
Licenciatura em Engenharia de Sistemas Informáticos

No âmbito da UC de Sistemas Embebidos em Tempo Real

# *Home Automation*

André Cardoso & Leonel Fernandes
18848 & 18850

Barcelos, 15 de janeiro de 2022

# Contents

# Contextualização

Em contexto da Unidade Curricular de Sistemas Embebidos em Tempo Real é proposto o desenvolvimento de soluções no tema de *Home Automation* - Automação do Lar.

A solução desenvolvida deve incorporar os seguintes sistemas:

- Sistema A – Controlo de Iluminação Interior
- Sistema B – Controlo de Climatização
- Sistema C – Sistema Acesso ao Estacionamento
- Sistema D – Sistema de Segurança[1]

Adicionalmente, a solução deve implementar as seguintes funções:

- Sensor Ultrassônico para maior precisão.
- Interrupt.
- Multitasking utilizando **RTOS**[2] ou Timer.
- Analisar **performance do programa**.

[1] Alarme de Segurança
[2] Real Time Operating System

# Introdução

A execução do trabalho prático proposto requer a utilização da plataforma Arduino. Foram utilizados, assim como diversos outros componentes compatíveis com a plataforma, de forma a resolver os 4 sistemas e as variadas funções adicionais requeridas.

Para o Sistema de controlo de iluminação interior, o Arduino deverá ser capaz de analisar os valores lidos de um fotorresistor, para assim alterar a intensidade de um LED.

Para o Sistema de controlo de Climatização, o Arduino deverá ser capaz de analisar os valores lidos de um sensor de temperatura e, através da utilização de um display LCD, apresentá-los de forma legível.

Para o Sistema de Acesso ao Estacionamento, o Arduino deverá ser capaz de analisar o input de um utilizador através de um comando de infravermelhos e assim alterar o posicionamento de um motor Servo, sendo que também deverá ser possível suspender o movimento do mesmo.

Para o Sistema de Segurança, o Arduino deverá ser capaz de analisar o input de um sensor de movimento e de seguida acionar um alarme sonoro.

Será ainda necessário, devido às restrições em termos de Hardware, a junção de sistemas por pares para execução em simultâneo das suas funções.

Todos estes sistemas deverão ter em conta as funções adicionais de necessária implementação em pelo menos um dos sistemas.

# Análise de Requisitos

## Requisitos Não Funcionais

Para este projeto verificamos que os Requisitos Não Funcionais se aplicam a todos os sistemas desenvolvidos, sendo eles:

- Processamento em Tempo Real
- Consumo energético inferior a 5 Watts

## Requisitos Funcionais

### Sistema A

- Deteção da Intensidade de Luz Solar
- Regulação da Intensidade de Luz Artifical[1] em função da Intensidade da Luz Solar

### Sistema B

- Deteção da Temperatura Ambiente
- Regulação da Temperatura Ambiente
  - Através de Ventilação[2]
  - Em função da Temperatura Ambiente
- Enunciar o Estado da Ventilação
  - LED de cor Verde - Temperatura 'Estável'
  - LED de cor Vermelha - Arrefecimento Ambiente

[1] LED
[2] Ventoínha

- Apresentação por LCD[3] de:
  - Estado da Ventoínha (ON/OFF)
  - Última Temperatura Lida
- Controlo de Luminosidade do LCD

### Sistema C

- Controlo Remoto de uma Barra de Acesso a Parque de Estacionamento
  - Abertura
  - Fecho
- Capacidade de Suspensão da Atividade atual da Barra
- Deteção de Movimento - Sensor Ultrassônico

### Sistema D

- Deteção de Movimento Intrusivo
- Sinalização de Intrusões
  - Sonora - Alarme
  - Visual - LED Intermitente
- Capacidade de desarmar o Alarme

[3]Liquid Crystal Display

# Especificação do Sistema

## Hardware - Arduino Uno R3


Figura 2.1: Arduino Uno R3

O design da PCB do Arduino Uno utiliza componentes SMD[1].

O micro-controlador inserido nesta placa é o **ATmega328** - MCU[2] da família AVR[3] e Arquitetura Harvard (armazena o programa e as variáveis em unidades diferentes.). É um dispositivo de 8-bits, isto é, a sua arquitetura é capaz de gerir 8 sinais em paralelo.

Este chip tem 3 tipos de memória:

- **Flash**: 32kB. Utilizada para armazenar o código compilado da aplicação, dado ser memória **Não Volátil**.
- **SRAM**: 2kB. Armazenamento de variáveis. **Volátil**.
- **EEPROM**: 1kB. Armazenamento de dados, que necessitam persistência. **Não Volátil**.

[1]Surface Mount Device
[2]Micro Controller Unit
[3]Alf and Vegard's RISC

Para armazenamento do código compilado, dado que a MCU não é capaz de comunicar diretamente por USB, precisa de uma 'ponte' que converta os sinais do host USB para a interface UART[4] - o **ATmega16U2**.

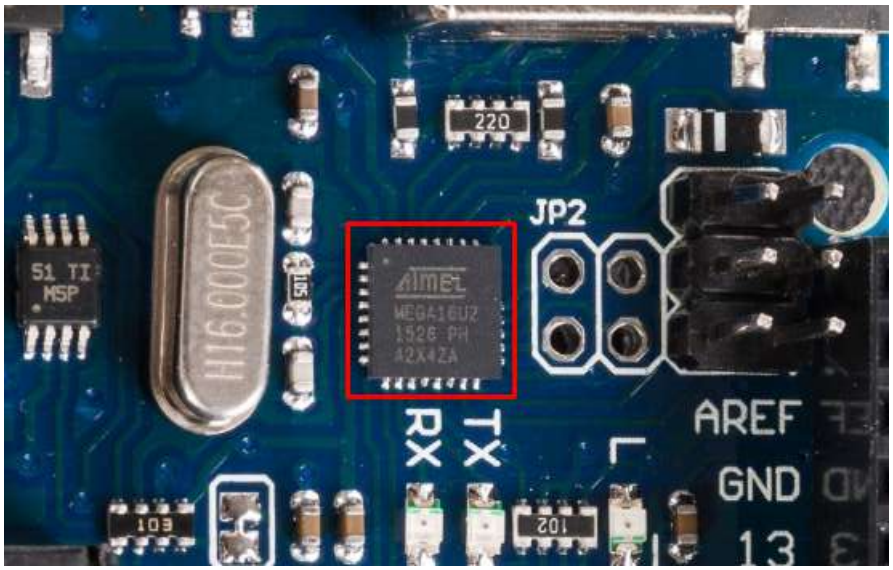



Figura 2.2: ATmega16U2

## Software - C/C++

Para facilidade de uso pelos consumidores o *chipset* (ATmega328) é previamente programado com um ***bootloader***. Este é basicamente o controlador do processo de 'arranque' da placa Arduino.

Dado que o código é desenvolvido numa arquitetura diferente da arquitetura destino (x86_x64 para Arm), o código é compilado em *cross-compiling* e denominado de ***firmware***. É carregado via USB como discutido anteriormente e apenas uma vez, dada a persistência da memória.

[4]Universal Asynchronous Receiver-Transmiter

Exemplo de *cross-compiling* de programa(em ambiente Linux baseado em Debian/Ubuntu).

```
// Install packages
sudo apt-get install gcc-arm-linux-gnueabi
g++-arm-linux-gnueabi

// Compile
arm-linux-gnueabi-gcc helloworld.c -o helloworld-arm
-static
// Run
./helloworld-arm
// Read Architecture
readelf -h helloworld-arm
```

# Modelo de Conceção

Para desenvolvimento do projeto em mãos consideramos que o mais lógico seria seguir um modelo de desenvolvimento 'Waterfall'.

O modelo Waterfall obedece a seguinte estrutura:

1. Requisitos
2. Arquitetura
3. Codificação
4. Teste
5. Manutenção

Apesar de ser um modelo de conceção pouco utilizado atualmente, dado o nosso caso de uso acreditamos que se enquadra perfeitamente dado que:

- Não haverá alteração de Requisitos
- O Hardware já está em nossa posse
  (Sendo a **necessidade de assunção** deste uma das desvantagens apontadas)

# Hardware

## Sistema A – Controlo de Iluminação Interior

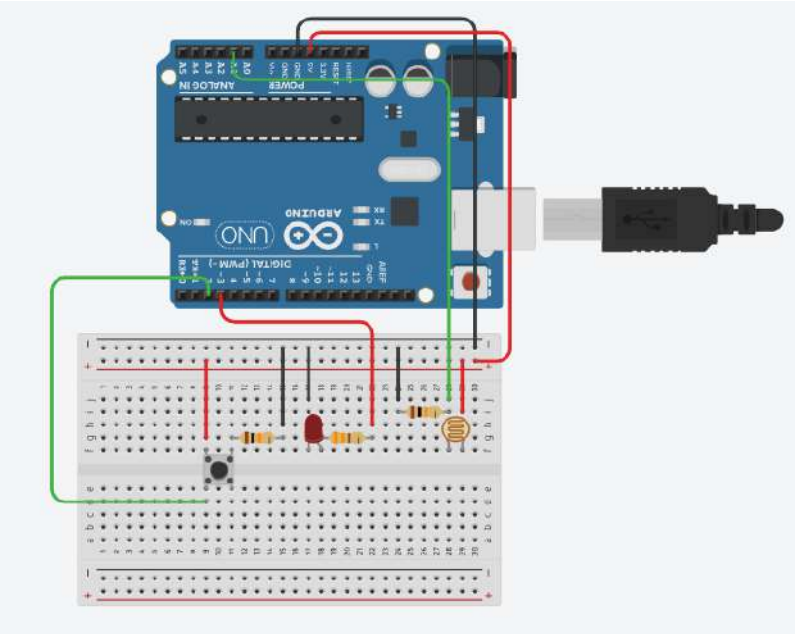


Figura 4.1: Esquema do Sistema A

| Componentes | Quantidade |
|---|---|
| Fotorresistor | 1x |
| Botão | 1x |
| Arduino | 1x |
| Breadboard | 1x |
| LED | 1x |
| Resistência 330 Ohm | 1x |
| Resistência 10k Ohm | 2x |

Figura 4.2: Componentes Utilizados

A resistência de 330 Ohm encontra-se ligada ao LED.

As duas resistências de 10k Ohm encontram-se ligadas ao fotorresistor e ao botão.

O Arduino recebe o input do fotorresistor através do pino analógico A1.

O LED está ligado à porta nº 3 que serve como output.

O Arduino recebe o input do botão através da porta nº 2. O Botão serve de interrupt do sistema.

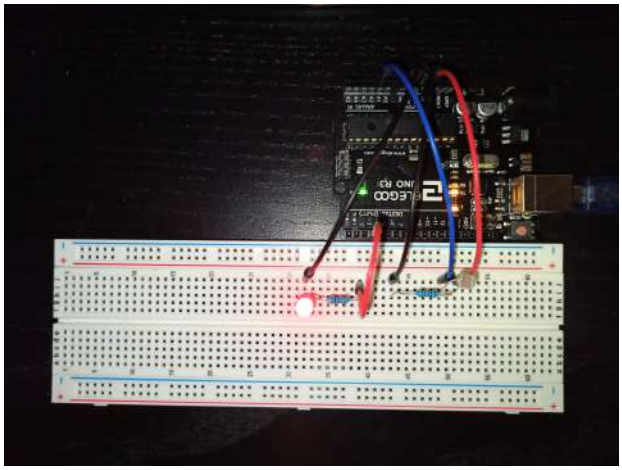


Figura 4.3: Esquema montado do Sistema A

# Sistema B – Controlo de Climatização

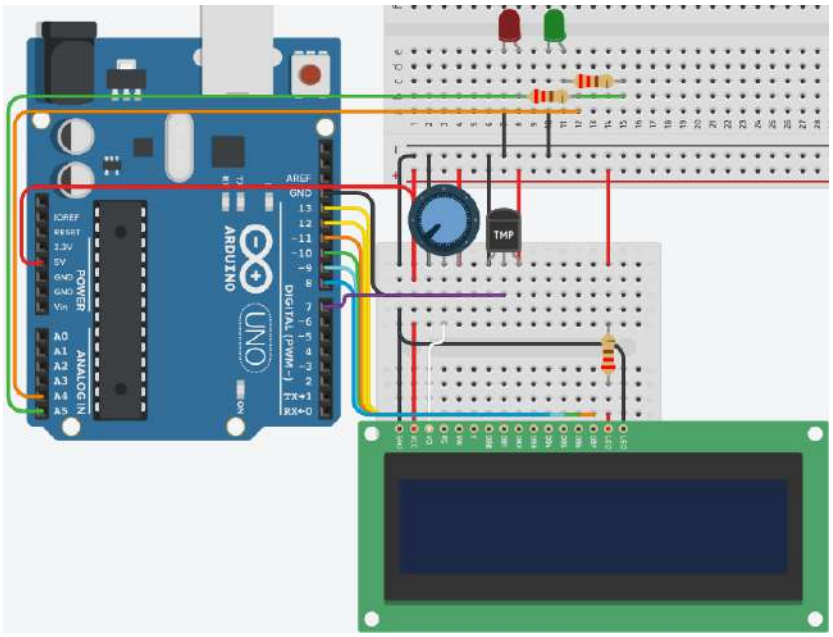



Figura 4.4: Esquema do Sistema

| Componentes | Quantidade |
|---|---|
| Arduino | 1x |
| LCD 16x2 | 1x |
| Potenciómetro 10kΩ | 1x |
| Sensor de Temperatura DHT11 | 1x |
| LED Vermelho | 1x |
| LED Verde | 1x |
| Resistência 220Ω | 3x |

Tabela 4.1: Lista dos Componentes

O LCD está ligado à placa Arduino do seguinte modo:
(Da esquerda para a direita do LCD)

- GCC – GND
- VCC – 5V
- V0 – Pino Analógico do Potenciómetro
- RS – 13
- RW – GND
- E – 12
- D0 > D3 – Desconectados
- D4 – 8
- D5 – 9
- D6 – 10
- D7 – 11

O sensor de temperatura, que no esquema físico corresponde ao modelo DHT11, encontra-se ligado à porta digital nº 7.

Os LED's Verde e Vermelho encontram-se ligados às portas analógicas 5 e 4 respetivamente.

## Sistema C – Sistema Acesso ao Estacionamento

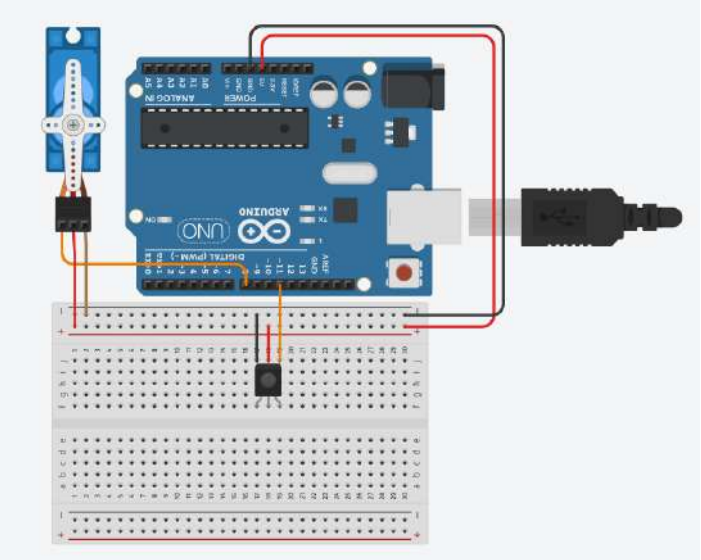


Figura 4.5: Esquema do Sistema C

| Componentes | Quantidade |
|---|---|
| Sensor Infravermelhos | 1x |
| Motor Servo | 1x |
| Arduino | 1x |
| Breadboard | 1x |
| Comando Infravermelhos | 1x |

Figura 4.6: Componentes Utilizados

O Sensor infravermelhos encontra-se ligado à porta nº 11 para envio dos dados.

O motor Servo encotra-se ligado à porta nº 8 para ser controlado pelo arduino.

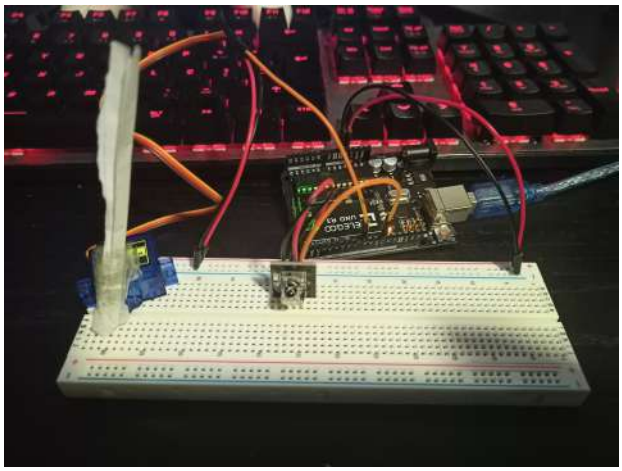


Figura 4.7: Esquema montado do Sistema A

## Sistema D – Sistema de Segurança

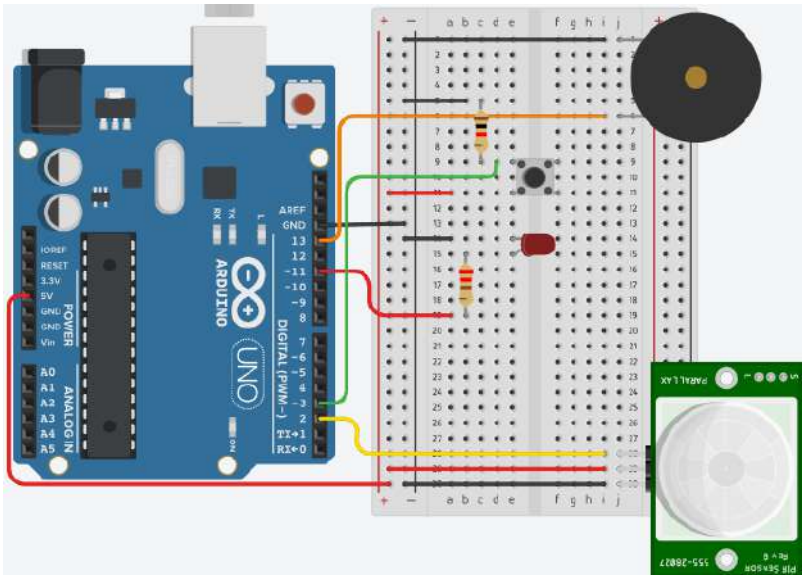



Figura 4.8: Esquema do Sistema

| Componentes | Quantidade |
|---|---|
| Arduino | 1x |
| LED Vermelho | 1x |
| Alarme Sonoro | 1x |
| Botão | 1x |
| Resistência 220Ω | 1x |
| Resistência 1kΩ | 1x |

Tabela 4.2: Lista dos Componentes

O botão encontra-se ligado à porta digital nº 3.

O sensor PIR está ligado à porta digital nº2.

O LED Vermelho encontra-se ligado à porta analógica nº1.

As portas 2 e 3 apresentam capacidade de Interrupt.

## Sistema A + C

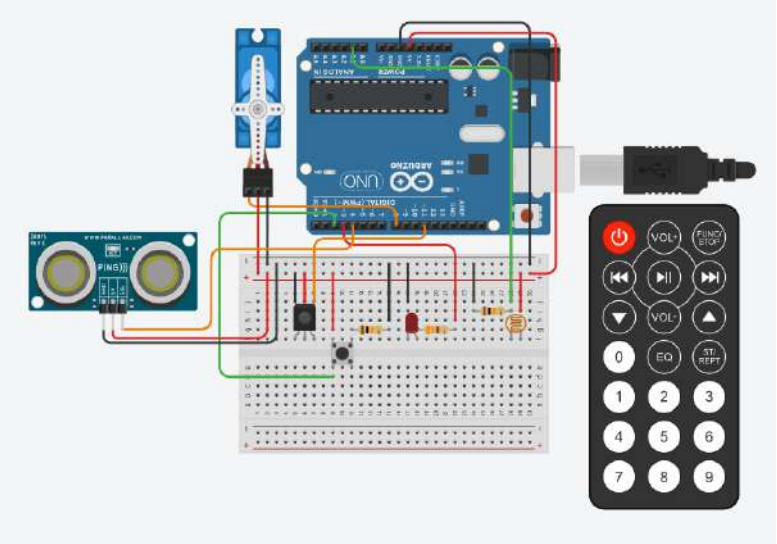


Figura 4.9: Esquema do Sistema

| Componentes | Quantidade |
|---|---|
| Arduino | 1x |
| LED Vermelho | 1x |
| Resistência 330Ω | 1x |
| Resistência 10kΩ | 2x |
| Botão | 1x |
| Sensor IR | 1x |
| Comando IR | 1x |
| Sensor Ultrassónico | 1x |
| Fotorresistor | 1x |
| Servomotor | 1x |

Tabela 4.3: Lista dos Componentes

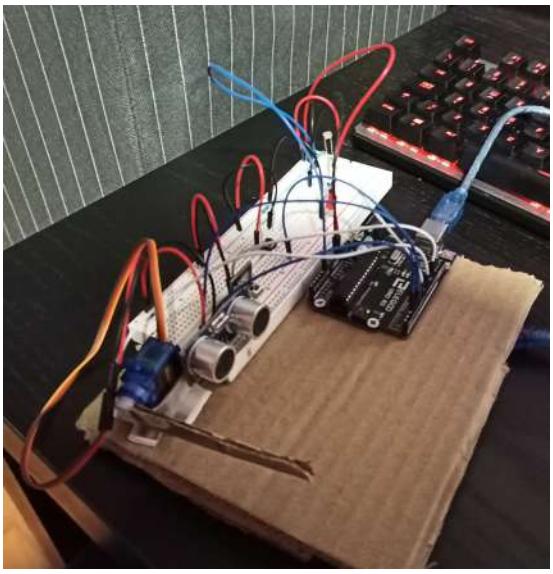


Figura 4.10: Sistema Montado

## Sistema B + D

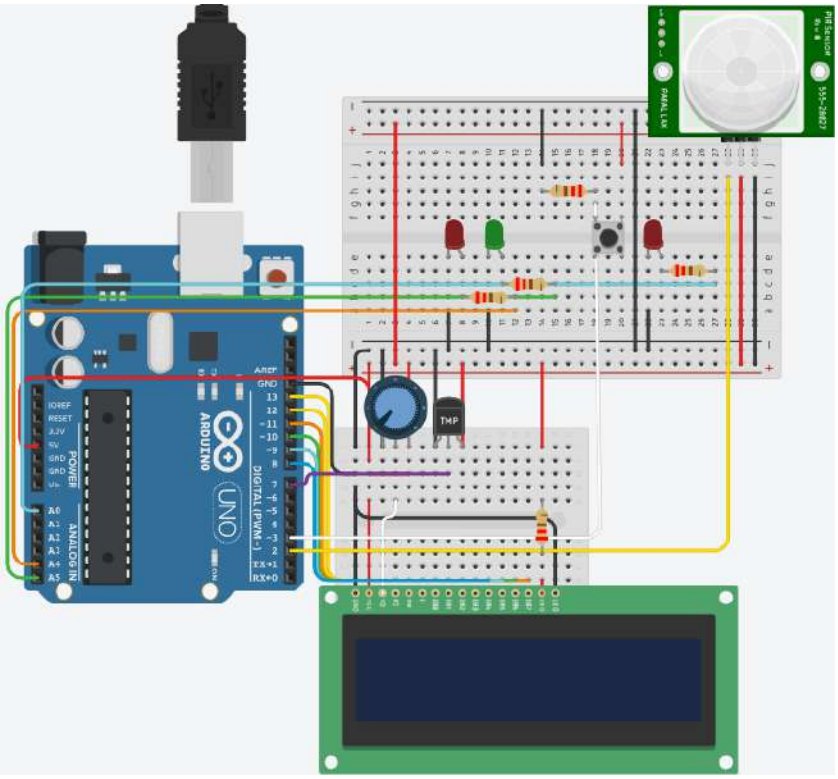



Figura 4.11: Esquema do Sistema

# Software

## Sistema A – Controlo de Iluminação Interior

O Sistema A tem como objetivo o controlo de um LED através de um fotorresistor. Foi ainda incluído um interrupt que fuciona através de um botão que desabilita o funcionamento do LED. O código elaborado foi o seguinte:

```
void loop() {
  //Ler o valor de luz que o sensor deteta
  readValue = analogRead(sensorPin);  1
  //Caso o valor de luz seja inferior a 200
  if (readValue < 200){

    //Desligar o LED
    analogWrite(ledPin, 0);  2

  }
  //Caso o valor de luz seja entre 200 e 500
  else if(readValue >= 200 && readValue < 500){

    //Ligar o LED com 1/4 da sua potência
    analogWrite(ledPin, 64);  3

  }
  //Caso o valor de luz seja entre 500 e 800
  else if(readValue >= 500 && readValue < 800){

    //Ligar o LED com 1/2 da sua potência
    analogWrite(ledPin, 128);  4

  }
  //Caso o valor de luz seja superior a 800
  else{

    //Ligar o LED com potência total
    analogWrite(ledPin, 255);  5

  }
}
```

Figura 5.1: Código para os níveis o LED

- 1 - O valor do Fotorresistor é lido através da função analogRead no pino onde está ligado o mesmo.
- 2,3,4,5 - Para o caso do LED estar compreendido dentro dos valores do "if", o mesmo é acendido com diferentes valores de potência.

```
pinMode(buttonPin, INPUT_PULLUP);
attachInterrupt(0, myISR, HIGH); // Interrupt
```

Figura 5.2: Código para o setup do interrupt


Figura 5.3: Código para a função do interrupt

- 1 - O LED é desligado durante o interrupt.
- 2 - Enquanto o botão se encontra pressionado, este "while" mantém a função de interrupt a correr.

## Sistema B – Controlo de Climatização

É requisitado que dada uma temperatura, obtida através do sensor DHT, e um intervalo de temperaturas máxima e mínima (25 e 20, respetivamente), o sistema seja capaz de manter uma temperatura ambiente controlada com recurso a uma ventoínha.

Como tal, sempre que a temperatura ultrapassar a temperatura **máxima**, a ventoínha deve ser acionada até que a temperatura registada seja inferior à temperatura **mínima**.

```
// Program Flow
void loop() {
  float temp;
  temp = temp_control();
  lcd_update(temp);
//  Delay for readability and stability
  delay(1000);
//  Free memory to avoid overflow
  free(&temp);
}
```

Figura 5.4: Função loop()

Esta função controla o 'flow' do Programa. Como se pode verificar, está definido o controlo de temperatura seguido da atualização do LCD. Por fim, um delay para estabilidade das leituras e apresentação do LCD.

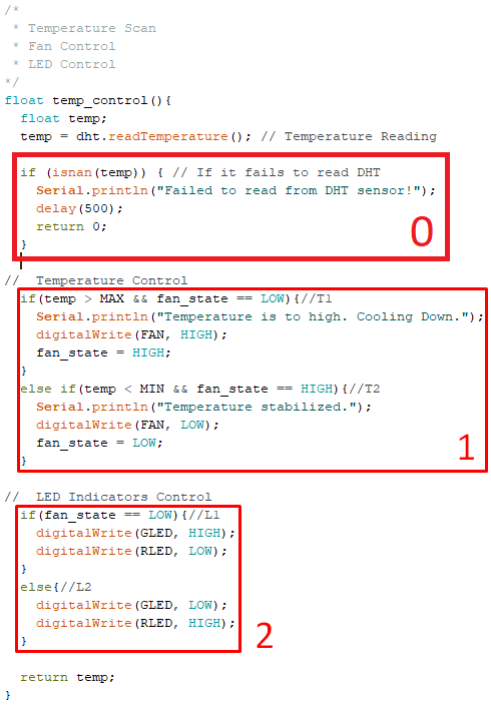



Figura 5.5: Função temp_control()

0. Falha de leitura da temperatura pelo sensor DHT
1. Definição do estado da ventoínha em função da temperatura observada.
    - Transição entre estado **estável** e **arrefecimento**
    - Transição entre estado **arrefecimento** e **estável**
2. Definição do estado do sistema utilizando LEDs em função do estado da ventoínha.

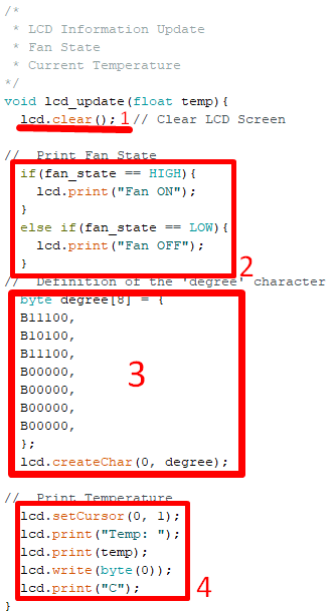



Figura 5.6: Função lcd_update()

1. Limpeza do ecrã do LCD e consequente reposiçãodo cursor.
2. Apresentação do estado atual da Ventoínha.
3. Definição do caracter ‘º’.
4. Apresentação da última temperatura lida.

## Sistema C – Sistema Acesso ao Estacionamento

O Sistema C tem como objetivo o controlo de um motor servo através de um comando de infravermelhos. O motor servo servirá como "portão". Deverá ser possível levantar e baixar o portão, assim como suspender o seu movimento. O programa apresenta ainda "multitasking" por "timer", para ser possível ler os valores do comando e alterar a posição do servo simultaneamente. O código elaborado foi o seguinte:

```
// Códigos dos botões utilizados no comando IR
#define desliga -23971
#define cima -28561
#define baixo -8161
```

Figura 5.7: Código para o comando IR

```
// Pino para Sensor Infravermelho
#define RECV_PIN 11
IRrecv irrecv(RECV_PIN);
decode_results results; // Definir a variável para os dados do recetor
```

Figura 5.8: Definição do sensor IR

```
// Função para ler sensor infravermelho
void remoteReadFunction(){
  if(IrReceiver.decode(&results)){
      code = results.value;
      Serial.println(code);
      switch(code)
      {
      case cima:
        if(motor.read() >= 0 && motor.read() <= 90){
          state = 1;
          Serial.println("sobe");
        }
        break;
      case baixo:
        if(motor.read() >= 0 && motor.read() <= 90){
          state = 0;
          Serial.println("desce");
        }
        break;
      case desliga:
          state = 2;
          break;
      default:
        break;
      }
      irrecv.resume();
    }
}
```

1

2

Figura 5.9: Função de leitura de valor do Sensor IR

- 1 - Leitura do valor recebido pelo sensor de Infravermelhos.
- 2 - Caso o código seja um dos códigos definidos a variável "state" é alterada, esta variável serve para controlar o funcionamento do servo.

```
//Função para alterar posição do Servo
void motorWriteFunction(){
  if (state == 1 && motor.read()<= 89){
      motor.write(motor.read() + 1);
      Serial.println(motor.read());
    }
    else if (state == 0 && motor.read()>= 1){
      motor.write(motor.read() - 1);
      Serial.println(motor.read());
    }
}
```

Figura 5.10: Função para controlo do Servo

Esta função serve para alterar a posição do servo em 1 grau por cada iteração do programa, apenas no caso do "state" se encontrar igual a "0" ou "1".

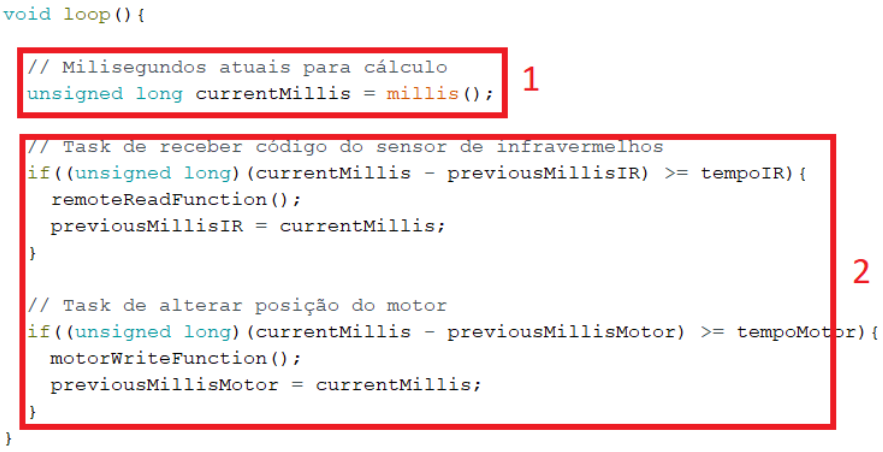
```
// - Multitasking -
// Milisegundos Anterior
unsigned long previousMillisIR = 0;
unsigned long previousMillisMotor = 0;
// Intervalos de tempo
int tempoIR = 10;
int tempoMotor = 50;
// Funções
void remoteReadFunction();
void motorWriteFunction();
```

1

2

Figura 5.11: Definições para multitasking

- 1 - Definidos os "delays" entre iterações de cada uma das funções.
- 2 - As funções para multitasking encontram-se aqui definidas.




```
void loop(){

  // Milisegundos atuais para cálculo
  unsigned long currentMillis = millis();

  // Task de receber código do sensor de infravermelhos
  if((unsigned long)(currentMillis - previousMillisIR) >= tempoIR){
    remoteReadFunction();
    previousMillisIR = currentMillis;
  }

  // Task de alterar posição do motor
  if((unsigned long)(currentMillis - previousMillisMotor) >= tempoMotor){
    motorWriteFunction();
    previousMillisMotor = currentMillis;
  }
}
```

Figura 5.12: Loop multitasking

- 1 - São verficados sempre os milisegundos atuais para execução das funções por timer.
- 2 - Execução das funções apenas se se encotrarem no intervalo de tempo correto.

## Sistema D – Sistema de Segurança

É fundamental para o sistema que a cada movimento detetado seja 'disparado' um alarme sonoro em conjunto com um LED intermitente. Mais ainda, deve ser possível terminar o alarme com o pressionar de um botão.

Como tal, considerando a importância de cada ação, foi decidida a implementação de ambas em torno do conceito de interrupção do sistema.

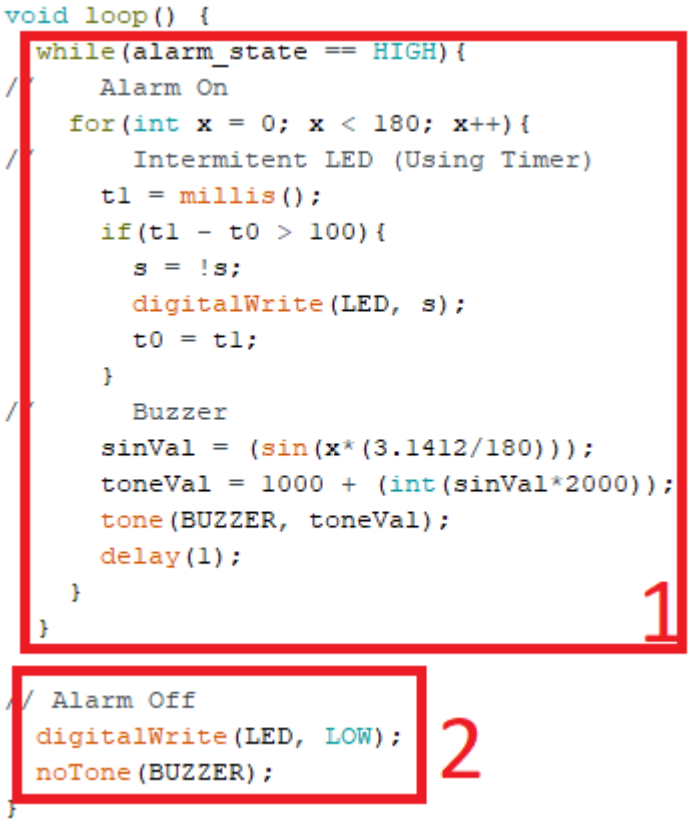



Figure 5.13: Função loop()

Podemos verificar que o Programa apenas executa:

1. A funcionalidade de alarme, quando o alarme estiver acionado
2. Suspensão de alarme

As seguintes funções são chamadas por interrupts acionados pelo Sensor PIR e um botão, respetivamente.

```
/*
 * Interrupt Functions Definitions
*/

// Set Alarm On
void AlarmOn(){
  if(alarm_state == LOW){
    Serial.println("Alarm On");
    alarm_state = HIGH;
  }
}                                    1

// Set Alarm Off
void AlarmOff(){
  if(alarm_state == HIGH){
    Serial.println("Alarm Off");
    alarm_state = LOW;
  }
}                                    2
```

Figure 5.14: Funções de Chamadas de Interrupt

1. **AlarmOn()** – Chamada por movimento detetado pelo Sensor.
2. **AlarmOff()** – Chamada por pressão de botão.

## Sistema A + C

Para junção dos sistemas A e C em termos de código foi apenas utilizado multitasking por timer para poder fazer correr as funções de ambos os sistemas em sincronia. Foi ainda adicionado um sensor ultrassónico para poder fazer a paragem do servomotor.

```
void loop(){

  // Milisegundos atuais para cálculo
  unsigned long currentMillis = millis();

  // Task de receber código do sensor de infravermelhos
  if((unsigned long)(currentMillis - previousMillisIR) >= tempoIR){
    remoteReadFunction();
    previousMillisIR = currentMillis;
  }

  // Task de alterar posição do motor
  if((unsigned long)(currentMillis - previousMillisMotor) >= tempoMotor){
    motorWriteFunction();
    previousMillisMotor = currentMillis;
  }

  // Task de controlo do LED
  if((unsigned long)(currentMillis - previousMillisControlLed) >= tempoLed){
    ledControlFunction();
    previousMillisControlLed = currentMillis;
  }

  // Task de controlo do Sensor Ultrassonico
  if((unsigned long)(currentMillis - previousMillisUltrassonic) >= tempoUltrassonic){
    ultrassonicFunction();
    previousMillisUltrassonic = currentMillis;
  }

}
```

Figura 5.15: Loop Multitasking

```
// Função Sensor Ultrassonico
void ultrassonicFunction(){
  // Verifica a distancia ao sensor ultrassonico
  digitalWrite(triggPin, LOW);
  delayMicroseconds(2);
  digitalWrite(triggPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggPin, LOW);
  duration = pulseIn(echoPin, HIGH);

  // Caso a distancia seja menor que o valor 250 para o servo
  if(duration <= 250){
    state = 2;
  }
}
```

Figura 5.16: Função Sensor Ultrassónico

## Sistema B + D

Para execução dos dois sistemas em tempo real, utilizamos a biblioteca FreeRTOS disponível para Arduino.

```
  /*
   * Tasks
  */
//  System D
1 xTaskCreate(TaskBuzzer, "Sound Buzzer", configMINIM
  , 2, &HandleBuzzer);
2 xTaskCreate(TaskBlink, "Blink LED", configMINIMAL_S
  , 2, &HandleBlink);
//  System B
3 xTaskCreate(TaskTempControl, "Temperature Control",
  , 0, NULL);
```

Figure 5.17: Criação das Tarefas

1. Tarefa para Alarme Sonoro
   - Prioridade – 2
   - Handle – HandleBuzzer
2. Tarefa para Alarme Visual (LED Intermitente)
   - Prioridade – 2
   - Handle – HandleBlink
3. Tarefa para Alarme Sonoro
   - Prioridade – 0 (Iddle)
   - Handle – NULL (Não é necessário)

Nesta junção os sistemas mantêm o mesmo funcionamento e como tal o Sistema D continua a fazer uso de Interrupts:

```
void ISRAlarmOn(){
  BaseType_t xHigherPriorityTaskWoken = pdFALSE;
  if(alarm_state == LOW){
    Serial.println("Alarm On");
    alarm_state = HIGH;
//    Call Task
  1 vTaskNotifyGiveFromISR(HandleBuzzer, &xHigherPriorityTaskWoken);
//2   vTaskNotifyGiveFromISR(HandleBlink, &xHigherPriorityTaskWoken);
  }
}
```

Figure 5.18: Função Interrupção (Deteção Intruso)

1. Chamada/Notificação da Tarefa TaskBuzzer() (utilizando a 'Handle' associada)
2. Chamada/Notificação da Tarefa TaskBlink() (utilizando a 'Handle' associada)

```
void ISRAlarmOff(){
  if(alarm_state == HIGH){
    Serial.println("Alarm Off");
    alarm_state = LOW;
  }
}
```

Figure 5.19: Função Interrupção (Desarme de Alarme)

Apesar da disponibilidade da biblioteca FreeRTOS, após alguma pesquisa e experiências, posso afirmar que o Arduino Uno, a placa em utilização, não está preparada a 100% para lidar com Sistemas Operativos em Tempo Real, um dos problemas que revelou foi a não execução da TaskBuzzer() na situação da TaskBlink() estar descomentada. Também verifiquei que não era possível chamar qualquer uma das Tasks se a outra não fosse também criada na inicialização do programa. No entanto, foi possível executar ambas as funções mesmo que não ao mesmo tempo.

Não encontrei resposta para este problema, mesmo após uma pesquisa exaustiva.

Assim sendo, foi feito uso da TaskBuzzer() apenas:

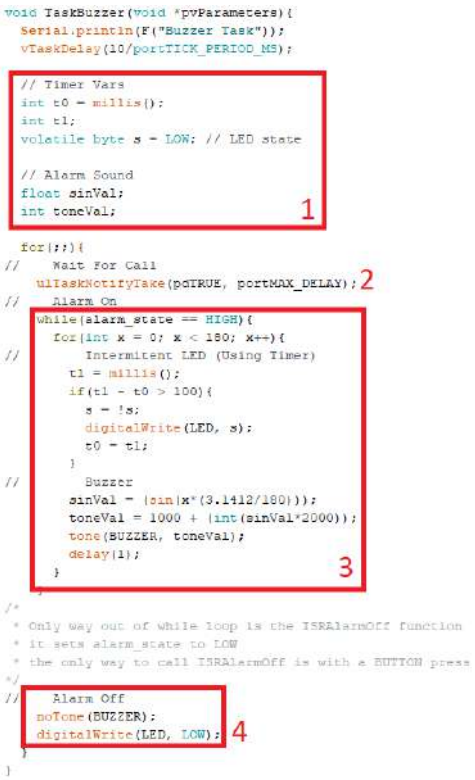



Figure 5.20: Caption

1. Variáveis para Alarme Sonoro e Visual
2. A tarefa aguarda neste momento por ser chamada
3. Execução do Alarme
4. Execução do Desarme

# Testes

Gravações dos Sistemas em Funcionamento em Anexo.

## Sistema A – Controlo de Iluminação Interior

Com um nível de luz entre 0 e 200 o LED encontra-se desligado.

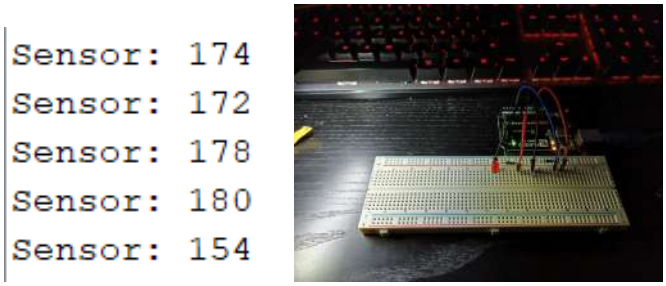



Figura 6.1: LED desligado no nivel inicial

Com um nível de luz entre 200 e 500 o LED encontra-se ligado com um valor de 64.

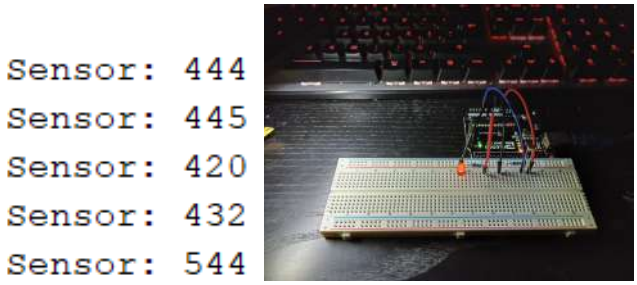



Figura 6.2: LED ligado no nivel 1

Com um nível de luz entre 500 e 800 o LED encontra-se ligado com um valor de 128.

```
Sensor: 698
Sensor: 702
Sensor: 784
Sensor: 768
Sensor: 750
```

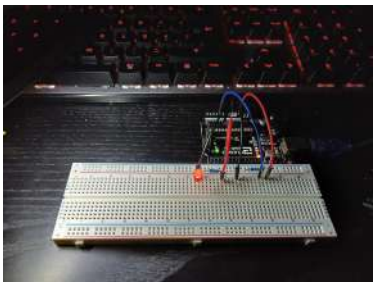


Figura 6.3: LED ligado no nivel 2

Com um nível de luz entre luz superior a 800 o LED encontra-se ligado com a potência máxima disponível.

```
Sensor: 967
Sensor: 982
Sensor: 1029
Sensor: 1023
Sensor: 1000
```

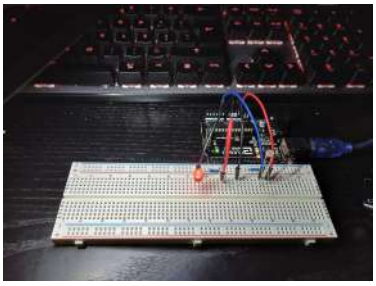


Figura 6.4: LED ligado no nivel máximo

## Sistema B – Controlo de Climatização

Para facilidade de teste, o intervalo de temperaturas é ajustado consoante necessário.

```
System B - Temperature Control
Temperature is to high. Cooling Down.
Temperature stabilized.
```

Figura 6.5: Serial Monitor

**Temperature is to high. Cooling Down.**
Temperatura ultrapassou máximo definido.

**Temperature stabilized.**
Temperatura alcançou mínimo definido.

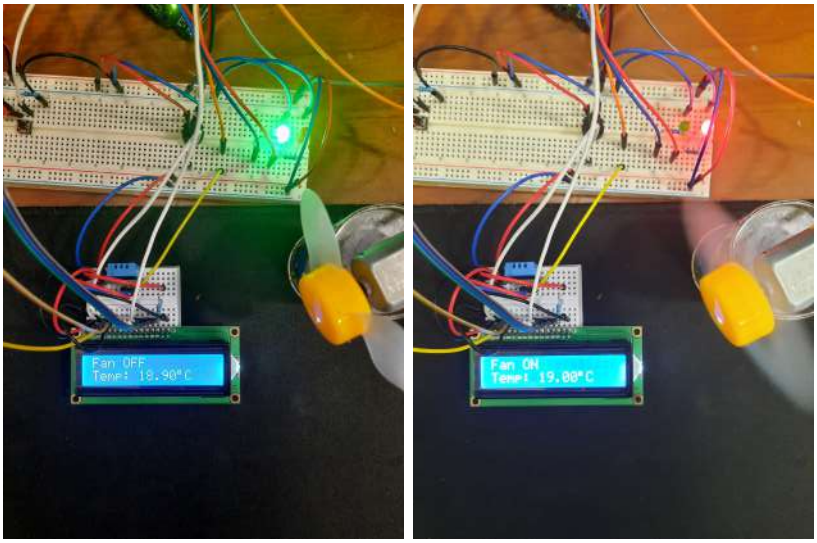



Figura 6.6: Stabilized vs Cooling Down

## Sistema C – Sistema Acesso ao Estacionamento

Quando é pressionado o botão "seta para cima" no comando, o servo faz com que o portão abra.

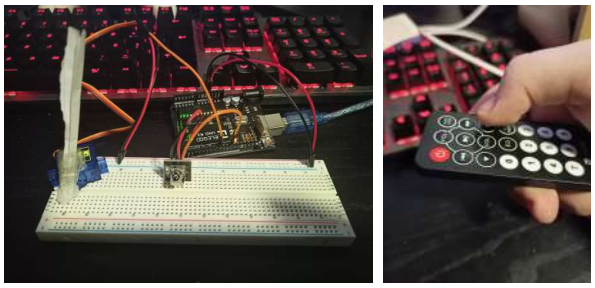


Figura 6.7: Portão Aberto

Quando é pressionado o botão "vermelho" no comando, o servo suspende o seu movimento.

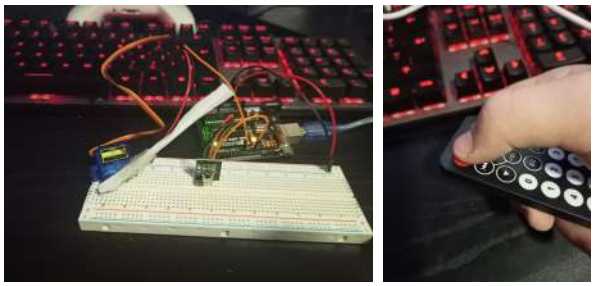


Figura 6.8: Portão Semi-aberto

Quando é pressionado o botão "seta para baixo" no comando, o servo faz com que o portão feche.

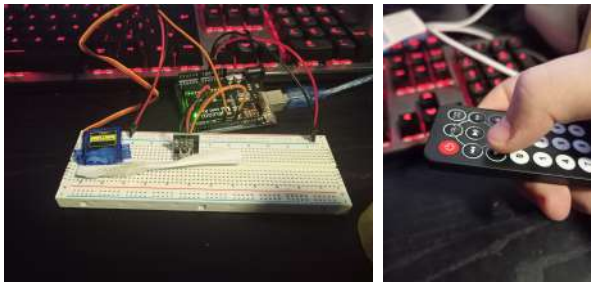


Figura 6.9: Portão Fechado

Foi verificado durante os testes que o sensor de infravermelhos nem sempre deteta os valores corretos do comando. Isto pode ser devido a interferências na corrente, pois a falha na deteção dos códigos é acentuada quando o servo se encontra em movimento.

## Sistema D – Sistema de Segurança

Quando o sensor PIR deteta movimento fica 'suspenso' durante um certo tempo em que não deteta qual movimento. Terminando este intervalo, o sensor está 'ativo' novamente pelo que pode ocorrer uma repetição da sinalização de 'Alarm On', caso o alarme não seja desarmado nesse meio tempo. No entanto, isto não acontece - como pode verificar no vídeo em anexo - é feita uma verificação para caso o alarme já esteja ativo.

Também é verdade para o desarme do alarme com o pressionar do botão. Múltiplos pressionares não desarmam múltiplas vezes o alarme.

```
System D - Alarm System
Alarm On
Alarm Off
```

Figura 6.10: Serial Monitor

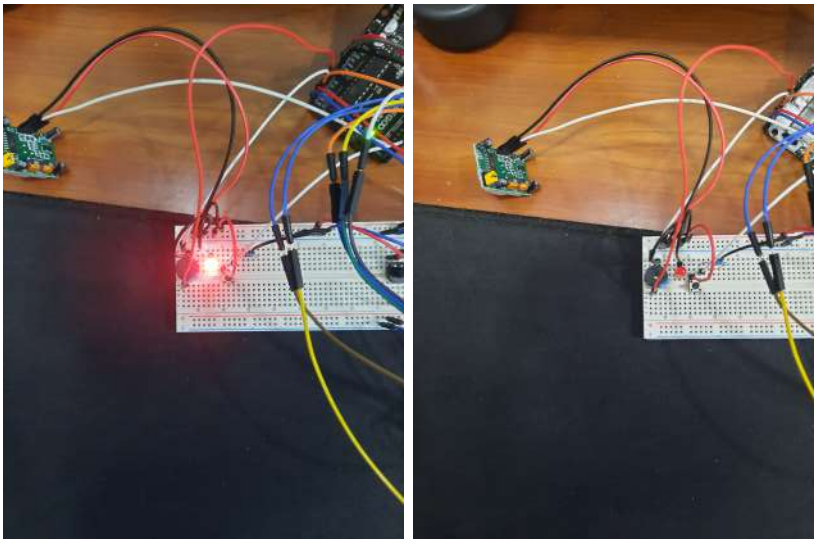


Figura 6.11: On vs Off

# Performance de Programa - Path

Para realizar a avaliação em questão foram construídas duas configurações do Sistema de Climatização.

```
  unsigned long t0 = micros();
//  Temperature Control
  if(temp > MAX && fan_state == LOW){//T1
    Serial.println("Temperature is to high. Cooling Down.");
    digitalWrite(FAN, HIGH);
    fan_state = HIGH;
  }
  else if(temp < MIN && fan_state == HIGH){//T2
    Serial.println("Temperature stabilized.");
    digitalWrite(FAN, LOW);
    fan_state = LOW;
  }


//  LED Indicators Control
  if(fan_state == LOW){//L1
    digitalWrite(GLED, HIGH);
    digitalWrite(RLED, LOW);
  }
  else{//L2
    digitalWrite(GLED, LOW);
    digitalWrite(RLED, HIGH);
  }

  unsigned long t1 = micros();
  Serial.print("Timer: ");
  Serial.print(t1 - t0);
  Serial.println(" µs");

  return temp;
```

Figura 7.1: Melhor Path

| Temp. | Fan | Path |
|---|---|---|
| > MAX | LOW | T1 = True<br>L1 = False & L2 = True |
| > MAX | HIGH | T1 = False & T2 = False<br>L1 = False & L2 = True |
| > MIN && < MAX | LOW | T1 = False & T2 = False<br>L1 = True |
| > MIN && < MAX | HIGH | T1 = False & T2 = False<br>L1 = False & L2 = True |
| < MIN | LOW | T1 = False & T2 = False<br>L1 = True |
| < MIN | HIGH | T1 = False & T2 = True<br>L1 = True |

```
  int t0 = micros();
//  Temperature Control
  if(temp > MAX && fan_state == LOW){//T1
    Serial.println("Temperature is to high. Cooling Down.");
    digitalWrite(FAN, HIGH);
    fan_state = HIGH;
  }
  if(temp < MIN && fan_state == HIGH){//T2
    Serial.println("Temperature stabilized.");
    digitalWrite(FAN, LOW);
    fan_state = LOW;
  }


//  LED Indicators Control
  if(fan_state == LOW){//L1
    digitalWrite(GLED, HIGH);
    digitalWrite(RLED, LOW);
  }
  if(fan_state == HIGH){//L2
    digitalWrite(GLED, LOW);
    digitalWrite(RLED, HIGH);
  }

  int t1 = micros();
  Serial.print("Timer: ");
  Serial.print(t1 - t0);
  Serial.println(" µs");
  return temp;
```

Figura 7.2: Pior Path

| Temp. | Fan | Path |
|---|---|---|
| > MAX | LOW | T1 = True & T2 = False<br>L1 = False & L2 = True |
| > MAX | HIGH | T1 = False & T2 = False<br>L1 = False & L2 = True |
| > MIN && < MAX | LOW | T1 = False & T2 = False<br>L1 = True & L2 = False |
| > MIN && < MAX | HIGH | T1 = False & T2 = False<br>L1 = False & L2 = True |
| < MIN | LOW | T1 = False & T2 = False<br>L1 = True & L2 = False |
| < MIN | HIGH | T1 = False & T2 = True<br>L1 = True & L2 = False |

| Estado | Melhor | Pior |
|---|---|---|
| Superior a Máximo | 280 µs | 300 µs |
| Estável entre Intervalo | 20 µs | 20 µs |
| Arrefecimento Entre Intervalo | 20 µs | 20 µs |
| Inferior a Mínimo Arrefecimento | 200 µs | 220 µs |
| Estável Inferior a Mínimo | 20 µs | 20µs |

Tabela 7.1: Timer

Apesar de reduzido, é possível verificar uma diferença no desempenho.

# Conclusão

Os vídeos dos sistemas estão disponíveis a partir do seguinte link:
`https://youtube.com/playlist?list=PLbfrOZp9dsMTeufm7N9a-4TlIZPoooJzm`

Para concluir achamos que tudo o que foi pedido para a execução deste trabalho prático foi desenvolvido. Os diversos sistemas encontram-se com todas as suas funções principais implementadas e funcionais.

Foi necessária a junção dos Sistemas A + C e dos Sistemas B + D devido à limitação da quantidade de Arduinos, como tal, foi utilizado multitasking para fazer essa junção de forma eficaz e funcional.

No Sistema A foi implementado um interrupt como forma de um botão.

No Sistema C para além de multitasking foi implementada a utilização do Sensor Ultrassónico para suspender a movimentação do portão

No Sistema D foram implementados dois interrupts, via Sensor PIR e botão.

No entanto foram encontrados alguns problemas na execução da tarefa pedida, nomeadamente no que toca à implementação do Sensor de Infravermelhos e de Multitasking por "FreeRTOS".

O Sensor de Infravermelhos é sensível a interferências, o que torna o sistema irresponsivo por vezes, principalmente durante o movimento do motor Servo. Outro dos problemas encontrados com o sensor Infravermelhos é que, devido ao seu funcionamento, algumas das portas "PWM" do Arduino perdem a sua função modular, podendo apenas enviar 0V ou 5V tornando impossível o controlo da voltagem do LED.

Como descrito anteriormente, na junção dos Sistemas B e D, não foi possível a utilização a 100% da biblioteca RTOS. Pelo que pude apurar isto deve-se maioritariamente à incapacidade do Arduino Uno.

Como nota final, achamos que este trabalho prático se adequa completamente aos conteúdos lecionados. É uma perfeita introdução ao tema de sistemas embebidos e ao seu desenvolvimento.