International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 10 Issue: 09 | Sep 2023 www.irjet.net p-ISSN: 2395-0072

Reproducing the Motion of Object AR Models in React.Js Using Tensorflow.Js and React Three Fiber

Atharva Borekar

Software Developer, Idox PLC, Pune, India

Abstract - *In recent years, our community has been using a variety of communication methods. Some of these are social networking sites like Instagram and Snapchat. One of the most popular reasons is the various camera filters (based on AR) used when creating videos and clicking photos. Some are simple, while others are difficult to reproduce; they all have complex methods and a large number of calculations that are performed many times per second. This article introduces you to the principles and core codes of AR-based filters with the help of example transformations in React.Js using Tensorflow.Js and React Three Fiber.*

Key Words: **Augmented Reality, React.Js, Tensorflow.Js, React Three Fiber, 3d Object Modelling**

1. Introduction

The way our society communicates and interacts with the digital world has changed over the years.

The spread of virtual platforms and applications has led to the use of new technologies, especially augmented reality (AR). Platforms like Instagram and Snapchat are capitalizing on the appeal of AR by combining beautiful cameras with interactive features that enhance the user experience. Ranging from the unusual to the complex, these filters are based on complex mathematics that need to be calculated every second.

This article delves into the complex world of AR-based filters and uncovers the code behind the complexities that make visualization so exciting.

We explore design patterns in React.Js that integrates TensorFlow.js and React Three Fiber. By introducing the product and process behind this model, we aim to reveal the core of AR-based filters and make them available to developers.

In this article, we will make a general introduction to the content of AR-based filters. Our purpose is not limited to the use of filters; it also covers the core code that makes the filters compatible with real-world conditions.

Using the power of React.js, TensorFlow.js and React Three Fiber, we demonstrate the effectiveness of designs that bring this content to life.

Node.js can be a powerful framework for building interactive and AR features. We will also demonstrate the TensorFlow.js integration that allows models to perform complex arithmetic operations in real time. To achieve this, React Three Fiber simplifies the integration of 3D images and video, increasing the overall accuracy of the AR experience.

We invite readers on a journey that encourages us to uncover the complex process of AR-based filters as well as find new ways to integrate this technology into many applications. By understanding the principles of AR evolution, developers can use this knowledge to create collaboration and communication in different formats.

In the next section, we'll introduce the basics of AR-based filters and show you the math behind how they work.

In summary, this article presents a combination of React.js, TensorFlow.js, and React Three Fiber for object modeling in AR. By unveiling the rules of

AR-based filters, we empower developers to go beyond the experience and embrace the real world.

2. Literature Review

The advancement of technology and its profound impact on modern life are blurring the lines between virtual and real, paving the way for new digital experiences[2]. Augmented reality (AR), a metaphor for embedding digital content in the physical world, has received a lot of attention in recent years. Known for their camera filters and interactive visuals, AR-based apps are already changing the way people interact with digital content[3].

The use of AR-based filters on popular social networks such as Instagram and Snapchat has attracted great interest. From pretty face distortions to environmental enhancements, these filters enhance visual interpretation and self-expression.
When users use these filters, they have a digital interaction with the world around them.
The magic of AR-based filters lie in their mathematical properties that create overlapping images of the real world[2]. Researchers and developers are looking for better ways to manage these filters, trying to find calculations that will facilitate real-time interaction. The combination of mathematics and creative expression has become the main focus of AR filter development.

Calculations to support AR filters cover a wide range of topics including but not limited to image recognition, object tracking, and adaptive vision[3].
Image recognition algorithms commonly used in deep learning, such as TensorFlow.js, allow filters to recognize and

track different faces or objects in the environment. These capabilities help align the digital support with the user's physical behavior or environment, making the entire AR experience more realistic.

In the software development world, libraries like React.js have become powerful tools for creating interactive user experiences.
React.js manifests simplify the creation of electronic components, allowing developers to change content when data changes. Developers can create flawless AR environments when combined with a library like React We Fiber that extends the power of React.js to 3D graphics.

React interface:
Using the graphical capabilities of Node.js and React Three Fiber, developers can create AR-based filters that respond to user actions and the environment in real time. While the documentation for the AR-based technology is rich, but there is still a large gap between React.js, TensorFlow.js, and React Three Fiber in terms of design to explore resources together. This article fills this gap by providing general guidelines for integrating these concepts into the development of AR-based reusable models.
By presenting a combination of techniques and calculations, this article will increase understanding of the evolution of AR filters and encourage innovation in the field.
This is why information about augmented reality is getting more attention due to the rise of AR-based filters in social media and other apps[3]. Powered by complex algorithms, these filters provide users with a digital experience that blends seamlessly with the physical world[2].
React.js, TensorFlow integration.
React Three Fiber brings a new direction to AR filter development by supporting design, technology integration, creativity and user interaction.
This article builds on current research to understand the potential of the technology and support future efforts in virtual reality.

3. Methodology

AR Realms uses various engines to perform various tasks in AR filter. Despite their volume, the essence of their mathematical thinking remains the same.

To create a prototype, the following steps must be performed:

1. Specify the object i.e. the person in our situation.
2. Use ML models to recognize human movement.
3. Calculate various factors such as points and angles in 3D geometric space.
4. Use angles and points from point 3 to switch to the 3D model.
5. Place the model in 3D space.

Steps:

- Knowing the current position of the body
  The basis of our research is knowing objects and their movements using advanced learning methods. BlazePose is a TensorFlow.js Model that helps predict the human body by detecting 33 skeletons of the human body. From the eyes to the fingers, these elements are important features for capturing movement. Leveraging BlazePose, our framework can recognize and track the movement of content as a leader to create immersive AR experiences.

- Calculation of points and angles:
  The mathematical logic behind AR filters are essential to creating realistic interactions. Calculating angles and points in 3-dimensional geometric space is the cornerstone of recycling. We find the angle between the bone points using the Math.atan2 function. This method must carefully calculate the angles of the different axes and convert radians to degrees. The resulting angles facilitate the movement of the 3D model.

- Integrating Angle Data with 3D Models:
  Integrating angles and points seamlessly into 3D models is an important step in enabling object models that replicate motion. Built on Three.js, React Three Fiber provides a powerful platform for rendering and visualizing 3D objects.
  Using the angles we can adjust the custom body of the 3D model. For example, adjust the rotation of the legs according to the angles to simulate real movement.

- Anti-aliasing algorithm for natural motion:
  We have developed a smoothing algorithm to improve the smoothness and naturalness of the copy. Real movement in the world is characterized by constant change. We can reduce the abrupt change of poses by calculating the average of the previous 10 points of the estimate. This approach provides great AR experiences and easy adaptations to product design.

Note:
The steps mentioned above need to happen "n" number of times per second where "n" should prominently fall in the range of 10-60 for optimal performance. All these steps are then run in a sequential manner and generate a move replicating 3d humanoid in a 3d space.

3.1 Technical Specifications

The Pose Estimation Component is a React-based web application that utilizes TensorFlow.js for real-time pose detection from a webcam feed. It estimates the human body's key points and calculates various angles between body parts to provide insights into the user's pose. This technical specification outlines the key functionalities, components, and implementation details of the Pose Estimation Component.

3.2 .Key Features

- Real-time Pose Detection:
  The component captures video from the user's webcam and performs real-time pose detection on the captured frames.
- Angle Calculation:
  It calculates angles between various body parts based on detected keypoints to assess the user's pose.
- Visualization:
  The component provides a visual representation of the user's pose on a canvas, displaying keypoints and lines connecting relevant body parts.
- Angle Averaging:
  It calculates the average angle values over a period to smooth out fluctuations in pose data.
- Modular Design:
  The component is designed to be modular and can be easily integrated into larger applications.

3.1. Dependencies

1. TensorFlow.js (tf.js): TensorFlow.js is used for the underlying machine learning model and pose estimation.
2. React: The component is built using React for user interface development.
3. React Webcam: This library is used to capture video from the user's webcam.
4. HTML5 Canvas: The Canvas element is used for rendering the pose estimation overlay
5. React Three Fiber: Library used to do 3d calculations and render model.

4. Component Architecture

- Initialization
  - The component initializes TensorFlow.js with the WebGL backend for GPU acceleration.
  - It sets up a `Webcam` component to capture video from the user's webcam.
  - A canvas element is created for rendering the video feed and pose estimation overlay.

- Pose Detection
  - The component loads a pre-trained pose detection model (BlazePose) using TensorFlow.js.
  - Pose estimation is performed continuously on each frame of the webcam feed.

- Data Processing
  - Keypoints representing various body parts are detected and extracted from the video frames.
  - The component calculates angles between keypoints based on predefined rules and part sets.
  - Angle averaging is applied to provide smoother angle data.

- Visualization
  - The detected keypoints are visualized on the canvas with blue dots.
  - Lines connecting relevant body parts are drawn in red to represent joint angles.
  - The canvas is updated in real-time to reflect the user's current pose.

- Data Export
  - The calculated pose data, including keypoints and angles, is made available for external components like react three scene via callback functions.

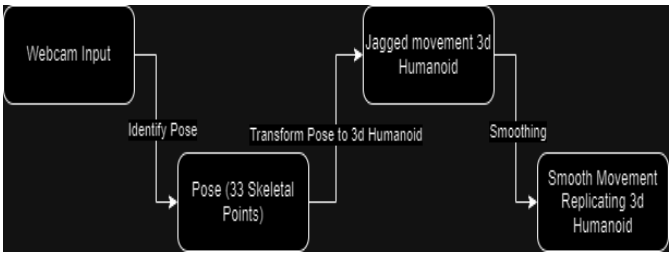

Figure 1: Architectural Flow

4.1 . Component Functions

1. getModel
   - This asynchronous function initializes and returns the pose detection model (BlazePose) using TensorFlow.js.
2. drawLines
   - Draws lines connecting keypoints on the canvas to represent joint angles.
3. drawKeypoints
   - Renders keypoints on the canvas as blue dots.
4. detect
   - Captures video frames from the webcam.
   - Performs pose estimation using the loaded model.
   - Calculates and visualizes keypoints and angles on the canvas.

5. runModel
   - Initiates the continuous execution of the `detect` function at regular intervals (e.g., every 10 milliseconds).
6. getAngles
   - Calculates angles between keypoints based on predefined rules and part sets.
7. getDirectAngle
   - Computes the direct angle between two points in 3D space.
8. angleBetweenLines
   - Calculates the angle between two lines formed by three points.
9. getAverage
   - Computes the average value of an angle over a set of historical angle measurements.

4.2 . Using the Pose Estimation Component

1. Include the component in a React application and provide callback functions to receive pose data (keypoints and angles).
2. Initialize the component, ensuring that it captures video from the user's webcam.
3. Start the pose detection process, which continuously updates the canvas with the user's pose.
4. Retrieve and utilize the calculated pose data for your application's specific requirements.

4.3 . Model (3d Humanoid) Architecture

1. Initialization
   The component imports the 3D model of a humanoid robot using the useGLTF hook from @react-three/drei. It sets up initial values for angle history, initializing an array for each angle element.
2. Angle History
   The component updates the angle history by pushing new angle values into the respective arrays for each angle element. Historical angle measurements are used for angle averaging, providing smoother animations.
3. Animation Logic
   The useFrame hook from @react-three/fiber is used to update the 3D model's bones and perform real-time animation.
   The component calculates the rotation angles for specific bones based on the averaged angle data from the Pose Estimation component.
   The rotation angles are applied to the 3D model's bones to mimic the user's pose.Start the pose detection process, which continuously updates the canvas with the user's pose.
4. Component Rendering
   The 3D model is rendered within a <group> element with the appropriate transformations (position, rotation, and scale).

4.3 . Model (3d Humanoid) Functions

1. updateAngles
   This function updates the angle history for each angle element based on the provided pose estimation data. It maintains a fixed-length history array for each angle element to enable angle averaging.
2. getAverage
   This function calculates the average value of angle measurements for a specific angle element by summing historical measurements and dividing by the number of measurements.

5. Results

The Model component presented here is designed to integrate a 3D model of a humanoid robot into a React application. It utilizes real-time pose estimation data to animate the 3D model, mimicking the user's movements and poses. Below, we discuss the key features and functionalities of this component and its expected outcomes.

5.1. Expected Outcome

When this Model component is integrated into a React application and supplied with real-time pose estimation data (the angles prop), it results in the following outcomes:

1. The 3D model of the humanoid robot will animate in real-time, responding to the user's movements and poses.
2. The arms, thighs, calves, pelvis, and head of the 3D model will mimic the corresponding movements of the user's body, creating an interactive and visually engaging experience.
3. The animations will be smooth and responsive, thanks to the angle averaging logic applied to historical angle measurements.

Developers can further customize and enhance the component to suit their application's specific requirements, such as integrating additional interactions or rendering multiple 3D models simultaneously.

Overall, the Model component provides a visually compelling way to visualize and interact with 3D models in real-time, making it a valuable tool for applications in fields like gaming, virtual reality, fitness tracking, and more.

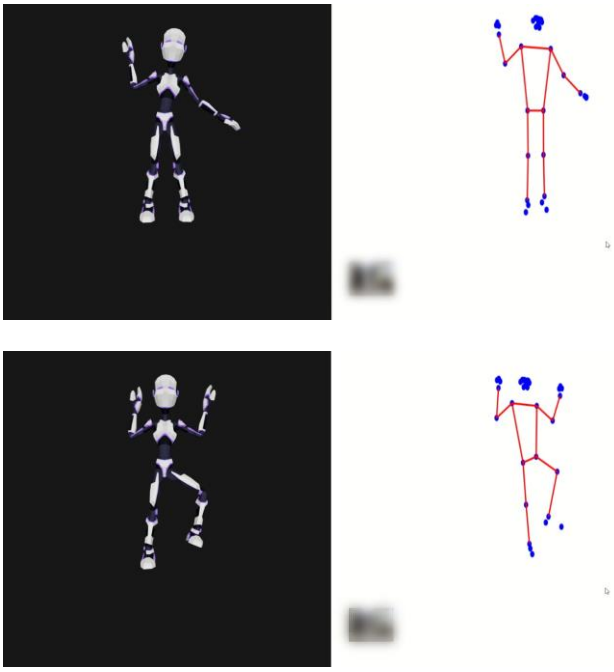

Figure 2: 3d Humanoid (right) imitating poses from left figure obtained through live camera feed (left) Blurred Part on bottom: Live Camera Feed

The Pose Estimation 3D Model Component offers a visually engaging way to visualize and animate a 3D model in response to real-time pose estimation data. Its modular design makes it easy to integrate into various applications, including fitness apps, virtual reality environments, and interactive simulations. Developers can leverage this component to enhance user experiences and create immersive applications that respond to users' movements and poses.

6. CONCLUSIONS

In the rapidly evolving field of Augmented Reality (AR), this research paper reveals a way to uncover the intricacies behind building motion-replicating object models using React.js, TensorFlow.js, and React Three Fiber.

By doing extensive research on the marriage of disparate technologies and math, we present the process that creates beautiful AR filters that have become an integral part of our digital interactions.

The advent of AR has changed the way society interacts with digital images, particularly through platforms like Instagram and Snapchat, where AR-based camera filters have become the importance of visual perception. Ranging from bad competition to complex environmental improvements, these filters engage users by merging the digital and physical worlds. But below the surface, these wonderful experiences are based on tough math.

Our research illuminates the multifaceted process of creating motion-replicating object models in AR.
Leveraging React.js, TensorFlow.js, and React Three Fiber, we're laying out a strong foundation for bringing AR filters to life. This connection not only enables developers and supporters to understand complex technologies but also encourages them to create digital experiences.

Our journey of discovery has many important stages.
We explore areas such as object recognition, motion tracking, angle and point calculation in 3D space. Application of the Math.atan2 function reveals the ability to derive the angles of skeleton elements as a bridge between mathematical theory and interactive magnification. The combination of creativity and technology is further expressed by integrating these angles into a 3D model using React Three Fiber.

To make it smooth and natural, we've added an aesthetic that helps deliver a more realistic AR experience.
From the middle of the previous estimate, we bridge the gap between sudden change and continuous change and make it useful in explaining the view from design standards.

In a nutshell, this research paper begins a journey in AR, technology, and mathematics, complete with a synthesis of the design model. Using React.js, TensorFlow.js, and React Three Fiber, we unravel the complex processes that drive AR filters and highlight the inherent potential of these technologies.
As AR continues to define human interaction with the digital environment, this research highlights the evolution of innovation, creativity and mathematical thinking.

7. Appendices

In the pursuit of creating a movement replicating object AR model using React.js, TensorFlow.js, and React Three Fiber, several critical code snippets and functions contribute to the overall framework's functionality and efficiency. This appendices section provides a detailed breakdown of the key code components involved in the creation of the movement replication model.

7.1. Appendix A: Angle Calculation Functions

To accurately replicate human movements in an AR context, the calculation of angles between skeletal points is essential. The following functions facilitate angle calculations and data processing within the framework.

```
// Constants for identifying different parts

 let partSet = new Set<string>()
 partSet.add("left-shoulder-left-elbow-left-wrist")
// ... (Other parts)
```

```
// Calculate the angle between two points
in radians and convert to degrees

const getDirectAngle = (start: {x:number, y:number,
z:number}, end: {x:number, y:number, z:number}, firstAxis:
string, secondAxis: string) =>
 (Math.atan2(
  start?.[firstAxis] - end?.[firstAxis],
  start?.[secondAxis] - end?.[secondAxis]
 ) * 180) / Math.PI;
```

7.2. Appendix B: Angle Calculation and Smoothing

The getAngles function computes angles between skeletal points, utilizing the previously defined partSet. Additionally, the function incorporates direct angle calculations for specific body parts. Smoothing algorithms are employed to ensure fluid and natural movements.

```
// Calculate angles between different skeletal points

function getAngles(pose: any) {
 const keypoints = pose.keypoints;
 const angles: any = {};

 for (let i = 0; i < keypoints.length; i++) {
  for (let j = 0; j < keypoints.length; j++) {
   for (let k = 0; k < keypoints.length; k++) {
    if (partSet.has(partIndexes[i] + "-" +
      partIndexes[j] + "-" +
      partIndexes[k]
     )
    ) {
     const a = keypoints[i];
     const b = keypoints[j];
     const c = keypoints[k];

     angles[
      partIndexes[i]-
      partIndexes[j]-
      partIndexes[k]
     ] = angleBetweenLines(a.x, a.y,
       b.x, b.y,
       c.x, c.y
      );
    }
   }
  }
 }
// ... (Other direct angle calculations)

return angles;
}

// Calculate the average of a given angle
  element's history

const getAverage = (angleElement: string) => {
 const sum = anglesHistory[angleElement]
        .reduce(
         (a: number, b: number) => a + b,
         0
        );
 let avg = sum;
 avg = avg / anglesHistory[angleElement]
      .length;
 return avg;
};
```

7.3 Appendix C: Angle Calculation Algorithm

The angleBetweenLines function calculates the angle between two lines defined by three points, utilizing vector arithmetic and trigonometry. The algorithm accurately computes angles while considering the orientation of the lines.

```
// Calculate the angle between two lines
  defined by three points
function angleBetweenLines(
 x1: number, y1: number,
 x2: number, y2: number,
 x3: number, y3: number
) {
 // Calculate the vectors representing
   the two lines
 const vec1 = [x1 - x2, y1 - y2];
 const vec2 = [x3 - x2, y3 - y2];

 // Calculate the dot product of
   the vectors
 const dotProduct = vec1[0] * vec2[0] +
          vec1[1] * vec2[1];

 // Calculate the lengths of the vectors
 const length1 = Math.sqrt(
          vec1[0] ** 2 +
          vec1[1] ** 2
         );
 const length2 = Math.sqrt(
          vec2[0] ** 2 +
          vec2[1] ** 2
         );

 // Calculate the angle between the
   vectors in radians
 const angle = Math.acos(
          dotProduct /
          (length1 * length2)
        );

 // Convert the angle to degrees
 const angleDegrees = (angle * 180) / Math.PI;
```

```
  // Determine the clockwise angle
     between the lines
  const det = vec1[0] * vec2[1] -
           vec1[1] * vec2[0];
  if (det > 0) {
    return 360 - angleDegrees;
  } else {
    return angleDegrees;
  }
}
```

REFERENCES

[1] On-device, Real-time Body Pose Tracking with MediaPipe BlazePose :Posted by Valentin Bazarevsky and Ivan Grishchenko, Research Engineers, Google Research

[2] Alzahrani, N.M. Augmented Reality: A Systematic Review of Its Benefits and Challenges in E-learning Contexts. Appl. Sci. 2020, 10, 5660. https://doi.org/10.3390/app10165660

[3] Yunqiang Chen et al 2019 J. Phys.: Conf. Ser. 1237 022082

[4] React.Js: www.reactjs.org

[5] React Three Fiber: https://docs.pmnd.rs/react-three-fiber/getting-started/introduction

[6] Tensorflow.Js: https://www.tensorflow.org/js