Studentų 50
Kompiuterių katedra

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Informatikos inžinerijos studijų programa

Kursinis darbas

T120B05 Kompiuterinių sistemų inžinerija

Apšvietimo stebėjimo sistema

ATLIKO:

Karolis Mockaitis _____ **IFC-1**
(Vardas Pavardė) (Parašas) (Grupė)

Rokas Petkus _____ **IFC-1**
(Vardas Pavardė) (Parašas) (Grupė)

DĖSTYTOJAS:

jaun. asist. DOBROVOLSKIS Algirdas _____
(Vardas Pavardė) (Parašas)

DARBAS ATIDUOTAS:

05 d. gruodžio mėn. 2024

KAUNAS, 2024

Užduotis

Gauta užduotis yra sukurti apšvietimo stebėjimo sistemą.

Naudojamas apšvietimo sensorius ir trys lemputės: mėlyna, žalia ir raudona. Žalia dega kai aplinkos apšvietimas yra „normalus“, mėlyna – kai „tamsu“, o raudona – kai „akina“. Naujausi apšvietimo sensoriaus rodmenys matomi tiek Android programoje, tiek WEB puslapyje. Apšvietimo tamsos ir akinimo ribos nustatomos iš Android programos.

Sprendimo architektūra/aprašymas

Sistemos architektūra buvo sukurta remiantis kelių komponentų sąveika. Pagrindiniai komponentai:

1. **Arduino mikrovaldiklis:**
    - Atsakingas už apšvietimo sensoriaus duomenų nuskaitymą.
    - Valdo trijų spalvų lemputes (mėlyna, žalia, raudona), pagal nustatytas ribas:
        - „Tamsu“ – mėlyna lemputė.
        - „Normalus apšvietimas“ – žalia lemputė.
        - „Akinantis apšvietimas“ – raudona lemputė.
    - Leidžia dinamiškai atnaujinti akinimo ir tamsos reikšmes per serijinį ryšį.
2. **Node.js serveris:**
    - Veikia kaip tarpininkas tarp Arduino ir klientinių aplikacijų (WEB bei Android).
    - Užtikrina, kad sensoriaus duomenys būtų transliuojami realiu laiku WebSocket pagalba.
    - Priima API užklausas akinimo ir tamsos reikšmių atnaujinimui ir siunčia jas Arduino.
3. **WEB sąsaja:**
    - Leidžia realiu laiku stebėti apšvietimo sensoriaus rodmenis.
    - Sukurta naudojant HTML, CSS ir JavaScript.
    - Naudoja WebSocket, kad užtikrintų nuolatinį duomenų atnaujinimą.
4. **Android programa:**
    - Skirta apšvietimo lygio stebėjimui ir akinimo ir tamsos reikšmių valdymui.
    - Palaiko realaus laiko duomenų atnaujinimą ir užtikrina paprastą sąveiką su vartotoju.
    - Naudoja REST API ir WebSocket ryšius duomenų perdavimui.
5. **Ryšiai tarp komponentų:**
    - Arduino bendrauja su Node.js serveriu per serijinį ryšį.
    - Node.js serveris perduoda duomenis WEB puslapiui ir Android programai per WebSocket bei REST API.
    - Android programa ir WEB sąsaja leidžia atnaujinti slenksčių reikšmes, kurios siunčiamos Arduino.

Ši architektūra užtikrina lanksčią ir patikimą sąveiką tarp skirtingų sistemos komponentų, leidžia efektyviai apdoroti ir vizualizuoti sensoriaus duomenis bei suteikia galimybę vartotojui paprastai valdyti sistemą.

Sprendimo programinis kodas

Šiame skyriuje pateikiamas pilnas visos sistemos kodas.

Arduino kodas

```
// Pins
const int ldrPin = A0;
const int redPin = 4;
```

```
const int greenPin = 5;
const int bluePin = 6;

// Thresholds
int darkThreshold = 300;  // Default
int brightThreshold = 700;  // Default

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

  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);

  // Initialize LEDs
  digitalWrite(redPin, LOW);
  digitalWrite(greenPin, LOW);
  digitalWrite(bluePin, LOW);
}
void loop() {
  // Read LDR value
  int ldrValue = analogRead(ldrPin);

  // Transmit LDR value
  Serial.print("LDR: ");
  Serial.println(ldrValue);

  // Control LEDs
  if (ldrValue < darkThreshold) {
    digitalWrite(bluePin, HIGH);
    digitalWrite(greenPin, LOW);
    digitalWrite(redPin, LOW);
  } else if (ldrValue > brightThreshold) {
    digitalWrite(redPin, HIGH);
    digitalWrite(greenPin, LOW);
    digitalWrite(bluePin, LOW);
  } else {
    digitalWrite(greenPin, HIGH);
    digitalWrite(redPin, LOW);
    digitalWrite(bluePin, LOW);
  }
  // Check for serial input to adjust thresholds
  if (Serial.available()) {
    String input = Serial.readStringUntil('\n');
    if (input.startsWith("dark:")) {
      darkThreshold = input.substring(5).toInt();
      //Serial.println("Dark threshold updated: " + String(darkThreshold));
    } else if (input.startsWith("bright:")) {
      brightThreshold = input.substring(7).toInt();
                //Serial.println("Bright    threshold    updated:    "    +
String(brightThreshold));
    }
  }
  delay(2000);
}
```

Node.js kodas

```
const express = require('express');
```

```
const { SerialPort } = require('serialport');
const { ReadlineParser } = require('@serialport/parser-readline');
const cors = require('cors');
const { WebSocketServer } = require('ws');
const path = require('path');
const app = express();
app.use(cors());
app.use(express.json());

// Serve static files
app.use(express.static(path.join(__dirname, 'public')));

// Serial port setup
const port = new SerialPort({
  path: 'COM7',
  baudRate: 9600
});
const parser = port.pipe(new ReadlineParser({ delimiter: '\n' }));

// WebSocket server setup
const wss = new WebSocketServer({ noServer: true });

let ldrValue = 0;
let thresholds = { dark: 300, bright: 700 };

// Broadcast function for WebSocket
const broadcast = (message) => {
  wss.clients.forEach(client => {
    if (client.readyState === client.OPEN) {
      client.send(JSON.stringify(message));
    }
  });
};

// Read data from Arduino
parser.on('data', (data) => {
  const match = data.trim().match(/LDR:\s*(\d+)/);
  if (match) {
    ldrValue = parseInt(match[1], 10);
    //console.log(`New LDR value: ${ldrValue}`);
    // Broadcast the updated LDR value to all WebSocket clients
    broadcast({ ldrValue });
  }
});

// REST API Endpoints
app.get('/ldr', (req, res) => {
  res.json({ ldrValue, thresholds });
});

app.post('/thresholds', (req, res) => {
  thresholds.dark = req.body.dark;
  thresholds.bright = req.body.bright;
  port.write(`dark:${thresholds.dark}\n`);
  port.write(`bright:${thresholds.bright}\n`);
  res.json({ success: true });
});

// WebSocket upgrade
```

```
const server = app.listen(3000, () => {
  console.log('Server running on http://localhost:3000');
});
server.on('upgrade', (request, socket, head) => {
  wss.handleUpgrade(request, socket, head, (ws) => {
    wss.emit('connection', ws, request);
  });
});
```

GUI

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Apsvietimo stebejimas</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <h1>Apšvietimo stebėjimas</h1>
  <p id="ldrValue">Apšvietimo lygis: Laukiama duomenų...</p>
  <script src="script.js"></script>
</body>
</html>
```

GUI stilius

```
body {
  font-family: Arial, sans-serif;
  text-align: center;
  padding: 20px;
}

h1 {
  color: #333;
}
p {
  font-size: 1.5rem;
  color: #555;
}
```

JavaScript automatiniam reikšmės atnaujinimui

```
document.addEventListener("DOMContentLoaded", () => {
  const ldrDisplay = document.getElementById("ldrValue");

  // Establish WebSocket connection
  const socket = new WebSocket("ws://localhost:3000");

  socket.onopen = () => {
    console.log("WebSocket connected");
  };

  socket.onmessage = (event) => {
    const data = JSON.parse(event.data);
    if (data.ldrValue !== undefined) {
      ldrDisplay.textContent = `Apšvietimo lygis: ${data.ldrValue}`;
    }
  };

  socket.onclose = () => {
```

```
    console.log("WebSocket disconnected");
  };
});
```

Android kodas

```
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.widget.SeekBar;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;
import org.json.JSONObject;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.nio.charset.StandardCharsets;

public class MainActivity extends AppCompatActivity {
    private final String apiUrl = "http://10.0.2.2:3000/ldr"; // API GET URL
    private TextView lightValueText;
    private TextView blueLightIndicator;
    private TextView greenLightIndicator;
    private TextView redLightIndicator;
    private TextView darkThresholdValue;
    private SeekBar darkThresholdSeekBar;
    private TextView blindingThresholdValue;
    private SeekBar blindingThresholdSeekBar;
    private int currentDarkThreshold = 300; // Default dark threshold
    private int currentBlindingThreshold = 700; // Default blinding threshold
    private int ldrValue = 0; // Current light sensor value
    private final Handler handler = new Handler();
    private final int updateInterval = 1000; // Update every second

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Initialize UI components
        lightValueText = findViewById(R.id.lightValueText);
        blueLightIndicator = findViewById(R.id.blueLightIndicator);
        greenLightIndicator = findViewById(R.id.greenLightIndicator);
        redLightIndicator = findViewById(R.id.redLightIndicator);
        darkThresholdSeekBar = findViewById(R.id.darkThresholdSeekBar);
        darkThresholdValue = findViewById(R.id.darkThresholdValue);
        blindingThresholdSeekBar                                          =
findViewById(R.id.blindingThresholdSeekBar);
        blindingThresholdValue = findViewById(R.id.blindingThresholdValue);
        // Set SeekBar listeners
        darkThresholdSeekBar.setOnSeekBarChangeListener(new
SeekBar.OnSeekBarChangeListener() {
            @Override
            public void onProgressChanged(SeekBar seekBar, int progress,
boolean fromUser) {
                currentDarkThreshold = progress;
```

```
                darkThresholdValue.setText("Value: " + progress);
                // Send updated threshold values to the server
                sendThresholdsToArduino(currentDarkThreshold,
currentBlindingThreshold);
            }
            @Override
            public void onStartTrackingTouch(SeekBar seekBar) {}
            @Override
            public void onStopTrackingTouch(SeekBar seekBar) {}
        });
        blindingThresholdSeekBar.setOnSeekBarChangeListener(new
SeekBar.OnSeekBarChangeListener() {
            @Override
            public void onProgressChanged(SeekBar seekBar, int progress,
boolean fromUser) {
                currentBlindingThreshold = progress;
                blindingThresholdValue.setText("Value: " + progress);
                // Send updated threshold values to the server
                sendThresholdsToArduino(currentDarkThreshold,
currentBlindingThreshold);
            }
            @Override
            public void onStartTrackingTouch(SeekBar seekBar) {}
            @Override
            public void onStopTrackingTouch(SeekBar seekBar) {}
        });
        // Start fetching LDR data
        startFetchingLDRData();
    }

    private void startFetchingLDRData() {
        handler.post(fetchLDRDataRunnable);
    }

    private void stopFetchingLDRData() {
        handler.removeCallbacks(fetchLDRDataRunnable);
    }
    private final Runnable fetchLDRDataRunnable = new Runnable() {
        @Override
        public void run() {
            // Fetch LDR data
            new Thread(() -> {
                String result = fetchLDRData();
                if (result != null) {
                    try {
                        JSONObject jsonObject = new JSONObject(result);
                        ldrValue = jsonObject.getInt("ldrValue");

                        // Update UI based on LDR value
                        runOnUiThread(() -> updateIndicators());
                    } catch (Exception e) {
                        Log.e("MainActivity", "Error parsing LDR data", e);
                    }
                }
            }).start();

            // Schedule next fetch
            handler.postDelayed(this, updateInterval);
        }
```

```
    };
    private void updateIndicators() {
        // Update light value display
        lightValueText.setText("Current Light Value: " + ldrValue);

        // Update indicators based on thresholds
        if (ldrValue < currentDarkThreshold) {
            blueLightIndicator.setText("Blue: ON");
            greenLightIndicator.setText("Green: OFF");
            redLightIndicator.setText("Red: OFF");
        } else if (ldrValue >= currentDarkThreshold && ldrValue <=
currentBlindingThreshold) {
            blueLightIndicator.setText("Blue: OFF");
            greenLightIndicator.setText("Green: ON");
            redLightIndicator.setText("Red: OFF");
        } else {
            blueLightIndicator.setText("Blue: OFF");
            greenLightIndicator.setText("Green: OFF");
            redLightIndicator.setText("Red: ON");
        }
    }
    private String fetchLDRData() {
        try {
            URL url = new URL(apiUrl);
            HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
            connection.setRequestMethod("GET");
            int responseCode = connection.getResponseCode();
            if (responseCode == HttpURLConnection.HTTP_OK) {
                BufferedReader reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
                StringBuilder response = new StringBuilder();
                String line;
                while ((line = reader.readLine()) != null) {
                    response.append(line);
                }
                reader.close();
                return response.toString();
            }
        } catch (Exception e) {
            Log.e("MainActivity", "Error fetching LDR data", e);
        }
        return null;
    }
    private void sendThresholdsToArduino(int darkThreshold, int
brightThreshold) {
        new Thread(() -> {
            try {
                // URL for the Node.js server
                URL url = new URL("http://10.0.2.2:3000/thresholds"); // Use
the correct server address if different
                // Open a connection to the server
                HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
                connection.setRequestMethod("POST");
                connection.setRequestProperty("Content-Type",
"application/json");
                connection.setDoOutput(true);
                // Create JSON payload
```

```
                String jsonPayload = String.format("{\"dark\": %d,
\"bright\": %d}", darkThreshold, brightThreshold);
                // Send data to the server
                try (OutputStream os = connection.getOutputStream()) {
                    byte[] input =
jsonPayload.getBytes(StandardCharsets.UTF_8);
                    os.write(input, 0, input.length);
                }
                // Get the response code (to check if it was successful)
                int responseCode = connection.getResponseCode();
                if (responseCode == HttpURLConnection.HTTP_OK) {
                    Log.d("MainActivity", "Thresholds sent successfully.");
                } else {
                    Log.e("MainActivity", "Failed to send thresholds.
Response code: " + responseCode);
                }

            } catch (Exception e) {
                Log.e("MainActivity", "Error sending thresholds to Arduino",
e);
            }
        }).start();
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        stopFetchingLDRData();
    }
}
```

Rezultatų apibendrinimas

Šio kursinio darbo metu buvo sukurta apšvietimo stebėjimo sistema, kurią sudaro keli komponentai: Arduino mikrokontroleris su apšvietimo sensoriumi ir trijų spalvų lemputėmis, Node.js pagrįstas serveris, internetinė vartotojo sąsaja (WEB puslapis) ir Android programa.

Pagrindiniai darbo rezultatai:

- **Techniniai sprendimai:**
  - Naudojant Arduino mikrovaldiklį, buvo sukurta logika, leidžianti apdoroti apšvietimo sensoriaus duomenis ir valdyti mėlyną, žalią bei raudoną lemputes pagal nustatytus slenksčius.
  - Node.js serveris užtikrina ryšį tarp Arduino ir vartotojo sąsajų (WEB ir Android). Jis taip pat leidžia realiu laiku perduoti sensoriaus rodmenis bei atnaujinti slenksčių reikšmes.
  - WEB puslapis ir Android programa leidžia patogiai stebėti apšvietimo lygį bei keisti nustatymus.
- **Projekto architektūra:** Buvo naudojama klientas-serveris principu pagrįsta sistema, leidžianti užtikrinti sklandų duomenų perdavimą tarp skirtingų platformų.
- **Efektyvumas:** Sukurta sistema veikia realiu laiku, užtikrina patogų valdymą ir stebėjimą, o jos sprendimai yra lengvai pritaikomi įvairioms apšvietimo stebėjimo situacijoms.

Bendras įgyvendintas sprendimas yra efektyvus ir funkcionalus, leidžiantis vartotojams patogiai valdyti apšvietimo parametrus ir stebėti rodmenis realiu laiku.