LAB 4: Home Security System(Group 14)

Hunter Adelson (1189312), Rydan Chalmers (1175347),

Charlie Magri (1194399), Kunal Sharma (1206352), Haroon Shahbaz (1174177)

ENGG: 4420

Dr. Spachos

Introduction

In this lab, we delve into the design and implementation of a real-time security system. The implementation will be capable of detecting motion and then capturing images from the place where motion was detected.

Real time system design is important for a system like this to ensure the camera can provide a live update of the motion. This could allow for tracking of intruder or potential danger and alert you once this is detected. The setup task would be a soft task as setup can be done when the device is plugged in, or turned on and will not need to be activated until the sensor detects motion, which will likely be much later than when you are setting up the device. The Image capture, image processing, motion sensing, and image display will all be Hard tasks as they are vital to the performance of the system. If any of these tasks were to be delayed this would lead to system failure. The image taking task will be cyclic, running at predetermined intervals with the other tasks being event driven and being called on capture completion.

The project incorporates several real-time challenges, such as integrating motion detection via a PIR sensor, managing the image acquisition through an ArduCAM module, and displaying the captured images in real-time. Synchronization and communication between tasks are essential, requiring the use of an RTOS to ensure seamless operation. Additionally, the system design includes an alert mechanism, such as LED blinking or LCD warnings, to enhance its usability.

Background

The purpose of the device is to create a home security sensing system. Home security systems can be used for multiple applications. For example, this live feed system could be implemented outside of your house to ensure no unwanted guests are trying to enter or trespassing on property. This system could also strictly be inside and used to alert authorities when unexpected motion is detected. The implanted system is only the beginning of a full security system as commercial models have automatic authority alerts, danger detection with ai, or even simple buzzer or alarms upon detection.

The Camera module used was ArduCAM and we implemented this with the I2C3 port. The role of the camera module is to take pictures at a desired rate, set by the operating system. This camera module can support a variety of capture modes with up to 90fps for smooth motion capture.

The motion sensor used is the PIR Motion Sensor. This sensor uses a simple 3 prong system to connect to the board. The purpose of this board is to send high to the pin when motion is detected allowing the code to enter the intruder state.

The Tera Term is used to get print statements from the code. The data from the camera is interfaced with the terminal to give status updates and notify if any failures. This could mean either bad camera setup or unexpected behavior. Similarly, it notifies the user of what the processor is doing and general workflow. It is also used to send reset to the system in case of false alarm detected from the system.

The LCD board was used in a previous lab and will be used to display a live feed of the camera while an intruder is detected. It Also displays warning messages while motion is detected. While motion is not detected in the last frame while active is displayed.

Methodology

This lab involved the design and implementation of a real-time security system that integrated motion detection with live image capture and display capabilities. The process was carried out in three main stages: hardware setup, software implementation, and system integration, each focusing on meeting specific lab requirements. The primary goal was to design a system that could reliably detect motion, trigger image capture, and display the captured image on an LCD, all while adhering to real-time constraints. The steps taken in this lab spanned hardware configuration, software design, and iterative testing to ensure robust functionality.

To begin with, the system's hardware setup was carefully planned and executed. The components used included an ArduCAM module for image capture, a PIR motion sensor for motion detection, and an STM32 microcontroller for system control and processing. The ArduCAM was configured to communicate with the STM32 using I2C3 for sensor control and SPI4 for ArduChip control, enabling efficient data transfer and camera management. The PIR sensor was connected to the STM32's GPIO pin (PC8), with its onboard potentiometer adjusted to fine-tune the detection range and time delay. The motion detection system relied on GPIO interrupts to ensure a prompt response to motion events, with the oscilloscope being used to calibrate the sensor's settings. The captured images were displayed on an LCD screen connected to the STM32, using APIs provided by the board's BSP library.

The software implementation focused on dividing the system's functionality into modular components distributed across three main files: camera.c, freertos.c, and gpio.c. The file camera.c handled all camera-related

ArduCAM Camera Module

Connections:
I2C3 port: For controlling the image sensor.
SPI4 port: For controlling the ArduChip and data transfer.
Purpose: Capturing images upon motion detection and transferring them for live display.

PIR Motion Sensor

Connections:
GPIO pin PC8 for digital output.
Purpose: Detects motion and triggers the image capture task.

STM32 Board

Acts as the central controller for the system, running the RTOS and managing communication between components.

Oscilloscope

Used to calibrate the PIR motion sensor for accurate detection of motion events.

FreeRTOS

Manages task priorities, synchronization, and scheduling.

The camera and motion sensor were carefully connected to the STM32 board using their respective ports. The PIR sensor's output level changes were monitored to validate its operation, ensuring it correctly generated a motion event. The oscilloscope was utilized during this process to confirm signal stability and proper detection timing.

Code Implementation

The code was implemented in three main files camera.c, freestos.c, and gpio.c. Camera.c was responsible for capturing images from the ArduCAM module, decoding the JPEG images using the PicoJPEG library, and displaying them on an LCD screen. The full code fro camera.c and freertos.c can be found attached to the bottom of the report in the appendix the relavent code snippet from gpio.c is shown in its description. Below is an explanation of the key components and functionality for each of the files:

Camera.c

Initilization (camera_setup)

This function is responsible for initializing the ArduCAM, it detects the ArduCAM hardware, powers up the camera, and configures the OV5642 sensor. If the setup is successful, the cameraReady flag is set to indicate readiness for image capture.

Capturing an Image (camera_initiate_capture)

This function is responsible for initiating image capture. To start capture it calls arduchip_start_capture() to trigger the camera to start capturing. The function will then wait and continuously check if the capture has completed using arduchip_capture_done(), if cature takes too long it will exceed the CAPTURE_TIMEOUT and exits with a corresponding error. If successful and the capture is completed it will call camera_get_image() to retrieve and process the image.

Retrieving the image (camera_get_image)

This function retrieves the size of the captured image in the FIFO buffer using arduchip_fifo_length(), it then passes the size to write_fifo_to_buffer() to read the image data from the FIFO buffer and processes it.

Conversion and Displaying of the Image

JPEG decoding was done by using the write_fifo_to_buffer() to read the JPEG image data in chunks from the FIFO buffer and store it in pic_buffer or pic_buffer2 to handle any cases where data may be overwritten due to th ebuffers limited size. The JPEG data is then passed to picojpg_test() for decoding, which will convert it into raw RGB pixel data. The decoded bitap is passed to display_bitmap(), which scales and displays the image on the LCD using BSP_LCS_DrawPixel() function.

Handling Image Data (write_ fifo_to_buffer)

This function will process the captured image data by reading the JPEG image data in chunks using arduchip_burst_read() and stores it in the appropriate buffer, as detrmined by the test variable. The image

will then be decoded by creating a memory stream from the buffer and passing it to picojpg_test() for debugging. The decoded bitmap will then be checked to make sure it matches the expected size (BITMAP_SIZE). Finally display_bitmap() is called to render the image on the LCD screen.

Pixel Conversion (get_pixel) **and Bitmap Display** (display bitmap)

get_pixel extracts RGB values from raw pixel data. For RGB images it will directly extract the red, green, and blue components. Display_bitmap will display the bitmap on the LCD by interating over the image pixel-by-pixel, converting each pixel to the ARGB format expected by the LCD. It will then use BSP_LCD_DrawPixel() to draw the pixel at the correct location on the screen.

The flow of execution for camera.c is as follows:

- Initialization:camera_setup()
    - prepares the camera and sets up the system
- Capture:camera_initiate_capture()
    - starts capturing an image.
- Process Image: camera_get_image()
    - retrieves the captured image size. write_fifo_to_buffer() reads the image data from the camera, decodes it, and displays it on the LCD.

Freertos.c

Initialization

Initialization is done by the function MX_FREERTOS_Init(), this is responsible for setting up mutexes, semaphores, message queues, and FreeRTOD tasks. The semaphores are responsible for synchronizing task execution such as motion detection, displaying alerts, and image capture. The mutexes ensure thread-safe access to shared resources such as data and printf. Tasks are created with specific priorities and stack sizes.

Tasks

StartTaskCOMM:

The purpose of this task is to monitor serial communication to handle commands. It reads messages from the CommQueueHandle queue. If it detects the message “s” it will stop the alert system and log it.

StartTaskGUI:

This task is resposible for managing the LCD to display the system information and alerts. It will wait for the CaptureHandle semaphore to start capturing an image and display an alert when DisplayALertHandle is triggered. This task also clears the alert after a delay.

StartTaskBTN:

This will monitor for a button press to clear alerts. Specifically, it will wait for the ClearHandle semaphore and if it detects a button press will reset the alert and log “clear”.

StartTaskSensor:
StartTaskSensor will monitor for motion from the PIR sensor to detect the intruder, it reads GPIO pin PC8 (GPIO_PIN_8) . It will debounce the signal and, after a sustained motion, release the MontionSensorHandle semaphore and sets do_alert = 1.

StartTaskCamera:

This manages the camera for capturing images. It sets up the camera if it is not initialized and will wait for the FakeHandle semaphore to trigger a capture. to initiate the capture the task will release the CaptureHandle semaphore for the StartTaskGUI

StartTaskAlert:

This handles intruder alerts. It waits for the MotionSensorHandle semaphore to detect motion and blinks LEDs while do_alert is active. It will also signal other tasks via FakeHandle and DisplayAlertHandle to display alerts or capture images.

The flow of execution for freertos.c is as follows:

- **Motion Detection**:
    - StartTaskSensor detects motion on GPIO pin GPIO_PIN_8.
    - Releases the MotionSensorHandle semaphore and sets do_alert = 1.
- **Alert Handling**:

    - StartTaskAlert activates LEDs and releases semaphores (FakeHandle, DisplayAlertHandle) to trigger image capture and display alerts.
- **Image Capture**:
    - StartTaskCamera waits for the FakeHandle semaphore and releases CaptureHandle to signal the GUI to capture an image.
    - StartTaskGUI calls camera_initiate_capture() to process the image.
- **Clearing Alerts**:
    - StartTaskBTN monitors the button press to clear alerts, resets do_alert, and releases the ClearHandle semaphore for the GUI.
- **Serial Command Handling**:
    - StartTaskCOMM processes incoming messages. If 's' is received, it stops the alert system.

Gpio.c

Finally, gpio.c was the file that handles the GPIO pin connections and settings. This file was modified to include code to setup pin PC8 so that the PIR motion sensor could interface with the rest of the system. Below is a code snippet of the implementation.

```
//TODO: define a pin for the motion sensor (refer to how the button pin is defined)
//      set the pin as external interrupt and define the priority (Optional)
GPIO_InitStruct.Pin = GPIO_PIN_8;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_PULLDOWN;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
```

Conclusion

In the conclusion of this lab report, we reflect on the challenges, lessons, skills developed, and next steps throughout the implementation and testing of the home security system on the STM32 board. The most challenging aspect was correctly displaying images from the camera to the board. To be more specific the image processing aspect was very difficult to achieve. Multiple design approaches were explored until a PicoJPG library proved to work. Dispite this challenge all requirements were met and the project was successful.

We learned many lessons regarding designs of real time systems. This lab taught us the importance of timing between components, and in-depth semaphore use. For example, you had to make sure that between camera captures you needed to give the camera enough time to process the first image or else the next one would overwrite data leading to obstructions, additionally if a image takes to long to process it will delay subsequent design and slow the entire cycle. We also learned to test your components before any implementation as faulty parts lead to confusion during later prototyping.

Our technical skills in programming, especially in C for embedded systems, were improved, as was our ability to troubleshoot complex systems involving multiple software and hardware interactions. Additionally, this lab emphasized transferable skills such as problem-solving, critical thinking, and effective time management. Moving forward, we can further improve these skills by practicing more with RTOS applications, working on advanced control systems, and continuously refining our ability to adapt theoretical

knowledge to practical challenges in embedded systems. Overall, this lab activity provided a comprehensive experience in both technical and applied aspects of real-time control systems.

There are many potential next steps for this lab. One could be adding uploads of pictures to the computer to document all movement from the device. This would allow for proof if legal action was ever taken. You could also implement an image recognition model on a more powerful board that would begin analyzing frames after motion was detected. Finally, you could set up multiple models in a wireless sensor network to monitor a larger area. Then the data could be sent to a base station or your phone for live updates without needing to be at the location of a camera.

Tables and Pictures

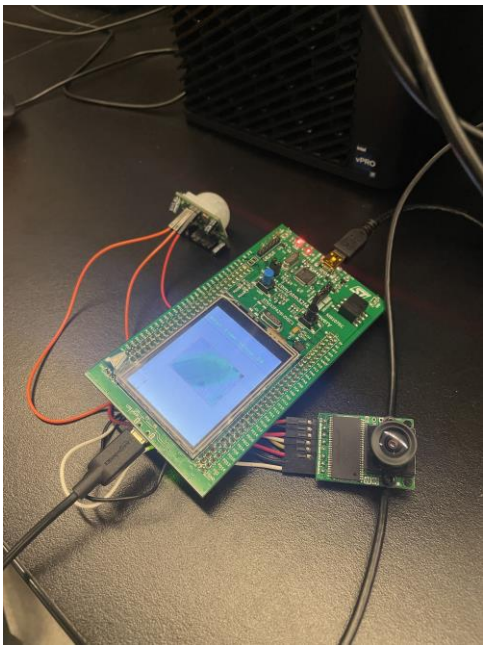

Figure 1 STM32 LCD display

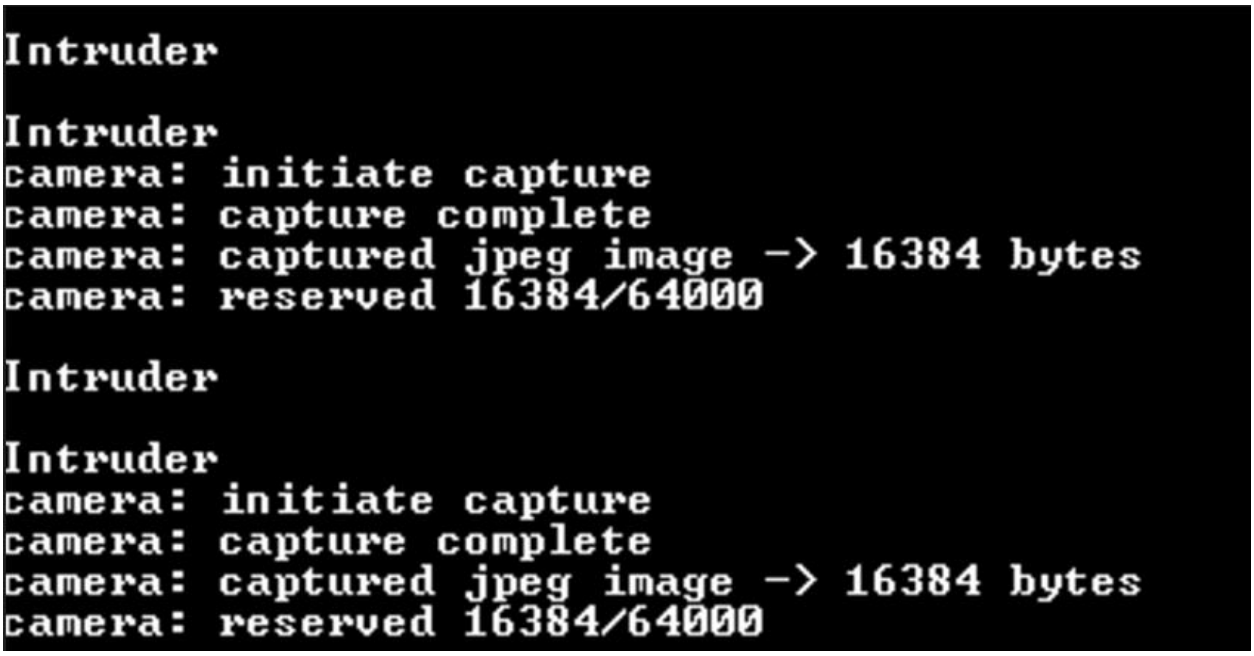

Figure 2 Tera Term output

Appendix

Camera.c

```
#include "camera.h"
#include "arducam.h"

#include "picojpeg_test.h"
#include "util.h"
#include "stlogo.h"
#include "stm32f429i_discovery_lcd.h"
#include "usbd_cdc_if.h"


#define MAX_PIC_SIZE 64000
#define BITMAP_SIZE 3600


static uint32_t captureStart;
uint8_t fifoBuffer[BURST_READ_LENGTH];
```

```
uint8_t fifoBuffer2[BURST_READ_LENGTH];
uint8_t pic_buffer[MAX_PIC_SIZE];
uint8_t pic_buffer2[MAX_PIC_SIZE];
uint8_t bitmap[BITMAP_SIZE];
int pic_index = 0;
int test = 0;
unsigned char cameraReady;

static void camera_get_image();
BaseType_t write_fifo_to_buffer(uint32_t length);

void camera_setup(){

    cameraReady = pdFALSE;

    if (   arduchip_detect()
        && arducam_exit_standby()
        && ov5642_detect()
    ) {

        osDelay(100);

        if (!ov5642_configure()) {
            printf("camera_task: ov5642 configure failed\n\r");
            return;
        } else {
            printf("camera: setup complete\n\r");
            cameraReady = pdTRUE;
            osDelay(100);
        }
    } else {
        printf("camera: setup failed\n\r");
        cameraReady = pdTRUE;
    }
}

/**
 * Capture an image from the camera.
 */
void camera_initiate_capture(){

    uint8_t done = 0;
```

```
    printf("camera: initiate capture\n\r");

    if (!cameraReady) {
        printf("camera: set up camera before capture\n\r");
    }

    /* Initiate an image capture. */
    if (!arduchip_start_capture()) {
        printf("camera: initiate capture failed\n\r");
        return;
    }

    /* wait for capture to be done */
    captureStart = (uint32_t)xTaskGetTickCount();
    while(!arduchip_capture_done(&done) || !done){

        if ((xTaskGetTickCount() - captureStart) >= CAPTURE_TIMEOUT) {
            printf("camera: capture timeout\n\r");
            return;
        }
    }

    printf("camera: capture complete\n\r");

    camera_get_image();

    return;

}

void camera_get_image(){

    /* Determine the FIFO buffer length. */
    uint32_t length = 0;
    if (arduchip_fifo_length(&length) == pdTRUE) {
        printf("camera: captured jpeg image -> %lu bytes\n\r", length);
        write_fifo_to_buffer(length);
    } else {
        printf("camera: get fifo length failed\n\r");
    }
```

```
    return;
}



static void get_pixel(int* pDst, const uint8_t *pSrc, int luma_only, int
num_comps)
{
    int r, g, b;

    // Case for grayscale (single component, just Y channel)
    if (num_comps == 1) {
        r = g = b = pSrc[0];  // If grayscale, all channels are the same
    }

    else if (luma_only) {

        int Y = pSrc[0];
        int U = pSrc[1];
        int V = pSrc[2];



        r = Y + 1.402 * (V - 128);
        g = Y - 0.344136 * (U - 128) - 0.714136 * (V - 128);
        b = Y + 1.772 * (U - 128);



        r = (r < 0) ? 0 : (r > 255) ? 255 : r;
        g = (g < 0) ? 0 : (g > 255) ? 255 : g;
        b = (b < 0) ? 0 : (b > 255) ? 255 : b;
    }
    // Case for RGB format (if the image is already RGB)
    else {
        r = pSrc[0];  // Red channel
        g = pSrc[1];  // Green channel
        b = pSrc[2];  // Blue channel
    }



    pDst[0] = r;
    pDst[1] = g;
    pDst[2] = b;
}
```

```
static void display_bitmap(const uint8_t * image, int width, int height, int
comps, int start_x, int start_y, int scale)
{

    int a[3];

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            get_pixel(a, image + (y * width + x) * comps, 0, comps);
            uint32_t argb = (0xFF << 24) | (a[0] << 16) | (a[1] << 8) | (a[0]);
            for (int yy = 0; yy < scale; yy++) {
                for (int xx = 0; xx < scale; xx++) {
                    BSP_LCD_DrawPixel(start_x + x * scale + xx, start_y + y *
scale + yy, argb);
                }
            }
        }
    }
    uint32_t old_col = BSP_LCD_GetTextColor();
    BSP_LCD_SetTextColor(LCD_COLOR_RED);
    BSP_LCD_DrawRect(start_x, start_y, width * scale, height * scale);
    BSP_LCD_SetTextColor(old_col);
}

BaseType_t write_fifo_to_buffer(uint32_t length)
{
    if (test % 2 == 0){
    int width, height, comp;
    uint16_t chunk = 0;
    unsigned int jpeg_size = length*sizeof(uint8_t);

    if (jpeg_size >= MAX_PIC_SIZE) {
        printf("camera: not enough memory (%d/%d)\n\r", jpeg_size, MAX_PIC_SIZE);
        return pdFALSE;
    } else {
        printf("camera: reserved %d/%d\n\r", jpeg_size, MAX_PIC_SIZE);
    }

    pic_index = 0;
    for (uint16_t i = 0; length > 0; ++i) {
```

```
        chunk = MIN(length, BURST_READ_LENGTH);
        arduchip_burst_read(fifoBuffer, chunk);
        for (uint16_t j = 0; j < chunk; j++) {
            pic_buffer[pic_index] = fifoBuffer[j];
            pic_index += 1;
        }
        length -= chunk;
    }

    FILE * pic_stream = fmemopen(pic_buffer, jpeg_size, "rb");  //jpg2tga will
close this, no need for fclose
    picojpg_test(pic_stream, &width, &height, &comp, bitmap);
    if(sizeof(bitmap) == BITMAP_SIZE){
        display_bitmap(bitmap, width, height, comp, 40, 100, 4);
    }else{
        return pdFALSE;
    }

    }
    else {
        int width, height, comp;
            uint16_t chunk = 0;
            unsigned int jpeg_size = length*sizeof(uint8_t);

            if (jpeg_size >= MAX_PIC_SIZE) {
                printf("camera: not enough memory (%d/%d)\n\r", jpeg_size,
MAX_PIC_SIZE);
                return pdFALSE;
            } else {
                printf("camera: reserved %d/%d\n\r", jpeg_size, MAX_PIC_SIZE);
            }

            pic_index = 0;
            for (uint16_t i = 0; length > 0; ++i) {

                chunk = MIN(length, BURST_READ_LENGTH);
                arduchip_burst_read(fifoBuffer2, chunk);
                for (uint16_t j = 0; j < chunk; j++) {
                    pic_buffer2[pic_index] = fifoBuffer2[j];
                    pic_index += 1;
                }
                length -= chunk;
            }
```

```
            FILE * pic_stream = fmemopen(pic_buffer2, jpeg_size, "rb"); //jpg2tga
will close this, no need for fclose
            picojpg_test(pic_stream, &width, &height, &comp, bitmap);
            if(sizeof(bitmap) == BITMAP_SIZE){
                display_bitmap(bitmap, width, height, comp, 40, 100, 4);
            }else{
                return pdFALSE;
            }

    }

    return pdTRUE;
}
```

Freertos.c

```
#include "FreeRTOS.h"
#include "task.h"
#include "cmsis_os.h"



/* USER CODE BEGIN Includes */
#include "usbd_cdc_if.h"
#include "stm32f429i_discovery.h"
#include "stm32f429i_discovery_lcd.h"
#include "spi.h"
#include "i2c.h"
#include "camera.h"

osMessageQId myQueue01Handle;
osMutexId myMutex01Handle;

/* USER CODE BEGIN Variables */
uint8_t RxData[256];
uint8_t Str4Display[50];

uint32_t data_received;

osThreadId TaskGUIHandle;
```

```
osThreadId TaskCOMMHandle;
osThreadId TaskBTNHandle;
osThreadId TaskSensorHandle;
osThreadId TaskAlertHandle;
osThreadId TaskCameraHandle;

osMessageQId CommQueueHandle;

osMutexId dataMutexHandle;
osMutexId printMutexHandle;

osSemaphoreId MotionSensorHandle;
osSemaphoreId DismissAlertHandle;
osSemaphoreId FakeHandle;
osSemaphoreId DisplayAlertHandle;
osSemaphoreId CaptureHandle;
osSemaphoreId ClearHandle;
int camera_on = 0;
int do_capture = 0;
int do_alert = 0;



typedef struct
{
    uint8_t Value[10];
    uint8_t Source;
}data;

data DataToSend={"Hello\0", 1};
data DataVCP={"VCP\0",2};




extern void MX_USB_DEVICE_Init(void);
void MX_FREERTOS_Init(void); /* (MISRA C 2004 rule 8.1) */

/* USER CODE BEGIN FunctionPrototypes */
void StartTaskCOMM(void const * argument);
void StartTaskBTN(void const * argument);
void StartTaskGUI(void const * argument);

void StartTaskCamera(void const * argument);
```

```
void StartTaskSensor(void const * argument);
void StartTaskAlert(void const * argument);

/* USER CODE END FunctionPrototypes */

/* Hook prototypes */

/* Init FreeRTOS */

void MX_FREERTOS_Init(void) {



    osMutexDef(dataMutex);
    dataMutexHandle = osMutexCreate(osMutex(dataMutex));

    osMutexDef(printMutex);
    printMutexHandle = osMutexCreate(osMutex(printMutex));

    //define the semaphores
    osSemaphoreDef(MotionSensor);
    MotionSensorHandle = osSemaphoreCreate(osSemaphore(MotionSensor), 1);

    osSemaphoreDef(DismissAlert);
    DismissAlertHandle = osSemaphoreCreate(osSemaphore(DismissAlert), 1);

    osSemaphoreDef(Fake);
    FakeHandle = osSemaphoreCreate(osSemaphore(Fake), 1);

    osSemaphoreDef(DisplayAlert);
    DisplayAlertHandle = osSemaphoreCreate(osSemaphore(DisplayAlert), 1);

    osSemaphoreDef(Capture);
    CaptureHandle = osSemaphoreCreate(osSemaphore(Capture), 1);

    osSemaphoreDef(ClearGUI);
    ClearHandle = osSemaphoreCreate(osSemaphore(ClearGUI), 1);



    osMessageQDef(myQueue01, 1, data);
    myQueue01Handle = osMessageCreate(osMessageQ(myQueue01), NULL);
```

```
    osMessageQDef(CommQueue, 1, &DataVCP);
    CommQueueHandle = osMessageCreate(osMessageQ(CommQueue), NULL);



    osThreadDef(TaskCOMM, StartTaskCOMM, osPriorityHigh, 0, 128);
    TaskCOMMHandle = osThreadCreate(osThread(TaskCOMM), NULL);

    osThreadDef(TaskBTN, StartTaskBTN, osPriorityLow, 0, 128);
    TaskBTNHandle = osThreadCreate(osThread(TaskBTN), NULL);

    osThreadDef(TaskGUI, StartTaskGUI, osPriorityAboveNormal, 0, 128);
    TaskGUIHandle = osThreadCreate(osThread(TaskGUI), NULL);



    osThreadDef(TaskCamera, StartTaskCamera, osPriorityNormal, 0, 128);
    TaskCameraHandle = osThreadCreate(osThread(TaskCamera), NULL);

    osThreadDef(TaskSensor, StartTaskSensor, osPriorityNormal, 0, 128);
    TaskSensorHandle = osThreadCreate(osThread(TaskSensor), NULL);

    osThreadDef(TaskAlert, StartTaskAlert, osPriorityNormal, 0, 128);
    TaskAlertHandle = osThreadCreate(osThread(TaskAlert), NULL);
}



void StartTaskCOMM(void const * argument)
{

  osEvent vcpValue;

  while(1)
  {
      vcpValue = osMessageGet(CommQueueHandle, osWaitForever);
      osMutexWait(dataMutexHandle, 0);
      memcpy(Str4Display,(char *)(((data *)vcpValue.value.p)->Value),
data_received+1);
      osMutexRelease(dataMutexHandle);

      //Type s to stop in serial terminal
```

```
      if (Str4Display[0] == 's')
      {
          printf("False Alarm\n\r");
          do_alert = 0;



      }
  }
}


void StartTaskGUI(void const * argument)
{
  while(1)
  {



      BSP_LCD_SetFont(&Font16);
      BSP_LCD_SetTextColor(LCD_COLOR_PURPLE);
      BSP_LCD_DisplayStringAtLine(2, (uint8_t *)" Real Time Group 14");
      BSP_LCD_SetTextColor(LCD_COLOR_BLACK);
      BSP_LCD_DrawHLine(0, 55, 240);

      osSemaphoreWait(CaptureHandle, osWaitForever);
      camera_initiate_capture();



      while(!(osSemaphoreWait(DisplayAlertHandle, 1))){
          BSP_LCD_SetTextColor(LCD_COLOR_RED);
          BSP_LCD_DisplayStringAtLine(4, (uint8_t *)" Criminal Detected:(");
          osSemaphoreRelease(ClearHandle);
      }



      osDelay(200);
      BSP_LCD_DisplayStringAtLine(4, (uint8_t *)"
");

  }
}

void StartTaskBTN(void const * argument)
```

```
{

  while(1) {

      osSemaphoreWait(ClearHandle, 0);
      if(HAL_GPIO_ReadPin(KEY_BUTTON_GPIO_PORT, KEY_BUTTON_PIN) == GPIO_PIN_SET){

          printf("clear\n\r");

          do_alert = 0;
          while(HAL_GPIO_ReadPin(KEY_BUTTON_GPIO_PORT,
KEY_BUTTON_PIN)==GPIO_PIN_SET);
      }
      osDelay(50);
  }
}

void StartTaskSensor(void const * argument)
{
    int i = 0;
    while (1)
    {
      if ((HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_8) == GPIO_PIN_SET)) // if the pin is
HIGH
      {
          i++;
          if(i > 6){
              i = 0;
              while ((HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_8) ==
GPIO_PIN_SET)){//wait for pin to go low
                  osSemaphoreRelease(MotionSensorHandle);
                  do_alert =1;
              }
          }
      }

      //osDelay(50);
    }
}



void StartTaskCamera(void const * argument)
{
```

```
    while (1) {

        if (!camera_on) {
            camera_setup();
            camera_on = 1;
        }

        //start capture if and only the intruder is detected
        printf("\n\r");

        if(osSemaphoreWait(FakeHandle, 1) && !do_alert){
            osDelay(2000);
        }
        else{
            printf("Intruder\n\r");
            osSemaphoreRelease(CaptureHandle);
            osDelay(100);

        }

    }
}


void StartTaskAlert(void const * argument)
{
    while (1) {

        osSemaphoreWait(MotionSensorHandle, osWaitForever);
        while(do_alert){
            osSemaphoreRelease(FakeHandle);
            osSemaphoreRelease(DisplayAlertHandle);


            HAL_GPIO_WritePin(GPIOG,LD3_Pin|LD4_Pin,GPIO_PIN_SET);
            HAL_Delay(200);
            HAL_GPIO_WritePin (GPIOG,LD3_Pin|LD4_Pin, 0); // LED OFF
            HAL_Delay(200);
        }
                //camera timing stuff
```

```
    }
}
```

Refrences

R. Muresan and K. Dong, *ENGG4420 Real-Time Systems Design Lab Manual*, 6th ed., University of Guelph, 2024.

K. Dong, "Lab 3: Hot Air Plant Control with RTOS," *ENGG4420 Real-Time Systems Design Lab Manual*, University of Guelph, 2024. [PDF document].