DOMOTIC - SOFTWARE

Project

Students: Pablo Sorrentino, Cristian Maquilon, Aldy Rhenals, Camila Ramirez

First Semester 2023
Due Date: 18th of April
ÉCOLE NATIONALE SUPÉRIEURE DE L'ÉLECTRONIQUE ET DE SES APPLICATIONS

Contents

1. Introduction

The Domotic Project is an innovative home automation project that uses three Blue Pill microcontrollers, connected via Bluetooth to an App, which in turn communicates with a server created for the project. The three microcontrollers are responsible for different functionalities: temperature and humidity sensing, motor control, and dimmer signal for controlling light brightness.

The first microcontroller in the system is equipped with a temperature and humidity sensor, which continuously measures the ambient temperature and humidity levels in the home. This data is transmitted to the mobile phone via Bluetooth for display and logging on the server. Users can remotely monitor the temperature and humidity levels in their home in real-time, enabling them to take appropriate actions if it is desired.

The second microcontroller in the system is responsible for motor control. It can drive and control various motorized appliances in the home, such as a motorized window blind or a motorized gate. The mobile phone interface allows users to remotely open or close these motorized appliances, providing convenient and flexible control options.

The third microcontroller in the system generates a dimmer signal for controlling the brightness of a light. It can receive commands from the mobile phone via Bluetooth to adjust the brightness level of a connected light source. This allows users to remotely control the brightness of a light, creating a customized lighting experience in their home.

All three microcontrollers are connected to a mobile phone via Bluetooth, which serves as the central control interface for the entire system. The mobile phone communicates with a server created for the project, which handles data logging, processing, and communication with the microcontrollers.

1.1. Blue Pill STM32F103C8T microcontroller

The Blue Pill STM32F103C8T microcontroller is a popular development board based on the STM32F103C8T6 microcontroller chip, which is part of the STM32 family of ARM Cortex-M3 based microcontrollers. The Blue Pill board has gained widespread popularity due to its affordability, versatility, and ease of use, making it a popular choice for a wide range of embedded systems projects, including hobbyist projects, DIY electronics, and industrial applications.

The STM32F103C8T6 microcontroller chip is manufactured by STMicroelectronics, a leading semiconductor company known for its high-quality microcontrollers. The chip of this microcontroller is based on the ARM Cortex-M3 processor, which is a powerful 32-bit processor with a rich instruction set and a high-performance core. The microcontroller operates at a clock speed of up to 72 MHz and has a built-in Flash memory of 64 KB for storing program code and 20 KB of SRAM for data storage.

One of the key features of the Blue Pill STM32F103C8T microcontroller is its versatile I/O capabilities. The board comes with a total of 37 I/O pins, which can be configured as GPIO (General-Purpose Input/Output) pins for various digital and analog operations. These pins can be used to interface with various sensors, actuators, displays, and other peripheral devices, making the Blue Pill board suitable for a wide range of applications.

The Blue Pill STM32F103C8T board also includes a number of built-in peripherals, such as UART (Universal Asynchronous Receiver/Transmitter), SPI (Serial Peripheral Interface), I2C (Inter-Integrated Circuit), PWM (Pulse Width Modulation), and more. These peripherals provide versatile communication and control options, allowing the microcontroller to interface with a wide range of external devices.

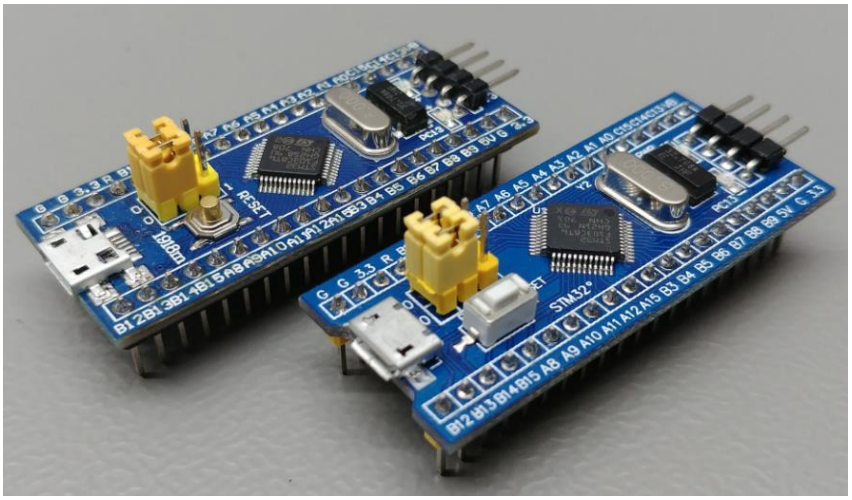

Figure 1: Top Layout of the Blue Pill STM32F103C8T.

The Blue Pill STM32F103C8T microcontroller also has a rich set of development tools and software libraries available, making it easy to develop and debug software for the microcontroller. STMicroelectronics provides a comprehensive development environment called STM32Cube, which includes a suite of software tools, libraries, and example code for developing firmware for STM32 microcontrollers.

To sum up, the Blue Pill STM32F103C8T microcontroller offers a powerful and versatile platform for developing a wide range of embedded systems projects. Its affordability, ease of use, and rich set of features make it a popular choice for hobbyists, DIY electronics enthusiasts, and professional developers alike. With its robust performance, extensive I/O capabilities, and support for various communication interfaces, the Blue Pill STM32F103C8T microcontroller is well-suited for diverse applications, including home automation, robotics, industrial control, and more.

1.2. Temperature and Humidity sensor: DHT11

The DHT11 sensor is a digital temperature and humidity sensor known for its simplicity and affordability. It requires only three pins for connection and operates on a single-wire digital communication protocol. With a temperature measurement range of 0°C to 50°C and a humidity measurement range of 20% to 90% RH, the DHT11 sensor provides reasonable accuracy of ±2°C for temperature and ±5% RH for humidity. It has a response time of around 2 seconds for temperature measurements and 2-5 seconds for humidity measurements. Its compact size, low power consumption, and durable construction make it a popular choice for various indoor environmental monitoring applications.

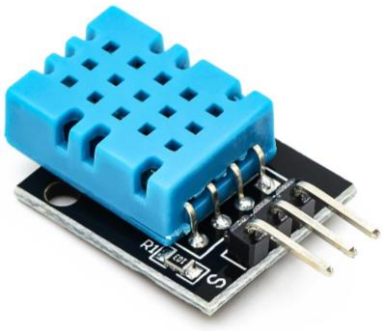

Figure 2: View of the final packaging of the DHT11.

Main Characteristics of the sensor:

- Temperature measurement range: 0°C to 50°C
- Humidity measurement range: 20% to 90% RH
- Accuracy: ±2°C for temperature, ±5% RH for humidity

- Fast response time: around 2 seconds for temperature, 2-5 seconds for humidity
- Operating Voltage: 3.3V to 5.5V
- Digital Output: Provides digital output for both temperature and humidity measurements
- Sampling Rate: Supports a sampling rate of up to 1 Hz (1 measurement per second)
- Sensor Type: Utilizes a capacitive humidity sensor and a thermistor for temperature measurement
- Communication Protocol: One-wire digital communication with a single data pin

1.3. Bluetooth communication: HC-05

The HC-05 Bluetooth module is a popular and widely used Bluetooth module that provides wireless communication capabilities for electronic devices. It is a versatile module that enables communication between microcontrollers, sensors, and other devices with Bluetooth-enabled devices, such as smartphones, tablets, and laptops. The HC-05 module is easy to use, with a simple serial interface for communication and configuration. It operates on the Bluetooth 2.0 standard and supports the Serial Port Profile (SPP), making it compatible with a wide range of devices and applications. The HC-05 module offers reliable communication, a range of up to 10 meters, and supports various baud rates for flexible communication speed. It is widely used in DIY projects, robotics, home automation, and other applications that require wireless communication capabilities.

Figure 3: *HC-05 module*.

- Bluetooth Version: Bluetooth 2.0 EDR (Enhanced Data Rate)
- Communication Interface: Serial UART (Universal Asynchronous Receiver/Transmitter)
- Operating Voltage: 3.3V to 5V
- Communication Range: Up to 10 meters (line of sight)
- Baud Rates: Supports various baud rates, including 9600, 19200, 38400, 57600, and 115200 bps
- Profiles: Supports the Serial Port Profile (SPP)
- Security: Provides encryption and authentication features for secure communication
- Pin Configuration: Features a 6-pin header with VCC, GND, TXD, RXD, EN (Enable), and STATE pins for easy interfacing with other devices

2. Implementation of the *Hardware*

The following three figures show all the pins used to carry out the project, including the modules and peripherals clarified in advance:

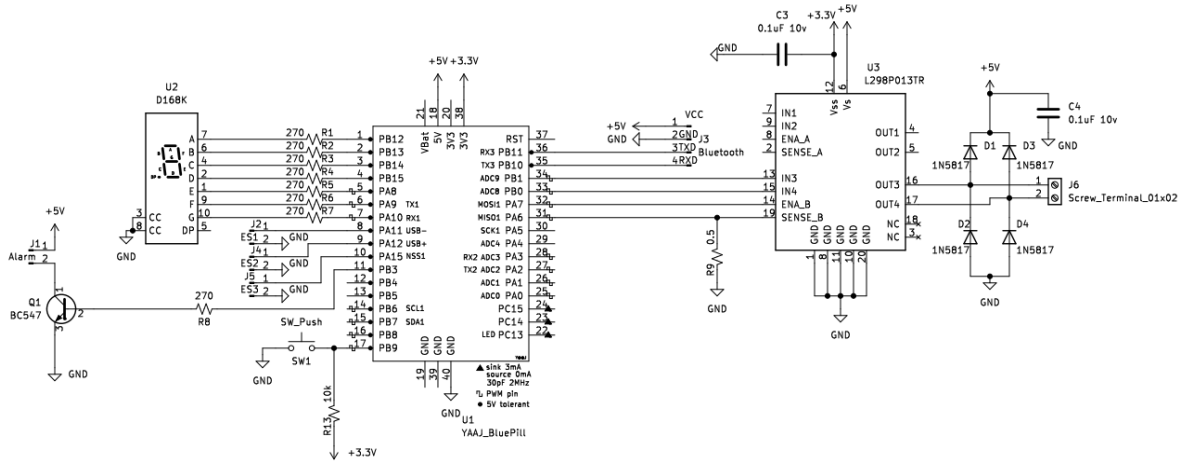

Figure 4: Pins of the Blue Pill used with the motor.

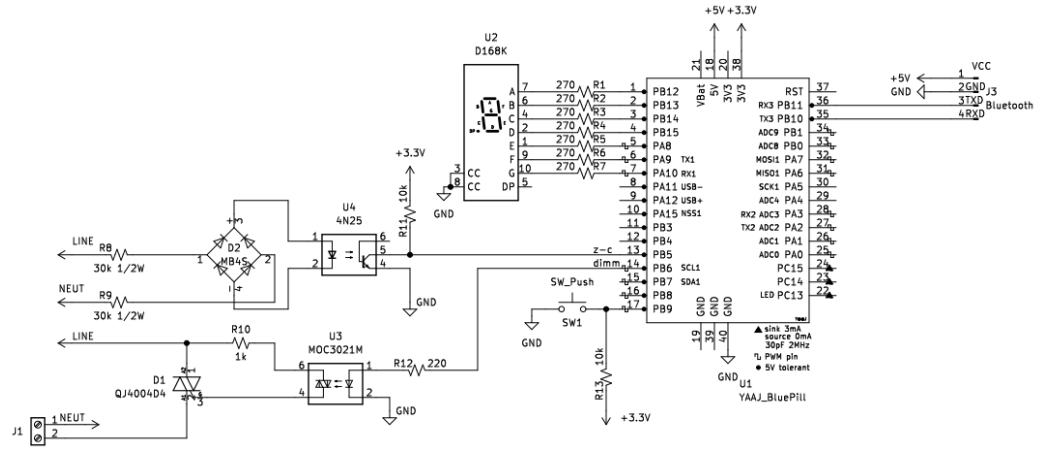

Figure 5: Pins of the Blue Pill used the dimmer.

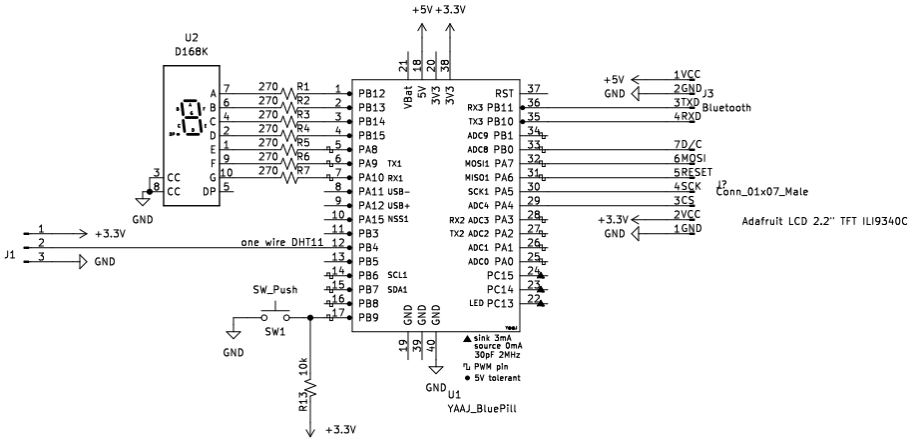

Figure 6: Pins of the Blue Pill used with the sensor.

3. Development of the *Software*

The programming of the device was carried out in a code developed in the STMCubeIDE software in C language, in addition the FreeRTOS library was implemented to work on a real-time operating system and the Hardware Abstraction Layer (HAL) provided by the STM. Each of the tasks will be explained in the next subsection:

4.1. Motor:

To begin with, the code for implementing the motor involved configuring the GPIO pins for motor control (IN1 and IN2 pins for direction control, and an ENABLE pin for motor enable/disable), and implementing appropriate motor control logic (forward, reverse, stop) in the main program loop. Additionally, input pins such as two limit switches were used for sensing the motor status.

In addition, the code for implementing the HC-05 Bluetooth module involved configuring the UART (Universal Asynchronous Receiver/Transmitter) interface for communication with the module, setting up appropriate baud rate (9600), data bits, stop bits, and parity settings, and handling UART interrupts for sending and receiving data. Data sent or received via the Bluetooth module were processed in the main program loop for controlling the motor logic.

To implement a 7-segment display module in STM32, the code included configuration of GPIO pins for segment control and connecting them to the correct STM32 pins and implementing logic to convert desired numbers or characters into corresponding segment patterns (with a proper pin connection). This logic was used for the alarm and switches.

```
int main(void)
{
  uint8_t limitSW_State_Open;
  uint8_t limitSW_State_Close;
  HAL_Init();
  SystemClock_Config();
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART3_UART_Init();
  /* USER CODE BEGIN 2 */
  // Configure motor control pins as GPIO outputs
    GPIO_InitTypeDef GPIO_InitStruct;
    __HAL_RCC_GPIOA_CLK_ENABLE(); // Enable GPIOA clock
    GPIO_InitStruct.Pin = MOTOR_EN_GPIO_PIN | MOTOR_IN1_GPIO_PIN |
MOTOR_IN2_GPIO_PIN;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(MOTOR_EN_GPIO_PORT, &GPIO_InitStruct);

    // 7 segments
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, 1);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, 0);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, 0);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12, 1);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, 1);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, 1);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, 1);
  while (1)
  {
```

```
	HAL_UART_Read_String(&huart3, message_rx, 20);
	sprintf(message_tx,"motor_R\r\n");
	HAL_UART_Transmit(&huart3, (uint8_t*)message_tx,strlen(message_tx),
HAL_MAX_DELAY);
	HAL_Delay(1000);
	HAL_GPIO_WritePin(MOTOR_EN_GPIO_PORT, MOTOR_EN_GPIO_PIN, GPIO_PIN_SET);//
Enable motor
	HAL_GPIO_WritePin(MOTOR_IN1_GPIO_PORT, MOTOR_IN1_GPIO_PIN,
GPIO_PIN_RESET);
	HAL_GPIO_WritePin(MOTOR_IN2_GPIO_PORT, MOTOR_IN2_GPIO_PIN,
GPIO_PIN_RESET);
	limitSW_State_Open = HAL_GPIO_ReadPin(Limit_SW2_GPIO_Port, Limit_SW2_Pin);
	limitSW_State_Close = HAL_GPIO_ReadPin(Limit_SW3_GPIO_Port,
Limit_SW3_Pin);
	if(limitSW_State_Open == 0){
		if(!strcmp(message_rx, "Open")){
			while(limitSW_State_Close == 1){
				HAL_GPIO_WritePin(MOTOR_IN1_GPIO_PORT,
MOTOR_IN1_GPIO_PIN, GPIO_PIN_SET);
				HAL_GPIO_WritePin(MOTOR_IN2_GPIO_PORT,
MOTOR_IN2_GPIO_PIN, GPIO_PIN_RESET);
				limitSW_State_Close =
HAL_GPIO_ReadPin(Limit_SW3_GPIO_Port, Limit_SW3_Pin);
			}
		}
	}
	if(limitSW_State_Close == 0){
		if(!strcmp(message_rx, "Close")){
			while(limitSW_State_Open == 1){
				HAL_GPIO_WritePin(MOTOR_IN1_GPIO_PORT,
MOTOR_IN1_GPIO_PIN, GPIO_PIN_RESET);
				HAL_GPIO_WritePin(MOTOR_IN2_GPIO_PORT,
MOTOR_IN2_GPIO_PIN, GPIO_PIN_SET);
				limitSW_State_Open =
HAL_GPIO_ReadPin(Limit_SW2_GPIO_Port, Limit_SW2_Pin);
			}
		}
	}
	HAL_GPIO_WritePin(MOTOR_EN_GPIO_PORT, MOTOR_EN_GPIO_PIN,
GPIO_PIN_RESET);// Disable motor
  }
}
```

4.2.Dimmer:

The code for implementing a dimmer and zero crossing signals for controlling light brightness in STM32 involved setting up a timer to generate a PWM signal for dimming control, configuring a GPIO pin as input for zero crossing detection, and implementing the logic to synchronize the PWM signal with the zero crossing signal. This included the duty cycle, and period for controlling the brightness level, as well as configuring interrupt or input capture to detect the zero crossing point of the AC waveform. Besides, we have used the same configurations seen for the 7 segment and the Bluetooth module.

```
MX_TIM2_Init();
HAL_TIMEx_OCN_Start_IT(&htim2, HAL_TIM_ACTIVE_CHANNEL_1);
while(1){
...
Dimmer = atoi(message_rx);
         for(Dimmer = 0; Dimmer < 2000; Dimmer +=10){
                TIM2->CCR1 = 2000 - (Dimmer*10);
                TIM2->CCR3 = TIM2->CCR1 +200;
                HAL_Delay(100);
         }
}
...
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){ //External Interrupt of the Z-C
       if(GPIO_Pin == Zero_Pin){
              if(HAL_GPIO_ReadPin(Zero_GPIO_Port, Zero_Pin) == 1){
                     if(Flag == 0){
                            TIM2->CNT = 0;
                            Flag = 1;
                     }
              }
              else{
                     Flag = 0;
              }
       }
}
void HAL_TIM_OC_DelayElapsedCallback(TIM_HandleTypeDef *htim){ //Dimmer Signal
       if (htim == &htim2 && htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1){
              HAL_GPIO_WritePin(Dimmer_GPIO_Port, Dimmer_PIN, SET);
       }
}
```

4.3. Sensor:

The code for implementing a DHT11 temperature and humidity sensor in STM32 involves several steps. First, a GPIO pin needs to be configured as an input for reading the sensor data. Next, a timing protocol is implemented to communicate with the sensor, including setting up a timer to generate precise timing intervals (Timer 1 in this case). The GPIO pin is initialized for data input, and the code needs to interpret the data bits sent by the sensor to obtain the temperature and humidity values. This includes accounting for the specific timing requirements of the DHT11 sensor, such as the start signal, data transmission, and checksum calculation.

In addition, apart from bluetooth and the 7 segments display, the code for implementing a TFT (Thin-Film Transistor) display for the DHT11 involved configuring the appropriate GPIO pins for SPI communication, setting up the SPI interface with DMA for efficient data transfer. The use of DMA can offload the data transfer from the CPU, improving performance and efficiency. Proper configuration and handling of DMA and SPI are important for smooth and reliable data transfer and display operation. We have used the SPI2 in correlation with the conveniences of the hardware and the pins used. For this display we also included the libraries ILI9341_STM32_Driver and ILI9341_GFX to configure the external module correctly.

```
...
int main(void){
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_SPI1_Init();
  MX_TIM1_Init();
  ILI9341_Init();
...
  while (1)
  {
  if(DHT11_Start()){
              RHI = DHT11_Read(); // Relative humidity integral
              RHD = DHT11_Read(); // Relative humidity decimal
              TCI = DHT11_Read(); // Celsius integral
              TCD = DHT11_Read(); // Celsius decimal
              SUM = DHT11_Read(); // Check sum
       }
       if (RHI + RHD + TCI + TCD == SUM){
              // Can use RHI and TCI for any purposes if whole number only needed
              tCelsius = (float)TCI + (float)(TCD/10.0);
              tFahrenheit = tCelsius * 9/5 + 32;
              RH = (float)RHI + (float)(RHD/10.0);
              // Can use tCelsius, tFahrenheit and RH for any purposes
       }
  ILI9341_FillScreen(WHITE);
  static char BufferText[30];
  temperature_C = tCelsius;
  sprintf(BufferText, "TC: %d°C", temperature_C);
  ILI9341_DrawText(BufferText, FONT3, 10, 10, BLACK, WHITE);
  sprintf(message_tx,"%u\r\n",temperature_C);
  HAL_UART_Transmit(&huart3, (uint8_t*)message_tx,strlen(message_tx),
HAL_MAX_DELAY);
```

4. Conclusion

In conclusion, the Domotic Project that utilizes three microcontrollers connected via Bluetooth with a cellphone, which in turn communicates with a server, offers a comprehensive and integrated solution for home automation. The three microcontrollers, each serving a specific purpose, work in synergy to enable efficient control and monitoring of temperature, humidity, motor, and light brightness in a home environment.

The microcontroller responsible for temperature and humidity sensing, such as the DHT11 sensor, provides real-time data on the environmental conditions inside the home, allowing for automated climate control and energy management. The microcontroller controlling the motor enables remote control and automation of devices such as fans, curtains, or blinds, adding convenience and comfort to the home environment. Lastly, the microcontroller handling the dimmer signal for controlling light brightness allows for custom lighting scenarios, energy-efficient lighting management, and mood setting in different rooms of the house.

The Bluetooth connectivity between the microcontrollers and the cellphone acts as a seamless bridge for remote control and monitoring from a mobile device. The mobile application allows the user to

interact with the microcontrollers, adjust settings, monitor sensor data, and control devices remotely, providing a user-friendly and convenient interface for home automation.

The server component of the project serves as a central hub for data storage, processing, and coordination. It facilitates communication between the microcontrollers and the mobile application, allowing for remote access, data logging, and data analysis for smart decision making.

The Domotic Project with its interconnected microcontrollers, Bluetooth communication, and server integration offers a robust and scalable solution for home automation. It brings numerous benefits such as increased comfort, convenience, energy efficiency, and remote access and control, enhancing the overall quality of life in a smart home environment.

To resume, the Domotic Project represents a successful implementation of a smart home automation system that combines the capabilities of multiple microcontrollers, Bluetooth communication, and server integration. With its diverse functionalities and seamless remote control and monitoring capabilities, the project offers a compelling solution for modern homes seeking automation and smart control over various aspects of the living environment.

One crucial aspect of future work is code optimization, error handling, and security enhancements. Optimizing the code running on the microcontrollers, mobile application, and server components can improve the overall efficiency and performance of the system. Implementing robust error handling mechanisms, such as error checking, data validation, and error recovery, can enhance the system's reliability and resilience to potential issues. Strengthening security measures, such as authentication, encryption, and access control, can safeguard against security threats and unauthorized access, ensuring the privacy and security of the system and its users.

Energy management is another area for future improvement. Implementing energy-saving techniques, such as sleep modes and power optimization, can reduce the system's energy consumption, making it more environmentally friendly and cost-effective. Finally, designing the system to be scalable and expandable, with modular and flexible architecture and standardized communication protocols, can allow for easy integration of new components in the future.

5. References

[1] ARM® Cortex®-M3 Processor Revision: r0p1 Technical Reference Manual Copyright © 2009, 2010, 2013, 2015 ARM. All rights reserved. ARM 100166

[2] Reference Manual for FreeRTOS version 10.0.0.

[3] https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

[4] https://stm32-base.org/boards/STM32F103C8T6-Blue-Pill.html

[5] https://components101.com/sites/default/files/component_datasheet/HC-05%20Datasheet.pdf

[6] https://components101.com/displays/7-segment-display-pinout-working-datasheet

[7] https://cdn-shop.adafruit.com/datasheets/ILI9341.pdf