TDE 2 - Performance em Sistemas Ciberfísicos

CESIA - Controle de Entradas e Saídas Independente do Ambiente

Integrantes : Fernanda Giacobbo, Julia de Souza Melo, Lara Eduarda Bredow, Pedro Antonio Serafim, Sofhia Chromiec

Repositório no Github: https://github.com/SofhiaC/TDE1-ControleDeOcupacao.git

1. Objetivo do Projeto

   O projeto tem como objetivo monitorar com precisão a movimentação de pessoas em determinado ambiente, utilizando dois sensores infravermelhos ativos para detectar entradas e saídas. O sistema é capaz de contar quantas pessoas estão presentes no local em tempo real, incrementando ou decrementando uma variável a partir da ordem de ativação dos sensores. Essa contagem é exibida simultaneamente em LEDs, display textual e em um painel (dashboard) digital - exemplo ao final da apresentação.

2. Justificativa

   O uso para implementação desse sistema é muito variável, se encaixando em estabelecimentos comerciais para estimativa de fluxo e lucro, em banheiros compartilhados para otimização da escala de limpeza, bibliotecas e salas de estudos para controle de capacidade, eventos e auditórios para controle da contagem de público de forma automática.

   Além disso, esse projeto também teria grande valor na época da pandemia da COVID-19 com relação ao controle de ocupação seguro nos mais variados ambientes da vida cotidiana.

3. Especificação do Sistema

   3.1. Hardware

   1 ESP32;
   1 Sensor de movimento IR Arduino;
   1 Receptor KY-02;
   1 LED infravermelho;
   2 LEDs vermelhos;
   1 LED amarelo;
   2 LEDs verdes;
   Resistores de 1K ohm;
   Resistores de 220 ohm;
   Display LDC 16x2;
   Cabos conectores fêmea x fêmea;
   Cabos conectores macho x macho;
   Cabos conectores macho x fêmea.

   A disposição dos sensores IR - componentes principais do projeto - funcionam de forma paralela para que a detecção seja dinâmica e completamente baseada na ordem da interrupção dos sensores.

3.2. Software

Para a simulação do projeto, foi utilizada a plataforma Wokwi e para desenvolvimento o grupo optou por utilizar o VS Code através da extensão *PlatformIO* que possibilitou um ótimo suporte para desenvolvimento em seu ambiente.

Como linguagem de programação, foi utilizado apenas C++ em todo o projeto, unido do uso de duas bibliotecas:

- LiquidCrystal
- Blynk

O código implementa a lógica de controle de ocupação com base na ordem de ativação de dois sensores infravermelhos. Ele define variáveis para controlar o tempo de detecção, o estado de espera entre sensores e o contador de pessoas presentes.

Na função *loop(),* o sistema realiza a leitura digital dos dois sensores. Quando o sensor 1 é ativado primeiro, o código inicia uma espera pelo sensor 2, dentro de um tempo limite de 3 segundos. Se essa sequência for completada corretamente, uma entrada é registrada, e o contador de pessoas é incrementado. Da mesma forma, se o sensor 2 for ativado primeiro e, em seguida, o sensor 1, o sistema interpreta como uma saída, e o contador é decrementado, desde que haja pelo menos uma pessoa no local.

Após cada detecção válida, a função *atualizarInterface()* é chamada para exibir a informação no LCD, atualizar os LEDs de status e enviar os dados ao aplicativo Blynk. O evento também é armazenado em uma string de histórico com o tipo de ação, número atual de pessoas e o tempo do evento.

Para evitar erros de leitura, o código inclui mecanismos de **timeout**: se o segundo sensor da sequência não for ativado dentro do tempo estipulado, a espera é cancelada e nenhum evento é registrado. A atualização dos LEDs físicos e virtuais é feita conforme a ocupação, limitando o número de LEDs acesos ao valor máximo pré-definido.

4. Diagrama da Arquitetura Ciberfísica

A lógica de funcionamento entre componentes e códigos funciona da seguinte forma:

Sensores → ESP32 → Blynk, LEDs e LCD

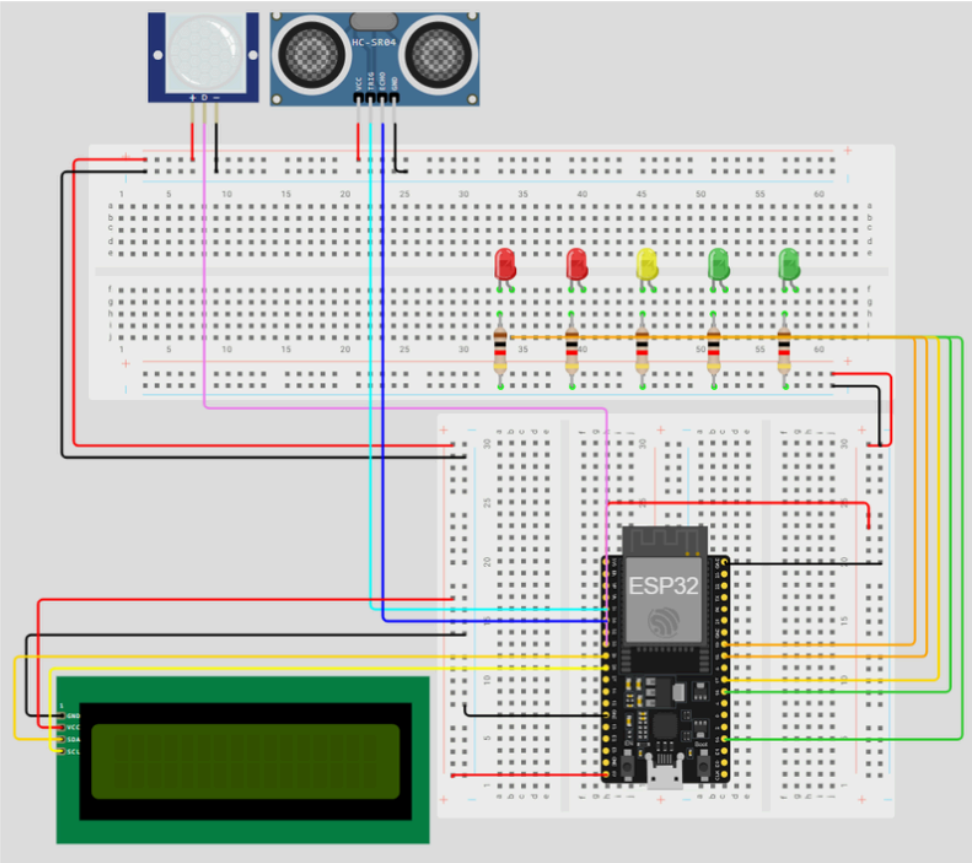

Imagem 1 retirada do sistema *Wokwi* de desenvolvimento

O sistema mostrado na Imagem 1 se refere ao que construímos virtualmente na plataforma *Wokwi* de simulação online. Nessa prototipagem temos todos os componentes implementados fisicamente seguindo o mesmo padrão, com exceção do sensor ultrassom que exemplifica o nosso uso de sensor IR composto separadamente por seu LED emissor e receptor KY-022.

5. Código-fonte Documentado

```
#define BLYNK_TEMPLATE_ID "TMPL2eYp6wwr_"
#define BLYNK_TEMPLATE_NAME "Midup"
#define BLYNK_AUTH_TOKEN "CqNfc1gSJKikV6zTHVBGNV_Att1HMb2W"

#include <Arduino.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <WiFi.h>
#include <BlynkSimpleEsp32.h>
#include <WebServer.h>

// LCD I2C no endereço 0x3F, com 16 colunas e 2 linhas
LiquidCrystal_I2C lcd(0x3F, 16, 2);
```

```
const int pinoEmissor = 14;
const int sensor1 = 13; // KY-022
const int sensor2 = 26; // Sensor reflexivo

const int leds[] = {15, 16, 17, 18, 19};
const int numLeds = 5;

const int blynkLedsVirtualPins[] = {V2, V3, V4, V5, V6};

unsigned long tempoSensor1 = 0;
unsigned long tempoSensor2 = 0;
bool esperandoSensor2 = false;
bool esperandoSensor1 = false;
unsigned long tempoLimite = 3000; // 3 segundos para timeout

char ssid[] = "Xiao";
char pass[] = "xiaooooo";

int pessoas = 0; // contador de pessoas presentes
String historico = "";

WebServer server(80);

void atualizarLeds(int pessoas) {
  int n = pessoas;
  if (n > numLeds) n = numLeds;
  if (n < 0) n = 0;

  for (int i = 0; i < numLeds; i++) {
    digitalWrite(leds[i], i < n ? HIGH : LOW);

    Blynk.virtualWrite(blynkLedsVirtualPins[i], i < n ? 255 : 0);
  }
}

void atualizarInterface(String evento) {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(evento);
  lcd.setCursor(0, 1);
  lcd.print("Pessoas: ");
  lcd.print(pessoas);
```

```
  atualizarLeds(pessoas);

  Blynk.virtualWrite(V1, pessoas);

  delay(2000);
  lcd.clear();
}

void setup() {
  Serial.begin(115200);

  pinMode(sensor1, INPUT);
  pinMode(sensor2, INPUT);
  for (int i = 0; i < numLeds; i++) {
    pinMode(leds[i], OUTPUT);
    digitalWrite(leds[i], LOW);
  }

  ledcSetup(0, 38000, 8);
  ledcAttachPin(pinoEmissor, 0);
  ledcWrite(0, 128);

  Wire.begin(33, 32);
  lcd.init();
  lcd.backlight();

  lcd.setCursor(0, 0);
  lcd.print("Iniciando WiFi...");
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);

  while (!Blynk.connected()) {
    delay(500);
    Serial.print(".");
  }

  lcd.clear();
  lcd.print("WiFi conectado!");
  delay(1000);
  lcd.clear();

  server.on("/", paginaWeb);
  server.begin();

  Serial.println("Servidor web iniciado.");
```

```
  Serial.print("IP do ESP32: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  Blynk.run();
  server.handleClient();

  int leitura1 = digitalRead(sensor1);
  int leitura2 = digitalRead(sensor2);
  unsigned long agora = millis();

  // Detecta sensor1 ativado primeiro
  if (leitura1 == LOW && !esperandoSensor2 && !esperandoSensor1) {
    tempoSensor1 = agora;
    esperandoSensor2 = true;
    Serial.println("Sensor 1 ativado, esperando sensor 2...");
  }

  // Detecta sensor2 ativado primeiro
  if (leitura2 == LOW && !esperandoSensor1 && !esperandoSensor2) {
    tempoSensor2 = agora;
    esperandoSensor1 = true;
    Serial.println("Sensor 2 ativado, esperando sensor 1...");
  }

  // Sequência entrada: sensor1 → sensor2
  if (esperandoSensor2 && leitura2 == LOW) {
    pessoas++;
    String evento = "Entrada detectada";
    Serial.println("📥 " + evento);
    historico += evento + " - " + String(pessoas) + " pessoas - " + String(agora /
1000) + "s\n";
    atualizarInterface(evento);
    Blynk.virtualWrite(V0, pessoas);
    esperandoSensor2 = false;
  }

  // Sequência saída: sensor2 → sensor1
  if (esperandoSensor1 && leitura1 == LOW) {
    if (pessoas > 0) pessoas--;
    String evento = "Saida detectada";
    Serial.println("📤 " + evento);
```

```
    historico += evento + " - " + String(pessoas) + " pessoas - " + String(agora / 1000) + "s\n";
    atualizarInterface(evento);
    Blynk.virtualWrite(V0, pessoas);
    esperandoSensor1 = false;
  }

  // Timeout para resetar espera
  if (esperandoSensor2 && (agora - tempoSensor1 > tempoLimite)) {
    esperandoSensor2 = false;
    Serial.println("⏱️ Timeout: sensor 2 não ativado a tempo");
  }
  if (esperandoSensor1 && (agora - tempoSensor2 > tempoLimite)) {
    esperandoSensor1 = false;
    Serial.println("⏱️ Timeout: sensor 1 não ativado a tempo");
  }

  delay(50);
}
```

6. Relatório de Desempenho e Testes

Para essa análise foram feitos testes em ambientes com baixa e média incidência de luz, com o uso de cronômetro para medição do tempo de resposta a partir de cada atuação nos sensores.

Foram realizados 20 testes (10 entradas e 10 saídas), utilizando movimentação com os dedos para a simulação de passagem devido a escala da implementação. As entradas e saídas foram feitas em diferentes velocidades para testar a confiabilidade do sistema. Também foi testado o comportamento com interrupção no meio da passagem (ex: pessoa parando entre os sensores, com apenas um sensor detectando movimento).

Teste	Ação	Resultado Esperado	Resultado Obtido	Status
1	Entrada	+1	0	Falha
2	Entrada	+1	+1	Sucesso
3	Entrada	+1	+1	Sucesso
4	Entrada	+1	+1	Sucesso
5	Entrada	+1	+1	Sucesso
6	Entrada	+1	+1	Sucesso

7	Entrada	+1	+1	Sucesso
8	Entrada	+1	+1	Sucesso
9	Saída	-1	0	Falha
10	Entrada	+1	+1	Sucesso
11	Saída	-1	-1	Sucesso
12	Saída	-1	-1	Sucesso
13	Saída	-1	0	Falha
14	Saída	-1	-1	Sucesso
15	Saída	-1	0	Falha
16	Saída	-1	-1	Sucesso
17	Saída	-1	-1	Sucesso
18	Saída	-1	-1	Sucesso
19	Saída	-1	-1	Sucesso
20	Entrada	+1	+1	Sucesso

Tabela 1 - Histórico de teste de sensores

Como resultado dos testes, temos que o tempo de resposta com sucesso se dá em 150ms, e sua precisão é de 80% e erros são mais possíveis de ocorrer se a passagem ocorrer com um curto período de tempo entre ela e a anterior. O sistema apresentou desempenho satisfatório em condições controladas, com precisão de 80%. Alguns ajustes são necessários para garantir maior robustez em ambientes com maior fluxo de pessoas ou com interferência externa.

7. Adicionais de Complexidade
   - 7.1. Registro de log's de operações
     O código envia todos os dados para o registro na plataforma *Blynk*, a qual registra como o usuário programar para ser exibido (ex: em forma de gráficos, texto ou apenas números)
   - 7.2. Interface de monitoramento em tempo real
     Com o auxílio da extensão *Blynk*, todos os dados são enviados via conexão wifi para o servidor e em seguida exibidos em tempo real no dashboard da aplicação *Blynk*, seja ela pelo celular ou computador.
   - 7.3. Transmissão eficiente de dados visuais

Com o auxílio da extensão *Blynk*, LEDs e o LCD, todos os dados de entradas e saídas são expostos visualmente.

7.4. Processamento eficiente de dados

Como os sensores são implementados de forma simples, se focando em seu controle de estado e, sem o uso de interrupções constantes, o sistema se torna eficiente durante todo seu tempo de uso. Além de não possuir o *delay()* em sua programação.