Video Model Integration Guide

📌 Overview

This document explains how CatchBarrels integrates video analysis models for biomechanical swing analysis, including:
- Pose detection models (MediaPipe, Abacus AI)
- Biomechanical overlay models (Reboot Motion, Dr. Kwon THSS)
- Joint tracking and visualization
- Future integration paths for advanced analysis

🎯 Current Architecture

1. Pose Detection Layer

CatchBarrels uses pose detection models to extract joint coordinates from swing videos.

Option A: MediaPipe Pose (Current - Browser-Based)

Location: `/nextjs_space/services/mediaPipePose.ts`

```
// Browser-based pose detection
export async function extractJointsWithMediaPipe(
  videoFile: File,
  options?: MediaPipeOptions
): Promise<JointData[]>
```

Pros:
- ✅ Runs entirely in browser (no server costs)
- ✅ Fast processing (< 5 seconds for 3-second video)
- ✅ Privacy-friendly (no video upload to external server)
- ✅ Works offline

Cons:
- ❌ Lower accuracy for occluded joints
- ❌ Limited to 2D pose estimation
- ❌ Requires modern browser with WebGL support

Use Cases:
- Real-time swing feedback
- Lightweight player analysis
- MVP/beta testing

Option B: Abacus AI Pose Service (Future - Server-Based)

Location: `/nextjs_space/services/abacusPoseClient.ts`

```
// Server-based pose detection via Abacus AI
export async function analyzeVideoWithAbacus(
  videoUrl: string,
  options?: AbacusOptions
): Promise<JointData[]>
```

Status: 🚧 **Placeholder** (not yet implemented)

When to Use:
- Higher accuracy pose detection needed
- 3D pose estimation required
- Advanced biomechanical features (e.g., joint angles, velocities)
- Production-scale analysis

Environment Variables Required:

```
ABACUS_POSE_ENDPOINT=https://api.abacus.ai/v1/pose
ABACUS_POSE_API_KEY=your_api_key_here
```

Implementation Notes:
- Abacus AI provides enterprise-grade pose estimation with higher accuracy
- Supports multi-camera calibration
- Can handle complex occlusions
- Returns normalized joint coordinates in world space
- DeepAgent can help integrate this when the Abacus Pose API is ready

---

2. Joint Data Format

All pose detection outputs are converted to a standardized format:

Location: `/nextjs_space/lib/types.ts`

```
export interface JointFrame {
  frame: number;        // Frame number (0-indexed)
  timestamp: number;    // Time in seconds
  joints: {
    [key: string]: {    // e.g., "leftShoulder", "rightHip"
      x: number;        // Normalized 0-1 (left to right)
      y: number;        // Normalized 0-1 (top to bottom)
      z?: number;       // Depth (optional, for 3D)
      confidence: number; // 0-1
    }
  };
}

export type SwingJointSeries = JointFrame[];
```

Joint Names (MediaPipe Standard):
- **Lower Body:** `leftAnkle`, `rightAnkle`, `leftKnee`, `rightKnee`, `leftHip`, `rightHip`
- **Core:** `pelvis` (midpoint of hips)
- **Upper Body:** `leftShoulder`, `rightShoulder`, `leftElbow`, `rightElbow`, `leftWrist`, `rightWrist`
- **Head:** `nose`, `leftEye`, `rightEye`, `leftEar`, `rightEar`

---

3. Scoring Engine Integration

Location: `/nextjs_space/lib/scoring/newScoringEngine.ts`

The scoring engine consumes joint data and calculates:
- **Momentum Transfer Score** (0-100)
- **Ground Flow, Power Flow, Barrel Flow** sub-scores
- **Kinematic Sequence** analysis
- **Timing metrics** (load duration, swing duration, A:B ratio)

```
export function scoreSwing(
  jointSeries: SwingJointSeries,
  level: PlayerLevel
): ScoringResult
```

Key Features Extracted:
1. **Hip Rotation Velocity** (from pelvis/hip joint tracking)
2. **Torso Rotation Velocity** (from shoulder-to-hip axis)
3. **Hip-Shoulder Separation** (angle between hip and shoulder lines)
4. **Kinematic Sequence** (pelvis → torso → arms → bat)
5. **Weight Shift** (lateral movement of center of mass)

🔧 Adding Advanced Biomechanical Overlays

Option 1: Reboot Motion Models

Reboot Motion provides biomechanical reference swings from pro players.

Integration Steps:

1. **Obtain Reboot API Access**
   - Contact Reboot Motion for API credentials
   - Request access to their baseball swing dataset
   - Get model swing JSON data (joint coordinates + metadata)

2. **Store Model Swings in Database**

Add to Prisma schema:

```
model ModelSwing {
  id            String   @id @default(cuid())
  playerName    String   // e.g., "Aaron Judge"
  playerLevel   String   // "MLB", "College", etc.
  handedness    String   // "R" or "L"
  videoUrl      String?  // Optional video file
  jointData     Json     // SwingJointSeries format
  playerHeight  Int?     // Height in inches
  exitVelo      Int?     // Exit velocity (mph)
  launchAngle   Float?   // Launch angle (degrees)
  createdAt     DateTime @default(now())
}
```

1. **Create Comparison API Endpoint**

Location: `/nextjs_space/app/api/videos/[id]/compare-model/route.ts`

```
export async function GET(request: NextRequest, { params }: { params: { id:
string } }) {
  const videoId = params.id;
  const { modelId } = await request.json();

  // Fetch user swing joints
  const userSwing = await prisma.video.findUnique({
    where: { id: videoId },
    select: { skeletonData: true }
  });

  // Fetch model swing joints
  const modelSwing = await prisma.modelSwing.findUnique({
    where: { id: modelId },
    select: { jointData: true }
  });

  // Calculate differences
  const comparison = compareSwings(userSwing.skeletonData, modelSwing.jointData);

  return NextResponse.json({ comparison });
}
```

1. **Visualize Overlays**

Location: `/nextjs_space/components/joint-overlay-compare.tsx`

```
export function JointOverlayCompare({ userSwing, modelSwing }: Props) {
  // Render both swings side-by-side or overlayed
  // Use color coding:
  // - Green: User joints that match model well
  // - Red: User joints with significant deviation
  // - Blue: Model reference joints
}
```

Option 2: Dr. Kwon THSS (The Hitting Solution System)

Dr. Kwon's THSS provides biomechanical "ideal" joint positions for each phase of the swing.

Integration Steps:

1. **Define THSS Ideal Positions**

Location: `/nextjs_space/lib/kwon-thss-ideals.ts`

```
export interface THSSIdeal {
  phase: 'stance' | 'load' | 'launch' | 'contact' | 'follow';
  joints: {
    [key: string]: {
      x: number; // Normalized position
      y: number;
      tolerance: number; // Acceptable deviation
    }
  };
  description: string;
}

export const THSS_IDEALS: THSSIdeal[] = [
  {
    phase: 'stance',
    joints: {
      leftHip: { x: 0.45, y: 0.60, tolerance: 0.05 },
      rightHip: { x: 0.55, y: 0.60, tolerance: 0.05 },
      // ... other joints
    },
    description: 'Balanced stance with weight centered'
  },
  // ... other phases
];
```

1. **Compare User Swing to THSS Ideals**

```
export function compareToTHSS(
  userSwing: SwingJointSeries,
  phase: SwingPhase
): ComparisonResult {
  const ideal = THSS_IDEALS.find(i => i.phase === phase);
  const userFrame = findPhaseFrame(userSwing, phase);

  const deviations = Object.entries(ideal.joints).map(([joint, idealPos]) => {
    const userPos = userFrame.joints[joint];
    const distance = Math.sqrt(
      Math.pow(userPos.x - idealPos.x, 2) +
      Math.pow(userPos.y - idealPos.y, 2)
    );

    return {
      joint,
      deviation: distance,
      withinTolerance: distance <= idealPos.tolerance
    };
  });

  return { deviations, overallMatch: ... };
}
```

1. **Visualize THSS Overlay**

Add a “THSS Mode” toggle to the video player that shows:
- **Gray ghost overlay** of ideal joint positions
- **Green circles** for user joints within tolerance
- **Red circles** for user joints outside tolerance
- **Yellow lines** connecting user joint to ideal position (showing deviation)

Option 3: Chakra System (Energy Flow Visualization)

Visualize energy/momentum transfer through the body using joint velocity data.

Implementation:

Location: `/nextjs_space/lib/chakra-visualization.ts`

```
export function calculateChakraFlow(
  jointSeries: SwingJointSeries
): ChakraFlowData {
  // Calculate joint velocities
  const velocities = jointSeries.map((frame, i) => {
    if (i === 0) return null;
    const prev = jointSeries[i - 1];

    return {
      frame: i,
      jointVelocities: Object.entries(frame.joints).map(([joint, pos]) => {
        const prevPos = prev.joints[joint];
        const dx = pos.x - prevPos.x;
        const dy = pos.y - prevPos.y;
        const dt = frame.timestamp - prev.timestamp;

        return {
          joint,
          velocity: Math.sqrt(dx * dx + dy * dy) / dt
        };
      })
    };
  }).filter(Boolean);

  // Identify "energy centers" (chakra points)
  const chakras = [
    { name: 'ground', joints: ['leftAnkle', 'rightAnkle'] },
    { name: 'hips', joints: ['leftHip', 'rightHip'] },
    { name: 'core', joints: ['pelvis'] },
    { name: 'chest', joints: ['leftShoulder', 'rightShoulder'] },
    { name: 'arms', joints: ['leftElbow', 'rightElbow', 'leftWrist', 'rightWrist'] }
  ];

  // Calculate energy flow through each chakra over time
  const flowData = chakras.map(chakra => ({
    name: chakra.name,
    energyOverTime: velocities.map(v => {
      const chakraVelocities = v.jointVelocities.filter(
        jv => chakra.joints.includes(jv.joint)
      );
      return {
        frame: v.frame,
        energy: chakraVelocities.reduce((sum, jv) => sum + jv.velocity, 0) / chakraVe-
locities.length
      };
    })
  }));

  return { flowData };
}
```

Visualization:

- Draw colored circles at each chakra point
- Circle size = energy magnitude
- Circle color = energy direction (inward/outward)
- Draw “energy lines” connecting chakras in sequence

🚀 Future Enhancements

1. Multi-Camera Calibration

For 3D pose estimation, integrate multiple camera angles:

```
interface MultiCameraSetup {
  cameras: Array<{
    id: string;
    angle: 'front' | 'side' | '45deg' | 'overhead';
    videoUrl: string;
    calibration: {
      focalLength: number;
      principalPoint: [number, number];
      distortion: number[];
    };
  }>;
}

export async function triangulate3DPose(
  setup: MultiCameraSetup
): Promise<SwingJointSeries3D>
```

2. Bat Tracking

Add bat path visualization using object detection:

```
export interface BatTrackingData {
  frames: Array<{
    frame: number;
    batTip: { x: number; y: number; z?: number };
    batKnob: { x: number; y: number; z?: number };
    angle: number; // Bat angle relative to horizontal
  }>;
}
```

3. Ball Tracking

Track pitch location and trajectory:

```
export interface BallTrackingData {
  release: { x: number; y: number; z: number; timestamp: number };
  contact: { x: number; y: number; z: number; timestamp: number };
  trajectory: Array<{ x: number; y: number; z: number; timestamp: number }>;
  velocity: number; // mph
  spinRate: number; // rpm
}
```

📊 Model Integration Comparison

Feature	MediaPipe (Current)	Abacus AI (Future)	Reboot Motion	Dr. Kwon THSS
Pose Detection	✅ Yes	✅ Yes	❌ No (provides reference data)	❌ No (provides ideals)
2D Tracking	✅ Yes	✅ Yes	✅ Yes	✅ Yes
3D Tracking	❌ No	✅ Yes	✅ Yes (if available)	✅ Yes (idealized)
Real-time	✅ Yes	❌ No	N/A	N/A
Cost	🟢 Free	🟡 Paid	🟡 Paid	🟢 Free (if licensed)
Accuracy	🟡 Good	🟢 Excellent	🟢 Excellent	N/A
Use Case	MVP/real-time	Production	Comparison	Coaching

🛠️ Implementation Checklist

Phase 1: Current (MediaPipe)

- [x] Browser-based pose detection
- [x] Joint extraction and normalization
- [x] Basic joint overlay visualization
- [x] Scoring engine integration

Phase 2: Enhanced Visualization

- [ ] Side-by-side comparison view
- [ ] Overlay opacity controls
- [ ] Joint-by-joint deviation highlighting
- [ ] Frame-by-frame scrubbing

Phase 3: Advanced Models

- [ ] Abacus AI pose service integration
- [ ] Reboot Motion API integration
- [ ] THSS ideal position overlays
- [ ] Chakra/energy flow visualization

Phase 4: Multi-Modal

- [ ] Multi-camera 3D reconstruction
- [ ] Bat tracking with object detection
- [ ] Ball tracking integration

- [ ] Exit velocity prediction from biomechanics

📞 Support & Resources

- **MediaPipe Docs:** https://google.github.io/mediapipe/solutions/pose
- **Reboot Motion:** Contact for API access
- **Dr. Kwon THSS:** Review published research papers
- **Abacus AI Pose API:** Coming soon - DeepAgent will help integrate when available

---

📝 Notes for Developers

1. **Always normalize joint coordinates** to 0-1 range for consistency
2. **Handle missing joints gracefully** (use confidence thresholds)
3. **Smooth joint trajectories** to reduce jitter (e.g., Kalman filter)
4. **Calibrate for player height** when comparing to models
5. **Test with multiple camera angles** to ensure robustness
6. **Cache model swings** to avoid repeated API calls
7. **Use web workers** for heavy joint processing to avoid blocking UI

Last Updated: Phase 2 Implementation
Document Version: 1.0