# *SmartCity*

## Relatório do Trabalho Prático Final de Sistemas Embebidos em Tempo Real (SETR)

**Discentes**: Adelino Daniel da Rocha Vilaça - a16939
António Jorge Magalhães da Rocha - a26052
António Rafael Henriques Ferreira - a26402

**Docente**: Pedro José Gonçalves da Silva Cunha

**Licenciatura em Engenharia de Sistemas Informáticos (PL)**

**3º ano**

Barcelos | dezembro, 2024

# Índice

# Índice de Figuras

# Introdução e Contextualização

Uma *Smart City* (cidade inteligente) pode ser entendida como um ecossistema urbano em que a recolha e análise de dados, juntamente com sistemas de comunicação e controlo, são utilizados para otimizar a prestação de serviços públicos e melhorar a qualidade de vida dos cidadãos. Em vez de dependerem de estruturas isoladas e reativas, as infraestruturas urbanas como os transportes, a iluminação, o abastecimento de água, a energia e a segurança pública passam a funcionar de forma interligada, apoiadas por tecnologias que se adaptam continuamente às necessidades da população e às condições ambientais envolventes (Caragliu et al., 2011; Lim et al., 2019; Piro et al., 2014).

Neste contexto de transformação, destacam-se quatro componentes essenciais: a gestão do estacionamento, a irrigação inteligente dos parques públicos, o controlo da iluminação ambiental e a gestão do tráfego condicionado. Cada solução funciona num domínio específico, mas partilham a mesma lógica subjacente de integração e automatização.

A gestão do estacionamento utiliza sensores e redes de comunicação para detetar lugares de estacionamento disponíveis e informar os condutores em tempo real, reduzindo o congestionamento e as emissões poluentes causadas pela procura prolongada de lugares livres. Ao mesmo tempo, as autoridades municipais podem processar estes dados para ajustar os preços, definir horários ou planear futuras intervenções nas vias públicas (Al-Turjman, & Malekloo, 2019; Barriga et al., 2019; Khanna, & Anand, 2018).

A rega inteligente de parques públicos utiliza dados sobre a humidade do solo, previsões meteorológicas e algoritmos de controlo para garantir que a água é fornecida em quantidades adequadas e nas horas ideais. Em contraste com os métodos tradicionais, esta abordagem evita o desperdício de um recurso escasso e pode ser monitorizada e ajustada remotamente, contribuindo assim para a sustentabilidade ambiental e minimizando os custos de manutenção (Aldegheishem et al., 2022; Gimpel et al., 2021; Gualpa et al., 2023).

O controlo da iluminação ambiental baseia-se na implementação de iluminação pública eficiente - frequentemente lâmpadas LED ligadas a sensores de movimento ou de luz. A intensidade da luz pode ser ajustada de acordo com a hora do dia, a presença de pessoas ou o traçado da rua, garantindo segurança noturna e poupança de energia. O mesmo sistema permite a rápida deteção e comunicação de avarias, facilitando a manutenção e evitando interrupções prolongadas do serviço (Gagliardi et al., 2020; Polimeni, & Iorgulescu, 2018).

A gestão do tráfego condicionado tem como objetivo regular dinamicamente o fluxo de veículos nas estradas urbanas, tendo frequentemente em conta as horas de ponta, incidentes

imprevistos ou eventos especiais. Através da utilização de câmaras, sensores e sinais de trânsito inteligentes, o tráfego pode ser distribuído de forma mais uniforme, os tempos de espera nos cruzamentos podem ser reduzidos e os níveis de poluição podem ser diminuídos. O registo sistemático de dados também permite às autoridades relevantes planear melhorias na rede rodoviária de forma mais sustentável (Li et al., 2017).

No seu conjunto, estas quatro áreas ilustram como vários aspetos da vida quotidiana numa *Smart City* - quer se trate da procura de estacionamento, da manutenção de espaços verdes, da supervisão da iluminação pública ou da regulação da circulação de veículos - se tornam cada vez mais eficientes, seguros e sustentáveis. A recolha e análise de dados em tempo real, combinadas com tecnologias de comunicação e controlo, promovem uma utilização mais racional dos recursos e respostas mais rápidas aos desafios urbanos do dia a dia. Consequentemente, uma *Smart City* não é apenas um centro urbano tecnologicamente avançado, mas sim um espaço concebido para promover o bem-estar da comunidade, equilibrando o crescimento económico, a conservação ambiental e uma melhor experiência urbana para todos (Lim et al, 2019; Zhou et al., 2023).

## Análise de Requisitos

Este projeto *Smart City* reúne várias soluções tecnológicas que, no seu conjunto, procuram otimizar recursos, aumentar a segurança e melhorar o conforto dos cidadãos. Para atingir estes objetivos, foram desenvolvidos quatro sistemas principais: O Sistema A, focado na iluminação exterior inteligente; o Sistema B, que assegura a climatização automática dos ambientes; o Sistema C, responsável pela segurança através de um alarme baseado na deteção de movimento; e o Sistema D, que ilustra uma aplicação adicional para o controlo de acessos, mas que pode ser adaptado para outros fins, como a gestão do tráfego ou do estacionamento.

Em termos de iluminação, o Sistema A lê a luminosidade ambiente através de sensores apropriados e ajusta a intensidade dos LEDs, utilizando escalas de luminosidade que vão desde níveis muito baixos até à intensidade máxima. Esta abordagem permite poupanças significativas de energia, uma vez que a iluminação só é aumentada quando é efetivamente necessária, mantendo também a segurança e o conforto nos períodos de pouca luz natural.

Por outro lado, o Sistema B utiliza sensores de temperatura para gerir um processo de arrefecimento automático através de um ventilador, que só se liga quando a temperatura ultrapassa os 24 °C e se desliga quando desce abaixo dos 20 °C. Este mecanismo reduz os consumos desnecessários e mantém as instalações dentro de um intervalo de temperatura confortável,

enquanto um ecrã LCD e dois LEDs informam os utilizadores sobre o estado do ventilador e a temperatura atual.

A segurança é outro dos pilares fundamentais da proposta, materializada no Sistema C. Aqui, a deteção de movimento através de um sensor PIR faz disparar, sempre que o sistema é armado, um alarme sonoro e luminoso durante dez segundos, com pausas de cinco segundos até ser desarmado por um botão. Estas indicações são registadas no monitor de série, permitindo ao responsável ver quando o alarme foi acionado e se o sistema foi corretamente desativado. Este mesmo método de registo facilita a análise de eventuais incidentes e dá apoio à tomada de decisões sobre a fiabilidade do espaço.

Por último, o sistema D demonstra a versatilidade das aplicações que podem ser integradas numa cidade inteligente. Esta mesma arquitetura de leitura de sinais pode ser facilmente alargada a funções como o controlo de semáforos, a deteção de veículos ou mesmo a automatização de barreiras em parques de estacionamento. A aplicação deste módulo, equipado com um servo ou motores específicos, permite uma melhor organização das infraestruturas urbanas, uma vez que o seu funcionamento automático reduz os atrasos e depende menos da intervenção humana.

Além destes requisitos funcionais, que definem claramente o que cada subsistema faz, existem aspetos não funcionais que garantem a qualidade, escalabilidade e eficiência do projeto. Por exemplo, o funcionamento contínuo e fiável depende da estabilidade das leituras dos sensores, do processamento inteligente dos dados e de mecanismos de proteção contra ruídos elétricos ou falhas de hardware. A eficiência energética - destacada no controlo da iluminação e na gestão da climatização - não se limita à otimização de cada subsistema, mas estende-se à minimização do consumo global de toda a solução. O projeto baseia-se em componentes económicos, como microcontroladores e sensores comuns, e está estruturado para ser de fácil manutenção, permitindo que partes do sistema sejam substituídas ou atualizadas sem afetar as restantes. Esta modularidade também permite a expansão de funcionalidades - seja aumentando o número de postes de iluminação, adicionando mais sensores de temperatura ou introduzindo novas aplicações de segurança. A adoção de normas de interoperabilidade, como protocolos de comunicação e bibliotecas padrão, garante a compatibilidade com outras plataformas, tanto para monitorização em tempo real como para recolha de dados históricos.

# Especificação do Sistema

## Sistema A

O Sistema foi desenvolvido para ajustar automaticamente a intensidade de iluminação de um espaço exterior, pertencente à *SmartCity*, baseando-se na luminosidade detetada por um sensor LDR / Fotoresitor. Assim é possível ter um sistema dinâmico e de eficiência energética. O sistema utiliza um LED para simular a iluminação do espaço, até 5 níveis de intensidade configurados, mas facilmente escalável dependendo das necessidades de cada *SmartCity*.

### Especificações Funcionais:

As funcionalidades principais do sistema são:

- Leitura da Luz Ambiente;
- Ajuste dinâmico da intensidade do LED;
- Monitorização via Serial Monitor / Monitor de Série.

### Especificações Não Funcionais:

- Confiabilidade;
- Desempenho;
- Eficiência Energética;
- Simplicidade;
- Escalabilidade.

## Sistema B

O Sistema B foi implementado para gerir automaticamente o processo de arrefecimento de um espaço, integrando-se na SmartCity através de sensores de temperatura. Deste modo, é possível manter as instalações num intervalo de conforto, ligando a ventoinha sempre que a temperatura ultrapassar os 24 °C e desligando-a abaixo dos 20 °C. Esta abordagem reduz consumos desnecessários e prolonga a vida útil dos equipamentos, enquanto um ecrã LCD e dois LEDs

(vermelho e verde) proporcionam feedback imediato do estado do sistema, informando sobre a temperatura atual e se o arrefecimento está ativo ou não.

### Especificações Funcionais:

As funcionalidades principais do sistema são:

- Leitura de Temperatura Ambiente;
- Controlo Dinâmico do Ventilador (liga acima de 24 °C e desliga abaixo de 20 °C);
- Apresentação do Estado (através de LCD e LEDs).

### Especificações Não Funcionais:

- Confiabilidade
- Desempenho
- Eficiência Energética
- Simplicidade
- Escalabilidade

## Sistema C

O Sistema C foi elaborado para reforçar a segurança de um determinado espaço, recorrendo a um sensor PIR que deteta movimento e aciona um alarme sonoro e luminoso sempre que o sistema estiver armado. O alarme mantém-se ativo por dez segundos, seguindo-se pausas de cinco segundos, repetindo este ciclo até ser desarmado por um botão. Além disso, todas as ocorrências são registadas no monitor de série, possibilitando ao responsável analisar em que momento o alarme disparou e confirmar se o processo de desativação foi devidamente concluído. Este registo também apoia a investigação de eventuais incidentes e contribui para avaliar a fiabilidade do local onde o sistema está instalado.

### Especificações Funcionais:

As funcionalidades principais do sistema são:

- Deteção de Movimento através de Sensor PIR
- Ativação do Alarme Sonoro e Luminoso (durante 10s, com pausas de 5s)
- Desarme por Botão (interrompe de imediato o ciclo de alarme)
- Registo de Ocorrências no monitor de série (hora de disparo, estado de desarme)

### Especificações Não Funcionais:

- Confiabilidade: O sistema deve assegurar elevada robustez e minimizar falsos alarmes, garantindo um funcionamento correto e ininterrupto em ambiente real.
- Desempenho: A resposta à deteção de movimento deve ser rápida, de modo a acionar o alarme sem atrasos significativos.
- Eficiência Energética: Ainda que seja um sistema de segurança, deve otimizar o consumo de energia, nomeadamente quando está em modo de espera (standby).
- Simplicidade: A instalação e utilização do sistema devem ser intuitivas, facilitando a manutenção e a operação diária por utilizadores não especialistas.
- Escalabilidade: Deve permitir a integração de componentes adicionais (mais sensores, outros tipos de alarmes) sem grandes modificações no hardware ou no software, de forma a acomodar diferentes cenários de segurança.

## Sistema D

O Sistema foi realizado para controlar o movimento de um portão (ou barreira) através de sinais de infravermelhos (IR), simulando assim uma funcionalidade de acesso automatizado no contexto de uma SmartCity (acessos restritos). Ao receber um sinal de um comando remoto, o servomotor ajusta a posição (por exemplo, aberto/fechado), enquanto o sensor IR deteta o código correspondente. Esta abordagem permite uma interação rápida e prática, demonstrando, de forma simples, como um sistema de portões inteligentes pode ser implementado em qualquer cenário de cidade moderna, bastando alterar as configurações para os códigos específicos do comando IR em uso.

### Especificações Funcionais:

As funcionalidades principais do sistema são:

- Receção de Sinais Infravermelhos: Leitura do código enviado pelo comando remoto para identificar instruções de abrir ou fechar.
- Controlo Dinâmico de Servomotor: Ajuste de posição (ex.: 0° ou 90°) conforme os comandos de abertura ou encerramento do portão.
- Visualização e Depuração: Impressão do valor do código IR (em formato hexadecimal, por exemplo) no monitor de série, permitindo validar o correto funcionamento.

### Especificações Não Funcionais:

- Confiabilidade: O sistema deve responder de forma consistente aos sinais IR garantindo que o portão (servo) se mova adequadamente.
- Desempenho: O reconhecimento dos códigos e a rotação do servo devem ocorrer em tempo oportuno, sem atrasos injustificados para o utilizador.
- Eficiência Energética: Mesmo utilizando um servomotor e sensor IR, o consumo global deve manter-se otimizado, principalmente quando o servo se encontra inativo.
- Simplicidade: Tanto a implementação (uso de bibliotecas IRremote e Servo) como a operação (receber sinal e mover o servo) devem ser diretas, facilitando testes e manutenções.
- Escalabilidade: Possibilidade de acrescentar mais códigos ou outros atuadores (por exemplo, vários servos) sem alterar drasticamente a lógica base, permitindo uma adaptação fácil a diferentes cenários de acesso automatizado numa SmartCity.

## Modelo de Conceção

### Modelo de refinamento sucessivo

O modelo de refinamento sucessivo, sendo o escolhido na elaboração deste trabalho prático, pode ser entendido como uma estratégia de desenvolvimento incremental, em que a complexidade do sistema é construída passo a passo. Em vez de definir todo o projeto de forma rígida na fase inicial, as funcionalidades ou requisitos são introduzidos progressivamente, com cada iteração validada antes de passar à seguinte. Esta filosofia é particularmente adequada aos cenários *Smart City* discutidos ao longo deste trabalho prático, em que cada sistema - iluminação, controlo climático, segurança ou gestão de acessos - pode inicialmente ser padronizado de forma simples e gradualmente enriquecido com novas funcionalidades ou níveis de integração (Young et al., 1998; Dragomir, 2010; Lombardi et al., 2012; Andreani et al., 2019).

Ao aplicar o modelo de refinamento sucessivo, a equipa de desenvolvimento pode avaliar, em cada iteração, se o sistema satisfaz as expectativas do utilizador e mantém a coesão entre os diferentes módulos. Exemplificando, inicialmente, pode ser implementado o controlo básico de um LED com base na luz ambiente. Posteriormente, podem ser adicionados o registo de dados em tempo real, capacidades de controlo remoto e uma interface gráfica mais abrangente. Este método também permite a deteção precoce de problemas, possibilitando correções específicas sem

comprometer todo o projeto (Young et al., 1998; Dragomir, 2010; Lombardi et al., 2012; Andreani et al., 2019).

Além disso, este modelo promove a colaboração contínua, em que cada iteração proporciona uma oportunidade de validação e recolha de feedback, facilitando a incorporação de novas ideias, a reavaliação de prioridades e o ajuste das especificações. No contexto de uma cidade inteligente, em que cada subsistema pode partilhar dados ou recursos com outros, esta abordagem incremental revela-se especialmente valiosa, uma vez que a integração faseada ajuda a garantir a compatibilidade dos módulos e a gestão eficiente dos recursos. Assim, o refinamento sucessivo torna-se não só uma forma de desenvolver soluções mais robustas e seguras, mas também um catalisador para a inovação e a melhoria contínua ao longo de todo o seu ciclo de vida (Young et al., 1998; Dragomir, 2010; Lombardi et al., 2012; Andreani et al., 2019).

# Construção do Sistema

## 1. Sistema A:

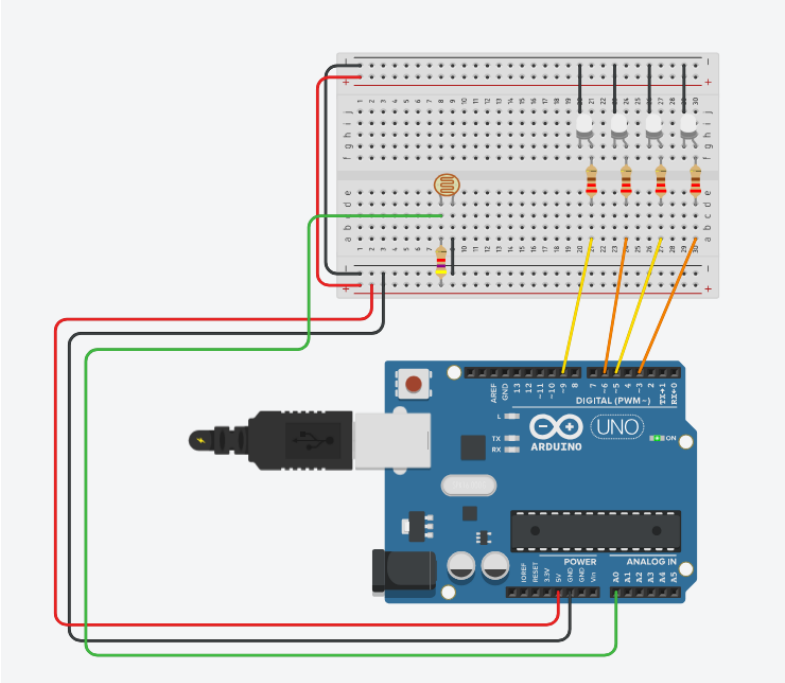


*Figura 1 - Sistema A – Tinkercad*

***Descrição***: Neste sistema, utilizando o *TinkerCad*, implementamos um circuito com 4 LEDS, 4 Resistências de 220 Ohms para controlo de picos de Tensão, 1 Fotoresistor com uma resistência de 4.7kOhms para controlo de pico de Tensão / amenizar ruido de sinal. Os pinos escolhidos para conectar os LEDs ao Arduíno (3, 5, 6 e 9) são os denominados pinos de PWM (*Pulse-Width Modulation*), que servem de controlo da intensidade de luminosidade dos LEDs. O pino A0 servirá como pino analógico de ligação ao Fotoresitor.

## 2. Sistema B:

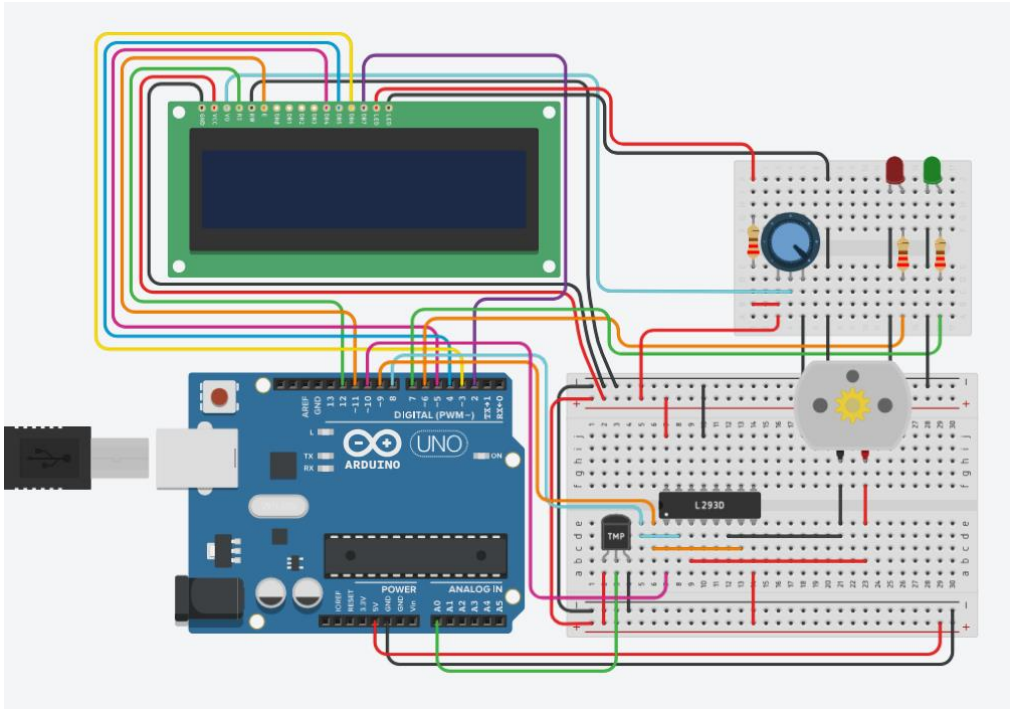


*Figura 2 - Sistema B – Tinkercad*

**Descrição**: Neste sistema, utilizando o *TinkerCad*, implementamos um circuito com 1 LCD (Hatachi), 2 LEDs (1 Vermelho e 1 Verde) com 2 resistências de 220 Ohms para controlo de pico de Tensão, 1 Potenciómetro com 1 resistência de 220 Ohms associada para controlar pico de voltagem tanto para o potenciómetro como para o *backlight* do LCD, 1 sensor TMP36 para servir como nosso sensor de temperatura, 1 Motor DC e 1 L293D para controlar a velocidade do motor. Os pinos escolhidos para conectar o LCD ao Arduíno seguem a documentação da Hatachi, segue abaixo:

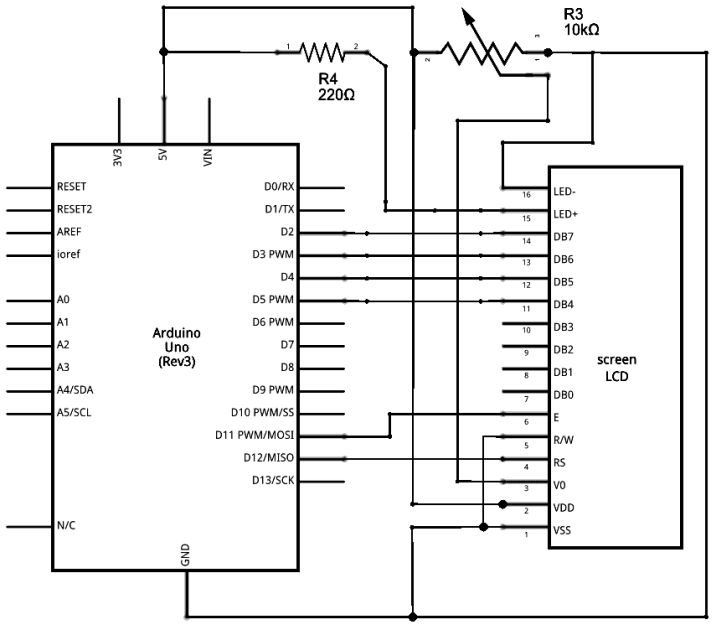


*Figura 3 - Esquema entre ligação do Arduíno e o LCD*

| **Pino** | **Função** |
|---|---|
| LED- | Cátodo do Backlight (GND) |
| LED+ | Ânodo do Backlight (VCC) |
| DB7 – DB0 | Pinos de Dados / Sinal |
| E | Ativar (Envio de Sinais ao LCD) |
| RW | Leitura / Escrita (GND para ficar em apenas modo Escrita) |
| RS | Comandos / Dados |
| V0 | Contraste |
| VDD | 5V / VCC |
| VSS | GND / Terra |

Para o sensor TMP36, apenas nos devemos preocupar com a polaridade do VCC e do GND, visto que o pino de sinal é sempre o do meio – melhor dizendo, é importante salientar que no circuito de teste, utilizamos tanto um TMP36 como um DHT11, estando a diferença na manipulação entre sinais analógicos (TMP36 no A0) e/ou sinais digitais (DHT11 no D13).

Para o L293D, também seguimos a documentação fornecida do mesmo para efetuar as ligações ao Arduíno, segue abaixo:

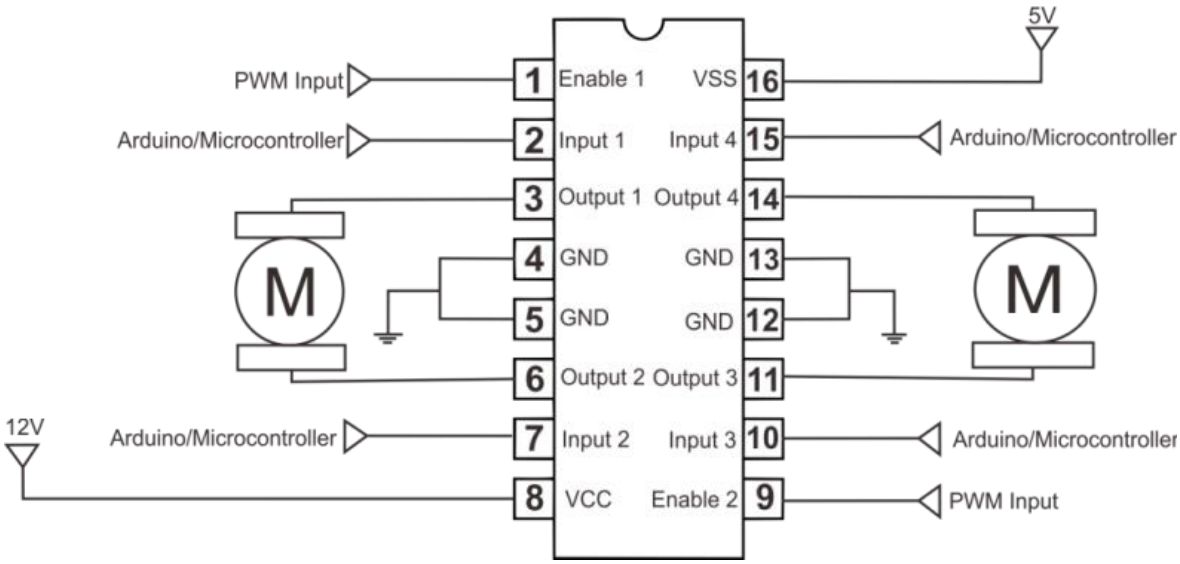


*Figura 4 - Esquema de ligações entre Arduíno, Motor e L293D*

Dado que só utilizamos 1 Motor DC, foram apenas utilizados os pinos de Input 1 e 2 para conexão ao Arduíno, com o pino Enable 1 também conectado, e os pinos de Output 1 e 2 para ligação ao Motor correspondente.

## 3. Sistema C:

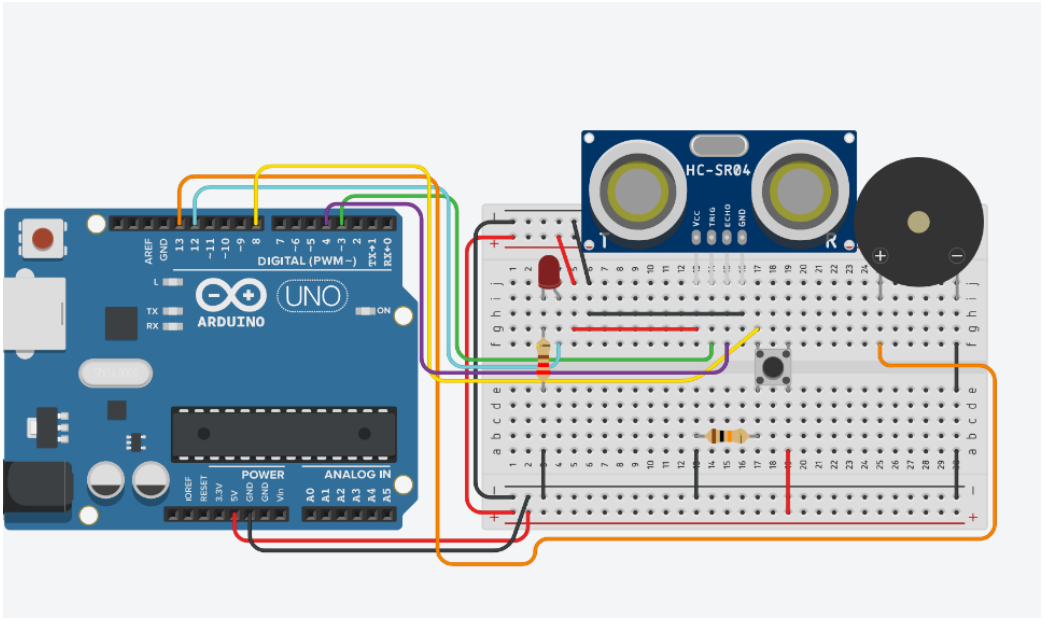


*Figura 5 - Sistema C - Tinkercad*

**Descrição**: Neste sistema, utilizando o *TinkerCad*, implementamos um circuito com 1 LED Vermelho com uma resistência de 220 Ohms para controlo de picos de Tensão, 1 Botão com uma resistência de 10 k Ohms associada, 1 Sensor de Ultrassons (HC-SR04) e 1 Buzzer (Ativo). Os pinos escolhidos para conectar o HC – SR04 (ERRO : PIR) e o Buzzer ao Arduíno seguem a documentação (ERRO: Hatachi), segue abaixo:

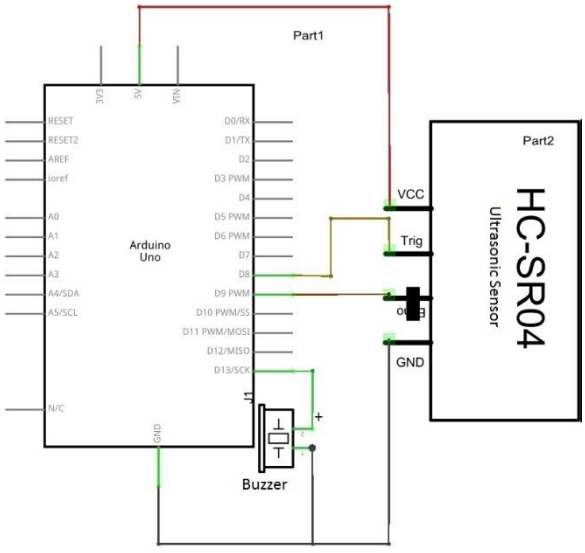


*Figura 6 - Esquema entre ligação do Arduíno, Sensor de Ultrassons e Buzzer*

Devemos ter em conta a polaridade no caso do HC-SR04, mas não no caso do Buzzer, visto este ser ativo, ou seja, não possui polaridade e o tom é constante. No caso do HC-SR04, temos os pinos VCC e GND, pino Trig para receção do sinal do Arduino para ativar a sua medição de distância e o pino de Echo para receção do sinal no Arduino, tendo em conta o tempo que demorou para "ecoar".

## 4. Sistema D:

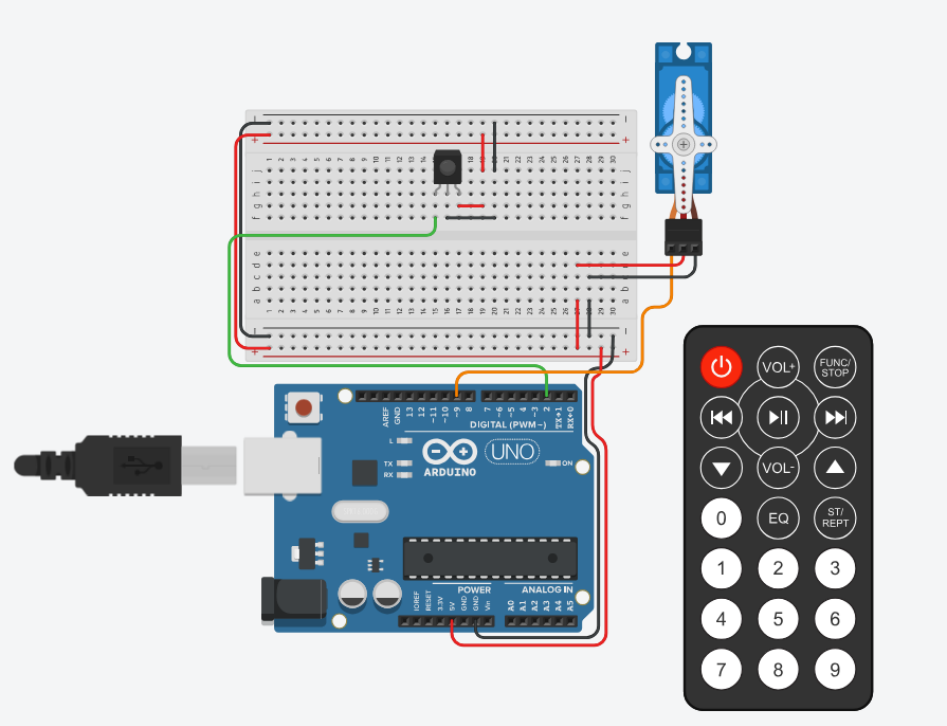


*Figura 7 - Sistema D - Tinkercad*

**Descrição**: Neste sistema, utilizando o *TinkerCad*, implementamos um circuito com 1 sensor de infravermelhos, 1 Servomotor e um Comando como emissor de sinais infravermelhos. Tanto o servomotor como o sensor de infravermelhos utilizam sinais digitais, logo ocupam pinos digitais no Arduíno, respetivamente pinos 9 e 2.

# Codificação

## 1. Sistema A:

```ino
// Definição dos pinos
#define LDR A0
#define LED1 3
#define LED2 5
#define LED3 6
#define LED4 9

void setup() {
  // Configuração dos pinos como saída
  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);
  pinMode(LED3, OUTPUT);
  pinMode(LED4, OUTPUT);
  Serial.begin(9600); // Inicialização da comunicação serial
}

void loop() {
  int LDRStatus = analogRead(LDR); // Leitura do valor do sensor LDR
  Serial.print("Nivel atual da luminosidade: ");
  Serial.println(LDRStatus);

  // Brilho
  if (LDRStatus < 200) {
    analogWrite(LED1, 0); // Brilho nulo (0%)
    analogWrite(LED2, 0);
    analogWrite(LED3, 0);
    analogWrite(LED4, 0);
    }
  else if (LDRStatus >= 200 && LDRStatus < 400) {
    analogWrite(LED1, 64); // Brilho baixo (25%)
    analogWrite(LED2, 64);
    analogWrite(LED3, 64);
    analogWrite(LED4, 64);
    }
  else if (LDRStatus >= 400 && LDRStatus < 600) {
    analogWrite(LED1, 128); // Brilho médio (50%)
    analogWrite(LED2, 128);
    analogWrite(LED3, 128);
    analogWrite(LED4, 128);
  }
  else if (LDRStatus >= 600 && LDRStatus < 800) {
    analogWrite(LED1, 192); // Brilho alto (75%)
    analogWrite(LED2, 192);
    analogWrite(LED3, 192);
    analogWrite(LED4, 192);
  }
  else if (LDRStatus >= 800) {
    analogWrite(LED1, 255); // Brilho máximo (100%)
    analogWrite(LED2, 255);
    analogWrite(LED3, 255);
    analogWrite(LED4, 255);
  }

  delay(500); // Pequeno atraso para estabilidade
}

```

*Figura 8 - Código do Sistema A*

**Descrição**: Primeiramente, definimos os pinos que iremos utilizar, neste caso os 4 LEDs e o fotoresistor (LDR) e dividimos o código em 2 ramificações: *Setup* e *Loop*.

Iniciando o void *Setup*, configuramos os 4 LEDs como OUPUT, mas ajustamos o LDR dentro do void Loop e não dentro do void Setup porque precisamos que a leitura dos dados seja feita constantemente para atualização da temperatura em tempo real.

Já dentro do void Loop, temos a leitura dos dados fornecidos pelo sensor e dois "Serial.Print" para visualização no Serial Monitor do Arduíno IDE, onde iremos executar este código.

Depois temos 5 condições de teste para aumentar ou diminuir a intensidade da luminosidade fornecida pelos LED's, dependendo do valor fornecido pelo sensor.

## 2. Sistema B:

```ino
#include <LiquidCrystal.h>

#define LEDVERM 6   // LED Vermelho (arrefecimento)
#define LEDVERD 7    // LED Verde (estabilizado)
#define IN1 8        // Entrada 1 do L293D (controle de direção)
#define IN2 9        // Entrada 2 do L293D (controle de direção)
#define ENA 10       // Pino de ativação (ENA) do L293D (controle de velocidade)
#define TMP A0       // Sensor TMP36

LiquidCrystal LCD(12, 11, 5, 4, 3, 2); // RS, E, D4, D5, D6, D7

void setup() {

  pinMode(LEDVERM, OUTPUT);
  pinMode(LEDVERD, OUTPUT);

  // Configuração das entradas do L293D
  pinMode(IN1, OUTPUT); // Pino 8 do L293D
  pinMode(IN2, OUTPUT); // Pino 9 do L293D
  pinMode(ENA, OUTPUT); // Pino 1 do L293D (controle de velocidade)

  // Iniciar comunicação com LCD
  LCD.begin(16, 2); // Configurar o LCD com 16 colunas e 2 linhas

  // Mensagem inicial no LCD
  LCD.print("Sistema B");
  delay(2000);
  LCD.clear();
}

void loop() {

  int leituraTMP = analogRead(TMP);

  // Converter a leitura para a temperatura em Celsius
  float voltagem = leituraTMP * (5.0 / 1023.0);
  float temperatura = (voltagem - 0.5) * 100.0;
  //float temperatura = max(0, (voltagem - 0.5) * 100.0);


  // Mostrar a temperatura no LCD
  LCD.setCursor(0, 0); // Primeira linha

  if (temperatura > 24) {
    LCD.print("Fan ON      ");

    // Ativar o motor e ajustar a velocidade
    analogWrite(ENA, 128);  // Ativar o motor na velocidade máxima
    digitalWrite(IN1, HIGH); // Definir a direção do motor (sentido horário)
    digitalWrite(IN2, LOW);  // Definir a direção do motor

    digitalWrite(LEDVERM, HIGH); // Liga LED Vermelho
    digitalWrite(LEDVERD, LOW);  // Desliga LED Verde
  }
  else if (temperatura < 20) {
    LCD.print("Fan OFF     ");

    // Desliga o motor
    analogWrite(ENA, 0);    // Desativa o motor
    digitalWrite(IN1, LOW); // Garante que o motor está desligado
    digitalWrite(IN2, LOW); // Garante que o motor está desligado

    digitalWrite(LEDVERM, LOW);  // Desliga LED Vermelho
    digitalWrite(LEDVERD, HIGH); // Liga LED Verde
  }
  else {
    LCD.print("Fan Stable  ");

    digitalWrite(LEDVERM, LOW);  // Desliga LED Vermelho
    digitalWrite(LEDVERD, HIGH); // Liga LED Verde
  }

  // Segunda linha para temperatura
  LCD.setCursor(0, 1); // Segunda linha
  LCD.print("Temp: ");
  LCD.print(temperatura);
  LCD.print((char)223); // Símbolo de grau
  LCD.print("C  ");

  delay(1000); // Atualização a cada 1 segundo
}
```

*Figura 9 - Código do Sistema B*

**Descrição**: Para este sistema, tivemos de instalar uma biblioteca externa denominada por "LiquidCrystal", bastante utilizada, focalizada para manipular entradas e saídas de dados relativos ao LCD (Liquid Crystal Display).

Inicialmente, definimos os pinos para os dois LEDs, os dois inputs do L293D tal como o pino de ativação do mesmo e o sensor de Temperatura (TMP36).

Para definir os pinos que iremos utilizar para o LCD, utilizamos uma função da biblioteca LiquidCrystal, a "LiquidCrystal LCD(arg)".

Para o void Setup, configuramos o tipo de dados utilizados para os LEDs, L293D e o tamanho do display que iremos utilizar (16x2)

No método do void loop, começamos por recolher dados analógicos do sensor TMP36, convertendo-os em valores de temperatura. Assumimos que os dados fornecidos pelo sensor possuem um limite máximo de 1023, utilizamos uma tensão de 5V fornecida pelo Arduíno através da porta USB do computador e aplicamos um fator de escala de 100. Finalmente, apresentamos a temperatura na posição (0x, 0y) no ecrã.

De seguida, testamos algumas condições que se verificam verdadeiras, como ligar a ventoinha (motor dc), caso a temperatura ultrapasse os 24ºC, desligar a ventoinha caso baixe dos 20ºC, e manter-se estável caso nenhuma das acima se verifique (escalável, permitindo, por exemplo, manter uma velocidade mínima do motor para manter a temperatura fresca), tudo para a 1ª linha do LCD.

Por fim, na 2ª linha do Display, fazemos print ao valor concreto da temperatura com o símbolo "º"

## 3. Sistema C:

```ino
// Pinos
const int TRIG = 3;       // Pino Trigger do HC-SR04
const int ECHO = 4;       // Pino Echo do HC-SR04
const int LEDVERM = 12;   // LED vermelho
const int BUZZ = 13;      // Buzzer
const int BTN = 8;        // Botão para desarmar

// Variáveis de controle
bool alarmArmed = true;         // Estado do alarme (armado por padrão)
bool lastButtonState = HIGH;    // Estado anterior do botão
unsigned long lastDebounceTime = 0; // Para debounce
unsigned long debounceDelay = 50;   // Tempo para debounce
unsigned long lastAlarmTime = 0;    // Para o controle do alarme
unsigned long alarmStartTime = 0;   // Marca o início do alarme

bool isAlarmActive = false;     // Indica se o alarme está ativo ou no intervalo
const int DISTANCE_THRESHOLD = 50; // Distância mínima (em cm) para ativar o alarme

void setup()
{
  pinMode(TRIG, OUTPUT);        // Configura o pino Trigger como saída
  pinMode(ECHO, INPUT);         // Configura o pino Echo como entrada
  pinMode(LEDVERM, OUTPUT);     // Configura o LED como saída
  pinMode(BUZZ, OUTPUT);        // Configura o buzzer como saída
  pinMode(BTN, INPUT_PULLUP);   // Configura o botão como entrada com pull-up interno
  Serial.begin(9600);           // Inicia a comunicação serial
}

void loop()
{
  // Leitura do estado do botão com debounce
  int buttonState = digitalRead(BTN);
  if (buttonState == LOW && lastButtonState == HIGH && (millis() - lastDebounceTime) > debounceDelay) {
    alarmArmed = !alarmArmed; // Alterna o estado do alarme
    lastDebounceTime = millis(); // Atualiza o tempo do debounce

    // Debug: Estado atual do alarme
    if (alarmArmed)
    {
      Serial.println("Alarme ARMADO.");
    } else {
      Serial.println("Alarme DESARMADO.");
      isAlarmActive = false; // Desativa qualquer alarme ativo
      digitalWrite(LEDVERM, LOW); // Garante que o LED esteja apagado
      noTone(BUZZ);               // Garante que o buzzer esteja desligado
    }
  }
  lastButtonState = buttonState; // Atualiza o estado anterior do botão

  // Verifica o estado do sensor ultrassônico se o alarme estiver armado
  if (alarmArmed)
  {
    int distance = measureDistance(); // Mede a distância

    if (distance > 0 && distance <= DISTANCE_THRESHOLD && !isAlarmActive)
    {
      // Inicia o alarme
      isAlarmActive = true;
      Serial.println("Movimento detectado! Alarme ativado.");
    }

    // Ciclo contínuo de 10 segundos de som seguido de 5 segundos de pausa
    if (isAlarmActive)
    {
      unsigned long currentTime = millis();

      // Controle do ciclo de tocar/pausa
      static bool isAlarmPlaying = true; // Indica se está a tocar ou em pausa
      static unsigned long cycleStartTime = currentTime; // Tempo de início do ciclo atual

      if (isAlarmPlaying && currentTime - cycleStartTime <= 10000)
      {
        // Tocar o alarme (LED pisca e buzzer toca)
        digitalWrite(LEDVERM, millis() % 500 < 250 ? HIGH : LOW); // LED pisca a cada 250ms
        tone(BUZZ, 1000); // Sinal sonoro com frequência de 1000Hz
      }
      else if (isAlarmPlaying && currentTime - cycleStartTime > 10000)
      {
        // Termina o ciclo de tocar e inicia a pausa
        isAlarmPlaying = false;
        cycleStartTime = currentTime; // Reinicia o tempo do ciclo
        digitalWrite(LEDVERM, LOW); // Garante que o LED esteja apagado
        noTone(BUZZ);               // Silencia o buzzer
        Serial.println("Pausa do alarme.");
      }
      else if (!isAlarmPlaying && currentTime - cycleStartTime >= 5000)
      {
        // Termina a pausa e volta a tocar
        isAlarmPlaying = true;
        cycleStartTime = currentTime; // Reinicia o tempo do ciclo
        Serial.println("Alarme retomado.");
      }
    }
  }

  delay(10); // Pequeno atraso para estabilidade
}

// Função para medir a distância com o sensor HC-SR04
int measureDistance()
{
  digitalWrite(TRIG, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG, LOW);

  long duration = pulseIn(ECHO, HIGH); // Mede o tempo de ida e volta do sinal ultrassônico
  int distance = duration * 0.034 / 2; // Calcula a distância em cm

  Serial.print("Distância medida: ");
  Serial.print(distance);
  Serial.println(" cm");
  return distance;
}
```

*Figura 10 - Código do Sistema C*

**Descrição**: Para o sistema C, começamos por inicializar os pinos que iremos utilizar como constantes, visto ser algo que não irá alterar-se, e também inicializamos as variáveis de controlo para podermos manipulá-las mais à frente.

No void *Setup*, configuramos os pinos para o sensor de ultrassons, o LED, o *Buzzer* e o Botão.

Para o *Loop*, começamos por verificar o estado do botão através do *Debounce* para impedir "ruído" de sinais relativos ao mesmo, como por exemplo, após carregar no botão uma vez, ele desativar como suposto, mas devido a oscilações mecânicas/eletrónicas, voltar a ativar (HIGH -> LOW -> HIGH), causando problemas no bom funcionamento do sistema pretendido.

Depois verificamos a condição do status do alarme "if (alarmArmed)", e caso essa condição se verifique, é medida a distância lida pelo HC-SR04 e caso essa distância seja maior que 0, ultrapasse o valor de *ThresHold* estipulado e se o alarme até essa instância estiver desarmado, é ativado e é também enviado uma mensagem de output através do Serial.Print, a informar que o alarme está ativado.

Para o ciclo contínuo do alarme, manipulamos a subtração entre o tempo atual de ativação e o tempo de início da ativação do alarme, necessitando que esse valor final seja menor que 10000 (10s) e caso essa condição seja verdadeira, inicializamos a intermitência do sinal do LED, para que se ilumine a cada 250ms e o *Buzzer* inicialize o toque corrente. Caso o valor final ultrapasse os 10000, é inicializada a pausa de 5s, em que o LED e o *Buzzer* são desligados. Após os 5s (>= 5000), retorna-se ao ciclo do alarme a tocar, reiniciando o currentStartTime para que permaneça neste ciclo até existir paragem humana.

Na função measureDistance, começámos com um *delay* de 2ms para garantir que valores residuais ou ruídos não sejam lidos. A seguir, enviamos um pulso através do *Trig* durante 10ms. Após esse pulso, é desligado para análise dos dados obtidos até esse instante. O tempo em que o *Echo* permanece em *High* (ativo) é recolhido e calculamos a distância baseando-nos na velocidade do som no ar, atendendo às condições do ambiente de teste (340ms ou 0.034 cm/microssegundos). Igualmente, dividimos o valor por 2 pois representa o pulso até receção, sendo uma viagem de ida e volta, e nós só queremos o tempo de ida para calcular a distância real do objeto.

Por fim são efetuados os Serial.Print para visualização dos valores de distância no Serial Monitor.

## 4. Sistema D:

```ino
/*
    IR Remote with Servo Motor Control

    Print out the received IR value and move a servo motor when a specific code is received.
*/

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

// Define pins
const int IR_RECEIVE_PIN = 2; // IR receiver pin
const int SERVO_PIN = 9;      // Servo motor pin

// Create a servo object
Servo gateServo;

// Define the IR code for the specific button (e.g., "5")
const uint32_t OPEN_GATE_CODE = 0xF50ABF00; // Replace with your specific code
const uint32_t CLOSE_GATE_CODE = 0xF708BF00; // Replace with your specific code

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

  // Initialize IR receiver
  IrReceiver.begin(IR_RECEIVE_PIN, ENABLE_LED_FEEDBACK); // Start the receiver

  // Initialize the servo
  gateServo.attach(SERVO_PIN);
  gateServo.write(0); // Start with the gate closed (angle 0)

  Serial.println("Ready to receive IR signals...");
}

void loop() {
  // Check if an IR signal has been received
  if (IrReceiver.decode()) {
    uint32_t receivedCode = IrReceiver.decodedIRData.decodedRawData;

    // Print the received IR code
    Serial.print("Received IR Code: ");
    Serial.println(receivedCode, HEX);

    // Control the servo motor based on the received IR code
    switch (receivedCode) {
      case OPEN_GATE_CODE: // Code for opening the gate
        Serial.println("Opening the gate...");
        gateServo.write(90); // Move servo to 90 degrees (open gate)
        delay(2000);         // Hold position for 2 seconds
        break;

      case CLOSE_GATE_CODE: // Code for closing the gate
        Serial.println("Closing the gate...");
        gateServo.write(0);  // Move servo back to 0 degrees (close gate)
        delay(2000);         // Hold position for 2 seconds
        break;

      default: // Unrecognized code
        Serial.println("Unrecognized code.");
        break;
    }

    // Resume IR receiver for the next signal
    IrReceiver.resume();
  }
}
```

*Figura 11 - Código do Sistema D*

**Descrição**: Para o sistema D, precisámos de instalar 2 bibliotecas, a IRemote (que nos permitirá manipular o sensor de Infravermelhos) e a Servo (que nos permitirá manipular o servomotor), configurando os pinos para o sensor IR (IR_RECEIVE_PIN) e para o Servomotor (SERVO_PIN).

Começamos por criar um objeto utilizando a lib Servo denominado por gateServo, que usaremos para controlar o movimento do servo.

Inicializamos as variáveis que usaremos para abrir e fechar o portão, utilizando o servo, com valores aleatórios, visto que necessitaremos de voltar a alterar estas entradas para os valores específicos, transmitidos pelo comando via infravermelhos.

No void *Setup*, introduzimos a comunicação com o Serial Monitor via 9600hz (Canal de Comunicação escolhido), estabelecemos a receção de dados via sensor IR tendo um Enable Feedback ativo para visualizarmos o sensor a piscar quando recebe um sinal. Também configuramos o servomotor com posição inicial de 0 e enviamos um print ao Serial Monitor.

No void Loop, principiamos por verificar se a condição de termos recebido um sinal é verdadeira e, caso seja, armazenamos esse valor em receivedCode, prosseguindo aos prints do valor em hexadecimal. Para controlar o servomotor, utilizamos um *switch-case*: se o valor dado como input for o valor de abrir o portão (OPEN_GATE_CODE), o servomotor irá rodar 90º mantendo a sua posição por 2 segundos; contudo caso o valor de input seja para fechar o portão (CLOSED_GATE_CODE), o servomotor volta à posição 0.

Por fim, o sensor IR é reiniciado para estar pronto para receção do próximo sinal.

# Testes / Resultados

## 1. Sistema A:

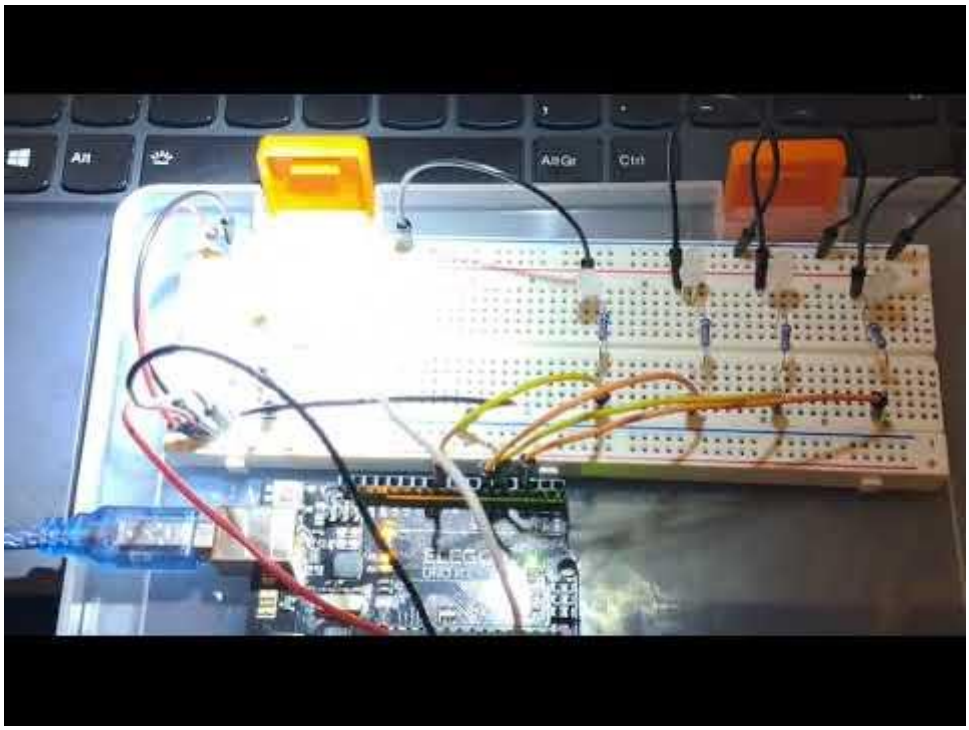

*Figura 12 – Vídeo do mockup do Sistema A*

## 2. Sistema B:

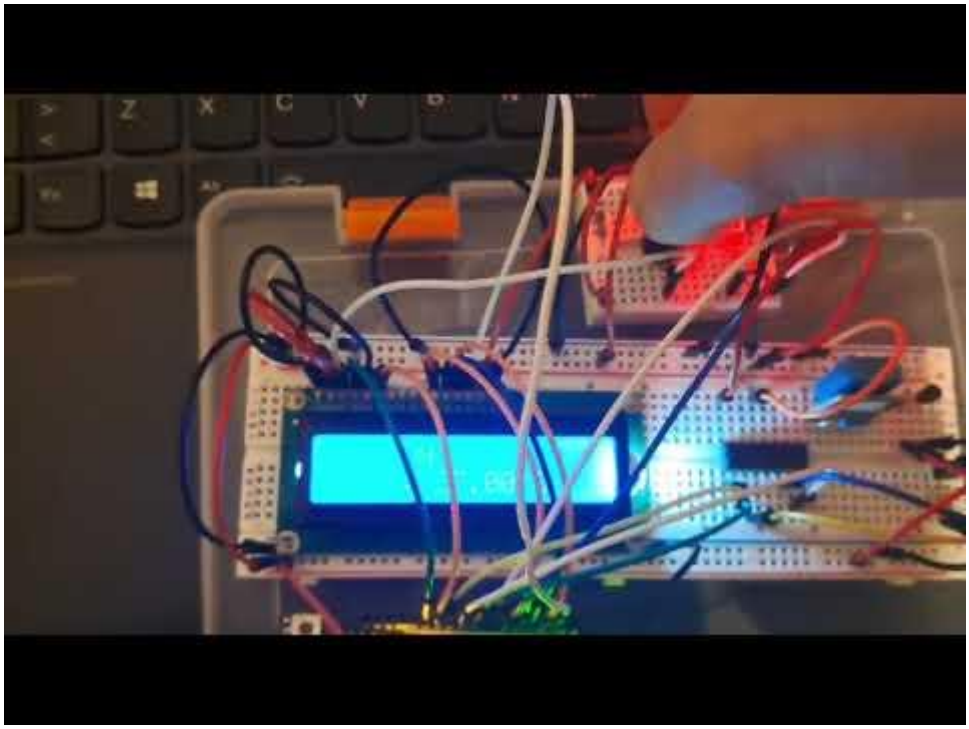

*Figura 13 – Vídeo do mockup do Sistema B*

## 3. Sistema C:

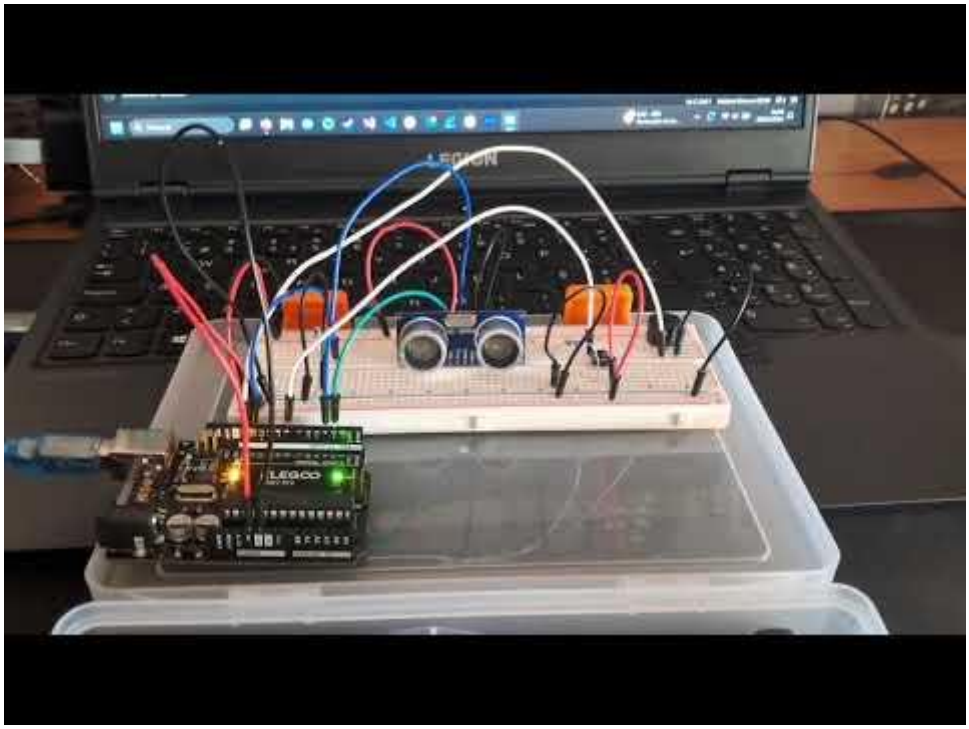

*Figura 14 – Vídeo do mockup do Sistema C*

## 4. Sistema D:

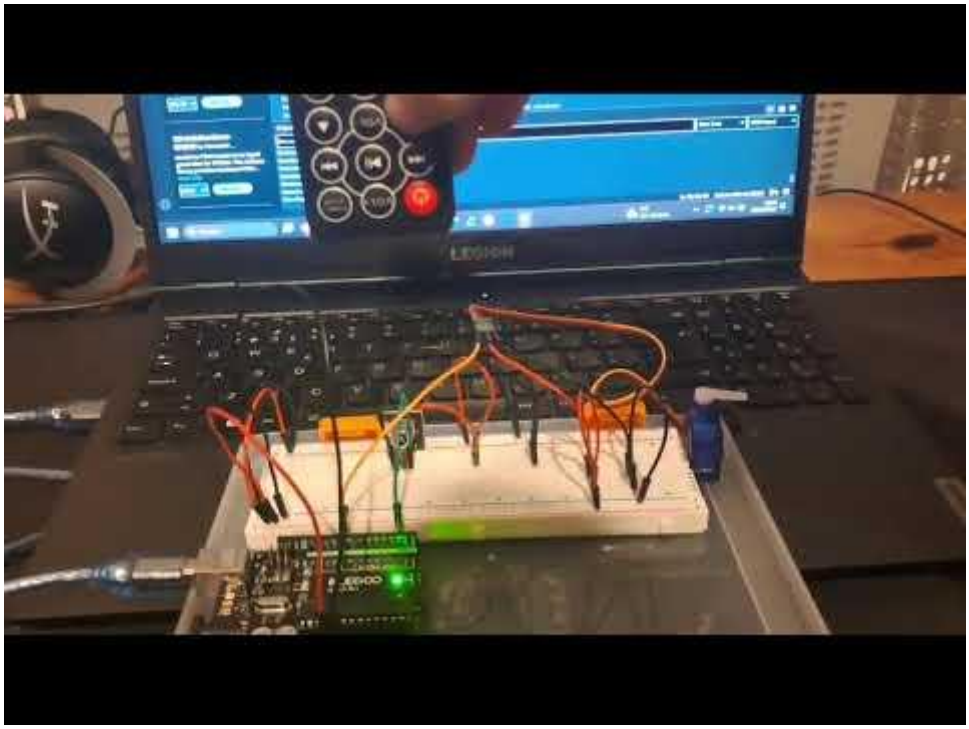

*Figura 15 – Vídeo do mockup do Sistema D*

# Conclusão

O desenvolvimento de sistemas inteligentes para ambientes urbanos, como demonstrado nos quatro exemplos (iluminação, climatização, segurança e possíveis extensões), ilustra como as cidades podem evoluir para modelos mais eficientes, mais seguros e mais sustentáveis. Cada sistema analisado - quer se trate da gestão dos consumos energéticos da iluminação pública, do controlo optimizador da climatização, do reforço da segurança contra as intrusões ou do apoio ao controlo de acessos - evidencia a importância da integração de tecnologias de monitorização e de ação em tempo real.

Além de responder às exigências de um mundo cada vez mais urbano e tecnológico, esta abordagem promove a adoção de práticas de engenharia focadas tanto na satisfação das necessidades imediatas da comunidade como na capacidade de escalar e adaptar estes projetos a desafios futuros. A utilização de sensores e atuadores comuns (e.g. LDRs, sensores PIR, ventoinhas, LEDs, ecrãs e controladores como o Arduíno) mostra também que, mesmo com soluções de baixo custo, é possível criar protótipos funcionais e versáteis que podem ser gradualmente melhorados e integrados em arquiteturas mais amplas de *Smart City*.

Igualmente, o trabalho apresentado sublinha a importância de equilibrar os requisitos funcionais (o que cada sistema deve fazer) com os requisitos não funcionais (como cada sistema deve operar, em termos de fiabilidade, desempenho, eficiência, simplicidade e escalabilidade). Esta combinação de práticas não só garante o funcionamento efetivo da solução hoje, como também facilita a manutenção, a expansão e a integração de novas funcionalidades no futuro.

Os quatro sistemas desenvolvidos demonstram diferentes aplicações num ambiente de prototipagem (Tinkercad) e em *mockup física* para uma *SmartCity*. No Sistema A, quatro LEDs e um fotoresistor regulam os níveis de luminosidade usando PWM. No Sistema B, um LCD, LEDs e um sensor TMP36 gerem o controlo da climatização, enquanto o motor DC é ativado através do driver L293D. No sistema C, um sensor ultrassónico e uma campainha formam um sistema de alarme, controlado por um botão com funcionalidade de debounce. Finalmente, no sistema D, um sensor de infravermelhos e um servo permitem o controlo de acessos restritos através da abertura e o fecho de uma "barreira de acesso" através de sinais infravermelhos enviados por um comando remoto.

Integrados num ecossistema de *Smart City*, todos estes sistemas apresentam eficiência energética, segurança e automação, contribuindo para uma cidade mais sustentável e funcional, para a qualidade de vida geral e para a utilização responsável dos recursos disponíveis.

Ao longo do desenvolvimento dos sistemas tivemos alguns constrangimentos devido à falta de alguns componentes inicialmente idealizados, como por exemplo no sistema B, onde inicialmente decidimos por controlar o Motor DC através de um díodo e de um transístor NPN mas visto que tínhamos em posse um L293D decidimos utilizar o mesmo para um controlo do motor mais facilitado. No sistema C, inicialmente tínhamos desenvolvido um sistema com base num sensor PIR mas foi necessário uma reformulação do sistema pois apenas tínhamos em posse um HC-SR04 (Sensor de ultrassons). Também no sistema D, inicialmente idealizamos desenvolver uma aplicação para integrar os 3 sistemas estipulados a desenvolver pelo Exmo. Professor de SETR, mas infelizmente não tínhamos em nossa posse um HC-05 (Módulo que permitiria comunicações com o Arduíno via Bluetooth) apesar de termos esquematizado a possibilidade de utilizar a placa integrada

do portátil para servir de comunicador, mas essa opção viu-se como bastante complexa devido à falta de bibliotecas que não estivessem em deterioração ( devido a serem de versões antiquas).

## Referências Bibliográficas

Aldegheishem, A., Alrajeh, N., García, L., & Lloret, J. (2022). SWAP: Smart WAter Protocol for the irrigation of urban gardens in Smart Cities. *IEEE Access*, PP, 1-1. https://doi.org/10.1109/ACCESS.2022.3165579.

Al-Turjman, F., & Malekloo, A. (2019). Smart parking in IoT-enabled cities: A survey. *Sustainable Cities and Society*. https://doi.org/10.1016/J.SCS.2019.101608.

Andreani, S., Kalchschmidt, M., Pinto, R., & Sayegh, A. (2019). Reframing technologically enhanced urban scenarios: A design research model towards human centered smart cities. *Technological Forecasting and Social Change*. https://doi.org/10.1016/J.TECHFORE.2018.09.028.

Barriga, J., Sulca, J., León, J., Ulloa, A., Portero, D., Andrade, R., & Yoo, S. (2019). Smart Parking: A Literature Review from the Technological Perspective. *Applied Sciences*. https://doi.org/10.3390/app9214569.

Caragliu, A., Del Bo, C., & Nijkamp, P. (2011). Smart Cities in Europe. *Journal of Urban Technology*, 18, 65 - 82. https://doi.org/10.1080/10630732.2011.601117.

Dragomir, I., Ober, I., & Percebois, C. (2010). System to software level model refinement in embedded system design. Université de Toulouse - IRIT.

Gagliardi, G., Lupia, M., Cario, G., Tedesco, F., Gaccio, F., Lo Scudo, F., & Casavola, A. (2020). Advanced Adaptive Street Lighting Systems for Smart Cities. *Smart Cities*. https://doi.org/10.3390/smartcities3040071.

Gimpel, H., Graf-Drasch, V., Hawlitschek, F., & Neumeier, K. (2021). Designing smart and sustainable irrigation: A case study. *Journal of Cleaner Production*, 315, 128048. https://doi.org/10.1016/J.JCLEPRO.2021.128048.

Gualpa, T., Caiza, G., Ayala, P., Garcia, C., & Garcia, M. (2023). Smart I4.0-Based Irrigation System for Optimization in Water Management: A case study. *2023 IEEE 28th International Conference on*

*Emerging Technologies and Factory Automation (ETFA)*, 1-8. https://doi.org/10.1109/ETFA54631.2023.10275443.

Khanna, A., & Anand, R. (2018). IoT based smart parking system. *2016 International Conference on Internet of Things and Applications (IOTA)*, 266-270. https://doi.org/10.1109/IOTA.2016.7562735.

Li, Z., Shahidehpour, M., Bahramirad, S., & Khodaei, A. (2017). Optimizing Traffic Signal Settings in Smart Cities. *IEEE Transactions on Smart Grid*, 8, 2382-2393. https://doi.org/10.1109/TSG.2016.2526032.

Lim, Y., Edelenbos, J., & Gianoli, A. (2019). Identifying the results of smart city development: Findings from systematic literature review. *Cities*, 95, 102397. https://doi.org/10.1016/J.CITIES.2019.102397.

Lombardi, P., Giordano, S., Farouh, H., & Yousef, W. (2012). Modelling the smart city performance. *Innovation: The European Journal of Social Science Research*, 25, 137 - 149. https://doi.org/10.1080/13511610.2012.660325.

Piro, G., Cianci, I., Grieco, L., Boggia, G., & Camarda, P. (2014). Information centric services in Smart Cities. *J. Syst. Softw.*, 88, 169-188. https://doi.org/10.1016/j.jss.2013.10.029.

Polimeni, J., & Iorgulescu, R. (2018). Smart city initiatives: Street lights. *Smart Cities and Regional Development (SCRD) Journal*, *2*(2), 59-64. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3413466

Young, J., MacDonald, J., Shilman, M., Tabbara, A., Hilfinger, P., & Newton, A. (1998). Design and specification of embedded systems in Java using successive, formal refinement. *Proceedings 1998 Design and Automation Conference. 35th DAC. (Cat. No.98CH36175)*, 70-75. https://doi.org/10.1145/277044.277058.

Zhou, Y., Xiao, F., & Deng, W. (2023). Is smart city a slogan? Evidence from China. *Asian Geographer*, 40, 185 - 202. https://doi.org/10.1080/10225706.2022.2052734.