Relatório

Sistemas Embebidos em Tempo Real

Alunos:

João Fernandes Nº18825

António Oliveira Nº18833

Carlos Martins Nº18836

Professor: Paulo Macedo

Licenciatura em Engenharia de Sistema Informáticos

Barcelos, janeiro, 2021

Índice

Índice de Figuras

1. Introdução

Este trabalho tem como objetivo o desenvolvimento de um conjunto de sistemas embebidos em tempo real integrados para uma Smart Home com funcionalidades especificas, das quais, iluminação, climatização, parking e segurança.

Para o desenvolvimento destes sistemas foi utilizada a linguagem de programação C com a utilização de bibliotecas utilizadas para a programação de sistemas arduino como, por exemplo, a LiquidCrystal. Além disso, alguns destes sistemas foram desenvolvidos fisicamente cumprindo todos os requisitos ou com algumas adaptações e os restantes foram desenvolvidos na plataforma tinkercad que é um simulador de arduino que possui uma grande variedade de funcionalidades, dispositivos e equipamentos.

Posto isto, este relatório vai ser realizado à medida que o trabalho vai avançando, de forma a organizar melhor os conteúdos.

2. Análise de Requisitos

Projeto “Home Automation” para realizar funcionalidades específicas: Iluminação, Climatização, Parking e Segurança.

Sistema	Requisitos Funcionais
A	No controle a luminosidade do espaço interior, em função da luz solar, é regulado através de um sensor, a luminosidade, garantindo uma iluminação constante e uma maior eficiência energética.
B	Para realizar o controlo da climatização, uma ventoinha é acionada para o arrefecimento do espaço em função da temperatura fornecidos pelo sensor de temperatura.
C	Para o sistema de parking, um controlo remoto controla a barra que abre e fecha, o acesso ao parque de estacionamento.
D	Para o sistema de segurança, um sensor de movimento deteta o movimento de intrusos, acionando um sinal luminoso e sonoro.

Sistema	Requisitos Não-Funcionais
A	Inputs: Sensor LDR. Outputs: Leds de iluminação interior. Funcionalidade: Minimiza custos de eletricidade. Interface com o utilizador: Sensor LDR que regula a intensidade da luz e um LED verde que indica uma temperatura regulada e um LED vermelho que indica que a ventoinha se encontra em funcionamento. Performance: Otimização e redução de custo da eletricidade. Propósito: Transmitir conforto e otimização dos custos de energia.
B	Inputs: LED vermelho e verde, sensor temperatura. Outputs: LCD 16 X 2 que mostra a temperatura e o mostra o estado da ventoinha. Funcionalidade: Climatização dentro de casa. Interface com o utilizador: O controlo de climatização através de uma ventoinha, e um LCD que mostra a temperatura e o estado da ventoinha. Performance: Atualização do ecrã LCD para saber a temperatura ambiente. Propósito: Manter a temperatura ambiente.
C	Inputs: Controle Remoto infravermelhos. Outputs: Motor Servo Funcionalidade: Sistema útil para estacionamento de forma a controlar remotamente.

	Interface com o utilizador: Para o acesso ao estacionamento, existe um controlo remoto, onde o utilizador consegue controlar a barra de acesso. Performance: Sensor que deteta movimento a uma certa distância evitando acidentes graves. Propósito: Facilidade no estacionamento.
D	Inputs: Botão de desarme do alarme. Outputs: Sinal luminoso (led) e sinal sonoro característico de um alarme. Funcionalidade: Melhor segurança na casa. Interface com o utilizador: Para o sistema de segurança o utilizador consegue pressionar num botão para desarmar o alarme. Performance: Sensibilidade do sensor para detetar movimentos mais precisos. Propósito: Aumentar a segurança da casa

3. Especificação do sistema

Deve incluir:

- A informação sobre a luminosidade e a temperatura ambiente no interior da casa;
- Movimentos bruscos quando ligado o sistema de segurança;
- Dados sobre a temperatura após atuar o sistema de climatização;
- LCD 16 X 2 que mostra os dados, botão de desarme e controlo remoto;
- Realizar a climatização e o controlo de luminosidade, seja de noite ou de dia, transmitir maior segurança quando o utilizador permanece ausente e a utilidade de uma barra de acesso para o estacionamento do automóvel;
- Sensores de temperatura, movimento e de luminosidade são sistemas que ficam em execução para que os sistemas funcionem.

4. Desenvolvimento da Arquitetura

- Todos os componentes utilizados servirão para realizar uma “home automation”, tornando uma casa comum, numa casa inteligente. Se todos os requisitos e especificações satisfazerem as condições, teremos os sistemas pretendidos.
- Arduíno, sensor de temperatura, movimento e luminosidade, cabos, motor servo, ventoinha, display LCD 16 X 2, botão (pressão), sinal sonoro e luminoso e por fim leds.
- Arduíno IDE e TinkerCad

4.1. Arquitetura Sistema A

Figura 1 - Arquitetura Sistema A

4.2. Arquitetura Sistema B

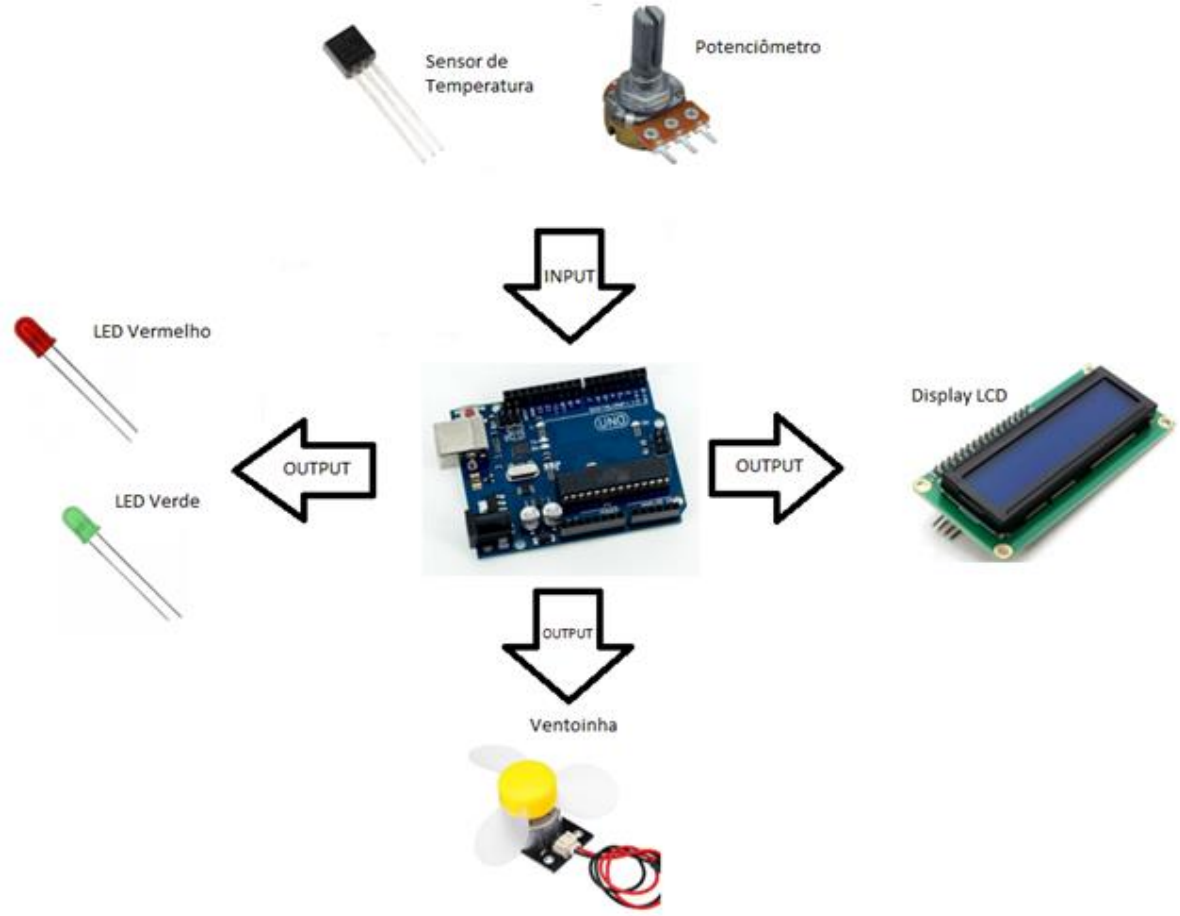

Figura 2 - Arquitetura Sistema B

4.3. Arquitetura Sistema C

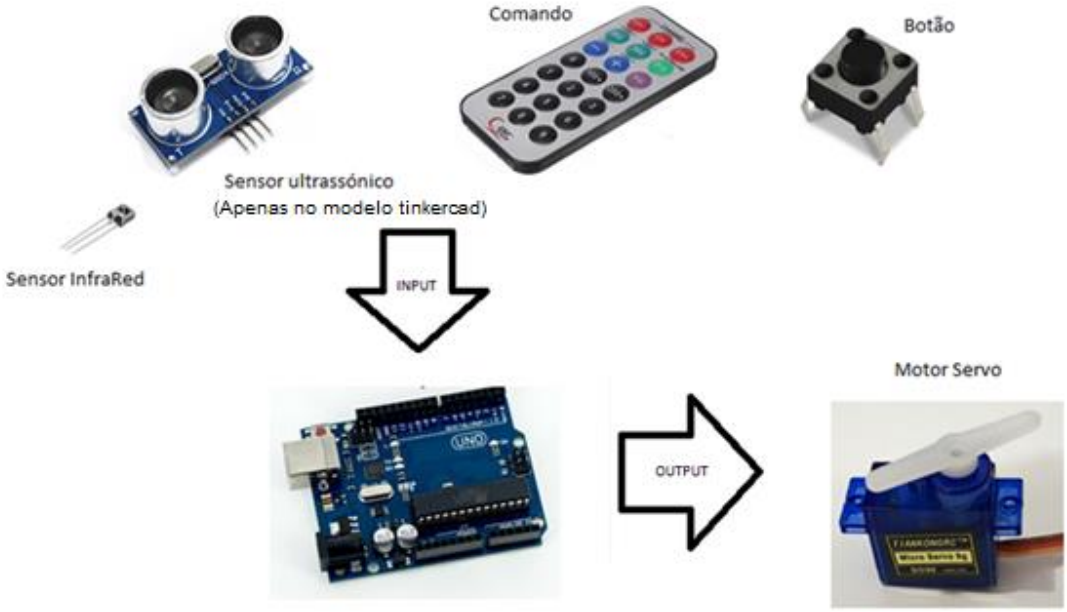

Figura 3 - Arquitetura Sistema C

4.4. Arquitetura Sistema D

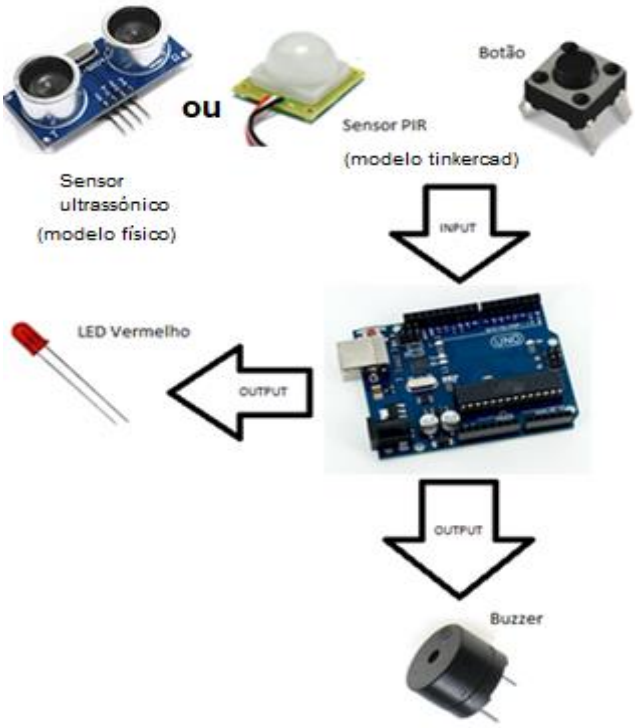

Figura 4 - Arquitetura Sistema D

5. Modelo de Conceção

O modelo utilizado no trabalho prático foi o ***waterfall.*** Foram definidos para o desenvolvimento dos Sistemas uma abordagem ***Top-down***:

- **Requisitos** funcionais e não funcionais para cada sistema;
- Uma **arquitetura** com um esquema prévio para o desenvolvimento de cada sistema;
- A implementação de um **código** que conseguisse cumprir o que foi imposto inicialmente;
- **Testes**, tanto do código, como na montagem do circuito como no simulador TinkerCad;
- Por fim uma **manutenção** para a correção de alguns erros, otimização do sistema e atualização do código.

Figura 5 - Modelo Waterfall

6. Construção do sistema

6.1. Sistema A – Controlo de iluminação interior

Para simular este sistema, utilizamos um LED e um sensor LDR. O objetivo é controlar a luminosidade do LED de acordo com a iluminação do espaço. Para saber qual a luminosidade do LED, foi definido escalas de intensidade de luz de forma a que, conforme a iluminação do espaço, detetado pelo sensor LDR, o LED ligue e permaneça com uma certa intensidade de luz. Os valores atribuídos para o sensor LDR foram:

- para o LED não ligar - < 200;
- para uma intensidade de 64 - ≥ 200 e < 500;
- para uma intensidade de 128 - ≥ 500 e < 800;
- para uma intensidade de 255 - ≥ 800;

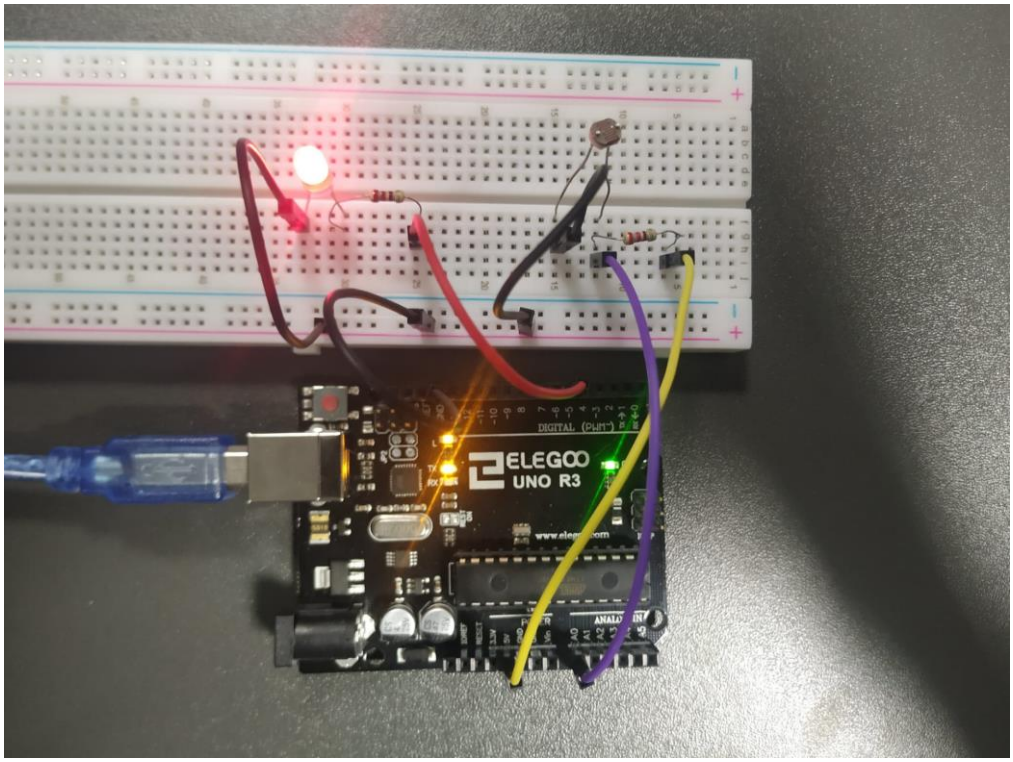

Figura 6 - Construção do Sistema A

6.1.2. Código

```
int ldrPin = A0; // Sensor LDR na porta Analógica 0
int ledPin = 5;  // Porta LED
int ldrValue = 0; // Variável que guarda o valor obtido pelo LDR

void setup() {
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  ldrValue = analogRead(ldrPin); // Ler o valor do LDR
  Serial.print("Valor entrada LDR:");
  Serial.println(ldrValue);

  // Conjunto de condições que alteram o estado do LED de acordo com o
  //valor do LDR

  if (200 > ldrValue)
  {
    digitalWrite(ledPin, 0);
    Serial.println("Luminosidade alterada para 0");
  }
  else if (ldrValue >= 200 && ldrValue < 500)
  {
    digitalWrite(ledPin, 64);
    Serial.println("Luminosidade alterada para 64");");
  }
  else if(ldrValue >= 500 && ldrValue < 800)
  {
    digitalWrite(ledPin, 128);
    Serial.println("Luminosidade alterada para 128");");
  }
  else if(ldrValue > 800)
  {
    digitalWrite(ledPin, 255);
    Serial.println("Luminosidade alterada para 255");");
  }
}
```

6.2. Sistema B – Controlo de climatização

Neste sistema é pretendido desenvolver um controlo de temperatura ambiente através de uma ventoinha que é acionada para arrefecer o espaço onde se situa, em função dos valores de temperatura que são obtidos pelo sensor de temperatura.

Quando o sensor de temperatura detetar 25 graus celsius a ventoinha liga e desliga sempre que a temperatura é inferior a 20 graus celsius. Para saber quando está a arrefecer o LED vermelho liga e quando a temperatura estabiliza o LED verde liga. Na 1ª linha do LCD mostra o estado da ventoinha, se ela está ON ou OFF e na 2ª linha mostra a temperatura atual. O potenciômetro regula a luminosidade do Display LCD.

Desenvolvemos este exercício no tinkercad porque o LCD e o sensor de temperatura que tínhamos disponíveis não estavam a funcionar corretamente.

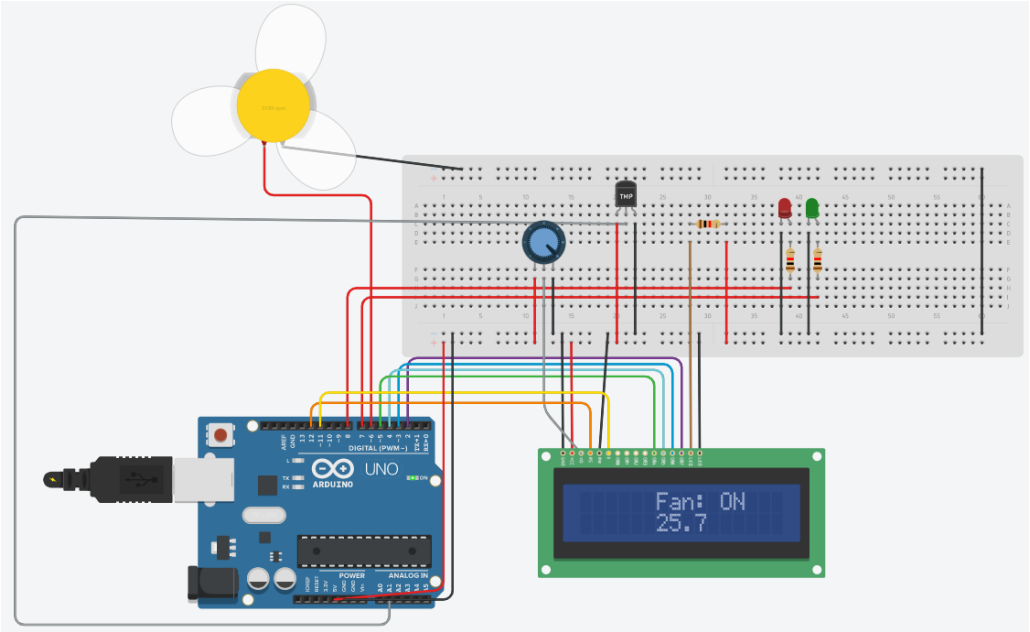

Figura 7 - Construção do sistema B

6.2.2. Código

```
#include <LiquidCrystal.h>
// componentes
/*  motor (ventoinha)
 *  sensor de temperatura
 *  lcd (a mostrar se a fan esta on/off e a temp)
 *  led vermelho (liga quando está a ventoinha ligada)
 *  led verde (liga quando a temp está estabilizada)
 *  potenciometro (para controlar a luminosidade do lcd)
 */
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

int fanPin;
int lcdPin;
int ledCooling = 8; // led vermelho - liga quando está a ventoinha ligada
int ledStabilized = 7; // led verde - liga quando a temperatura está
estabilizada
int maxTemp = 25; // temperatura maxima tolerada antes de ligar a
ventoinha
int minTemp = 20; // temperatura minima que quando atingida desliga a
ventoinha
int tempPin = A1;
int motorPin = 6;
float temp;


void setup() {
  // put your setup code here, to run once:
  pinMode(fanPin, OUTPUT);
  pinMode(lcdPin, OUTPUT);
  pinMode(motorPin, OUTPUT);
  lcd.begin(16, 2); //Liga o lcd com as dimensões (linhas,colunas)
  lcd.setCursor(6, 0); // Define onde o texto vai ser escrito
  lcd.print("Fan: ");
  Serial.print("Fan: ");
  temp_regulator ();
  Serial.begin(9600); // Starts the serial communication
}
```

```
void loop() {
  // put your main code here, to run repeatedly:

  // temp >25º liga
  // temp <20º desliga

  temp = analogRead(tempPin); // le valor do sensor de temperatura

  // converte o valor em uma temperatura em ºC
  temp = temp * 5 / 1024;
  temp = (temp - 0.5) * 100;

  lcd.setCursor(6, 0);
  lcd.print("Fan: ");
  Serial.print("Fan: ");

  temp_regulator ();

  lcd.setCursor(6, 1);
  lcd.print(temp, 1);
  Serial.print("Temp: ");
  Serial.println(temp);

  delay(500);

}
void temp_regulator (){
  if (temp > maxTemp) {
    Serial.println("ON ");
    lcd.print("ON ");
    digitalWrite(ledCooling, HIGH);
    digitalWrite(ledStabilized, LOW);
    digitalWrite(motorPin, HIGH);

  } else if (temp < minTemp) {
    Serial.println("OFF");
    lcd.print("OFF");
    digitalWrite(ledCooling, LOW);
    digitalWrite(ledStabilized, HIGH);
    digitalWrite(motorPin, LOW);

  } else {
    Serial.println("OFF");
    lcd.print("OFF");
    digitalWrite(ledCooling, HIGH);
    digitalWrite(ledStabilized, LOW);
    digitalWrite(motorPin, LOW);
  }
}
```

6.3. Sistema C – Sistema de acesso ao estacionamento (Físico)

Este sistema permite que um comando controle uma barra de acesso a um parque de estacionamento. Na simulação, o comando infrared controla o motor servo da seguinte forma:

- O botão 0 do comando levanta a barra 90º verticalmente;
- O botão 1 do comando desce a barra até aos 0º horizontalmente;

Além disso, existe um botão adicional (que não conseguimos implementar no comando) que quando é pressionado provoca uma interrupção utilizando a função attachInterrupt.

De modo a reduzir a velocidade do motor Servo (que simula a barra de entrada/saída de um estacionamento) utilizamos um ciclo for em que a posição do mesmo altera a cada 25ms.

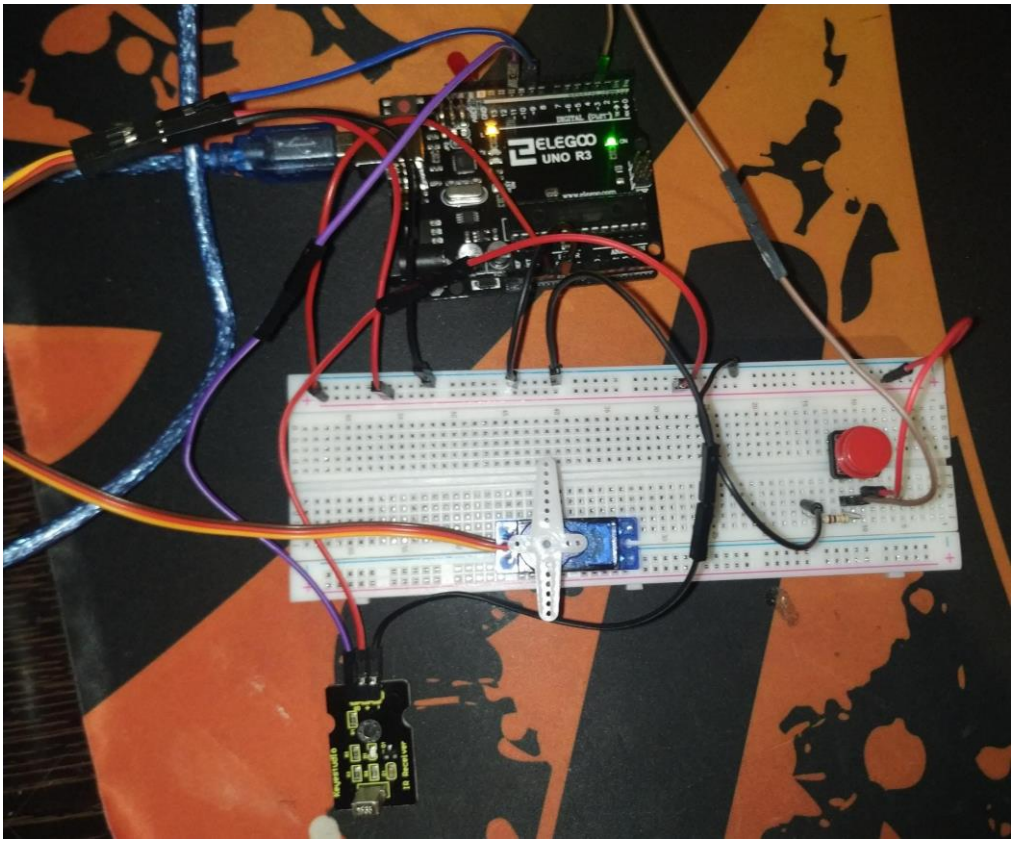

Figura 8 - Construção do Sistema C

6.3.1. Código

```
#include <IRremote.h>
#include <Servo.h>

#define subir 34935 //Botao 0
#define descer 8415 //Botao 1

int receiver_pin = 11; // Pino do sensor IR
Servo motor; // Declarar variável servo
int pos = 180; // Posiçao para o SERVO (Usada para definir a velocidade
do servo)

int buttonPin = 3; // Pino do botão utilizado para suspender o movimento
unsigned long lastInterrupt; //Variavel para gerar o interrupt

int state = 0; // Variavel para obter o estado do botão (Clicado ou não)

IRrecv receiver(receiver_pin);
decode_results output;


void setup()
{
  Serial.begin(9600);


  /* Infravermelhos */
  receiver.enableIRIn();
  /* ---- */
  motor.write(pos); // Definir posição inicial
  motor.attach(9); //Ligar o SERVO à porta 9
  pinMode(buttonPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(buttonPin), suspender, FALLING);
  //Chamar o interrupt com a função suspender quando o estado do botão
  //passa de HIGH para LOW (Falling)
}

void loop()
{
  if (receiver.decode(&output)) // Detetar sinais IR
  {
    unsigned int value = output.value;
    Serial.println(value);
    receiver.resume();
```

```
switch (value) // Switch case para trabalhar sinais IR recebidos
    {
        receiver.resume();

      case subir:

        subirBarra();
        break;

      case descer:
        descerBarra();
        break;
    }
  }
}

//Função utilizada para suspender a barra
// É chamada no interrupt, ou seja, quando o botao é clicado o botão fica
// HIGH e o state passa a 1 o que causa a que o motor dê detach, quando é
// clicado novamento o state passa a 0 e o servo pode continuar o
// movimento
void suspender() {

  state = !state;
  if (state == 1)
    motor.detach();
  else
    motor.attach(9);
}
// Funcões utilizadas para controlar a velocidade de movimento do servo
// Altera a posição a cada 15ms
void subirBarra(int position)
{
// verifica se o portão está fechado(pos != 180) e se tiver sobe o portão
  if (pos != 180) {
    if (position >= 0)
      pos = position;
    else
      pos = 0;
    for (; pos < 180; pos += 1)
    {
      motor.write(pos);
      delay(15);
    }
  }
}
```

```
void descerBarra()
{
// verifica se o portão está aberto(pos != 0) e se tiver desce o portão
  if (pos != 0) {
    for (pos = 180; pos > 0; pos -= 1)
    {
      motor.write(pos);
      delay(15);
    }
  }
}
```

6.4. Sistema C – Sistema de acesso ao estacionamento (Tinkercad)

Desenvolvemos também este sistema em Tinkercad de modo a implementarmos um sensor ultrassónico na sua construção. Este sensor ultrassónico verifica se não está ninguém dentro de uma distância definida por nós (2 metros) e se a condição for satisfeita o servo começa a descer, se não o servo não desce, além disso, se a condição for satisfeita, mas de repente entrar alguém dentro do radar de deteção o servo começa automaticamente a subir.

A justificação para implementar o sensor ultrassónico neste sistema no Tinkercad e não fisicamente foi que o sensor ultrassónico estava a causar algum tipo de interferência com o sensor IR. Supomos que a causa disto seja devido ao número de cálculos que o sistema tem de executar para calcular a distância a que um objeto se encontra do sensor ultrassónico a cada vez que o servo está a descer.

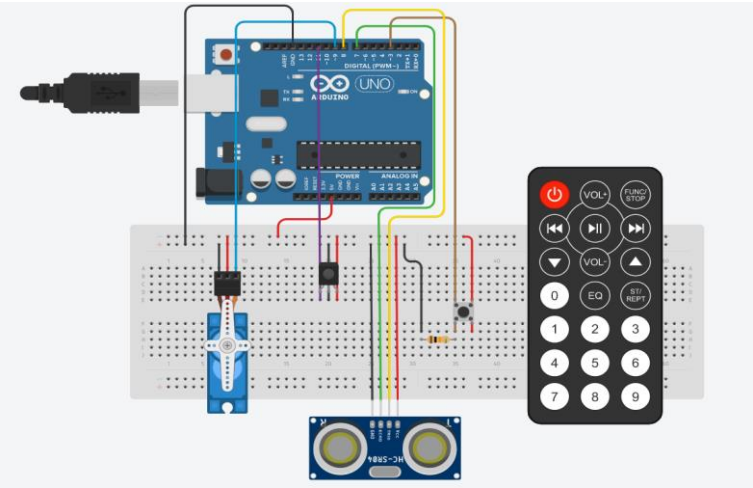

Figura 9 - Construção do sistema C (Tinkercad)

6.4.1. Código

Para manter não estar a mostrar praticamente o mesmo código vamos só deixar apenas as alterações feitas o código anterior. O código completo encontra-se no seguinte link: https://pastebin.com/iUW9nXqD

```
// Declaramos estas variáveis globais

// Variaveis do sensor ultrassónico
int trigPin = 8; // Associa o pin trig à porta 8
int echoPin = 7; // Associa o pin echo à porta 7

long duration, distance; // Variaveis utilizadas para calcular a
// distancia e a duração da onda


// A função descerBarra também sofre várias alterações para
// acomodar a distância

for (pos = 180; pos > 0; pos -= 1)
  {
    digitalWrite(trigPin, HIGH); //Enviar onda ultrassónica
    delayMicroseconds(10); //Durante 10ms
    digitalWrite(trigPin, LOW); //Para de enviar onda
    duration = pulseIn(echoPin, HIGH);
    //Calcular a distancia (em cm) baseada na velocidade do som.
    distance = (duration/58.2)*2;

    if(distance > 200) {
        motor.write(pos);
        delay(25);
    }
    else if(distance <= 200){

        subirBarra(pos);
        break;
    }
    pos -= 1;
  }
  }
}
```

6.5. Sistema D – Sistema de segurança (Tinkercad)

Foi criado um sistema de segurança para a deteção de movimentos através de um sensor PIR. No momento que deteta movimento é acionado um sinal luminoso através de um LED vermelho, um sinal sonoro que seja característico de um alarme e dura 10 segundos e um botão que permita desarmar o alarme. Vale a pena notar que este sinal sonoro pode parecer um pouco distorcido no tinkercad, porque no tinkercad um segundo não equivale realmente a um segundo verdadeiro.

Como o arduino não suporta multiprocessamento tivemos que utilizar multitasking (simulamos isto com a função micros) para conseguirmos obter o LED a piscar e um sinal sonoro em simultâneo.

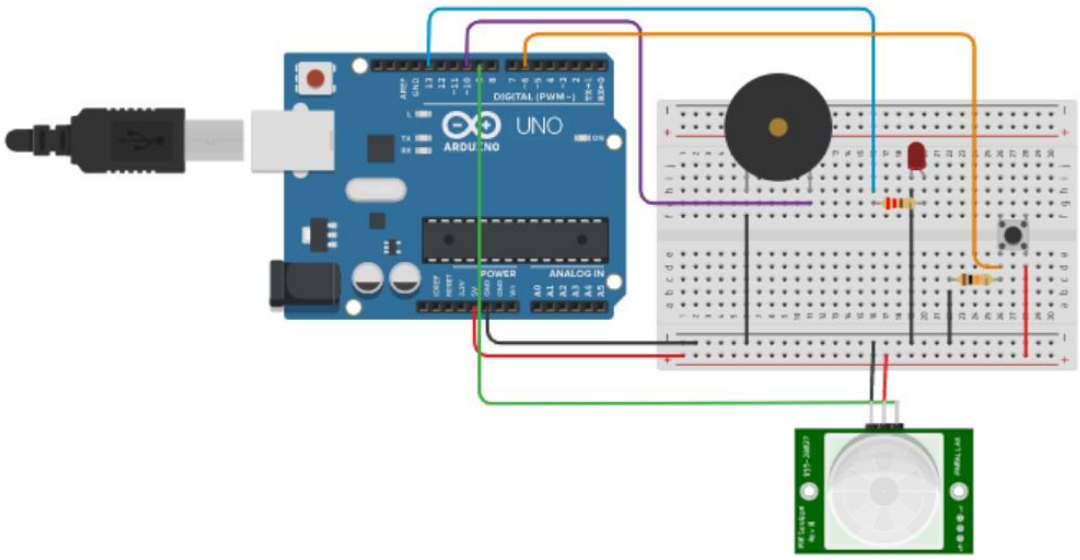

Figura 10 - Construção do Sistema D (Tinkercad)

6.5.1. Código

```
// Definir os pinos para os diferentes componentes
int pirSensor = 9;
int buzzer = 10;
int ledPin = 13;
int buttonPin = 6;

// Estado do sistema, se estiver 1 significa que está ligado e o alarme
está ativo
// se tiver 0 o sistema está desligado e o alarme está desativado
int status = 1;
// Estado do botão, se tiver HIGH é porque foi pressionado para
interromper o alarme
int buttonStatus = LOW;
// Estado do alarme, se estiver 1 é porque o alarme foi disparado
int alarmStatus = 0;
// Estado do pir, o valor é HIGH quando deteta movimento
int verifica;

int LED1_state = LOW; // Definir o estado do led no inicio do programa
para LOW
int BUZZER_state = LOW; // Definir o estado do buzzer  no inicio do
programa para LOW
unsigned long ledTime = millis(); // Guardar o tempo da ultima vez que o
led piscou
unsigned long buzzerTime = millis(); // Guardar o tempo da ultima vez que
o buzzer emitiu um sinal sonoro
unsigned long alarmTime = 0;

long intervalLed = 300; //Piscar o led a cada 300ms
long intervalBuzzer = 400; // Sinal sonoro a cada 400ms

void setup() {
  Serial.begin(9600);
  pinMode(pirSensor, INPUT);
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {

  unsigned long currentTime = millis();
  alarmTime = millis();

  buttonStatus = digitalRead(buttonPin);
  verifica = digitalRead(pirSensor);
```

```
// Se o alarme estiver ativo e for detetado movimento
// o alarme dispara
if (status == 1) {
    if (verifica == HIGH || alarmStatus == 1) {
      alarmStatus = 1;

      // utilizado no multitasking, verifica se esta na fatia de tempo do
led para piscar
      if (currentTime - ledTime > intervalLed) {
        LED1_state = !LED1_state;
        digitalWrite(ledPin, LED1_state);

        ledTime = currentTime;
      }
      // utilizado no multitasking, verifica se esta na fatia de tempo do
buzzer para ativar
      if (currentTime - buzzerTime > intervalBuzzer) {
        tone(buzzer, 1000, 100);
        delay(200);
        tone(buzzer, 800, 200);
        noTone(buzzer);

        buzzerTime = currentTime;
      }

    }
  }

  // Se o botão for pressionado desliga o alarme
  if (buttonStatus == HIGH) {
    status = !status;

    if (status == 0) {
      alarmStatus = !alarmStatus;
    } else
      alarmStatus = 0;
    digitalWrite(buzzer, LOW);
  }
  // Se tiverem passados 10segundos e o botao clicado tiver sido clicado
  if (alarmTime >= 10000 || buttonStatus == HIGH) {
    digitalWrite(ledPin, LOW); // Desliga o led
    noTone(buzzer); // Desliga o buzzer
  }

  delay(50);
}
```

6.6. Sistema D – Sistema de segurança (Físico)

Fisicamente, temos praticamente o mesmo sistema sendo que a única diferença é que tivemos de substituir o sensor IR por um sensor ultrassónico, porque não tínhamos nenhum disponível a funcionar. Isto causou ligeiras diferenças no código porque o sensor é acionado a partir de uma certa distância e não quando deteta um movimento em todo o seu radar de deteção.

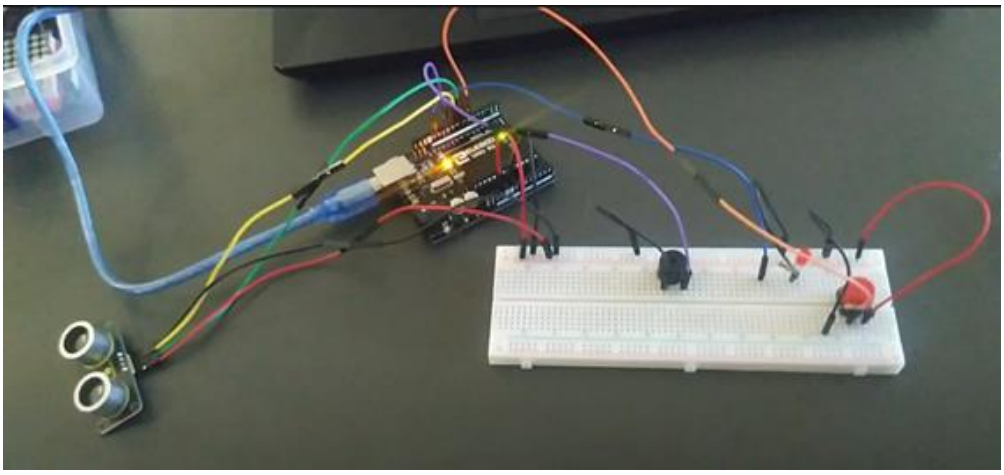

Figura 11 - Construção do sistema D (Físico)

6.6.1. Código

Para manter não estar a mostrar praticamente o mesmo código vamos só deixar apenas as alterações feitas o código anterior. O código completo encontra-se no seguinte link: https://pastebin.com/d3Cp1FPK

```
// Substituimos a variável do sensor PIR por estas duas variáveis para
incorporar o sensor //ultrassónico

// Definir os pinos para os diferentes componentes
int trigPin = 8;
int echoPin = 7;

// Acrescentamos estas variáveis
long duration, distance; // Variaveis utilizadas para calcular a
// distancia e a duração da onda

// Removemos a variável verifica
int verifica;

//Substituimos a condição que verifica se o sensor PIR foi ativado por
// uma condição que verifica se foi detetada alguma coisa dentro da área
// de deteção definida por nós

 if (distance <= 200 && distance > 2 || alarmStatus == 1)
```

7. Testes/Resultados

Para o sistema A, como montámos fisicamente, gravámos um vídeo em que é possível verificar a composição do circuito assim como os testes realizados ao mesmo. Os testes realizados foram apontar a lanterna de um telemóvel ao sensor LDR a diferentes distâncias para a intensidade detetada ser maior ou menor dependendo da distância. Vídeo

Para o sistema B não o montámos fisicamente pelas razões já abordadas nos tópicos anteriores, mas como desenvolvemos o circuito no tinkercad o mesmo está acessível e testável a partir do seguinte link: tinkercad

Para o sistema C foi realizada uma versão no tinkercad (tinkercad) e uma versão física (vídeo). Os testes realizados para o modelo físico foram subir o servo, descer o servo e pressionar o botão para interromper o movimento do servo. No tinkercad temos praticamente o mesmo circuito, mas com ligeiras diferenças no código porque incorporámos um sensor ultrassónico que deteta se existe algo dentro do campo de deteção definido que neste caso são 2 metros.

Para o sistema D também desenvolvemos uma versão física (vídeo) e uma versão no tinkercad (tinkercad). A diferença da versão física para a do tinkercad é que na versão física substituímos o sensor PIR por um sensor ultrassónico (e algum código por causa da diferença de sensores). Os testes realizados para a versão física foram passar a mão a uma distância detetável pelo sensor ultrassónico e desativar o alarme no botão.

8. Análise de performance do sistema C

De forma a medir a performance de uma forma constante sem variáveis que podem interferir na medição, foi feita a medição de performance numa função que não tem interferência de um utilizador e não depende de funções externas.

Na função de descer a barra, “descerBarra()”, criamos uma variável que irá guardar o tempo desde a execução do programa dado pela função “micros()” e após a execução de todo o código da função, guardamos numa nova variável o tempo no fim da execução de forma a calcular o tempo que demorou a executar, utilizando o cálculo tempo final – tempo inicial.

Como podemos ver no código A o tempo de execução foi 3420176 microssegundos, ou seja, maior devido a fatores como a instanciação de uma função auxiliar, ao contrário do código B que mantém o código sem utilizar uma função auxiliar, obtendo um tempo de execução de 3411556 microssegundos.

Código A

```
void descerBarra()
{
  if (pos != 0) {
    timeStart = micros();
  for (pos = 180; pos > 0; pos -= 1)
  {
    ultrasonicSensor();
    duration = pulseIn(echoPin, HIGH);
    distance = (duration/58.2)*2;

    if(distance > 200) {
        motor.write(pos);
        delay(25);
    }
    else if(distance <= 200){
        subirBarra(pos);
        break;
    }
```

```
  }
    timeEnd = micros();
    Serial.println("Time:");
    Serial.println(timeEnd - timeStart);
  }
}
void ultrasonicSensor(){
  digitalWrite(trigPin, HIGH); //Enviar onde ultrassonica
  //Durante 10ms
  for (int i = 0; i<10; i++)
    delayMicroseconds(1);
  digitalWrite(trigPin, LOW); //Para de enviar onda
}
```

```
Time:
3420176
```

Figura 12 - Tempo de execução do código não otimizado

Código B

```
void descerBarra()
{
  if (pos != 0) {
    timeStart = micros();
    for (pos = 180; pos > 0; pos -= 1)
    {
      digitalWrite(trigPin, HIGH); //Enviar onde ultrassonica
      delayMicroseconds(10); //Durante 10ms
      digitalWrite(trigPin, LOW); //Para de enviar onda
      duration = pulseIn(echoPin, HIGH);
      distance = (duration/58.2)*2;
      if(distance > 200) {
          motor.write(pos);
          delay(25);
      }
      else if(distance <= 200){
          subirBarra(pos);
          break;
      }
    }
    timeEnd = micros();
    Serial.println("Time:");
    Serial.println(timeEnd - timeStart);
  }
}
```

```
Time:
3411556
```

Figura 13 - Tempo de execução do sistema não otimizado

9. Conclusão

Chegamos ao fim do semestre e cremos que foi uma disciplina trabalhosa, mas que trabalhamos com todo o gosto. Foi uma disciplina diferente das outras todas que estávamos habituados, porque nunca antes tínhamos mexido em circuitos eletrónicos com a liberdade que nos foi dada nesta cadeira e ainda exploramos e conhecemos componentes que não conhecíamos.

Foi também um trabalho em que deu para usar no nosso "instinto de engenheiro", porque passamos algumas dificuldades com a falta de componentes funcionais, mas demos a volta a isso utilizando outros componentes de modo a cumprir o mesmo objetivo, como por exemplo, no sistema D a utilização de um sensor ultrassónico no lugar de um sensor PIR.

Para finalizar concluímos que o trabalho foi uma mais-valia muito grande porque conseguimos desenvolver o nosso conhecimento e aplicá-lo praticamente todo, senão todo o conteúdo lecionado nas aulas.