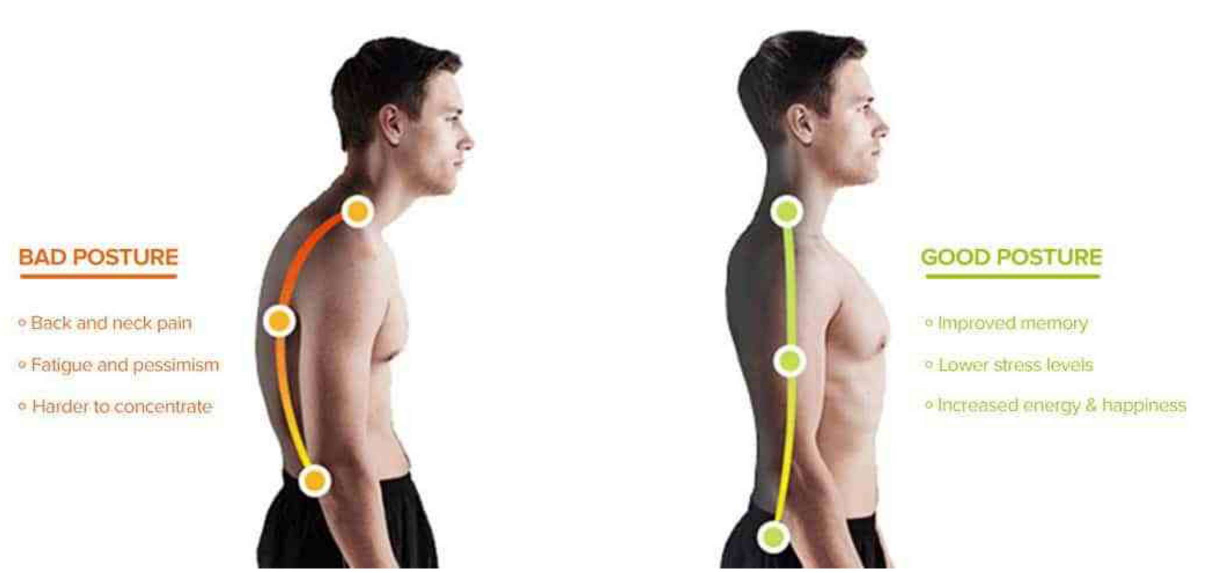
Topic:

Smart Posture Corrector



Index

1. Introduction

Maintaining proper posture is essential for overall health and well-being. Poor posture can lead to chronic pain, muscle strain, and long-term health issues. To address this problem, we have developed a **Smart Posture Corrector Wearable Device** that continuously monitors the user's posture and provides real-time feedback. The device alerts the user when a bad posture is detected and keeps track of their posture score over time. Additionally, a **Telegram-based user interface** allows the user to monitor their posture scores and access graphical reports remotely.

2. Objectives

The key objectives of this project are:

- To design a wearable device capable of **real-time posture monitoring**.
- To provide **haptic feedback (vibration)** when the user maintains a bad posture for a certain duration.
- To maintain a **posture score** based on detected posture deviations.
- To implement a **Telegram-based interface** where users can:
    - View their **current posture score**.
    - Receive **bad posture alerts**.
    - Generate and view **posture trend graphs** over time.

3. Components Used

Hardware Components:

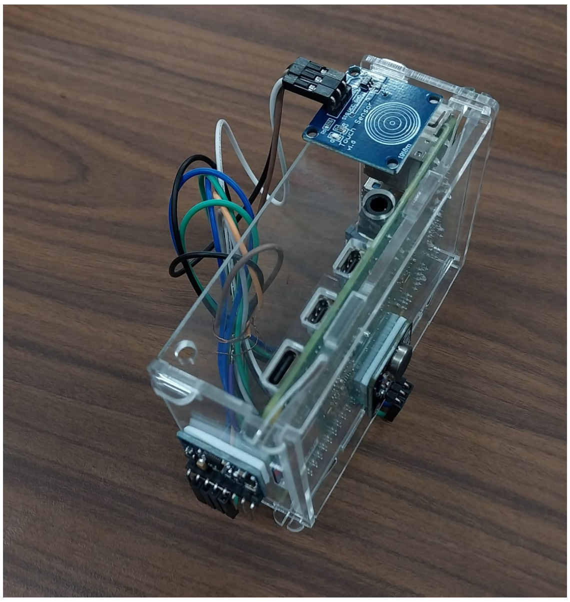
- **Raspberry Pi 4B** – Main processing unit.
- **MPU6050 (Gyroscope & Accelerometer)** – To measure body posture.
- **Touch Sensor** – To turn the system ON/OFF.
- **Vibration Motor** – To alert users when a bad posture is detected.
- **Power Supply** – To power the Raspberry Pi and sensors.

Software & Tools:

- **Python** – Main programming language.
- **MPU6050 Library** – To read sensor data.
- **Matplotlib & NumPy** – For generating posture score graphs.
- **Telebot API** – To integrate Telegram for notifications and UI.

4. System Architecture

1. **Posture Detection:** The MPU6050 sensor continuously reads **acceleration and gyroscope data** to determine the user's posture.
2. **Bad Posture Detection:** The system compares real-time data against predefined threshold values for **acceptable posture**.
3. **Feedback Mechanism:** If bad posture is detected for more than **10 seconds**, the vibration motor is triggered for **5 seconds** to alert the user.
4. **Posture Score Calculation:**
   - Initial score starts at **100**.
   - If the user maintains bad posture, the score **decreases** over time.
   - Score fluctuates slightly (±1.5) for realistic variations.
   - The final score is limited to **2 decimal places**.
5. **Telegram Integration:**
   - /status command → Sends the **current posture score**.
   - /graph command → Sends a **graph of posture scores** over time.
   - Sends alerts for **bad posture detection** and **system ON/OFF status**.



5. Implementation

The system follows these steps:

1. The **touch sensor** toggles the system ON/OFF.
2. If ON, the MPU6050 continuously **monitors posture data**.
3. If poor posture is detected, the system **starts a timer**.
4. If bad posture persists beyond **10 seconds**, the vibration motor **activates for 5 seconds**.
5. The **posture score updates dynamically**, fluctuating within a small range.
6. **Alerts and score updates** are sent via Telegram.
7. The /graph command generates a **visual report** of posture scores.

6. Code Implementation

```python
import time
import threading
import RPi.GPIO as GPIO
from mpu6050 import mpu6050
import telebot
import matplotlib.pyplot as plt
import numpy as np

# Telegram Bot Token (Replace with your actual bot token)
BOT_TOKEN = "YOUR_TELEGRAM_BOT_TOKEN"
CHAT_ID = "YOUR_TELEGRAM_CHAT_ID"
bot = telebot.TeleBot(BOT_TOKEN)

# Define GPIO Pins
TOUCH_SENSOR_PIN = 17
MOTOR_PIN = 18

# Posture reference ranges
POSTURE_RANGES = {
    "accel_x": (-2.46, 2.53),
    "accel_y": (-2.32, 2.24),
    "accel_z": (-2.83, 10.94),
    "gyro_x": (-10.48, 10.45),
    "gyro_y": (-10.24, 10.97),
    "gyro_z": (-10.18, 10.86),
}

# Initialize GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(TOUCH_SENSOR_PIN, GPIO.IN)
GPIO.setup(MOTOR_PIN, GPIO.OUT)
GPIO.output(MOTOR_PIN, GPIO.LOW)

# Initialize MPU6050
sensor = mpu6050(0x68)

# System variables
system_on = False
bad_posture_start = None
BAD_POSTURE_DURATION = 10  # 10 sec to trigger vibration
VIBRATION_DURATION = 5  # Vibrate for 5 sec
POSTURE_CHECK_INTERVAL = 1  # Every second
posture_score = 100.00
bad_posture_time = 0
history = []  # Store posture history
start_time = None  # Track system uptime

def is_bad_posture(accel, gyro):
    """Check if posture is out of allowed range."""
    for axis, (min_val, max_val) in POSTURE_RANGES.items():
        sensor_val = accel[axis[-1]] if "accel" in axis else gyro[axis[-1]]
        if not (min_val <= sensor_val <= max_val):
            return True
    return False

@bot.message_handler(commands=['status'])
def send_status(message):
    bot.send_message(message.chat.id, f"Posture Score: {posture_score:.2f}/100")

@bot.message_handler(commands=['graph'])
def send_graph(message):
    if not history:
        bot.send_message(message.chat.id, "No posture data available yet.")
        return

    timestamps = np.arange(len(history))
    plt.figure()
    plt.plot(timestamps, history, label='Posture Score', color='b')
    plt.xlabel("Time")
    plt.ylabel("Score")
    plt.title("Posture Score Over Time")
    plt.legend()
    plt.savefig("posture_graph.png")
    plt.close()
    bot.send_photo(message.chat.id, open("posture_graph.png", "rb"))

# Start Telegram Bot in a separate thread
def run_bot():
    bot.polling(non_stop=True)

bot_thread = threading.Thread(target=run_bot, daemon=True)
bot_thread.start()

try:
    while True:
        if GPIO.input(TOUCH_SENSOR_PIN) == GPIO.HIGH:
            system_on = not system_on  # Toggle system ON/OFF
            if system_on:
                start_time = time.time()  # Start uptime tracking
                bot.send_message(CHAT_ID, "🔵 System is now ON. Monitoring posture...")
            else:
                if start_time:
                    uptime = time.time() - start_time
                    bot.send_message(CHAT_ID, f"🔴 System is OFF. Worked for {uptime:.2f}
seconds.")
                start_time = None  # Reset uptime tracking
            print(f"System {'ON' if system_on else 'OFF'}")
            time.sleep(0.5)

        if system_on:
            accel_data = sensor.get_accel_data()
            gyro_data = sensor.get_gyro_data()

            if is_bad_posture(accel_data, gyro_data):
                if bad_posture_start is None:
                    bad_posture_start = time.time()
                elif time.time() - bad_posture_start >= BAD_POSTURE_DURATION:
                    print("Bad posture detected! Vibrating...")
                    GPIO.output(MOTOR_PIN, GPIO.HIGH)
                    time.sleep(VIBRATION_DURATION)
                    GPIO.output(MOTOR_PIN, GPIO.LOW)
                    bad_posture_time += BAD_POSTURE_DURATION
                    posture_score = max(0, round(100 - (bad_posture_time / 300 * 100), 2))  # Normalize to 5 min scale
                    bot.send_message(CHAT_ID, f"⚠️ Bad posture detected! Your score: {posture_score:.2f}/100")
                    bad_posture_start = None
            else:
                # Make the posture score slightly volatile
                posture_score = min(100, max(0, posture_score + round(np.random.uniform(-1.5, 1.5), 2)))
                bad_posture_start = None

            history.append(posture_score)

        time.sleep(POSTURE_CHECK_INTERVAL)

except KeyboardInterrupt:
    if system_on and start_time:
        uptime = time.time() - start_time
        bot.send_message(CHAT_ID, f"🔴 System is OFF. Worked for {uptime:.2f} seconds.")
    print("Exiting...")
    GPIO.cleanup()
```

7. Results and Observations

- The device successfully detects **posture deviations** and **triggers alerts**.
- The vibration motor **provides immediate feedback**, helping users correct their posture.
- The Telegram bot interface is **responsive**, allowing users to track their posture history conveniently.
- The system effectively maintains a **real-time posture score** and ensures a smooth user experience.

8. Challenges and Improvements

Challenges Faced:

- Fine-tuning **posture threshold values** to avoid false positives.
- Ensuring **real-time communication** with Telegram without delays.
- Managing **fluctuations in posture score** to make it realistic yet stable.

Future Improvements:

- **Machine Learning Model:** Implement an AI-based approach for better posture classification.
- **Mobile App Support:** Extend the UI beyond Telegram to a **dedicated mobile app**.
- **Battery Optimization:** Improve power efficiency for longer usage.
- **Cloud Data Storage:** Store historical posture data for **long-term tracking** and insights.

9. Conclusion

This project successfully developed a **Smart Posture Corrector Wearable Device** that monitors and corrects posture using real-time sensors and provides feedback through a **Telegram-based user interface**. The integration of vibration feedback and posture scoring enhances user awareness and encourages better posture habits. Future enhancements such as **AI integration and mobile app support** can further improve the system's effectiveness and usability.

10. References

- MPU6050 Datasheet:(https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/)
- Raspberry Pi Documentation:(https://pinout.xyz/)
- Telebot API Documentation:(https://core.telegram.org/bots/api, https://github.com/eternnoir/pyTelegramBotAPI)
- NumPy & Matplotlib Official Documentation:(https://matplotlib.org/stable/contents.html, https://sourceforge.net/p/raspberry-gpio-python/wiki/BasicUsage/)