POLITÉCNICA

UNIVERSIDAD POLITÉCNICA DE MADRID

DOMÓTICA DE UNA VIVIENDA MEDIANTE PLACA STM32F407

SISTEMAS ELECTRÓNICOS DIGITALES

Realizado por:
Raúl Herranz Rodríguez - EE403 - 54928
Carlos Murillo Pagador - EE403 - 54933
Alejandro Sacristán Jiménez – EE403 - 54936

Repositorio: https://github.com/asacristanj/Trabajo-Microprocesadores-SED-2021

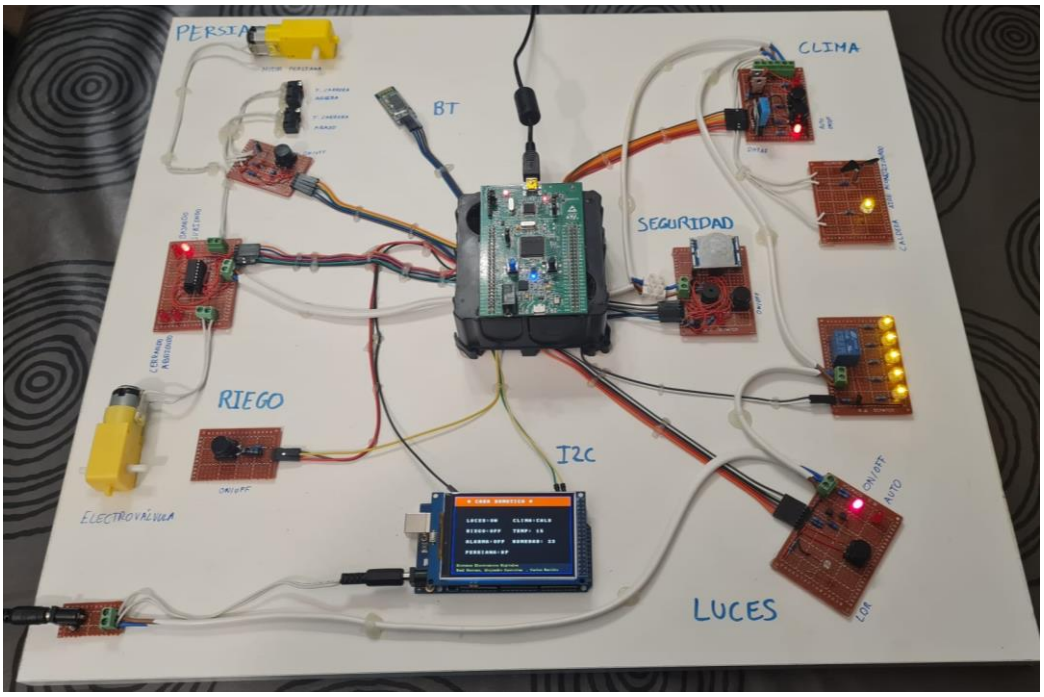
Índice

INTRODUCCIÓN

Este proyecto se trata de la realización de una aplicación domótica con el microprocesador STM32F407. El control domótico alcanzará diversas funciones dentro de una propia casa: el control de: luces, persianas, temperatura y alarma. Todo esto, se simulará físicamente mediante una maqueta en la que se usarán botones, diversos sensores, finales de carrera, pequeños motores, LEDs, etcétera.

Además de poder realizar el control mediante los botones, se ha añadido un módulo bluetooth (concretamente HC-06), para que todas las funcionalidades se puedan monitorear y controlar desde un dispositivo móvil, siendo más cómodo y accesible su uso.

La maqueta también contará con un display TFT en el que se mostrará diversas informaciones, como por ejemplo la temperatura a la que se encuentra la casa, el estado de las luces (on, off o automático) o si la alarma está activada o desactivada.



Estrategia y algoritmos

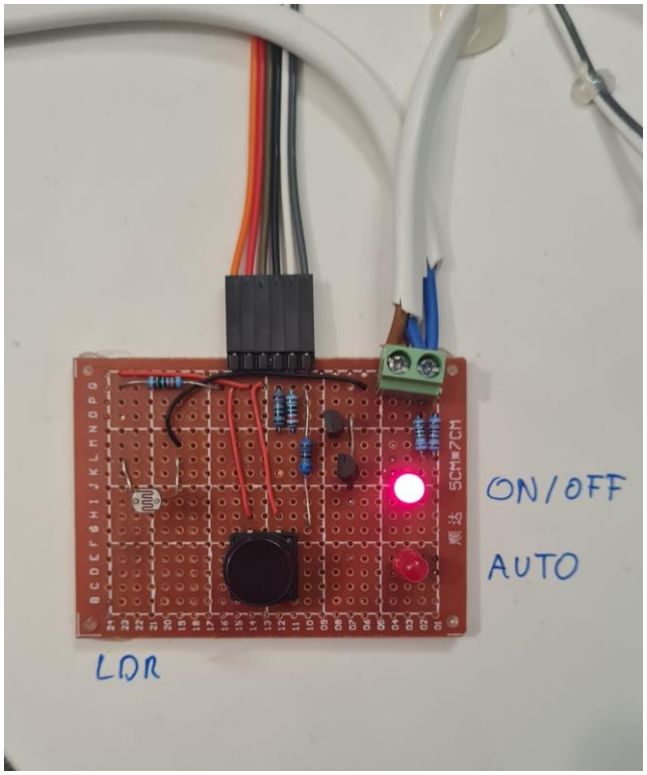
La estrategia se basará en dividir el control domótico en módulos. Para ello se ha creado un archivo de cabecera ".h" por cada módulo para que el código se pueda entender con más claridad. En el correspondiente "main.c" se ejecutarán las funciones de los anteriores archivos que necesiten repetirse en bucle y se gestionarán todas las interrupciones. A continuación, se mostrarán los diferentes componentes que controla la

placa en cada módulo y las herramientas seguidas en cada uno, su funcionamiento específico se detallará más adelante:

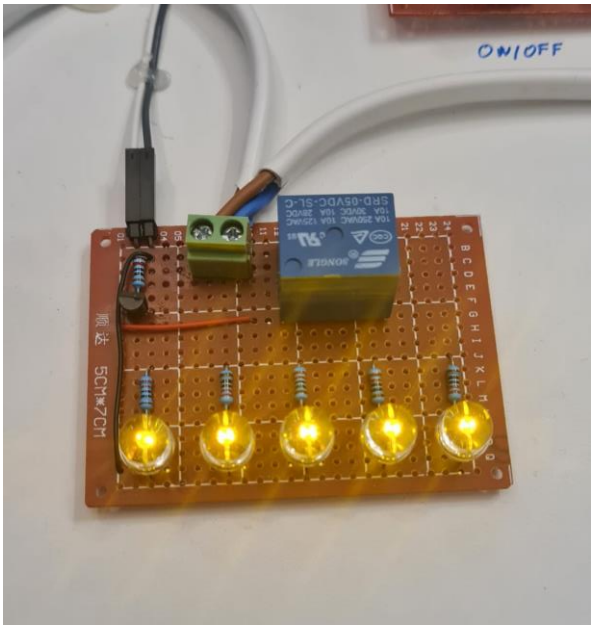
Control de luces

Este módulo se encargará del control de las luces de la casa. El hardware que se utilizará para simularlo será el siguiente:

Por un lado, en una PCB universal perforada, se han colocado: un botón, un LDR y dos LEDs. El botón cuenta con una resistencia de pull-down, el fotorresistor con una resistencia en serie de 1kΩ y los LEDs con sus debidas resistencias de protección y dos transistores 2N2222 que actúan como interruptores, alimentándolos así con una fuente de alimentación externa.



Por otro lado, en otra PCB, se ha colocado lo que simularía las luces de la casa: consta de 5 LEDs, cada uno con su resistencia de protección y un relé que los enciende o apaga. La bobina de control del relé está conectado a un transistor que actúa como interruptor.

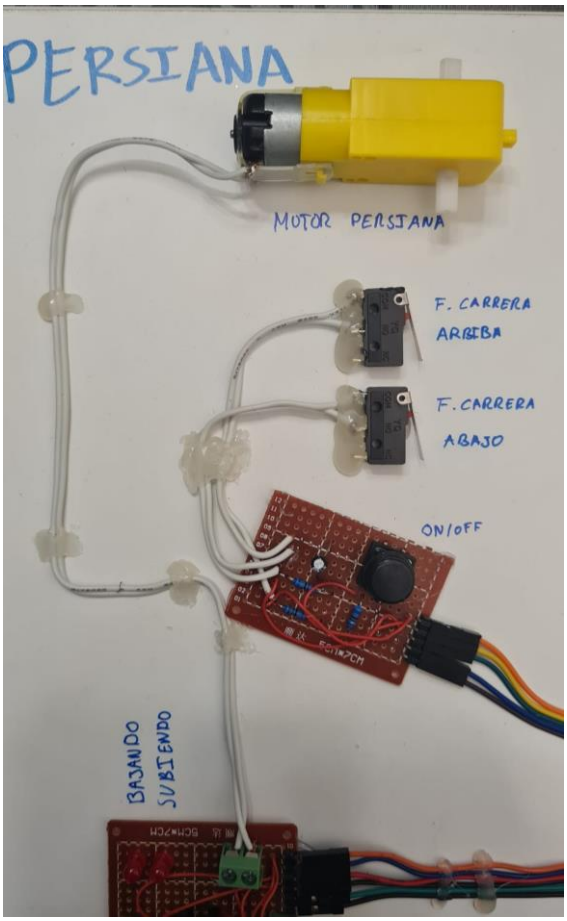


En cuanto al software, las luces podrán estar en 3 estados: apgadas (OFF), encendidas (ON) y en automático (AUTO). Se usará un convertidor analógico-digital para el correcto funcionamiento del modo automático con el LDR, el cual se ha decidido que funcione mediante polling. Se realizarán 5 medidas cada 300 milisegundos y se efectuará una media. Esa media es la que se comparará con el umbral correspondiente, apagando las luces si hay demasiada luz o viceversa.

Control de persianas

Este módulo encargado de controlar las persianas estará compuesto por una PCB en la que se colocará un botón con resistencia de pull-down y dos finales de carrera con resistencia pull-down también que simularán el tope de arriba y debajo de la persiana. En otra PCB, se ha soldado el integrado L293, cuyo funcionamiento es un puente en H y permitirá a un motor girar en ambos sentidos, simulando así la subida y la bajada de una persiana.

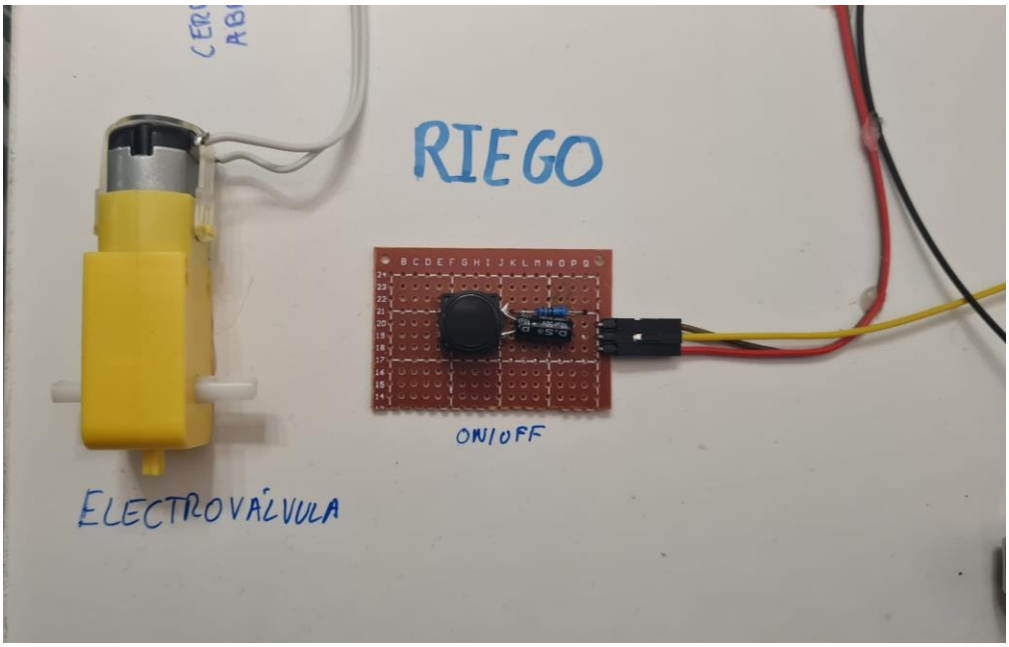
En cuanto al software, la persiana podrá encontrarse en 3 estados: bajando (DOWN), subiendo (UP) o parada (STOP).



Control de riego

Este módulo será el que gestione el riego. Para ello se ha decidido usar un motor que puede girar en ambos sentidos (controlado también por el integrado L293) para que simule el funcionamiento de una electroválvula (un sentido simula la apertura de la electroválvula y el otro simula el cierre) y un botón con resistencia de pull-down que active la apertura o el cierre de esta.

En cuanto al software, se ha utilizado un temporizador, para que el giro ya sea hacia un sentido o hacia el otro tenga 1 segundo de duración. La “electroválvula” podrá encontrarse en varios estados: abierta, cerrada, abriéndose o cerrándose.

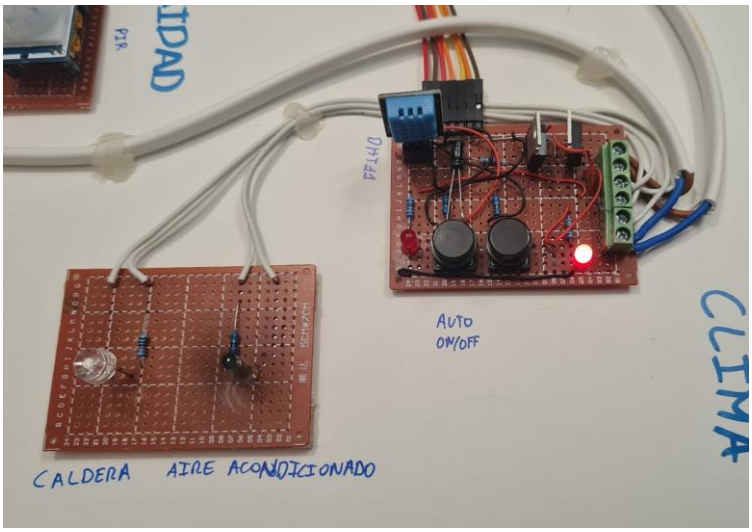


Control de temperatura

Este módulo gestionará el clima de la casa. Para ello se ha soldado en una PCB un sensor de temperatura y humedad (DHT11), dos LEDs que mostrarán el estado de la calefacción o del aire acondicionado y dos botones para activar o desactivar los modos.

Dos transistores se encargarán de actuar como interruptores controlando así en otra PCB un LED que simulará la caldera y una pequeña hélice que simulará el aire acondicionado. Los botones tienen resistencias de pull-down y los LEDs están en serie con sus respectivas resistencias de protección.

En cuanto al software, tendrá dos modos, uno automático que será activado o desactivado mediante la pulsación de uno de los botones y otro manual controlado por otro botón en el que se puede elegir poner la caldera o el aire acondicionado.



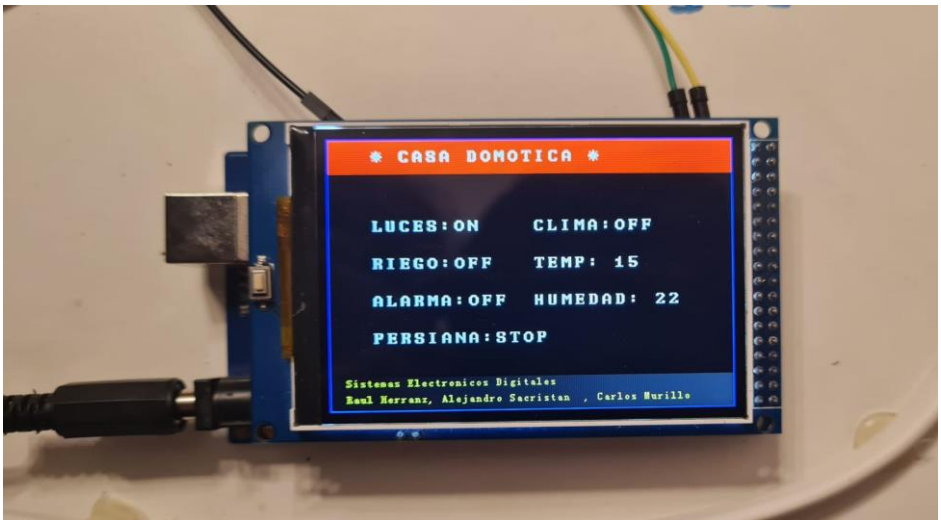
Control de seguridad

Este módulo se encargará de simular una alarma de seguridad de una casa. Para ello, se ha soldado en una PCB un sensor PIR para detectar la presencia o movimiento, un LED para indicar si la alarma está activada o no, un botón para encender o apagar la alarma y un buzzer que emitirá una alarma en caso de estar activada y haber detectado movimiento. Como en el resto de los módulos, el botón está conectado a una resistencia de pull-down, el LED tiene una resistencia de protección y el buzzer está controlado mediante un transistor 2N2222 que actúa como interruptor.

En cuanto al funcionamiento, la alarma se activará 5 segundos después de presionar el botón para dar tiempo a marcharse y en cuanto detecta movimiento, lanza la señal acústica de la alarma, simulada con un buzzer.

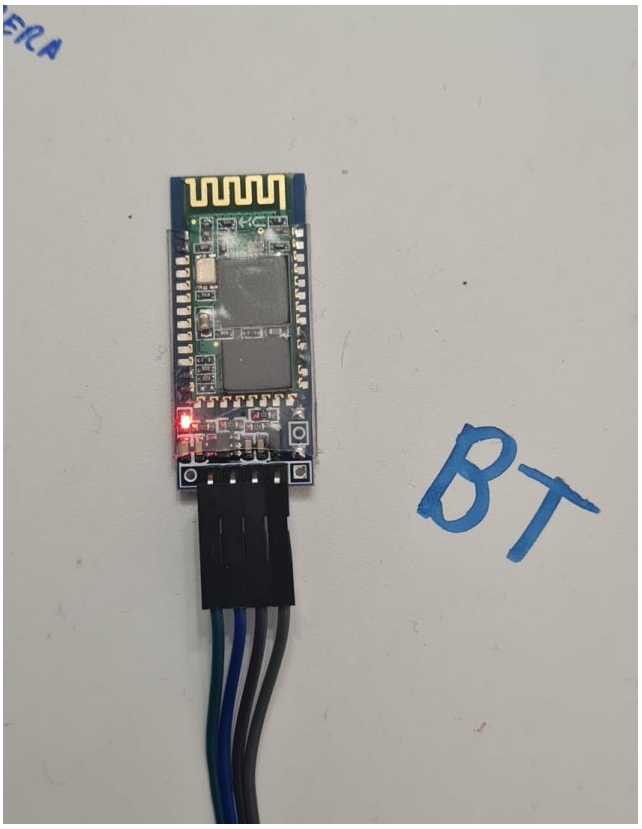
Pantalla TFT

Se utlizará una pantalla TFT de 320x480 píxeles que será controlada mediante un Arduino Mega. Este Arduino Mega, está conectado a la placa STM32 mediante una comunicación serie I2C. Se ha planteado de esta manera porque la pantalla sigue unas librerías muy específicas de Arduino que serían difíciles de traspasar a la placa principal. Además, así se agrega una comunicación serie I2C al proyecto.



Bluetooth

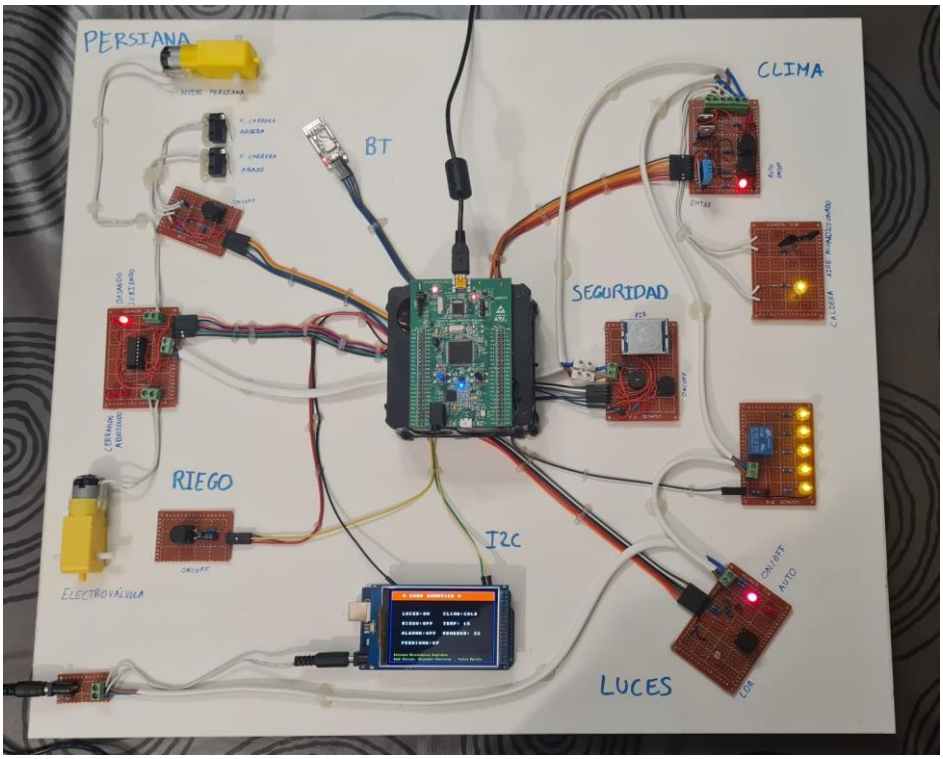
El módulo utilizado será el HC-06 y será programado mediante interrupciones, comunicándose con la placa mediante UART. Las señales serán enviadas mediante una aplicación de móvil consistente en una terminal de comunicaciones.



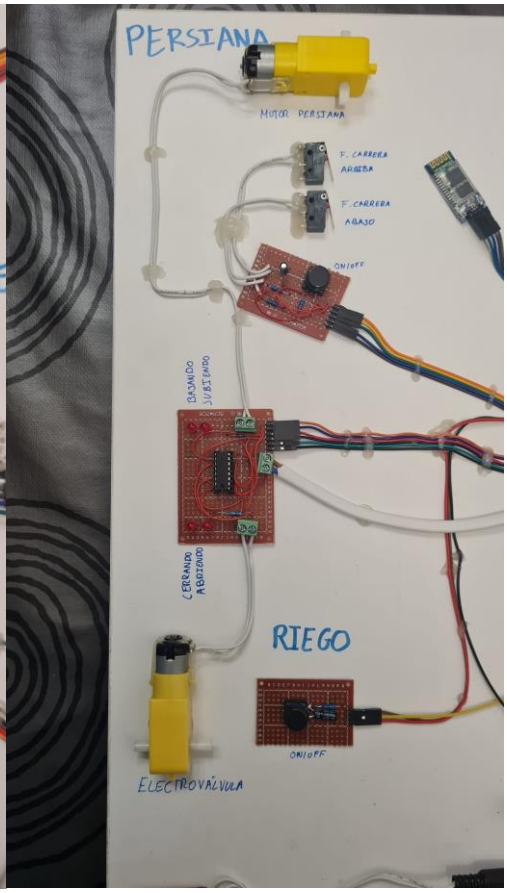
Extras y main

El módulo de controlador de motores no tiene asignado un fichero de cabecera para él solo, sino que se ha programado entre el módulo de control de riego y persianas.

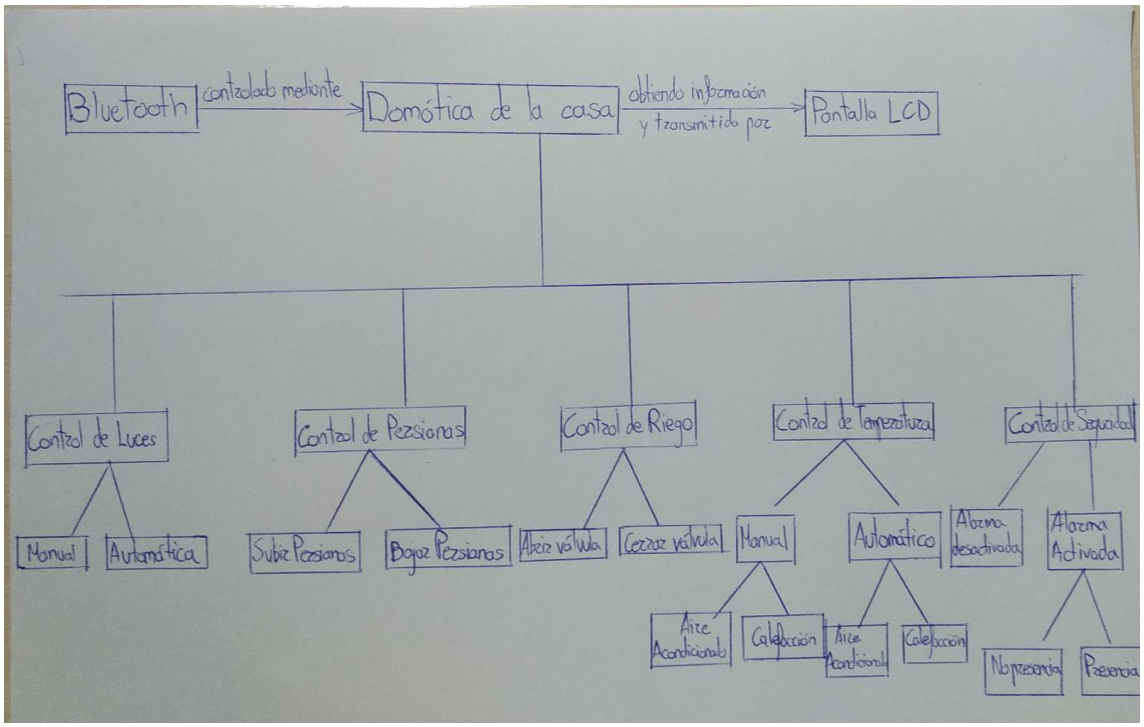
En el “main.c” se gestionarán todas las interrupciones de los botones mediante los Callback correspondiente, para que el sistema diferencie qué botón se está pulsando. Todos ellos cuentan con antirrebotes por software y, solo algunos que daban más problemas, llevan antirrebotes por hardware también, logrando así una respuesta más sólida.


PERSIANA
CLIMA
BT
SEGURIDAD
RIEGO
I2C
ELECTROVÁLVULA
LUCES


I2C


PERSIANA
MOTOR PERSIANA
F. CARRERA ARRIBA
F. CARRERA ABAJO
ON/OFF
RIEGO
ELECTROVÁLVULA

Diagramas



La estructura será bastante clara, realizando y probando cada bloque por separado hasta que se consiga el funcionamiento correcto y así poder incuir todas las funcionalidades de una forma más fácil, eficiente y clara.

Funcionamiento de interfaz y bloques funcionales:

Control de luces

Las luces tendrán dos modos: automático y manual. Al pulsar una vez el botón correspondiente se activará el modo manual y se encenderán los LEDs correspondientes a la simulación de las luces de la casa y se encenderá el LED indicador de que está activado el modo manual. Al pulsar otra vez el botón, se apagará el LED indicador de que está activado el modo manual, encendiéndose el del modo automático y el encendido de los LEDs correspondientes a la simulación de luces de la casa será controlado por el sensor LDR, el cuál por encima de un umbral de luminosidad las mantendrá apagadas y por debajo de él, las encenderá.

La programación se realizará en el fichero “control_luces.h”, que cuenta con varios métodos:

El primer método, es el que controla la máquina de estados de las luces, habiendo tres estados posibles: luces apagadas, luces encendidas y modo automático.

```
void setLuces(int n){
	if(n==0){
		HAL_GPIO_WritePin(GPIOD, GPIO_PIN_11, GPIO_PIN_RESET);
			HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
			HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
	}else if(n==1){
		HAL_GPIO_WritePin(GPIOD, GPIO_PIN_11, GPIO_PIN_SET);
			HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
			HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
	}else{
		HAL_GPIO_WritePin(GPIOD, GPIO_PIN_11, GPIO_PIN_RESET);
		HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
	    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
	}
	estado_luces=n;
	actualizar pantalla(1);
}
```

Relacionado con este método, se puede realizar uno que devuelva el estado de las luces, que será útil para devolver mensajes mediante bluetooth o para mostrar datos en la pantalla:

```
int getEstadoLuces(){
	return estado_luces;
}
```

Con el siguiente método se cambia el estado de las luces consecuentemente:

```
void cambiarEstadoLuces(){
		if(estado_luces==0){
			setLuces(1);
		}else if(estado_luces==1){
			setLuces(2);
		}else if(estado_luces==2){
			setLuces(0);
		}
	}
```

Por último, en el modo automático, se controlarán las luces mediante el LDR, tomando 5 medidas cada segundo:

```
void medirLDR(){
	if(estado_luces==2 && counter_luces>1000){
		counter_luces=0;
		media_ldr=0;
		tickstart_luces=HAL_GetTick();
		int i=0;
		for(i=0;i<5;i++){
			HAL_ADC_Start(&hadc1);
			HAL_ADC_PollForConversion(&hadc1, 100);
			adcval[i]=HAL_ADC_GetValue(&hadc1);
			HAL_ADC_Stop(&hadc1);
			media_ldr+=adcval[i];
		}
		media_ldr=media_ldr/5;
			if(media_ldr>umbral_luces){
				HAL_GPIO_WritePin(GPIOD, GPIO_PIN_11, GPIO_PIN_RESET);
			}else{
				HAL_GPIO_WritePin(GPIOD, GPIO_PIN_11, GPIO_PIN_SET);
			}
	}else{
		counter_luces=HAL_GetTick()-tickstart_luces;
	}
}
```

Control de persianas

El funcionamiento de este bloque comenzará con la pulsación del botón el cual hará girar el motor en un sentido simulando la subida o la bajada de la persiana, si se pulsa

el botón otra vez, el motor se parará y si se vuelve a pulsar a continuación el motor girará en el sentido contrario simulando la operación contraria de la anterior.

Si se pulsa el final de carrera ya sea hacia un sentido (bajar persiana) o hacia el otro (subir persiana), el motor se para y el siguiente giro lo realizará en el sentido que esté permitido. Por ejemplo, si girando hacia la izquierda llega al final de carrera, el siguiente giro será hacia la derecha.

El control de la máquina de estados se realizará mediante la siguiente función:

```
void setEstadoPersianas(int n){
        estado anterior persianas=estado persianas;
        estado_persianas=n;
        if(estado_persianas==0){
                 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
                 HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
        }else if(estado_persianas==1){
                HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET);
                HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);
        }else if(estado_persianas==2){
                HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
                HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_SET);
        }
        actualizar_pantalla(4);
}
```

Y el cambio de estados mediante:

```
void cambiarEstadoPersianas(){
        if(getEstadoPersianas()==0 && getEstadoAnteriorPersianas()==2){
                setEstadoPersianas(1);
        }else if(getEstadoPersianas()==0 && getEstadoAnteriorPersianas()==1){
                setEstadoPersianas(2);
        }else if(getEstadoPersianas()==0 && getEstadoAnteriorPersianas()==0){
                setEstadoPersianas(2);
        }else if(getEstadoPersianas()==1){
                setEstadoPersianas(0);
        }else if(getEstadoPersianas()==2){
                setEstadoPersianas(0);
        }
}
```

Control de riego

El funcionamiento de este módulo se basará en interrupciones mediante el pulsador o bluetooth. El motor permanecerá en reposo hasta que se produce la primera pulsación la cual activará el giro del motor hacia un sentido con la duración de 1 segundo establecido con el temporizador. Tras ese segundo, se detendrá el motor. Este movimiento simulará la apertura de la electroválvula de riego de la casa.

Al volver al pulsar el botón, el motor girará 1 segundo en sentido contrario simulando el cierre de la misma. La operación se repetirá tantas veces como se pulse el botón.

Como funciones a destacar, se muestra la máquina de estados y el temporizador de 1 segundo:

```
void setEstadoRiego(int n){
	estado_anterior_riego=estado_riego;
	estado_riego=n;
	tickstart_riego=HAL_GetTick();
	if(estado_riego==0){
		HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);
		HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
	}else if(estado_riego==1){
		HAL GPIO WritePin(GPIOD, GPIO PIN 12, GPIO PIN RESET);
		HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_SET);
	}
	else if(estado_riego==2){
			HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
			HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, GPIO_PIN_RESET);
		}
	actualizar_pantalla(2);
}
void temporizador_riego(){
	if(estado_riego==0 || estado_riego==1){
		if(HAL_GetTick()-tickstart_riego>1000){
			setEstadoRiego(2);
		}
	}
	}
```

Control de temperatura

Este módulo que controla el clima tendrá dos modos y dos botones, uno asociado para cada modo: manual y automático.

Con una pulsación en el botón manual se encenderá la calefacción es decir el LED que lo representa, con otra pulsación se apagará la calefacción (el LED) y se activará el aire acondicionado, es decir el ventilador que lo representa, finalmente con otra pulsación el sistema permanece en reposo y apagado.

El modo automático se encenderá con una pulsación a su correspondiente botón y con otra pulsación se apagará. Durante su encendido el sensor de temperatura y humedad registrará la temperatura de la sala cada segundo mediante un temporizador y en función de dicha medida, encenderá según sea necesario, el aire acondicionado (ventilador) o la calefacción (LED). El sensor DHT11 utiliza el temporizador interno de la placa TIM6 para leer sus la temperatura y humedad.

Se destacan las funciones del sensor DHT11 y el modo automático:

```
void delay(uint16_t time) {
	__HAL_TIM_SET_COUNTER(&htim6, 0);
	while ((__HAL_TIM_GET_COUNTER(&htim6)) < time)
		;
}

void Set_Pin_Output(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin) {
	GPIO_InitTypeDef GPIO_InitStruct = { 0 };
	GPIO_InitStruct.Pin = GPIO_Pin;
	GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
	GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
	HAL_GPIO_Init(GPIOx, &GPIO_InitStruct);
}

void Set_Pin_Input(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin) {
```

```
	GPIO_InitTypeDef GPIO_InitStruct = { 0 };
	GPIO_InitStruct.Pin = GPIO_Pin;
	GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
	GPIO_InitStruct.Pull = GPIO_PULLUP;
	HAL_GPIO_Init(GPIOx, &GPIO_InitStruct);
}

#define DHT11_PORT GPIOA
#define DHT11_PIN GPIO_PIN_2

void DHT11_Start(void) {
	Set_Pin_Output(DHT11_PORT, DHT11_PIN);  // set the pin as output
	HAL_GPIO_WritePin(DHT11_PORT, DHT11_PIN, 0);   // pull the pin low
	delay(18000);   // wait for 18ms
	HAL_GPIO_WritePin(DHT11_PORT, DHT11_PIN, 1);   // pull the pin high
	delay(20);   // wait for 20us
	Set_Pin_Input(DHT11_PORT, DHT11_PIN);    // set as input
}

uint8_t DHT11_Check_Response(void) {
	uint8_t Response = 0;
	delay(40);
	if (!(HAL_GPIO_ReadPin(DHT11_PORT, DHT11_PIN))) {
		delay(80);
		if ((HAL_GPIO_ReadPin(DHT11_PORT, DHT11_PIN)))
			Response = 1;
		else
			Response = -1; // 255
	}
	while ((HAL_GPIO_ReadPin(DHT11_PORT, DHT11_PIN)))
		;   // wait for the pin to go low

	return Response;
}

uint8_t DHT11_Read(void) {
	uint8_t i, j;
	for (j = 0; j < 8; j++) {
		while (!(HAL_GPIO_ReadPin(DHT11_PORT, DHT11_PIN)))
			;   // wait for the pin to go high
		delay(40);   // wait for 40 us
		if (!(HAL_GPIO_ReadPin(DHT11_PORT, DHT11_PIN)))   // if the pin is low
		{
			i &= ~(1 << (7 - j));   // write 0
		} else
			i |= (1 << (7 - j));  // if the pin is high, write 1
		while ((HAL_GPIO_ReadPin(DHT11_PORT, DHT11_PIN)))
			;  // wait for the pin to go low
	}
	return i;
}

void lectura_dht11() {
	DHT11_Start();
	Presence = DHT11_Check_Response();
	Rh_byte1 = DHT11_Read();
	Rh_byte2 = DHT11_Read();
	Temp_byte1 = DHT11_Read();
	Temp_byte2 = DHT11_Read();
	SUM = DHT11_Read();
	TEMP = Temp_byte1;
	RH = Rh_byte1;
	Temperature = (float) TEMP;
	Humidity = (float) RH;
}

void clima(){
	if(counter_clima>2000){
		counter_clima=0;
		tickstart_clima=HAL_GetTick();
		lectura_dht11();
		actualizar pantalla(6);
		if(controldelclima==1){
			if(Temperature>temperatura_objetivo){
				HAL_GPIO_WritePin(GPIOD, GPIO_PIN_9, GPIO_PIN_RESET);
```

```
                                        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_10,
GPIO_PIN_SET);
                        }else{
                                HAL_GPIO_WritePin(GPIOD, GPIO_PIN_9, GPIO_PIN_SET);
                                        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_10,
GPIO_PIN_RESET);
                        }
                }

        }else{
                counter_clima=HAL_GetTick()-tickstart_clima;
        }

}
```

Control de seguridad

El control de seguridad simulará una alarma de seguridad de una vivienda. Al pulsar el botón de activación transcurrirán 5 segundos y se activará la alarma, lo cual queda indicado con el encendido de un LED que si está encendido quiere decir que la alarma está activa y si está apagado quiere decir que la alarma está apagada. Esta operación está realizada mediante temporizadores.

A continuación, si la alarma está encendida y el sensor de presencia PIR detecta movimiento o presencia activará el buzzer, emitiendo una alarma, lo que significará que habrá intrusos en la casa. Para desactivar la alarma se pulsará el botón una vez esté activada la alarma.

Como función a destacar se muestra la máquina de estados de la alarma y la espera de 5 segundos tras pulsar el botón:

```
void setEstadoSeguridad(int n) {
        estado_seguridad = n;
        if (estado_seguridad == 0) {
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET);
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8,GPIO_PIN_RESET);
                alarma_seguridad = 0;
        } else if (estado_seguridad == 1) {
                alarma_seguridad = 0;
                tickstart_seguridad = HAL_GetTick();
        } else if (estado_seguridad == 2) {
                HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);
        }
        actualizar_pantalla(3);
}

void activar_sensor() {

        if (estado_seguridad == 1) {
                counter_seguridad = HAL_GetTick() - tickstart_seguridad;
                if ((counter_seguridad > 0) && (counter_seguridad < 500)) {
                        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);
                }else if ((counter_seguridad > 500) && (counter_seguridad < 1000)) {
                        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET);
                } else if (counter_seguridad > 5000) {
                        counter_seguridad = 0;
                        setEstadoSeguridad(2);
                }

        }
        }
```

Pantalla TFT

Sin duda, la comunicación con la pantalla ha sido el proceso más tedioso del proyecto. Se ha intentado pasar las funciones de las librerías de la pantalla en Arduino a funciones con el mismo nombre en CubeIDE. Se comunica el Arduino (esclavo) con la STM32 (maestro) mediante I2C. Como ejemplo de uso, algunas de las funciones en CubeIDE son:

```
//Limpiar pantalla:
void clrScr() {
	uint8_t op = 1;
	HAL_I2C_Master_Transmit(&hi2c1, 0x8 << 1, &op, 1, 3000);
}
//Seleccionar Color pantalla LCD
void setColor(int red, int green, int blue) {
	uint8_t color[4] = { 2, red, green, blue };
	HAL_I2C_Master_Transmit(&hi2c1, 0x8 << 1, (uint8_t*) color, 4, 3000);
}
//Seleccionar color fondo pantalla LCD
void setBackColor(int red, int green, int blue) {
	uint8_t color[4] = { 3, red, green, blue };
	HAL_I2C_Master_Transmit(&hi2c1, 0x8 << 1, (uint8_t*) color, 4, 3000);
}
//Dibuja un rectángulo pantalla LCD
void drawRect(int x1, int y1, int x2, int y2) {
	int desb[4] = { 0, 0, 0, 0 };
	if (x1 > 255) {
		desb[0] = 1;
	}
	if (y1 > 255) {
		desb[1] = 1;
	}
	if (x2 > 255) {
		desb[2] = 1;
	}
	if (y2 > 255) {
		desb[3] = 1;
	}
	uint8_t coord[9] = { 4, x1, y1, x2, y2, desb[0], desb[1], desb[2], desb[3] };
	HAL_I2C_Master_Transmit(&hi2c1, 0x8 << 1, (uint8_t*) coord, 9, 3000);
	}
```

Así, el Arduino recibe estos datos y los interpreta como:

```
void receiveEvent(int howMany) {
  int option;
  int cadena[20];
  char t[50];
  int k = 0;
  while (Wire.available()) {
    cadena[k] = Wire.read();
    Serial.print(cadena[k]);
    k++;
  }
  option = cadena[0];
  if (option == 1) {
    myGLCD.clrScr(); //Limpia pantalla
  } else if (option == 2) {
    myGLCD.setColor(cadena[1], cadena[2], cadena[3]);
  } else if (option == 3) {
    myGLCD.setBackColor(cadena[1], cadena[2], cadena[3]);
  } else if (option == 4) {
    for (int i = 5; i < 9; i++) {
      if (cadena[i] == 1) {
        cadena[i - 4] = cadena[i - 4] + 256;
      }
    }
```

Además, la pantalla mostrará los estados en los que se encuentran los diferentes módulos del proyecto y se actualizará cuando uno de estos cambie:

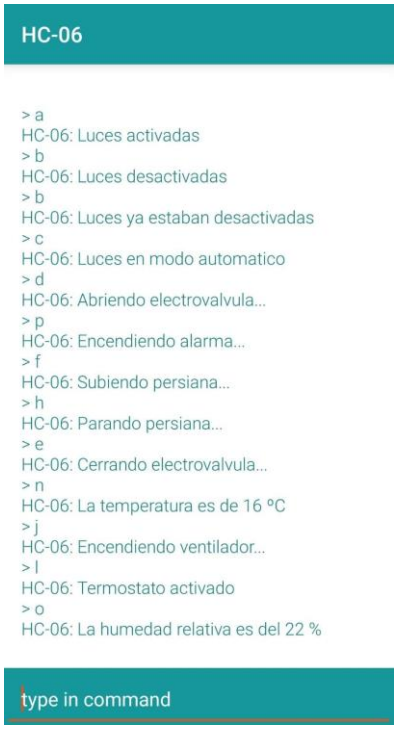
```
void actualizar_pantalla(int act){
	setColor(0, 0, 0);
	if(act==1){
		fillRect(146, 94, 210,110);
		setColor(255, 255, 255);
		setBackColor(0, 0, 0);
			if(getEstadoLuces()==0){
				print("OFF", 146, 94);
			}else if(getEstadoLuces()==1){
				print("ON", 146, 94);
			}else{
				print("AUTO", 146, 94);
			}
	}else if(act==2){
		fillRect(146, 138, 210,154);
		setColor(255, 255, 255);
			if(getEstadoAnteriorRiego()==0){
				print("ON", 146, 138);
			}
			else{
				print("OFF", 146, 138);
			}
	}else if(act==3){
		fillRect(162, 182, 226,198);
		setColor(255, 255, 255);
			if(getEstadoSeguridad()==0){
				print("OFF", 162, 182);
			}
			else{
				print("ON", 162, 182);
			}
	}else if(act==4){
		fillRect(194, 226, 258,242);
		setColor(255, 255, 255);
			if(getEstadoPersianas()==0){
				print("STOP", 194, 226);
			}
			else if(getEstadoPersianas()==1){
				print("DOWN", 194, 226);
			}else if(getEstadoPersianas()==2){
				print("UP", 194, 226);
			}
	}else if(act==5){
		fillRect(336, 94, 400,110);
		setColor(255, 255, 255);
		setBackColor(0, 0, 0);
		if(getControlClima()==1){
			print("AUTO", 336, 94);
		}else if(getEstadoClima()==0){
				print("OFF", 336, 94);
			}else if(getEstadoClima()==1){
				print("CALD", 336, 94);
			}else if(getEstadoClima()==2){
				print("AIRE", 336, 94);
			}
	}else if(act==6){
		fillRect(336, 138, 400,154);
		setColor(255, 255, 255);
		setBackColor(0, 0, 0);
		char bufff[2];
		sprintf(bufff, "%i", (int) Temperature);
		print(bufff, 336, 138);

		setColor(0, 0, 0);
		fillRect(384, 182, 470,198);
		setColor(255, 255, 255);
		setBackColor(0, 0, 0);
		sprintf(bufff, "%i", (int) Humidity);
		print(bufff, 384, 182);
	}
	}
```

Bluetooth

El módulo HC-06 está programado mediante interrupciones. Cuando se manda un caracter mediante la terminal en el móvil, se ejecuta la orden y se responde con la acción ejecutada, por ejemplo, para el caso de las luces:

```
void bluetooth(char recibido[]) {
	if (strcmp(recibido, "a") == 0) {
		if (getEstadoLuces() == 1)
			HAL_UART_Transmit(&huart6, (uint8_t*) tx_buffer,
					sprintf(tx_buffer, "Las luces ya estaban
activadas\n"),
					500);
		else {
			setLuces(1);
			HAL_UART_Transmit(&huart6, (uint8_t*) tx_buffer,
					sprintf(tx_buffer, "Luces activadas\n"), 500);
		}
	} else if (strcmp(recibido, "b") == 0) {
		if (getEstadoLuces() == 0)
			HAL_UART_Transmit(&huart6, (uint8_t*) tx_buffer,
					sprintf(tx_buffer, "Luces ya estaban
desactivadas\n"), 500);
		else {
			setLuces(0);
			HAL_UART_Transmit(&huart6, (uint8_t*) tx_buffer,
					sprintf(tx_buffer, "Luces desactivadas\n"), 500);
		}
	} else if (strcmp(recibido, "c") == 0) {
		if (getEstadoLuces() == 2)
			HAL_UART_Transmit(&huart6, (uint8_t*) tx_buffer,
					sprintf(tx_buffer, "Luces ya estaban en modo
automatico\n"),
					500);
		else {
			setLuces(2);
			HAL_UART_Transmit(&huart6, (uint8_t*) tx_buffer,
					sprintf(tx_buffer, "Luces en modo automatico\n"),
500);
			}
```

La interfaz con el usuario sería la siguiente:



Estos son todos los comandos disponibles:

Orden	Acción	Módulo
a	Activar luces	Luces
b	Desactivar luces	
c	Poner luces en modo automático	
d	Regar (abrir electroválvula)	Riego
e	Parar de regar (cerrar electroválvula)	
f	Subir persiana	Persianas
g	Bajar persiana	
h	Parar persiana	
i	Encender calefacción	Clima
j	Encender ventilador	
k	Apagar calefacción o ventilador	
l	Activar termostato	
m	Desactivar termostato	
n	Consultar temperatura actual	
o	Consultar humedad relativa actual	
p	Encender alarma	Seguridad
q	Apagar alarma	

Extras y main

En el código principal del bucle while, se encuentran las inetrrupciones producidas por los botones, así como las acciones que se realizarán por haberlos pulsado y otras funciones que necesitan estar ejecutandose en cada ciclo del while.

De esta manera, queda un prgrama principal fácil de entender:

```
while (1) {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
                switch (boton_presionado) {
                case 3:
                        if (debouncer(&button_int, GPIOA, GPIO_PIN_3)) {
                                cambiarEstadoLuces();
                        }
                        break;
                case 12:
                        if (debouncer(&button_int, GPIOE, GPIO_PIN_12)) {
                                setEstadoPersianas(0);
                        }
                        break;
                case 13:
                        if (debouncer(&button_int, GPIOA, GPIO_PIN_13)) {
                                cambiarEstadoPersianas();
                        }
                        break;
                case 10:
                        if (debouncer(&button_int, GPIOA, GPIO_PIN_10)) {
                                cambiarEstadoRiego();
                        }
                        break;
                case 7:
                        if (debouncer(&button_int, GPIOA, GPIO_PIN_7)) {
                                cambiarEstadoSeguridad();
                        }
                        break;
```

```
                case 9:
                        if (debouncer(&button_int, GPIOA, GPIO_PIN_9)) {
                                intruso_detectado();
                        }
                        break;
                case 14:
                        if (debouncer(&button_int, GPIOB, GPIO_PIN_14)) {
                                cambiarControlClima();
                        }
                        break;
                case 15:
                        if (debouncer(&button_int, GPIOB, GPIO_PIN_15)) {
                                cambiarEstadoClima();
                        }
                        break;
                }

                luces();
                riego();
                seguridad();
                clima();
        }
  /* USER CODE END 3 */
}
```

Las funciones de los antirrebotes, son las vistas en el laboratorio:

```
volatile int button_int = 0;
int boton_presionado = 0;
//Antirrebotes botones:
int debouncer(volatile int *button_int, GPIO_TypeDef *GPIO_port,
                uint16_t GPIO_number) {
        static uint8_t button_count = 0;
        static int counter = 0;

        if (*button_int == 1) {
                if (button_count == 0) {
                        counter = HAL_GetTick();
                        button_count++;
                }
                if (HAL_GetTick() - counter >= 20) {
                        counter = HAL_GetTick();
                        if (HAL_GPIO_ReadPin(GPIO_port, GPIO_number) != 1) {
                                button_count = 1;
                        } else {
                                button_count++;
                        }
                        if (button_count == 4) { //Periodo antirebotes
                                button_count = 0;
                                *button_int = 0;
                                return 1;
                        }
                }
        }
        return 0;
}
```