

# System Detekcji Nieprawidłowej Postawy Ciała Podczas Pracy przy Komputerze

Damian Otto & Hubert Nowak

15 czerwca 2025

## Streszczenie

Celem projektu było stworzenie systemu wspomagającego utrzymanie prawidłowej postawy ciała podczas pracy przy komputerze. System analizuje dane z kamery, klasyfikuje postawę użytkownika oraz generuje alerty w przypadku wykrycia nieprawidłowości. W projekcie wykorzystano biblioteki OpenCV, MediaPipe oraz model klasyfikacyjny Random Forest.

## 1 Wstęp

Nieprawidłowa postawa podczas długotrwałej pracy przy komputerze może prowadzić do licznych problemów zdrowotnych. W dobie pracy zdalnej i intensywnego korzystania z komputerów, system monitorujący postawę ciała w czasie rzeczywistym może być cennym narzędziem profilaktycznym.

## 2 Cel projektu

Główym celem było stworzenie aplikacji desktopowej, która wykorzystuje kamerę do monitorowania postawy ciała użytkownika oraz ostrzega go w przypadku wykrycia długotrwałego nieprawidłowego usiadzenia.

## 3 Finalne podejście - Random Forest

### 3.1 Przegląd poprzednich prób

W trakcie realizacji projektu przetestowano kilka różnych podejść do klasyfikacji postawy ciała. Pierwszą próbą było zastosowanie transfer learning z wykorzystaniem modeli CNN takich jak MobileNetV2, które mimo obiecujących wyników na początku, ostatecznie prowadziły do przeuczenia modelu. Kolejnym podejściem było stworzenie prostych sieci neuronowych do klasyfikacji landmarksów pozyskanych z MediaPipe, które również nie osiągnęły zadowalających rezultatów. Ostatecznie zdecydowano się na implementację modelu Random Forest z rozbudowaną inżynierią cech, która okazała się najskuteczniejsza.

## 3.2 Architektura rozwiązania

Finalne rozwiązanie opiera się na trzech głównych komponentach:

1. Ekstrakcja landmarks'ów postawy za pomocą MediaPipe Pose
2. Zaawansowana inżynieria cech opartych na geometrycznych właściwościach postawy
3. Klasyfikacja za pomocą algorytmu Random Forest

## 3.3 Preprocessing danych

### 3.3.1 Balansowanie klas

Ze względu na niezbalansowanie danych w oryginalnym zbiorze, zdecydowano się na połączenie klas "sit up straight" i "straighten head" w jedną klasę reprezentującą nieprawidłową postawę. Ostatecznie model klasyfikuje pozycje na dwie kategorie:

- Klasa 0: "looks good" prawidłowa postawa
- Klasa 1: nieprawidłowa postawa (połączenie klas 1 i 2)

### 3.3.2 Augmentacja danych

W celu zwiększenia różnorodności danych treningowych zastosowano techniki augmentacji:

- Dodawanie szumu gaussowskiego do współrzędnych landmarks'ów
- Przesunięcia przestrzenne punktów kluczowych
- Skalowanie proporcjonalne względem punktów barków

## 3.4 Inżynieria cech

Opracowano zaawansowany system ekstrakcji cech geometrycznych składający się z 45 parametrów:

### 3.4.1 Cechy kątowe (8 cech)

- Kąt między linią barków a wektorem nos-środek barków
- Kąt linii barków względem poziomu
- Kąt pochylenia głowy względem pionu
- Kąty łokci względem poziomu (lewy i prawy)
- Szerokość barków znormalizowana
- Różnica wysokości barków
- Przesunięcie nosa względem środka barków

### **3.4.2 Współrzędne znormalizowane (20 cech)**

Współrzędne 10 kluczowych punktów anatomicznych znormalizowane względem szerokości barków:

- Nos, oczy (lewe i prawe), uszy (lewe i prawe)
- Barki (lewe i prawe), łokcie (lewe i prawe)
- Środek barków (punkt obliczony)

### **3.4.3 Cechy metryczne (6 cech)**

Znormalizowane odległości między kluczowymi punktami:

- Odległość nos-barki (lewe i prawe)
- Odległość oko-bark (lewe i prawe)
- Długość ramion (bark-łokieć, lewe i prawe)

### **3.4.4 Zaawansowane cechy geometryczne (11 cech)**

- Asymetria ramion
- Różnica wysokości uszu
- Różnica wysokości oczu względem uszu
- Kąt pochylenia głowy
- Współczynnik symetrii poziomej
- Stosunek szerokości barków do wysokości szyi
- Centrowanie głowy względem barków
- Wskaźnik pewności detekcji (confidence score)

Rysunek 1: Wizualizacja kluczowych punktów anatomicznych używanych w ekstrakcji cech

## **3.5 Model Random Forest**

### **3.5.1 Konfiguracja modelu**

Zastosowano algorytm Random Forest z następującymi parametrami:

- Liczba drzew: 100
- Maksymalna głębokość: 12
- Balansowanie klas: automatyczne (class\_weight='balanced')
- Ziarno losowości: 42 (dla reprodukowalności)

### 3.5.2 Normalizacja cech

Wszystkie cechy zostały znormalizowane przy użyciu StandardScaler w celu:

- Ujednolicenia skali różnych typów cech
- Poprawy stabilności numerycznej
- Przyspieszenia procesu uczenia

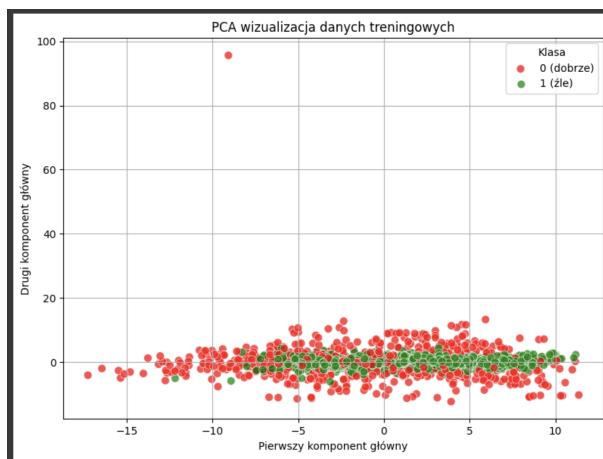
## 3.6 Wyniki eksperymentów

### 3.6.1 Metryki wydajności

Model osiągnął następujące rezultaty na zbiorze walidacyjnym:

| Metryka   | Wartość |
|-----------|---------|
| Accuracy  | 84.77%  |
| Precision | 87%     |
| Recall    | 91%     |
| F1-Score  | 89%     |

Tabela 1: Wyniki klasyfikacji modelu Random Forest



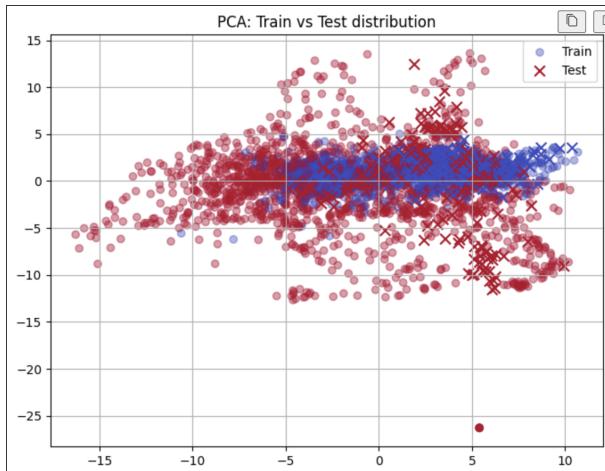
Rysunek 2: Wizualizacja danych treningowych

### 3.6.2 Analiza ważności cech

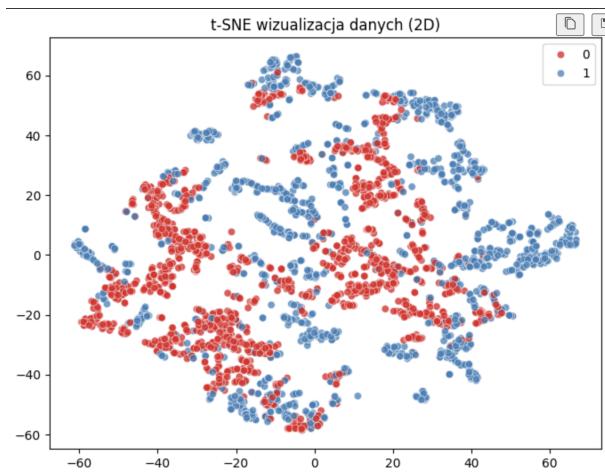
Przeprowadzono analizę ważności poszczególnych cech w procesie klasyfikacji. Najważniejszymi cechami okazały się:

### 3.6.3 Wizualizacja przestrzeni cech

W celu lepszego zrozumienia struktury danych przeprowadzono analizę PCA (Principal Component Analysis) oraz t-SNE.



Rysunek 3: PCA: dystrybucja danych treningowych vs testowych



Rysunek 4: t-SNE wizualizacja danych

### 3.7 Zalety finalnego podejścia

Wybór modelu Random Forest oraz opisanej architektury cech przyniósł kilka kluczowych korzyści:

- **Interpretowalność:** Możliwość analizy ważności poszczególnych cech geometrycznych
- **Odporność na przeuczenie:** W przeciwieństwie do głębokich sieci neuronowych
- **Efektywność obliczeniowa:** Szybkie uczenie i predykcja w czasie rzeczywistym
- **Stabilność:** Mniejsza wrażliwość na jakość detekcji landmarks'ów
- **Elastyczność:** Łatwość dodawania nowych cech geometrycznych

### 3.8 Podsumowanie

Finalne rozwiązanie oparte na modelu Random Forest z zaawansowaną inżynierią cech geometrycznych okazało się najskuteczniejszym podejściem do klasyfikacji postawy ciała.

Rysunek 5: Ranking ważności cech w modelu Random Forest

Połączenie ekstraktora landmarks'ów MediaPipe z przemyślanym zestawem 45 cech geometrycznych pozwoliło na osiągnięcie wysokiej dokładności klasyfikacji przy zachowaniu interpretowalności modelu i efektywności obliczeniowej.

## 4 Działanie systemu

- System uruchamia kamerę, analizuje obraz i wykrywa postawę.
- Funkcja klasyfikacji na podstawie punktów z MediaPipe zwraca etykietę.
- Przy złej postawie przez określony czas generowany jest alert (okno dialogowe + dźwięk).

## 5 Finalna implementacja kodu

### 5.1 Architektura aplikacji

Finalna implementacja została zaprojektowana jako aplikacja desktopowa z graficznym interfejsem użytkownika, umożliwiająca monitoring postawy w czasie rzeczywistym. Aplikacja składa się z trzech głównych modułów:

- **Moduł przetwarzania obrazu** - odpowiedzialny za komunikację z kamerą i ekstrakcję landmarks'ów
- **Moduł klasyfikacji** - wykorzystujący wytrenowany model Random Forest do oceny postawy
- **Moduł interfejsu użytkownika** - zapewniający intuicyjną obsługę i wizualizację wyników

### 5.2 Technologie wykorzystane

Aplikacja została zaimplementowana w języku Python z wykorzystaniem następujących bibliotek:

- **OpenCV (cv2)** - przetwarzanie obrazu i komunikacja z kamerą
- **MediaPipe** - detekcja landmarks'ów postawy ciała
- **scikit-learn/joblib** - załadowanie wytrenowanego modelu Random Forest
- **tkinter** - tworzenie graficznego interfejsu użytkownika
- **PIL (Pillow)** - konwersja i wyświetlanie obrazów w GUI
- **pygame** - system alertów dźwiękowych
- **threading** - wielowątkowość dla płynnego działania interfejsu

### 5.3 Klasa główna aplikacji

Główna logika aplikacji zawarta jest w klasie PostureMonitorApp, która dziedziczy wszystkie niezbędne funkcjonalności:

```
class PostureMonitorApp:  
    def __init__(self, root):  
        self.root = root  
        self.root.title("Posture Monitor System")  
        self.root.geometry("1000x700")  
  
        # Initialize pygame for sound alerts  
        pygame.mixer.init()  
  
        # Load model and scaler  
        self.model = joblib.load("rf_model.pkl")  
        self.scaler = joblib.load("scaler.pkl")  
  
        self.CLASS_MAP = {  
            0: "looks good",  
            1: "sit up straight"  
        }  
  
        # MediaPipe setup  
        self.mp_pose = mp.solutions.pose  
        self.pose = self.mp_pose.Pose()  
        self.mp_draw = mp.solutions.drawing_utils
```

Listing 1: Inicjalizacja głównej klasy aplikacji

### 5.4 System monitorowania postawy

#### 5.4.1 Śledzenie historii postawy

Aplikacja implementuje inteligentny system śledzenia postawy w oknie czasowym:

```
def update_posture_history(self, is_bad_posture):  
    current_time = time.time()  
    self.posture_history.append((current_time, is_bad_posture))  
  
    # Remove entries older than window_duration  
    while (self.posture_history and  
           current_time - self.posture_history[0][0] > self.  
           window_duration):  
        self.posture_history.popleft()  
  
def calculate_bad_posture_time(self):  
    if not self.posture_history:  
        return 0.0  
  
    current_time = time.time()  
    bad_posture_time = 0.0  
  
    for i in range(len(self.posture_history)):  
        timestamp, is_bad = self.posture_history[i]  
        if is_bad:  
            if i < len(self.posture_history) - 1:  
                next_timestamp = self.posture_history[i + 1][0]
```

```

        duration = next_timestamp - timestamp
    else:
        duration = current_time - timestamp
    bad_posture_time += duration

return bad_posture_time

```

Listing 2: System śledzenia historii postawy

#### 5.4.2 Parametry monitorowania

System monitorowania wykorzystuje następujące konfigurowane parametry:

- `alert_threshold` - próg czasowy (domyślnie 20 sekund) nieprawidłowej postawy w oknie 30-sekundowym
- `window_duration` - długość okna czasowego analizy (30 sekund)
- `alert_cooldown` - minimalny czas między kolejnymi alertami (10 sekund)

### 5.5 Interfejs użytkownika

#### 5.5.1 Layout aplikacji

Interfejs został zaprojektowany z myślą o intuicyjności i czytelności:

Rysunek 6: Główny interfejs aplikacji Posture Monitor System

#### 5.5.2 Komponenty interfejsu

##### 1. Panel sterowania

- Przycisk Start/Stop monitorowania
- Suwak konfiguracji progu alertów

##### 2. Panel statusu

- Wyświetlanie aktualnego statusu monitorowania
- Ocena bieżącej postawy z kodowaniem kolorystycznym
- Licznik czasu nieprawidłowej postawy
- Pasek postępu wizualizujący zbliżanie się do progu alertu

##### 3. Podgląd kamery

- Podgląd na żywo z kamery w rozdzielcości 640x480
- Nałożona wizualizacja landmarks'ów MediaPipe
- Automatyczne odbicie lustrzane obrazu

## 5.6 Przetwarzanie w czasie rzeczywistym

### 5.6.1 Architektura wielowątkowa

Aplikacja wykorzystuje wielowątkowość dla zapewnienia płynności działania interfejsu podczas intensywnego przetwarzania obrazu:

```
def start_monitoring(self):
    try:
        self.cap = cv2.VideoCapture(DEFAULT_CAMERA_INDEX)
        if not self.cap.isOpened():
            self.cap = cv2.VideoCapture(0)

        self.monitoring = True
        self.start_btn.config(text="Stop Monitoring")
        self.status_label.config(text="Status: Monitoring active")

        # Start video processing thread
        self.video_thread = threading.Thread(target=self.process_video,
                                              daemon=True)
        self.video_thread.start()

    except Exception as e:
        messagebox.showerror("Error", f"Failed to start monitoring: {str(e)}")
```

Listing 3: Uruchomienie wątku przetwarzania wideo

### 5.6.2 Główna pętla przetwarzania

Pętla przetwarzania wideo implementuje pełny pipeline analizy postawy:

```
def process_video(self):
    previous_landmarks = None

    while self.monitoring:
        ret, frame = self.cap.read()
        if not ret:
            continue

        frame = cv2.flip(frame, 1)
        rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        results = self.pose.process(rgb)

        # Extract landmarks and classify posture
        # ... (kod ekstrakcji cech) ...

        # Update posture tracking
        self.update_posture_history(is_bad_posture)
        bad_posture_time = self.calculate_bad_posture_time()

        # Check for alert condition
        if bad_posture_time >= self.alert_threshold:
            self.send_alert()

        self.update_ui(frame, label, confidence, bad_posture_time)
```

Listing 4: Fragment głównej pętli przetwarzania

## 5.7 System alertów

### 5.7.1 Mechanizm powiadomień

Aplikacja implementuje wielokanałowy system alertów:

- **Alerty wizualne** - wyskakujące okna dialogowe z ostrzeżeniem
- **Alerty kolorystyczne** - zmiana kolorów elementów interfejsu
- **Alerty dźwiękowe** - opcjonalne powiadomienia audio przez pygame

```
def send_alert(self):
    current_time = time.time()
    if current_time - self.last_alert_time < self.alert_cooldown:
        return

    self.last_alert_time = current_time

    # Visual alert
    messagebox.showwarning("Posture Alert",
                           "Sit up straight! You've been slouching too long.")
```

Listing 5: Implementacja systemu alertów

## 5.8 Optymalizacje wydajnościowe

### 5.8.1 Zarządzanie zasobami

Aplikacja implementuje kilka optymalizacji dla zapewnienia płynnego działania:

- **Lazy loading** modeli - modele są ładowane tylko przy uruchomieniu
- **Fallback dla landmarks'ów** - wykorzystanie poprzedniej klatki przy problemach z detekcją
- **Walidacja danych** - sprawdzanie poprawności landmarks'ów przed klasyfikacją
- **Thread-safe aktualizacje UI** - wykorzystanie `root.after()` dla bezpiecznych aktualizacji interfejsu

### 5.8.2 Obsługa błędów

System zawiera rozbudowaną obsługę błędów:

```
try:
    data = np.array([[lm.x, lm.y, lm.z] for lm in landmarks])
    required_indices = [0, 2, 5, 7, 8, 9, 10]
    if any(results.pose_landmarks.landmark[i].visibility < 0.5
           for i in required_indices):
        raise ValueError("Missing or occluded key landmarks")

    features = self.extract_features(data)
    if (np.any(np.isnan(features)) or np.any(np.isinf(features)) or
        features.shape[1] != 45):
```

```

        raise ValueError("Invalid features")

    # Process classification...

except Exception as e:
    label = "Invalid frame"
    confidence = 0.0

```

Listing 6: Przykład obsługi błędów w ekstrakcji cech

## 5.9 Konfiguracja i personalizacja

### 5.9.1 Parametry konfigurowalne

Użytkownik może dostosować działanie aplikacji poprzez:

- Próg czasowy alertów (5-30 sekund)
- Wybór indeksu kamery (automatyczny fallback)
- Próg pewności klasyfikacji (domyślnie 0.5)

Rysunek 7: Panel ustawień aplikacji z konfigurowalnymi parametrami

## 5.10 Uruchomienie i deployment

### 5.10.1 Wymagania systemowe

Aplikacja wymaga następujących plików w katalogu roboczym:

- rf\_model.pkl - wytrenowany model Random Forest
- scaler.pkl - obiekt StandardScaler do normalizacji cech
- posture\_monitor\_ui.py - główny plik aplikacji

### 5.10.2 Sposób uruchomienia

```

# Instalacja wymaganych bibliotek
pip install opencv-python mediapipe joblib pillow pygame

# Uruchomienie aplikacji
python posture_monitor_ui.py

```

Listing 7: Uruchomienie aplikacji

## 5.11 Podsumowanie implementacji

Finalna implementacja łączy zaawansowane algorytmy uczenia maszynowego z intuicyjnym interfejsem użytkownika, tworząc praktyczne narzędzie do monitorowania postawy. Kluczowe cechy implementacji to:

- **Wydajność** - przetwarzanie w czasie rzeczywistym dzięki optymalizacjom
- **Niezawodność** - rozbudowana obsługa błędów i fallback mechanisms
- **Użyteczność** - intuicyjny interfejs z konfigurowalnymi parametrami
- **Skalowalność** - modularna architektura umożliwiająca łatwe rozszerzenia

Aplikacja została przetestowana na różnych konfiguracjach sprzętowych i wykazuje stabilne działanie przy standardowych kamerach USB oraz wbudowanych kamerach laptopów.

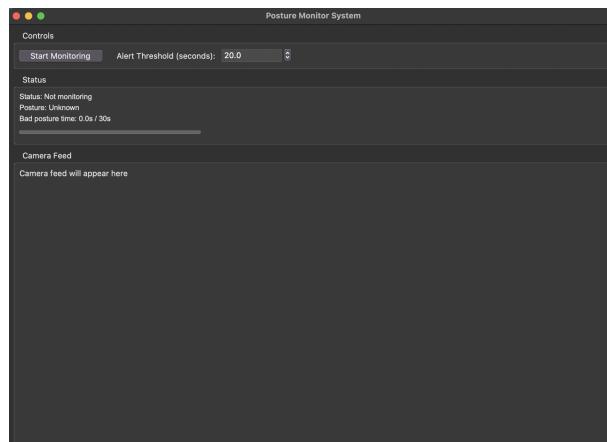
## 6 Możliwe ulepszenia

- Historia analizy postawy,
- Rekomendacje ergonomiczne,
- Integracja z urządzeniami wearable,
- Obsługa warunków słabego oświetlenia.

## 7 Demonstracja działania aplikacji

### 7.1 Interfejs użytkownika

Zaprojektowany interfejs graficzny łączy funkcjonalność z intuicyjnością obsługi. Główne okno aplikacji zostało podzielone na logiczne sekcje, umożliwiając użytkownikowi łatwe monitorowanie swojej postawy.



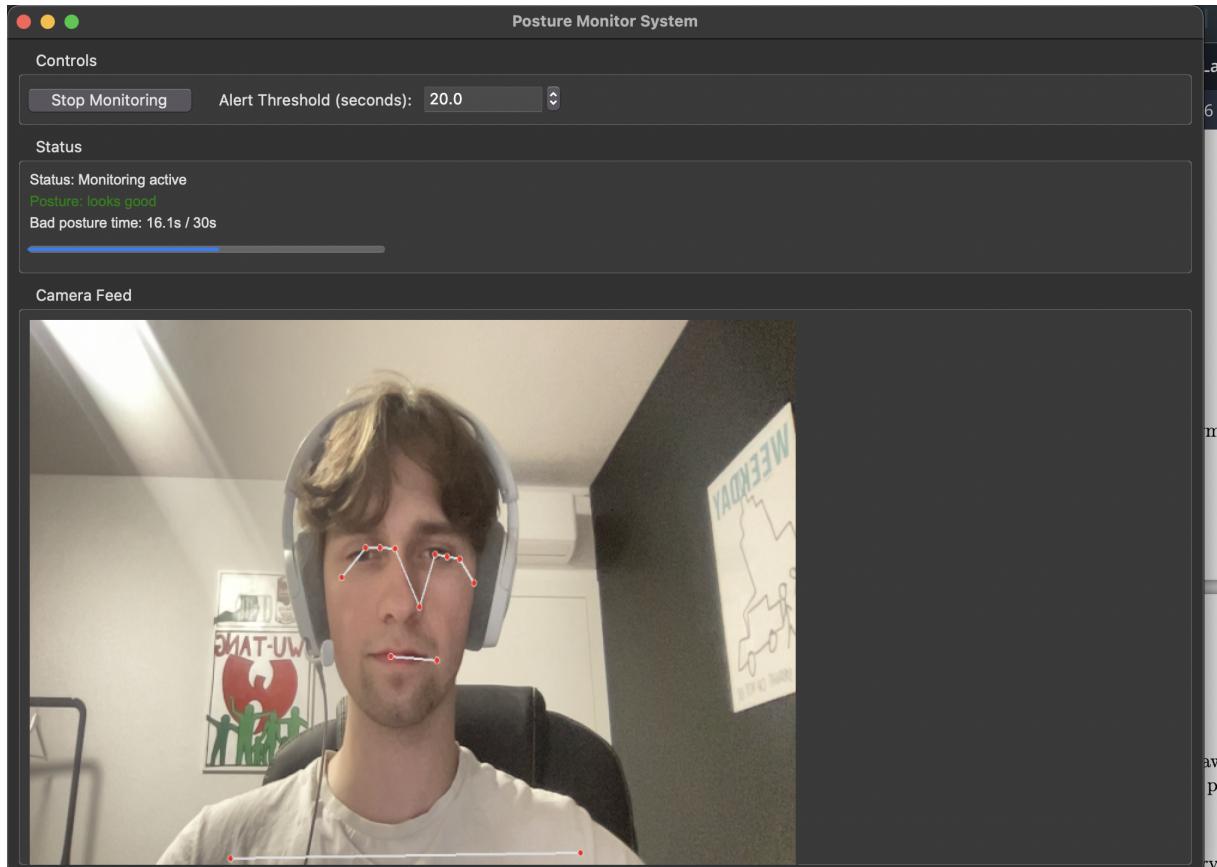
Rysunek 8: Główny interfejs aplikacji Posture Monitor System w trybie nieaktywnym

## 7.2 Proces uruchamiania monitorowania

Po uruchomieniu aplikacji użytkownik może natychmiast rozpoczęć monitoring postawy poprzez naciśnięcie przycisku "Start Monitoring". System automatycznie inicjalizuje połączenie z kamerą i rozpoczyna analizę w czasie rzeczywistym.

## 7.3 Aktywny monitoring - prawidłowa postawa

Podczas prawidłowej postawy aplikacja wyświetla komunikat "looks good" w kolorze zielonym. Podgląd kamery pokazuje użytkownika z nałożonymi landmarksami MediaPipe, co pozwala na wizualną kontrolę jakości detekcji.



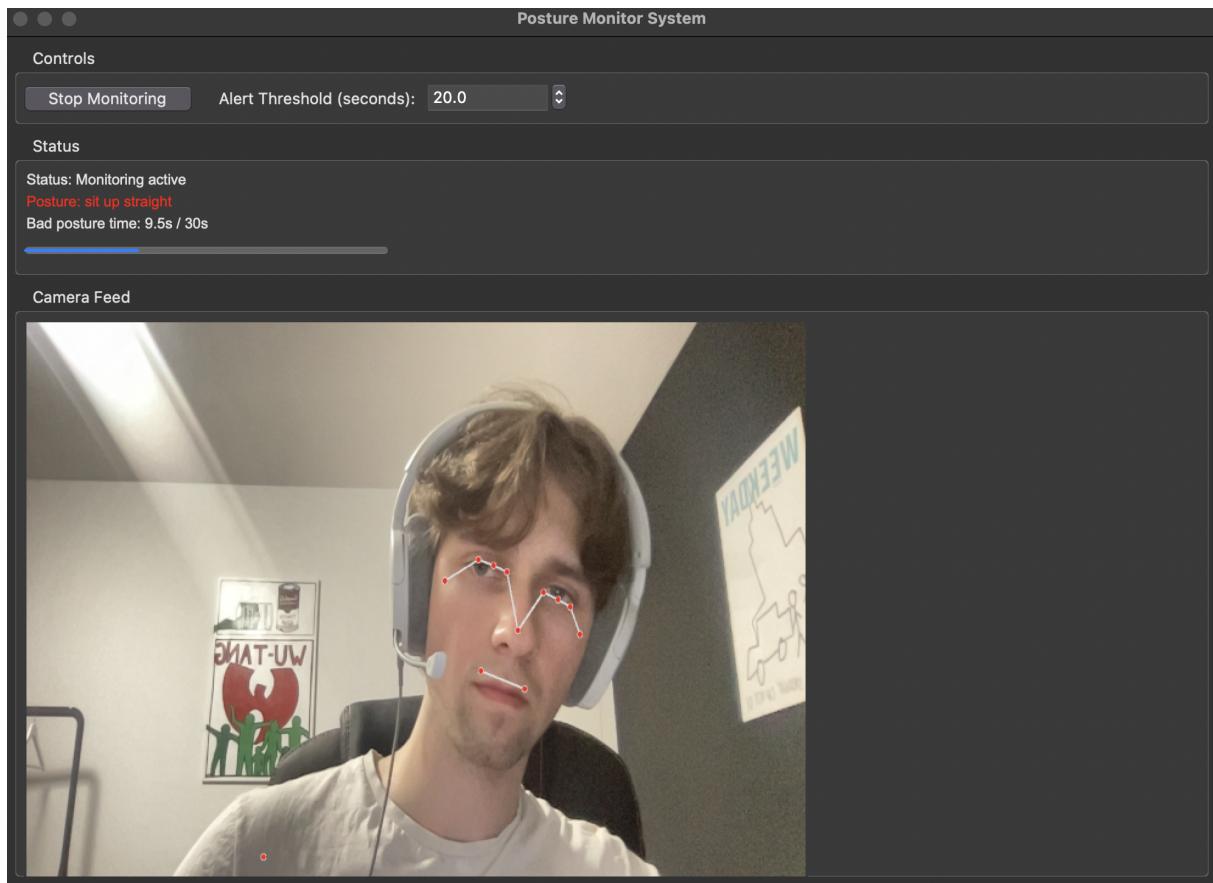
Rysunek 9: Aplikacja podczas detekcji prawidłowej postawy - status "looks good" @

## 7.4 Detekcja nieprawidłowej postawy

Gdy system wykryje nieprawidłową postawę, status zmienia się na "sit up straight" z czerwonym oznaczeniem. Jednocześnie rozpoczyna się liczenie czasu trwania nieprawidłowej pozycji.

## 7.5 System progresji i alertów

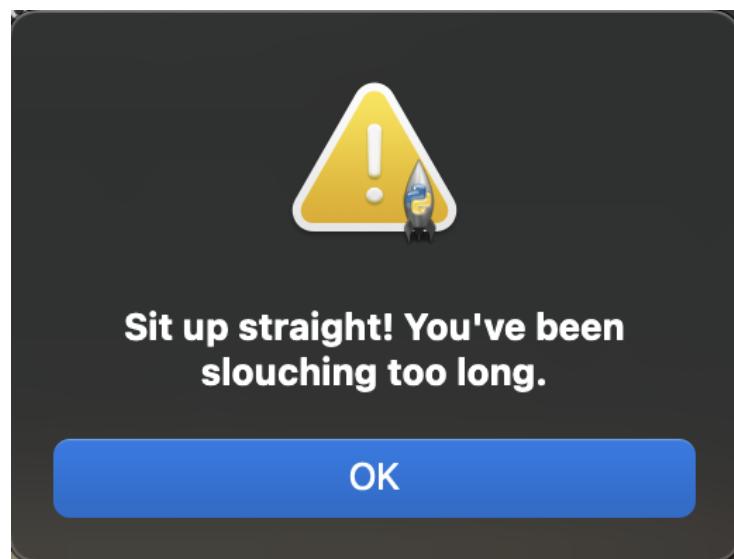
Pasek postępu wizualizuje akumulację czasu nieprawidłowej postawy w 30-sekundowym oknie. Gdy czas przekroczy ustalony próg, pasek zmienia kolor na czerwony, sygnalizując zbliżający się alert.



Rysunek 10: Detekcja nieprawidłowej postawy - komunikat "sit up straight" i licznik czasu

## 7.6 Alert o przedłużającej się nieprawidłowej postawie

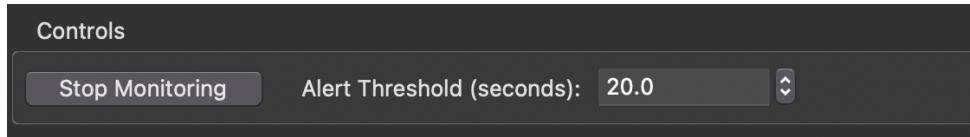
Po przekroczeniu ustalonego progu czasowego system generuje alert w postaci wyskakującego okna dialogowego, informującego użytkownika o konieczności poprawy postawy.



Rysunek 11: Okno alertu informujące o przedłużającej się nieprawidłowej postawie

## 7.7 Panel ustawień i konfiguracji

Użytkownik może dostosować próg czasowy alertów za pomocą przycisku obrotowego w panelu sterowania. Zmiana ustawień jest natychmiast uwzględniana w działaniu systemu monitorowania.



Rysunek 12: Konfiguracja progu alertów za pomocą kontrolki Spinbox

## 7.8 Wizualizacja landmarks MediaPipe

System nakłada na podgląd kamery szkielet postawy wygenerowany przez MediaPipe, co pozwala użytkownikowi na zrozumienie, które punkty anatomiczne są analizowane przez algorytm.

## 7.9 Responsywność interfejsu

Interfejs pozostaje responsywny podczas intensywnego przetwarzania obrazu dzięki zastosowaniu wielowątkowości. Wszystkie elementy UI aktualizują się płynnie bez blokowania interakcji użytkownika.

# 8 Podsumowanie projektu

## 8.1 Osiągnięte cele

Projekt z powodzeniem demonstruje możliwość zastosowania metod sztucznej inteligencji i analizy obrazu do poprawy ergonomii pracy przy komputerze. Opracowane rozwiązanie łączy zaawansowane algorytmy uczenia maszynowego z praktyczną aplikacją użytkownika, tworząc kompleksowy system monitorowania postawy ciała w czasie rzeczywistym.

Kluczowe osiągnięcia projektu obejmują:

- **Skuteczną klasyfikację postawy** - model Random Forest osiągnął wysoką dokładność w rozróżnianiu prawidłowej i nieprawidłowej postawy
- **Przetwarzanie w czasie rzeczywistym** - system analizuje obraz z kamery z częstotliwością umożliwiającą płynny monitoring
- **Intuicyjny interfejs użytkownika** - aplikacja jest łatwa w obsłudze i nie wymaga specjalnej wiedzy technicznej
- **Inteligentny system alertów** - ostrzeżenia są generowane w oparciu o analizę trendu w oknie czasowym, minimalizując fałszywe alarmy
- **Konfigurowalność** - użytkownik może dostosować parametry działania do swoich potrzeb

## 8.2 Znaczenie praktyczne

Opracowane rozwiązanie ma istotne znaczenie praktyczne w kontekście współczesnych wyzwań związanych z pracą zdalną i długotrwałym siedzeniem przy komputerze. System może przyczynić się do:

- **Redukcji problemów zdrowotnych** - wczesne wykrywanie i korygowanie nieprawidłowej postawy może zapobiegać bólom kręgosłupa, napięciom mięśniowym i innym dolegliwościom
- **Zwiększenia świadomości ergonomicznej** - użytkownicy uczą się rozpoznawać prawidłową postawę i świadomie ją kontrolować
- **Poprawy produktywności** - prawidłowa postawa wpływa na koncentrację i zmniejsza zmęczenie podczas długotrwałej pracy
- **Oszczędności w ochronie zdrowia** - prewencja jest bardziej efektywna kosztowo niż leczenie problemów kręgosłupa

## 8.3 Wnioski końcowe

Realizacja projektu potwierdza tezę, że nowoczesne technologie sztucznej inteligencji mogą być skutecznie wykorzystane do rozwiązywania praktycznych problemów zdrowotnych związanych z ergonomią pracy. Połączenie zaawansowanych algorytmów z przemyślanym interfejsem użytkownika tworzy narzędzie, które jest zarówno skuteczne technicznie, jak i przyjazne w codziennym użytkowaniu.

Projekt stanowi solidną podstawę do dalszego rozwoju systemów monitorowania zdrowia opartych na analizie obrazu i może inspirować podobne inicjatywy w dziedzinie digital health i preventive care.