
Internal DLL Error
Error: Flash Download failed - Target DLL has b
Flash Load finished at 16:58:36
Load "C:\\Keil\\ARM\\Examples\\Blinky\\lab\\Objec
Connecting: Mode=JTAG, Speed=1000000Hz
Erase Done.
Programming Done.
Verify OK.
Flash Load finished at 16:59:00
Type here to search



VII.Complete Hardware Implementation:





Link:

https://drive.google.com/file/d/1iwI26LoKNjAuSTRJF3CPtsMQFYmkD8sQ/view?usp=sharing

Code

```
#define PART_TM4C123GH6PM

#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <stdarg.h>

#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "inc/hw_pwm.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/timer.h"
#include "driverlib/pwm.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"

#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"

#define STACK_UART_TASK     (configMINIMAL_STACK_SIZE * 3)
#define STACK_DECISION_TASK (configMINIMAL_STACK_SIZE * 2)
#define STACK_DEFAULT       (configMINIMAL_STACK_SIZE * 2)

#define QUEUE_LEN_SENSOR    1
#define QUEUE_LEN_MOTOR     1
#define QUEUE_LEN_LED       1
#define QUEUE_LEN_LOG       10
#define MAX_LOG_MSG_LEN     80

QueueHandle_t xSensorDataQueue;
QueueHandle_t xMotorCommandQueue;
QueueHandle_t xLedCommandQueue;
QueueHandle_t xLogQueue;

#define FRONT_TRIG_PORT   GPIO_PORTB_BASE
#define FRONT_TRIG_PIN    GPIO_PIN_0
#define FRONT_ECHO_PORT   GPIO_PORTB_BASE
```

```
#define FRONT_ECHO_PIN    GPIO_PIN_1

#define LEFT_TRIG_PORT    GPIO_PORTE_BASE // EXAMPLE - REPLACE
#define LEFT_TRIG_PIN     GPIO_PIN_0      // EXAMPLE - REPLACE
#define LEFT_ECHO_PORT    GPIO_PORTE_BASE // EXAMPLE - REPLACE
#define LEFT_ECHO_PIN     GPIO_PIN_1      // EXAMPLE - REPLACE

#define RIGHT_TRIG_PORT   GPIO_PORTE_BASE // EXAMPLE - REPLACE
#define RIGHT_TRIG_PIN    GPIO_PIN_2      // EXAMPLE - REPLACE
#define RIGHT_ECHO_PORT   GPIO_PORTE_BASE // EXAMPLE - REPLACE
#define RIGHT_ECHO_PIN    GPIO_PIN_3      // EXAMPLE - REPLACE

#define ECHO_TIMER_PERIPH SYSCTL_PERIPH_TIMER2
#define ECHO_TIMER_BASE   TIMER2_BASE
#define ECHO_TIMER        TIMER_A

#define TIMEOUT_VAL       0xFFFFFFFFU

#define DIST_STOP_FRONT   200U
#define DIST_SLOW_FRONT   500U
#define DIST_TURN_SIDE    300U

#define EN_PORT          GPIO_PORTB_BASE
#define ENA_PIN          GPIO_PIN_6
#define ENB_PIN          GPIO_PIN_7
#define DIR_PORT         GPIO_PORTD_BASE
#define IN1_PIN          GPIO_PIN_0
#define IN2_PIN          GPIO_PIN_1
#define IN3_PIN          GPIO_PIN_2
#define IN4_PIN          GPIO_PIN_3
#define PWM_FREQ_HZ       5000U
#define PWM_MAX_DUTY      100
static uint32_t pwm_period;

#define SPEED_STOP        0
#define SPEED_SLOW        40
#define SPEED_NORMAL      70
#define SPEED_TURN        50

typedef enum {
    LED_STATE_OFF,
    LED_STATE_STOP,
    LED_STATE_SLOW,
    LED_STATE_NORMAL,
```

```
    LED_STATE_TURN_L,
    LED_STATE_TURN_R,
    LED_STATE_TIMEOUT
} LedState_t;

#define LED_PIN_RED       GPIO_PIN_1
#define LED_PIN_GREEN     GPIO_PIN_3
#define LED_PIN_BLUE      GPIO_PIN_2
#define LED_ALL_PINS      (LED_PIN_RED | LED_PIN_GREEN | LED_PIN_BLUE)

typedef struct {
    uint32_t front_dist;
    uint32_t left_dist;
    uint32_t right_dist;
} SensorData_t;

typedef struct {
    int16_t left_speed;
    int16_t right_speed;
} MotorCommand_t;

typedef char LogMessage_t[MAX_LOG_MSG_LEN];


static void InitConsole(void) {
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    UARTStdioConfig(0, 115200, SysCtlClockGet());
}

static void InitLED(void) {
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, LED_ALL_PINS);
    GPIOPinWrite(GPIO_PORTF_BASE, LED_ALL_PINS, 0);
}

static void InitUltrasonic(void) {
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
```

```
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE); // EXAMPLE - CHANGE IF NEEDED

    GPIOPinTypeGPIOOutput(FRONT_TRIG_PORT, FRONT_TRIG_PIN);
    GPIOPinWrite(FRONT_TRIG_PORT, FRONT_TRIG_PIN, 0);
    GPIOPinTypeGPIOOutput(LEFT_TRIG_PORT, LEFT_TRIG_PIN);
    GPIOPinWrite(LEFT_TRIG_PORT, LEFT_TRIG_PIN, 0);
    GPIOPinTypeGPIOOutput(RIGHT_TRIG_PORT, RIGHT_TRIG_PIN);
    GPIOPinWrite(RIGHT_TRIG_PORT, RIGHT_TRIG_PIN, 0);

    GPIOPinTypeGPIOInput(FRONT_ECHO_PORT, FRONT_ECHO_PIN);
    GPIOPinTypeGPIOInput(LEFT_ECHO_PORT, LEFT_ECHO_PIN);
    GPIOPinTypeGPIOInput(RIGHT_ECHO_PORT, RIGHT_ECHO_PIN);

    SysCtlPeripheralEnable(ECHO_TIMER_PERIPH);
    TimerConfigure(ECHO_TIMER_BASE, TIMER_CFG_PERIODIC);
    TimerLoadSet(ECHO_TIMER_BASE, ECHO_TIMER, 0xFFFFFFFF);
    TimerEnable(ECHO_TIMER_BASE, ECHO_TIMER);
}

static void InitMotors(void) {
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);

    GPIOPinTypeGPIOOutput(DIR_PORT, IN1_PIN | IN2_PIN | IN3_PIN | IN4_PIN);
    GPIOPinWrite(DIR_PORT, IN1_PIN | IN2_PIN | IN3_PIN | IN4_PIN, 0);

    GPIOPinConfigure(GPIO_PB6_M0PWM0);
    GPIOPinConfigure(GPIO_PB7_M0PWM1);
    GPIOPinTypePWM(EN_PORT, ENA_PIN | ENB_PIN);

    SysCtlPWMClockSet(SYSCTL_PWMDIV_8);
    pwm_period = (SysCtlClockGet() / 8) / PWM_FREQ_HZ;

    PWMGenConfigure(PWM0_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN |
PWM_GEN_MODE_NO_SYNC);
    PWMGenPeriodSet(PWM0_BASE, PWM_GEN_0, pwm_period);

    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0, 0);
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1, 0);

    PWMOutputState(PWM0_BASE, PWM_OUT_0_BIT | PWM_OUT_1_BIT, true);
    PWMGenEnable(PWM0_BASE, PWM_GEN_0);
}
```

```
void Log(const char* format, ...) {
    LogMessage_t msg;
    va_list args;
    va_start(args, format);
    vsnprintf(msg, MAX_LOG_MSG_LEN, format, args);
    va_end(args);
    msg[MAX_LOG_MSG_LEN - 1] = '\0';

    xQueueSend(xLogQueue, &msg, pdMS_TO_TICKS(10));
}


static uint32_t MeasureDistance10thCm(uint32_t ui32TrigPort, uint8_t ui8TrigPin,
                                      uint32_t ui32EchoPort, uint8_t ui8EchoPin)
{
    const uint32_t ticks_per_us = SysCtlClockGet() / 1000000;
    uint32_t start, end, elapsed_ticks;
    volatile uint32_t timeout_counter;

    GPIOPinWrite(ui32TrigPort, ui8TrigPin, ui8TrigPin);
    SysCtlDelay((SysCtlClockGet() / 3000000) * 10);
    GPIOPinWrite(ui32TrigPort, ui8TrigPin, 0);

    timeout_counter = (SysCtlClockGet() / 3) / 20;
    while (GPIOPinRead(ui32EchoPort, ui8EchoPin) == 0) {
        if (timeout_counter-- == 0) {
            return TIMEOUT_VAL;
        }
    }

    start = TimerValueGet(ECHO_TIMER_BASE, ECHO_TIMER);

    timeout_counter = (SysCtlClockGet() / 3) / 20;
    while (GPIOPinRead(ui32EchoPort, ui8EchoPin) != 0) {
         if (timeout_counter-- == 0) {
            return TIMEOUT_VAL;
        }
    }
    end = TimerValueGet(ECHO_TIMER_BASE, ECHO_TIMER);

    if (end < start) {
        elapsed_ticks = start - end;
```

```
    } else {
        elapsed_ticks = (0xFFFFFFFF - end) + start;
    }

    uint32_t elapsed_us = elapsed_ticks / ticks_per_us;

    return ((elapsed_us * 10) + 29) / 58;
}


static inline uint32_t duty2cmp(uint32_t duty) {
    if (duty > 100) duty = 100;
    return (duty >= 100) ? (pwm_period - 1) : (pwm_period * duty) / 100;
}

static void SetMotors(int16_t left, int16_t right) {
    const int16_t left_trim = 0;
    const int16_t right_trim = 0;

    if (left != 0) left += (left > 0) ? left_trim : -left_trim;
    if (right != 0) right += (right > 0) ? right_trim : -right_trim;

    if (left > 100) left = 100;
    if (left < -100) left = -100;
    if (right > 100) right = 100;
    if (right < -100) right = -100;

    uint32_t cmpL, cmpR;
    uint8_t dirL_pins = 0;
    uint8_t dirR_pins = 0;

    if (left == 0) {
        dirL_pins = 0;
        cmpL = 0;
    } else if (left > 0) {
        dirL_pins = IN1_PIN;
        cmpL = duty2cmp((uint32_t)left);
    } else {
        dirL_pins = IN2_PIN;
        cmpL = duty2cmp((uint32_t)(-left));
    }
    GPIOPinWrite(DIR_PORT, IN1_PIN | IN2_PIN, dirL_pins);
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0, cmpL);
```

```
    if (right == 0) {
        dirR_pins = 0;
        cmpR = 0;
    } else if (right > 0) {
        dirR_pins = IN3_PIN;
        cmpR = duty2cmp((uint32_t)right);
    } else {
        dirR_pins = IN4_PIN;
        cmpR = duty2cmp((uint32_t)(-right));
    }
    GPIOPinWrite(DIR_PORT, IN3_PIN | IN4_PIN, dirR_pins);
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1, cmpR);
}


void vSensorReadTask(void *pvParameters) {
    SensorData_t current_data;
    TickType_t xLastWakeTime;
    const TickType_t xFrequency = pdMS_TO_TICKS(200);

    xLastWakeTime = xTaskGetTickCount();

    while (1) {
        vTaskDelayUntil(&xLastWakeTime, xFrequency);

        current_data.front_dist = MeasureDistance10thCm(FRONT_TRIG_PORT,
FRONT_TRIG_PIN, FRONT_ECHO_PORT, FRONT_ECHO_PIN);
        vTaskDelay(pdMS_TO_TICKS(10));
        current_data.left_dist  = MeasureDistance10thCm(LEFT_TRIG_PORT, LEFT_TRIG_PIN,
LEFT_ECHO_PORT, LEFT_ECHO_PIN);
         vTaskDelay(pdMS_TO_TICKS(10));
        current_data.right_dist = MeasureDistance10thCm(RIGHT_TRIG_PORT,
RIGHT_TRIG_PIN, RIGHT_ECHO_PORT, RIGHT_ECHO_PIN);

        xQueueOverwrite(xSensorDataQueue, &current_data);
    }
}

void vDecisionTask(void *pvParameters) {
    SensorData_t sensed_distances;
    MotorCommand_t motor_cmd;
    LedState_t led_cmd;

    while (1) {
```

```
      if (xQueueReceive(xSensorDataQueue, &sensed_distances, portMAX_DELAY) ==
pdPASS)
      {
         motor_cmd.left_speed = SPEED_NORMAL;
         motor_cmd.right_speed = SPEED_NORMAL;
         led_cmd = LED_STATE_NORMAL;

         if (sensed_distances.front_dist == TIMEOUT_VAL ||
            sensed_distances.left_dist  == TIMEOUT_VAL ||
            sensed_distances.right_dist == TIMEOUT_VAL)
         {
            motor_cmd.left_speed = SPEED_STOP;
            motor_cmd.right_speed = SPEED_STOP;
            led_cmd = LED_STATE_TIMEOUT;
            Log("Sensor Timeout! F:%u L:%u R:%u\r\n", sensed_distances.front_dist,
sensed_distances.left_dist, sensed_distances.right_dist);
         }
         else if (sensed_distances.front_dist < DIST_STOP_FRONT)
         {
             Log("RED ZONE F:%u.%u L:%u.%u R:%u.%u - Trying Turn\r\n",
                sensed_distances.front_dist/10, sensed_distances.front_dist%10,
                sensed_distances.left_dist/10, sensed_distances.left_dist%10,
                sensed_distances.right_dist/10, sensed_distances.right_dist%10);

            if (sensed_distances.left_dist > sensed_distances.right_dist &&
sensed_distances.left_dist > DIST_TURN_SIDE) {
               motor_cmd.left_speed = -SPEED_TURN;
               motor_cmd.right_speed = SPEED_TURN;
               led_cmd = LED_STATE_TURN_L;
               Log("  Turning LEFT\r\n");
            } else if (sensed_distances.right_dist > DIST_TURN_SIDE) {
               motor_cmd.left_speed = SPEED_TURN;
               motor_cmd.right_speed = -SPEED_TURN;
               led_cmd = LED_STATE_TURN_R;
               Log("  Turning RIGHT\r\n");
             } else {
               motor_cmd.left_speed = SPEED_STOP;
               motor_cmd.right_speed = SPEED_STOP;
               led_cmd = LED_STATE_STOP;
               Log("  TRAPPED - STOPPING\r\n");
             }
         }
         else if (sensed_distances.front_dist < DIST_SLOW_FRONT)
         {
```

```
            motor_cmd.left_speed = SPEED_SLOW;
            motor_cmd.right_speed = SPEED_SLOW;
            led_cmd = LED_STATE_SLOW;
            Log("GREEN ZONE F:%u.%u L:%u.%u R:%u.%u - SLOW\r\n",
                sensed_distances.front_dist/10, sensed_distances.front_dist%10,
                sensed_distances.left_dist/10, sensed_distances.left_dist%10,
                sensed_distances.right_dist/10, sensed_distances.right_dist%10);
        }
        else
        {
            motor_cmd.left_speed = SPEED_NORMAL;
            motor_cmd.right_speed = SPEED_NORMAL;
            led_cmd = LED_STATE_NORMAL;
             Log("BLUE ZONE F:%u.%u L:%u.%u R:%u.%u - NORMAL\r\n",
                sensed_distances.front_dist/10, sensed_distances.front_dist%10,
                sensed_distances.left_dist/10, sensed_distances.left_dist%10,
                sensed_distances.right_dist/10, sensed_distances.right_dist%10);
        }

        xQueueOverwrite(xMotorCommandQueue, &motor_cmd);
        xQueueOverwrite(xLedCommandQueue, &led_cmd);
      }
   }
}

void vMotorControlTask(void *pvParameters) {
   MotorCommand_t received_cmd;

   while (1) {
      if (xQueueReceive(xMotorCommandQueue, &received_cmd, portMAX_DELAY) ==
pdPASS) {
         SetMotors(received_cmd.left_speed, received_cmd.right_speed);
      }
   }
}

void vLedControlTask(void *pvParameters) {
   LedState_t received_state;
   uint32_t led_pins = 0;
   TickType_t last_toggle_time = 0;
   bool toggle_state = false;

   while (1) {
```

```
      if (xQueueReceive(xLedCommandQueue, &received_state, pdMS_TO_TICKS(250)) ==
pdPASS)
      {
          switch(received_state) {
            case LED_STATE_STOP:      led_pins = LED_PIN_RED; break;
            case LED_STATE_SLOW:      led_pins = LED_PIN_GREEN; break;
            case LED_STATE_NORMAL:    led_pins = LED_PIN_BLUE; break;
            case LED_STATE_TURN_L:    led_pins = LED_PIN_RED | LED_PIN_GREEN; break;
            case LED_STATE_TURN_R:    led_pins = LED_PIN_RED | LED_PIN_BLUE; break;
            case LED_STATE_TIMEOUT:   led_pins = LED_PIN_RED; break;
            case LED_STATE_OFF:
            default:                  led_pins = 0; break;
          }

          if (received_state != LED_STATE_TIMEOUT) {
              GPIOPinWrite(GPIO_PORTF_BASE, LED_ALL_PINS, led_pins);
              last_toggle_time = xTaskGetTickCount();
          }
      }

       if (led_pins == LED_PIN_RED) {
          if ((xTaskGetTickCount() - last_toggle_time) >= pdMS_TO_TICKS(250))
          {
              toggle_state = !toggle_state;
              GPIOPinWrite(GPIO_PORTF_BASE, LED_PIN_RED, toggle_state ? LED_PIN_RED
: 0);
              last_toggle_time = xTaskGetTickCount();
          }
       }
   }
}

void vUartLogTask(void *pvParameters) {
   LogMessage_t msg_to_print;

   while (1) {
      if (xQueueReceive(xLogQueue, &msg_to_print, portMAX_DELAY) == pdPASS) {
         UARTprintf("%s", msg_to_print);
      }
   }
}


int main(void) {
```

```
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ);

    InitConsole();
    InitLED();
    InitUltrasonic();
    InitMotors();

    UARTprintf("\n\n--- FreeRTOS Autonomous Robot ---\r\n");
    UARTprintf("System Clock: %u Hz\r\n", SysCtlClockGet());
    UARTprintf("Initializing FreeRTOS...\r\n");

    xSensorDataQueue = xQueueCreate(QUEUE_LEN_SENSOR, sizeof(SensorData_t));
    xMotorCommandQueue = xQueueCreate(QUEUE_LEN_MOTOR, sizeof(MotorCommand_t));
    xLedCommandQueue = xQueueCreate(QUEUE_LEN_LED, sizeof(LedState_t));
    xLogQueue = xQueueCreate(QUEUE_LEN_LOG, sizeof(LogMessage_t));


    if (xSensorDataQueue == NULL || xMotorCommandQueue == NULL || xLedCommandQueue
== NULL || xLogQueue == NULL) {
        UARTprintf("FATAL: Failed to create queues!\r\n");
        while(1);
    }

    BaseType_t status;

    status = xTaskCreate(vSensorReadTask, "SensorTask", STACK_DEFAULT, NULL, 3, NULL);
    if (status != pdPASS) { UARTprintf("FATAL: Failed to create SensorTask!\r\n"); while(1); }

    status = xTaskCreate(vDecisionTask, "DecisionTask", STACK_DECISION_TASK, NULL, 2,
NULL);
     if (status != pdPASS) { UARTprintf("FATAL: Failed to create DecisionTask!\r\n"); while(1); }

    status = xTaskCreate(vMotorControlTask, "MotorTask", STACK_DEFAULT, NULL, 4, NULL);
     if (status != pdPASS) { UARTprintf("FATAL: Failed to create MotorTask!\r\n"); while(1); }

    status = xTaskCreate(vLedControlTask, "LedTask", STACK_DEFAULT, NULL, 1, NULL);
     if (status != pdPASS) { UARTprintf("FATAL: Failed to create LedTask!\r\n"); while(1); }

    status = xTaskCreate(vUartLogTask, "UartTask", STACK_UART_TASK, NULL,0, NULL);
     if (status != pdPASS) { UARTprintf("FATAL: Failed to create UartTask!\r\n"); while(1); }

    UARTprintf("Tasks created. Starting scheduler...\r\n");
```

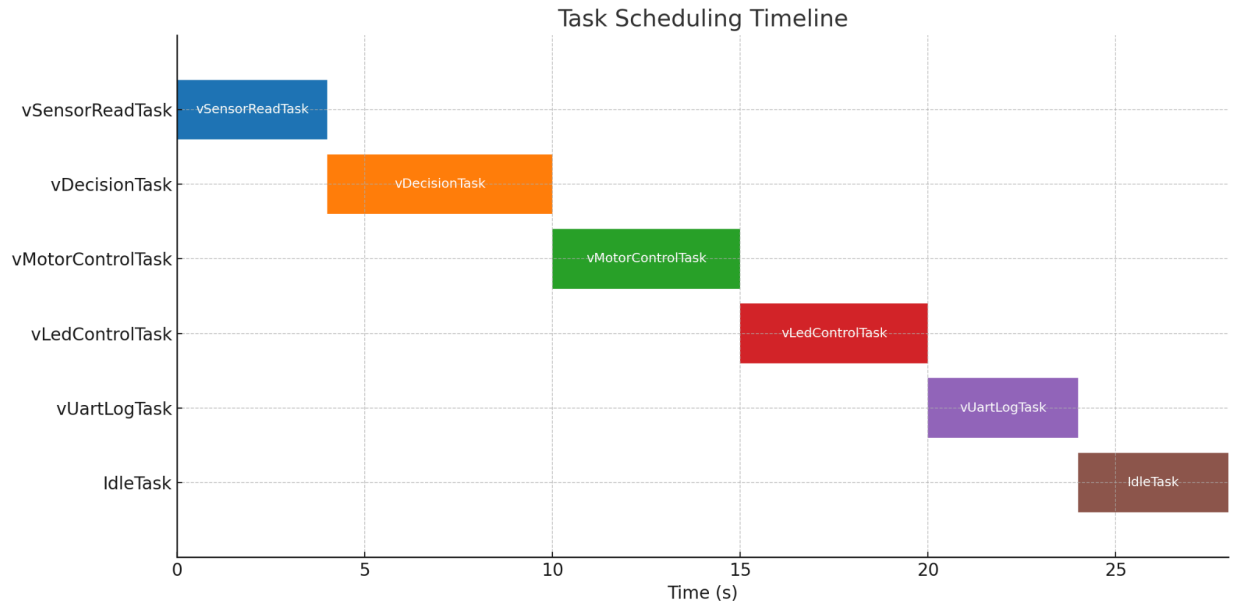
```
    vTaskStartScheduler();

    UARTprintf("FATAL: Scheduler returned!\r\n");
    while (1);
}


void vApplicationMallocFailedHook(void) {
    UARTprintf("FATAL: Malloc Failed!\r\n");
    taskDISABLE_INTERRUPTS();
    while(1);
}

void vApplicationStackOverflowHook(TaskHandle_t pxTask, char *pcTaskName) {
    (void)pxTask;
    UARTprintf("\r\nFATAL: Stack Overflow in task %s!\r\n", pcTaskName);
    taskDISABLE_INTERRUPTS();
     while(1);
}
```

Timing Table



Summary:
The code uses FreeRTOS to manage tasks, with one main task (UltrasonicTask) responsible for measuring distance and controlling LEDs and motors accordingly. The hardware is initialized and configured to enable the ultrasonic sensor to measure distances and adjust motor and LED states based on these measurements.

References:

1. A. B. Pinto, R. Barbosa, and M. F. Silva, "Autonomous Robot Control by Neural Networks," presented at CONTROLO 2014 - 11th Portuguese Conference on Automatic Control, Porto, Portugal, July 2014. DOI: 10.13140/2.1.3826.7524
2. M. S. H. Sunny, E. Hossain, T. N. Mimma and S. Hossain, "An autonomous robot: Using ANN to navigate in a static path," 2017 4th International Conference on Advances in Electrical Engineering (ICAEE), Dhaka, Bangladesh, 2017, pp. 291-296, doi: 10.1109/ICAEE.2017.8255369
3. K. Lenac, E. Mumolo, M. Nolich, and M. O. Noser, "Development of Embedded Devices in Real-Time Autonomous Robots," presented at the 28th International Conference on Information Technology Interfaces, 2006. DOI: 10.1109/ITI.2006.1708564. Conference Paper.
4. S. Harapanahalli, N. O. Mahony, G. Velasco Hernandez, S. Campbell, D. Riordan, and J. Walsh, "Autonomous Navigation of Mobile Robots in Factory Environment," Procedia Manufacturing, vol. 38, pp. 1524-1531, 2019, ISSN 2351-9789. [Online]. Available: