Interactive Design, Fine-tuning, and Rendering of Rational B-Spline Motions and Trajectories of Human Limbs and Joints

A report detailing the algorithms used in the development of the framework for joint/limb motion capture and the models/libraries used

by Amro Halwah

Table of Contents

Abstract

This paper presents an online pose detection framework to capture human motion data in order to design and synthesize mechanisms. Using a cheap optical device such as a webcam, joints on the human body can be tracked with the help of a pose detection model. The holistic model, by MediaPipe, can detect the spatial position of many different key points on the human body in real time. In recent years, people have leveraged this model to enable various modern life applications such as fitness analysis and gesture control. In this paper we have presented a tool which can be used to input key positions using gestures for mechanism synthesis, as well as for motion visualization using motion interpolation algorithms. This framework can also be used for capturing human motion data as well as to track joints and filter its trajectories. The structure of the application and motion theory are largely based on Rumit Desai's Kinect-based framework for human motion capture [8]. Desai's tool retrieves spatial joint data from Kinect, a device that detects motion using a RGB video camera and a depth sensor.

Chapter 1: Introduction and Background

1.1: Introduction

The motivation behind understanding human movement is inspired by the complexity of different movements and the subtle differences in a similar motion carried by different people. A motion can be affected by a person's joint mobility, muscular behavior, and limb lengths. Therefore, developing a web application that uses a webcam to capture joint data in real time will aid in the analysis of human motion as it varies from one person to another. Such human data motion is very useful when rehabilitating physically challenged people and analyzing gaits.

1.2: Motion Capture Systems

The analysis of human motion is a task that can be accomplished in different ways. With the help of mechanical capture systems, such as the Gypsy 7, a mechanical motion capture system where each human limb is represented by a mechanical segment on an exoskeleton, the relative motion at 14 different joints can be captured. This system is not prone to occlusion errors which occur in optical motion capture systems [10]. The Microsoft Kinect is an optical motion capture system that allows for the tracking of multiple subjects simultaneously. Kinect has a RGB camera which gives a color image of resolution 1920x1080 pixels. It also has as infra-red emitter that emits a fixed pattern of infra-red rays and an infra-red camera that detects the depth of an object based on the reflection of the infra-red rays. Its depth stream gives a depth image at a resolution of 512x424 pixels at 30 frames per second. The maximum depth measured by the sensor is 4.5 meters, but the quality of depth value degrades at distances greater than 4.0 meters [11]. The input for Desai's Kinect based framework is joint position data from the skeletal tracking feature of the Kinect SDK. In addition to recognizing human joint, the sensor can also recognize predefined gestures like opening a fist and closing a fist [12]. These two gestures are used to start and end the recording of a motion, respectively.

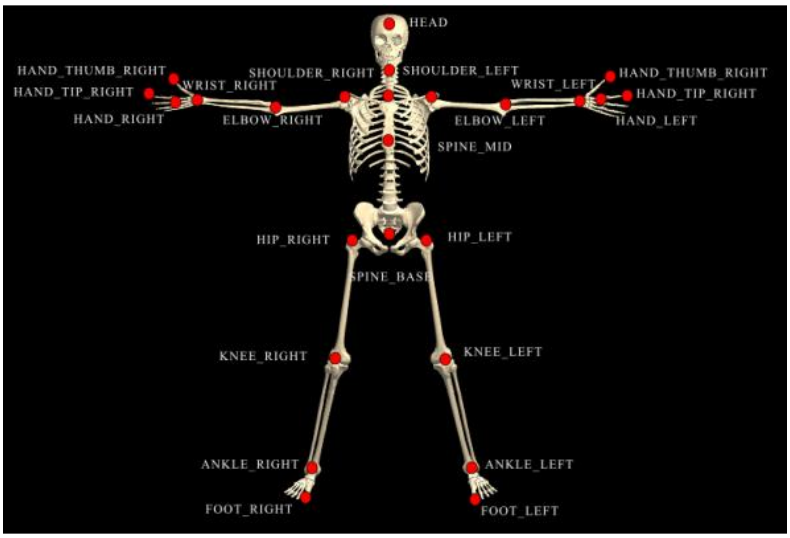
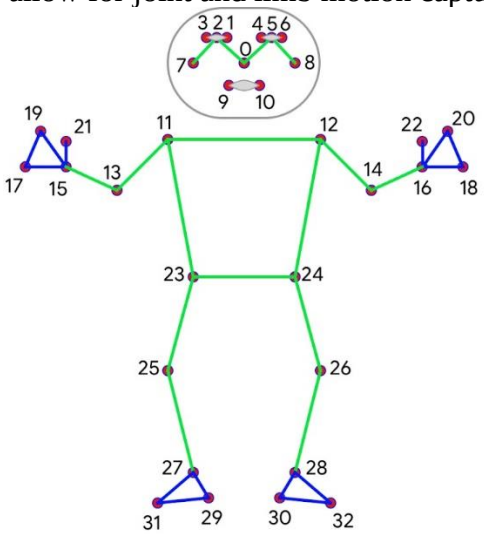


Figure 1.1: Kinect SDK Skeleton Joints

In this paper, the capture of human joint data is accomplished using the Holistic model by MediaPipe. Holistic is a machine learning model that can detect a total of 543 landmarks on the human body (33 pose landmarks, 468 face landmarks, and 21 hand landmarks per hand). The MediaPipe Holistic pipeline integrates separate models for pose, face, and hand components, which is made possible using a multi-stage pipeline. First, the human pose is estimated using Blazepose's pose detector and subsequent landmark model. Then, using inferred pose landmarks, three regions of interest (ROI) crops are derived - for the right hand, left hand, and face. Face and hand models are then applied to ROI crops of the full-resolution input frame to estimate their corresponding landmarks. All landmarks are merged with those of the pose model to yield the full 540+ landmarks that are outputted by the Holistic model [1]. In our tool we use 33 pose landmarks, as labelled in figure 1.2, and 21 right hand landmarks, as labelled in figure 1.3, as input to allow for joint and limb motion capture.



0. nose
1. right eye inner
2. right eye
3. right eye outer
4. left eye inner
5. left eye
6. left eye outer
7. right ear
8. left ear
9. mouth right
10. mouth left
11. right shoulder
12. left shoulder
13. right elbow
14. left elbow
15. right wrist
16. left wrist
17. right pinky knuckle #1
18. left pinky knuckle #1
19. right index knuclke #1
20. left index knuckle #1
21. right thumb knuckle #2
22. left thumb knuckle #2
23. right hip
24. left hip
25. right knee
26. left knee
27. right ankle
28. left ankle
29. right heel
30. left heel
31. right foot index
32. left foot index

Figure 1.2: Pose Landmarks

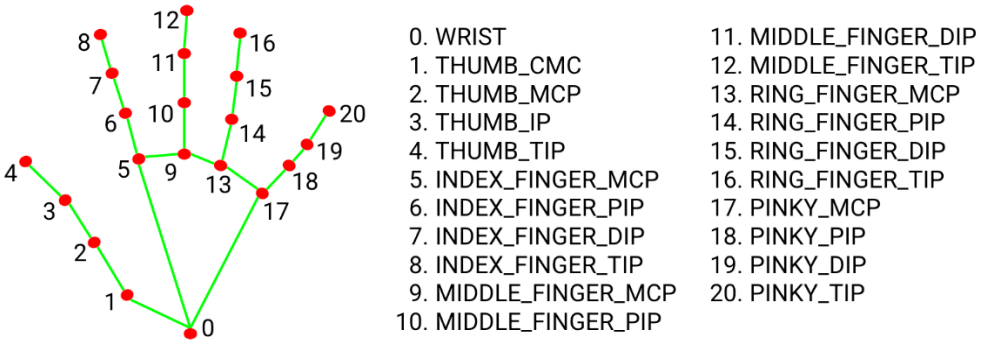


Figure 1.3: Right Hand Landmarks

The pose and hand detection models output x, y, z coordinates for each landmark detected. The x and y coordinates describe the planar position of a landmark. Both models can detect their respective landmarks in either pixels or meters. The pose detection model explicitly predicts two additional virtual key points that firmly describe the human body center, rotation, and scale as a circle. In addition, the midpoint of a person's hips, the radius of a circle circumscribing the whole person, and the incline angle of the line connecting the shoulder and hip midpoints are predicted [7]. This information allows for the prediction real-world 3D coordinates in meters with the origin at the center of the hips. For the hand detection model, the wrist is assumed to be the hand's approximate geometric center when reporting real-world 3D coordinates [5]. The two models can also report the landmarks positions in pixels with the origin of the x and y coordinates being the top left corner of the video's resolution rectangle, and the origin of the z-coordinate being the hip for pose detection and wrist for hand detection. The z-coordinate in pixels uses the same scaling as the x-coordinate [4]. This system lacks the ability to sense the depth of a joint since only an RGB camera is used, but it can predict the depth of a joint relative to a person's hip and the depth of a joint on the hand relative to the wrist. This was made possible by fitting the Generative 3D Human Shape and Articulated Pose Models (GHUM) to existing 2D pose dataset and extended with a real-world 3D key point coordinates in metric space to obtain 3D human body pose ground truth. During the fitting process the shape and the pose variables of GHUM were optimized such that the reconstructed model aligns with the image evidence. This includes 2D key point and silhouette semantic segmentation alignment as well as shape and pose regularization terms [6]. In this paper we use the x, y, and z coordinates of a joint in pixels.

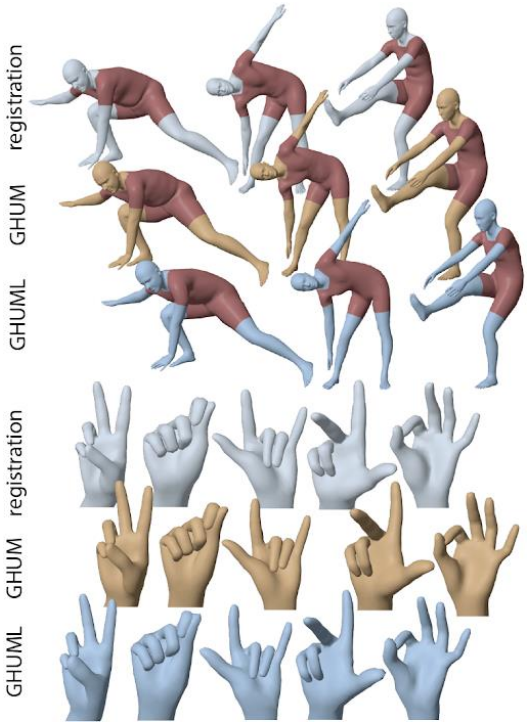


Figure 1.4: Generative 3D Human Shape and Articulated Pose Model

Since the joint detection is done using the input from a RGB camera which is a series of still images in 2D, this method shares the disadvantage of being prone to occlusion errors with Microsoft Kinect. The MediaPipe pose detection model's performance was measured using the Average Percentage of Detected Joints (PDJ) as a metric, which was found to be 91.8% [3]. The MediaPipe hand detection model's performance was measured using the Mean of Normalized Absolute Error (MNAE) by palm size, where the palm size is calculated as the distance between the wrist and first joint of the middle finger. The average MNAE was found to be 10.09% with a standard deviation of +/- 1.73%, where the MNAE of the validation dataset was found to be 6.0% upon re-annotation [2]. These two metrics show that joint detection using these models and an RGB camera has a high accuracy when measuring 2D human joint information, as these models were evaluated using 2D output information.

Chapter 2: Geometric Fundamentals and Motion Design Theory

The motion design algorithms implemented in our framework use dual quaternions for their spatial calculations. The reason being that dual quaternions combine the rotation and translation into a single entity. We can represent a moving frame (M), attached to a rigid moving body in Euclidean (E^3) space, to a fixed reference frame (F) as shown in Fig. 2.1. The position of M with respect to F is given by translation vector $d = (d_1, d_2, d_3)$ and the orientation of M can be represented by the 3x3 rotation matrix [R].

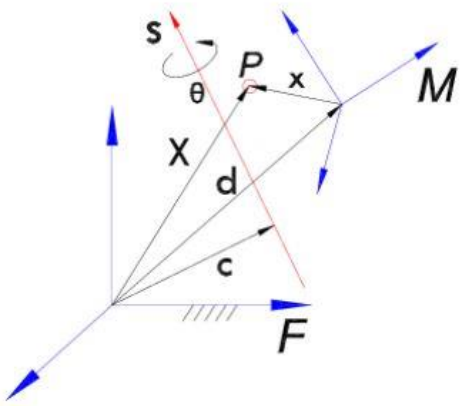


*Figure 2.1: Spatial displacement of **M** given by rotation axis **s**, angle of rotation θ, and translation vector **d** in reference frame **F**.*

Spatial displacement can be represented as a homogeneous transformation in frame **F:**

$$\begin{bmatrix} X \\ 1 \end{bmatrix} = \begin{bmatrix} & R & & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} \tag{2.1}$$

2.1 Quaternions

Rotations in E^3 space can be represented using a combination of an axis of rotation unit vector $s = (s_1, s_2, s_3)$ and an angle θ. These two parameters can then be used to define the Euler-Rodrigues parameters:

$$q_1 = s_1 sin\frac{\theta}{2}, q_2 = s_2 sin\frac{\theta}{2}, q_3 = s_3 sin\frac{\theta}{2}, q_4 = cos\frac{\theta}{2} \tag{2.2}$$

For rotations in Euclidian space to be represented as unit quaternions, the following relation should be satisfied:

$$q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1 \tag{2.3}$$

This relation can be thought of as a point lying on a unit 3-sphere (S^3) that is embedded in 4D space. From both eqn. 2.3 and eqn. 2.2, the rotation matrix $[R]$ defined in the previous section can be backed out by the following:

$$[R] = \frac{1}{S^2}\begin{bmatrix} q_4^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_4q_3) & 2(q_1q_3 + q_4q_2) \\ 2(q_2q_1 + q_4q_3) & q_4^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 + q_4q_1) \\ 2(q_3q_1 - q_4q_2) & 2(q_3q_2 + q_4q_1) & q_4^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \tag{2.4}$$

$$S^2 = q_1^2 + q_2^2 + q_3^2 + q_4^2$$

Therefore, a quaternion, q, can be thought of as a set of homogeneous coordinates that can represent the rotation matrix, $[R]$.

2.2 Dual Quaternions

Dual quaternions are similar to the homogeneous coordinates of q shown in section 2.1. In fact, both q and the translation vector, d, can be placed into a single quaternion. This new set of eight numbers can be used for the design of rational motions. This can be taken one step further using Study's parameters, which gives another set of eight homogeneous parameters (Q, Q^0). The parameter $Q = (Q_1, Q_2, Q_3, Q_4)$ represents the quaternion of homogeneous Euler parameters of rotation. The second parameter $Q^0 = (Q_1^0, Q_2^0, Q_3^0, Q_4^0)$ is the quaternion representing translation. Its components are represented by:

$$\begin{bmatrix} Q_1^0 \\ Q_2^0 \\ Q_3^0 \\ Q_4^0 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 0 & -d_3 & d_2 & d_1 \\ d_3 & 0 & -d_1 & d_2 \\ -d_2 & d_1 & 0 & d_3 \\ -d_1 & -d_2 & -d_3 & 0 \end{bmatrix}\begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \end{bmatrix} \tag{2.5}$$

The translation vector, d, can be backed out by eqn. 2.5 by means of eqn. 2.6, where the value of S is that same as in eqn. 2.4:

$$d = \frac{2}{S^2}\begin{bmatrix} Q_4^0 Q_1 - Q_1^0 Q_4 + Q_2^0 Q_3 - Q_3^0 Q_2 \\ Q_4^0 Q_2 - Q_2^0 Q_4 + Q_3^0 Q_1 - Q_1^0 Q_3 \\ Q_4^0 Q_3 - Q_3^0 Q_4 + Q_1^0 Q_2 - Q_2^0 Q_1 \end{bmatrix} \tag{2.6}$$

These parameters can also take on a dual vector form $\hat{Q} = Q + \epsilon Q^0$, where the symbol ϵ denotes the dual unit. Subsequently, eqns. 2.5 and 2.6 can be rewritten as:

$$Q^0 = \frac{1}{2} dQ \tag{2.7}$$

$$d = \frac{(Q^0)Q^* - Q(Q^0)^*}{QQ^*} \tag{2.8}$$

Note that d as shown in equations 2.7 and 2.8 is a vector quaternion with no scalar part, while $Q^* = -Q_1^0, -Q_2^0, -Q_3^0, -Q_4^0$ is the conjugate of Q such that $QQ^* = Q_1^2 + Q_2^2 + Q_3^2 + Q_4^2$. Eqns. 2.6 or 2.8 can be used to recover d from Q and Q^0, even when they do not necessarily satisfy the plucker condition as shown below:

$$Q_1 Q_1^0 + Q_2 Q_2^0 + Q_3 Q_3^0 + Q_4 Q_4^0 = 0 \tag{2.9}$$

When the plucker condition is satisfied, eqn. 2.8 reduces to:

$$d = \frac{2(Q^0)Q^*}{QQ^*} \tag{2.10}$$

Another way of finding dual quaternions is by using screw displacements. A dual vector can be used to define the screw axis as $\hat{s} = (\hat{s}_1, \hat{s}_2, \hat{s}_3)$, while a dual angle can also be used to define the angle of rotation and translation about said axis, $\hat{\theta} = \theta + \epsilon h$. From this, the four dual components of a dual quaternion can be represented by dual Euler parameters:

$$\hat{q}_1 = \hat{s}_1 sin\frac{\hat{\theta}}{2}, \hat{q}_2 = \hat{s}_2 sin\frac{\hat{\theta}}{2}\ \hat{q}_3 = \hat{s}_3 sin\frac{\hat{\theta}}{2}, \hat{q}_4 = cos\frac{\hat{\theta}}{2} \tag{2.11}$$

Where these four dual Euler parameters must satisfy the relation:

$$\hat{q}_1^2 + \hat{q}_2^2 + \hat{q}_3^2 + \hat{q}_4^2 = 1 \tag{2.12}$$

The dual quaternion, $\hat{q}$, that results from these set of equations is then a unit dual quaternion. Since the formulation of dual quaternions for spatial movement is invariant with respect to the change of both the fixed and moving reference frames, the dual orthogonal matrix $[\hat{R}]$ can be parametrized using dual Euler parameters [8]:

$$[\hat{R}] = \frac{1}{\hat{S}^2}\begin{bmatrix} \hat{q}_4^2 + \hat{q}_1^2 - \hat{q}_2^2 - \hat{q}_3^2 & 2(\hat{q}_1\hat{q}_2 - \hat{q}_4\hat{q}_3) & 2(\hat{q}_1\hat{q}_3 + \hat{q}_4\hat{q}_2) \\ 2(\hat{q}_2\hat{q}_1 + \hat{q}_4\hat{q}_3) & \hat{q}_4^2 - \hat{q}_1^2 + \hat{q}_2^2 - \hat{q}_3^2 & 2(\hat{q}_2\hat{q}_3 - \hat{q}_4\hat{q}_1) \\ 2(\hat{q}_3\hat{q}_1 - \hat{q}_4\hat{q}_2) & 2(\hat{q}_3\hat{q}_2 + \hat{q}_4\hat{q}_1) & \hat{q}_4^2 - \hat{q}_1^2 - \hat{q}_2^2 + \hat{q}_3^2 \end{bmatrix} \tag{2.13}$$

$$\hat{S}^2 = \hat{q}_1^2 + \hat{q}_2^2 + \hat{q}_3^2 + \hat{q}_4^2$$

$\hat{Q} = (\hat{Q}_1, \hat{Q}_2, \hat{Q}_3, \hat{Q}_4)$ may be referred to as a homogeneous dual quaternion, seeing as $\hat{Q}$ can be considered a set of four homogeneous dual coordinates that define a point in the image space [8].

2.3 Motion Design

2.3.1 Rational Screw Motion

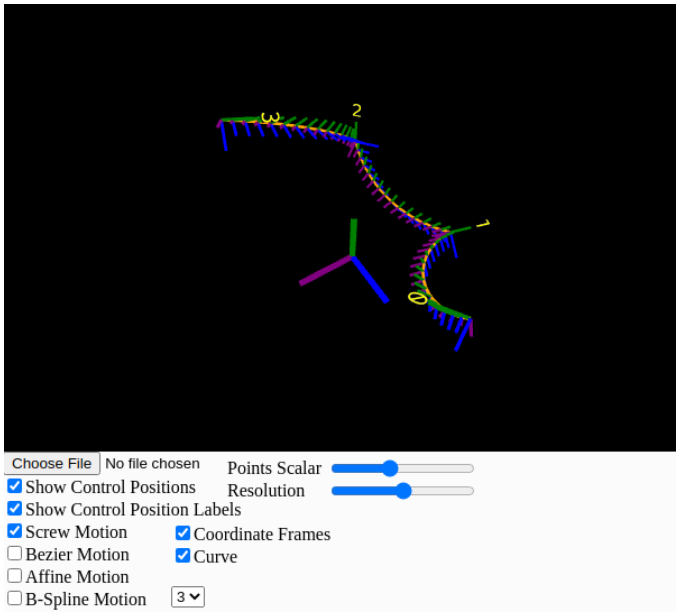


Figure 2.1: Debug rational screw curve and coordinate frame poses

Rational screw motion can be defined as a combination of both rotation and translation parameters. Doing so allows for an easier method of representing spatial displacements.

Additionally, a linear interpolation in dual quaternion space corresponds to a rational screw motion in the Euclidian space, E^3:

$$\hat{Q}(t) = (1-t)\hat{q}_0 + t\hat{q}_1 \tag{2.14}$$

$\hat{q}_0$ and $\hat{q}_1$ as shown in eqn. 2.14 are dual quaternions representing spatial displacement. The curve that this equation forms is shown in Figure 2.1.

2.3.2 Rational Bezier Motion

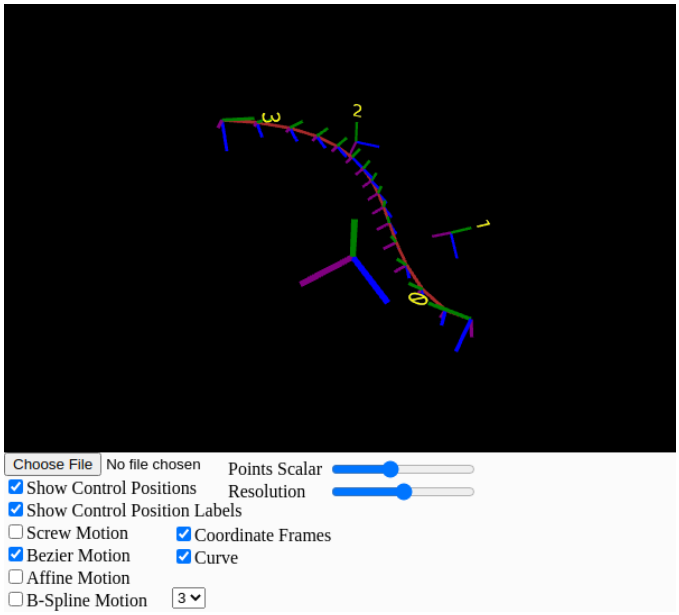


Figure 2.2: Debug rational Bezier curve and coordinate frame poses

A rational Bezier motion can be thought of as a special case of the rational B-spline motion later discussed in section 2.3.3. Here, the degree p of the Bezier motion is always equivalent to one less than the number of control points $(n-1)$:

$$\hat{Q}(t) = \sum_{i=0}^{n} B_i^n(t)\hat{Q}_i \tag{2.15}$$

Here, $\hat{Q}$ is a homogeneous dual quaternion as discussed in section 2.2, and $B_i^n(t)$ represents the Bernstein polynomial function.

2.3.3 Rational B-spline Motion

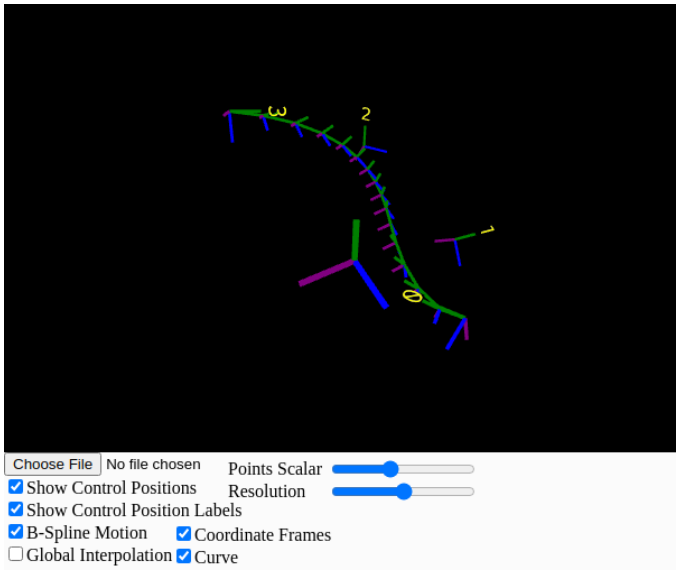


Figure 2.3: Debug rational B-spline curve and coordinate frame poses

The B-spline Motion algorithm used with dual quaternions is very similar to the algorithm used for generating non rational B-spline curves in cartesian space, as seen by the following equation:

$$\hat{Q}(t) = \sum_{i=0}^{n} N_{i,p}(t)\hat{Q}_i \tag{2.16}$$

The difference being that again, $\hat{Q}$ is a homogeneous dual quaternion with both real and dual parts. Consequently, the procedure for generating the motion curve data points would be run twice, once for the real part, and again for the dual part.

The B-spline basis function for a given degree p, $N_{i,p}$, is given as:

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{2.17}$$

The parameter u is known as a "knot", which is an element of the "knot vector" $U = \{u_0, \ldots, u_m\}$, where $m = n + p + 1$. Each consecutive element of this vector must be equal to or greater than the last. Typically, u_0 and u_m are chosen to be 0 and 1 respectively as was done here, however it is not necessary.

This algorithm also makes use of clamped ends, meaning that the curve begins and ends at the first and last control points. For a point to be clamped, a knot in the knot vector needs to have a multiplicity of $k = p + 1$, meaning that the knot needs to be repeated in sequence $p + 1$ times. (E.g., $U = \{0, 0, 0, 0.5, 1, 1, 1\}$ corresponds to clamped ends of a B-spline curve of degree p = 2 and n = 3 control points, as the first and last three $(p + 1)$ knots have the same value).

Several algorithms found in Chapter 2 of Piegl and Tiller's book [23] were used to calculate the basis functions of $N_{i,p}$. Algorithm A2.1, FindSpan() was used to determine the knot span index, and Algorithm A2.2, BasisFuns() was used to compute the nonvanishing basis functions.

2.3.4 Global Motion Interpolation

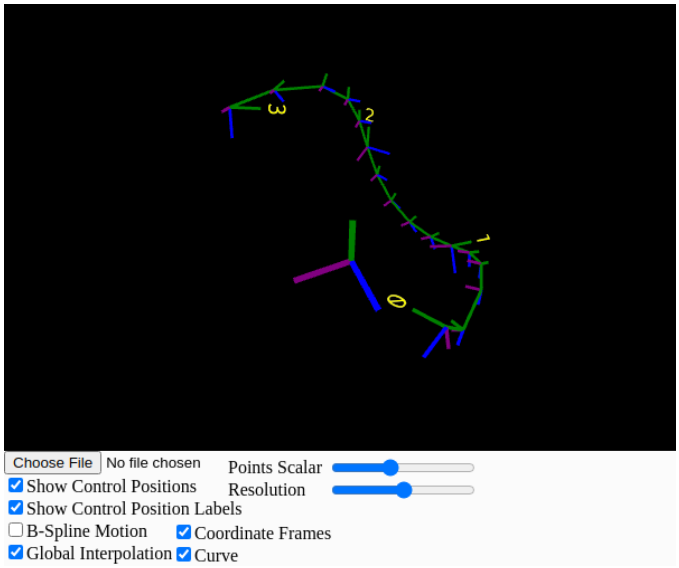


Figure 2.4: Debug global interpolation B-spline curve and coordinate frame poses

The main objective of this function is to generate a curve that passes through a set of given data points. Considering this, these data points cannot be used as the input for, say, a B-spline function. Instead, the control points required to generate such an interpolation curve needs to be calculated. These new control points can then be fed into a B-spline function to output the desired curve.

The program makes heavy use of algorithm A9.1 for Global Curve Interpolation To Point Data, as seen in Piegl and Tiller's The NURBS Book [23] Their approach to the problem begins with defining the given set of data points as the vector $\{Q_k\}$, where $k = 0, \ldots, n$, and $n = Q.length\ -\ 1$, or one less than the length of vector Q_k. These data points are to be interpolated using a p^{th}-degree non rational B-spline curve. Additionally, an appropriate knot vector, $U = \{u_0, \ldots, u_m\}$, where $m = n + p + 1$, should be determined by assigning a parameter value, $\bar{u}_k$, to each Q_k such that an $(n + 1) \times (n + 1)$ system of linear equations can be set up:

$$Q_k = C(\bar{u}_k) = \sum_{i=0}^{n} N_{i,p}(\bar{u}_k) P_i \tag{2.18}$$

In this system, P_i is the vector of newly generated control points with $(n + 1)$ unknowns, and $N_{i,p}(\bar{u}_k)$ is the vector of basis functions for a given $\bar{u}_k$.

It should be noted that this algorithm was intended for use with cartesian coordinates. Within Chapter 9 of Piegl and Tiller's book on Curve and Surface Fitting [23], there is no mention of an algorithm that works with quaternions, let alone dual quaternions. As such, some changes were made.

The choices of $\bar{u}_k$ and U are important as they affect the design and parametrization of the generated curve. Piegl and Tiller briefly covered some of the different methods used to calculate these parameters, as there are many. At the end of their discussion, they heavily suggested using the centripetal method for calculating $\bar{u}_k$ as it produces a more stable and desirable result. The caveats with this method become apparent when working with dual quaternions, however.

Eqn. 2.18 can be adjusted slightly such that the data matrix becomes $Q \in \mathbb{R}^{(n+1)\times r}$, where $n+1$ is the number of data points, and r is the number of coordinates of the coordinate system being used. For dual quaternions, r would be 8 as both its real and dual parts have 4 coordinates respectively. The generated control point matrix P should have the same form as Q, where $P \in \mathbb{R}^{(n+1)\times r}$, and the basis function matrix $N_{i,p}$ should be $N_{i,p} \in \mathbb{R}^{(n+1)\times(n+1)}$. To solve for the matrix P, one coordinate vector per data point is solved at a time such that Q and P become single column vectors (E.g., for cartesian coordinates, solve for x values of all data points, then for y and z):

$$[Q]_{(n+1)} = [N]_{(n+1)\times(n+1)}[P]_{(n+1)}$$

To perform operations on dual quaternions, the real and dual parts need to be separated and operated on individually. Only after can they be re-joined into one dual quaternion. For use in the algorithm, the dual quaternions taken from the data matrix were separated into their respective real and dual matrices. Each matrix would be used to generate their respective control point matrices, $P_{real} \in \mathbb{R}^{(n+1)\times 4}$ and $P_{dual} \in \mathbb{R}^{(n+1)\times 4}$, which would then be placed into a control point dual quaternion matrix, $P \in \mathbb{R}^{(n+1)\times 8}$. Matrix P, parameter value $\bar{u}_k$ and knot vector U are all that's needed to then calculate the interpolation curve. The problem with using the centripetal method is that it outputs a $\bar{u}_k$ and U based on the matrix data that it's given. By inputting both the real and dual parts of data into the algorithm separately, two different values of $\bar{u}_k$ and U are created. This is not ideal as to generate the curve, there needs to be one unified value for both. The less reliable but simpler equally spaced method of finding $\bar{u}_k$ was used instead, as it does not rely on the data being interpolated:

$$\begin{aligned} \bar{u}_0 &= 0 \qquad & \bar{u}_n &= 1 \\ \bar{u}_k &= \frac{k}{n} \qquad & k &= 1,\ldots,n-1 \end{aligned} \tag{2.19}$$

For additional simplicity, the knot vector was made to be equally spaced using the following:

$$\begin{aligned} u_0 = \ldots = u_p &= 0; \qquad & u_{m-p} = \ldots = u_m &= 1 \\ \bar{u}_{j+p} &= \frac{j}{n-p+1}, \quad & j &= 1,\ldots,n-p \end{aligned} \tag{2.20}$$

Chapter 3: Task Position Capture for Motion Visualization

This chapter discusses the methods by which task positions are captured to be used in motion design. Task positions are control positions which are used to design motions using motion design algorithms discussed in chapter 2. MediaPipe's Holistic model provides three-dimensional coordinates (X, Y, Z) in pixels, whereby the X and Y coordinates are measured from the top left corner of the video element and extend to the width and height of the element in pixels, respectively. The Z coordinate is measured from the center of the person's hip whereby a joint that is in between the hip and camera is negative and grows more negative as it gets closer to the camera. A joint that is behind the hip and far from camera is positive and grows more positive as it moves farther from the camera. The Z coordinate uses the same scaling as the X, which would be the width of the video element. In figure 3.1, a stick figure is used to demonstrate how the Z coordinate is measured for a body landmark. The left wrist, in red, is measured to be 100 pixels away from the hip center in the z-direction and is negative because it is in front of the hip, or between the camera and the hip. The right wrist is 80 pixels away from the hip center in the z-direction and is positive because it is behind the hip. Similarly, the hand's landmarks z-coordinates are measured from the position of the wrist, as the geometric center for the hand. The Holistic model provides position information in meters as well, however the x, y, z coordinates are measured with the origin being the person's hip. So, if a person was to hold his wrist in a fixed position relative to his hip and walk around in the scene, the x-y position of the wrist would not even change. Therefore, the unit of pixels was chosen to replicate the Kinect's ability for spatial capture. The Holistic model provides position information for 33 body landmarks and 21 landmarks per hand.

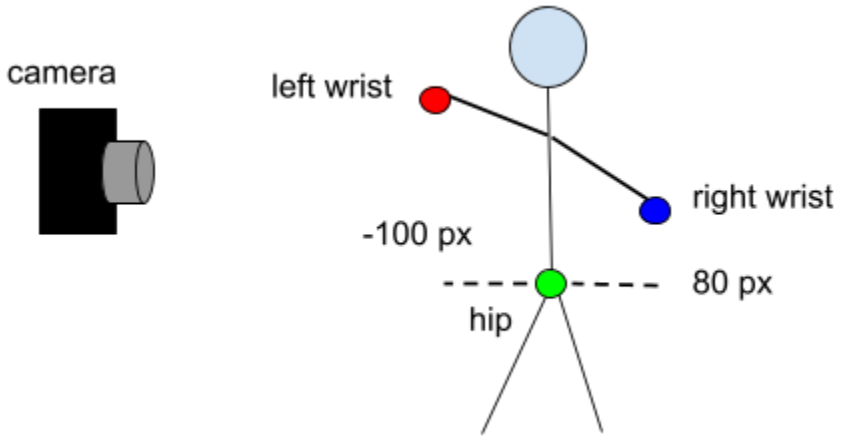


Figure 3.1: A visual aid for understanding the way MediaPipe Holistic measures the Z-coordinate

Using this information, we have created a method for interactively capturing task positions and using them as an input to various motion design algorithms. We have designed a framework which can record a rigid body's displacement and orientation to generate and visualize various

motions. This tool allows the user to select task positions from a continuous motion to use for a mechanism synthesis problem. In this chapter we will first discuss the method for generating task positions from the landmarks position data. Second, we will be discussing the key position recognition algorithm. Third, we will provide examples for motion generation and visualization using recorded key positions and continuous motion. The tool architecture and features will be discussed in chapter 5.

3.1 Pose Generation

A camera is an optical device with no built-in sensors to detect the depth of an object. A camera projects a 3D view onto a 2D view, thus resulting in the loss of the z-dimension which can be obtained if camera parameters and dimensions of the viewed object are known. Luckily, the scaling of the z-coordinate by the Holistic model allows us to visualize a change in a joint's depth in 3D space. The webcam used to capture task positions of a user has a fixed reference frame (F) for planar motion as shown in Fig. 3.2. The fixed reference frame for the z coordinate is the user's hip center. The representation of a rigid body displacement using a 4x4 homogenous matrix can result in the loss of body's rigidity as the rotation matrix can lose its orthonormality in the interpolation and approximation process, see Fillmore [17] and Röschel [18]. To overcome this problem, quaternions will be used to represent rotation and translation, see Shoemake[19], Bottema and Roth [13] and McCarthy[15], since they generate smooth and natural curves for interpolating positions of arbitrary positions.

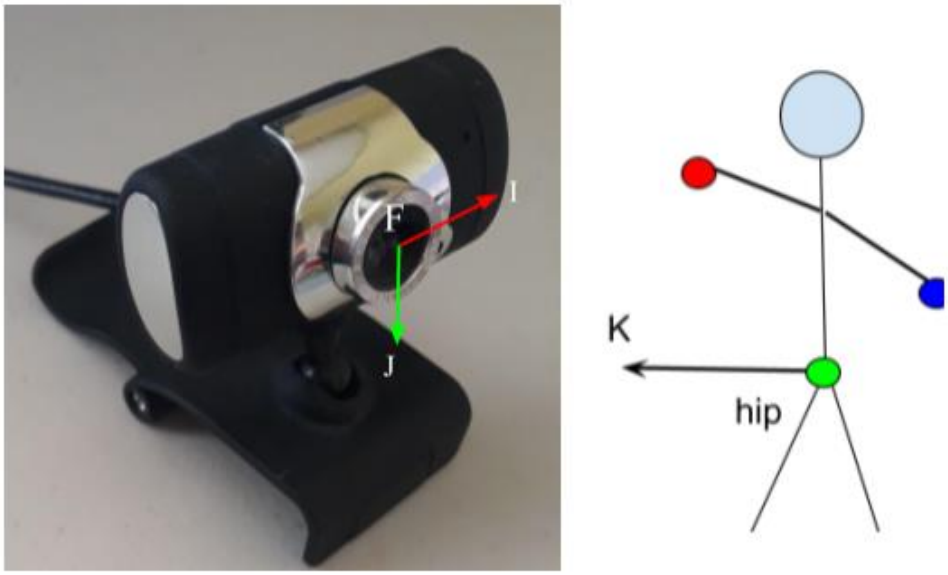


Figure 3.2: A camera uses a left-handed coordinate system to report joint data

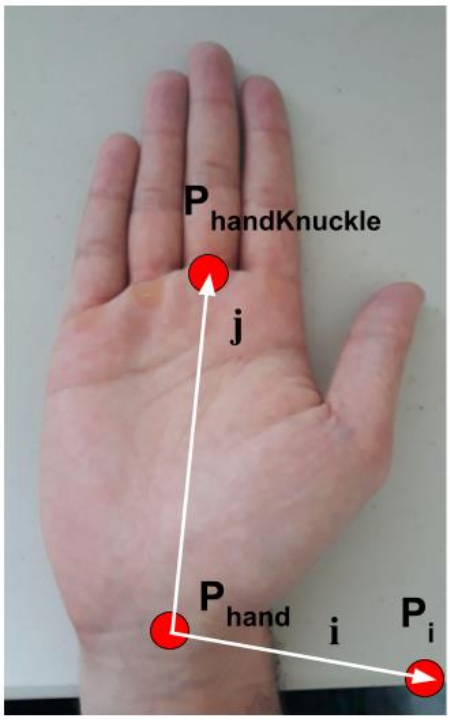


Figure 3.3: Orientation of moving frame (M) attached at the wrist for planar capture

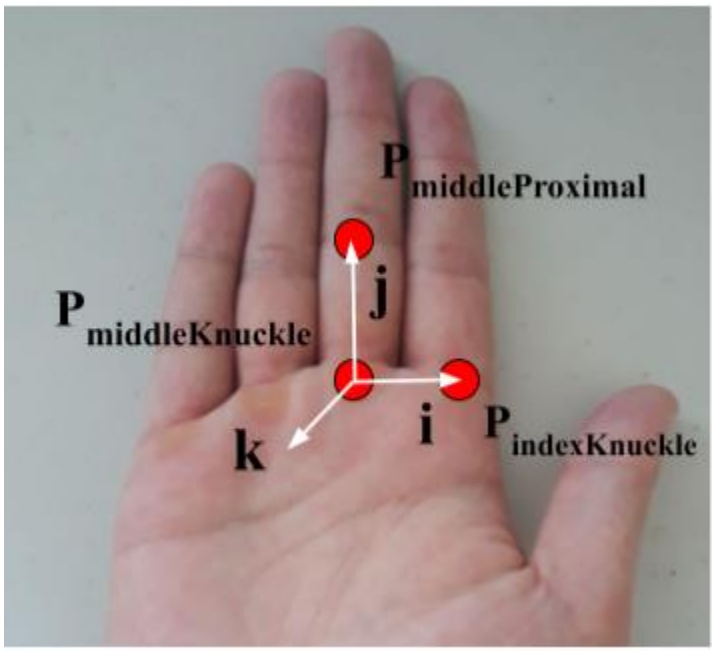


Figure 3.4: Orientation of moving frame (M) attached at the middle knuckle for spatial capture

We have assumed that the moving frame (M) is attached to a user's right hand as shown in Fig. 3.3 and 3.4. The plane of the palm provides the X-Y plane, and the Z-axis can be computed by taking the cross product of two unit vectors that we get in X-Y plane for spatial capture. For

planar capture, however, a vector is calculated using two joints to represent the y-axis, then rotated 90 degrees clockwise to obtain the x-axis as follows:

For planar capture,

$$P_i = (-P_{handKnuckle_y} + P_{hand_y} + P_{hand_x}, P_{handKnuckle_x} - P_{hand_x} + P_{hand_y}, 0)$$

$$j = \vec{u}/\|\vec{u}\|, \text{ where } \vec{u} = P_{handKnuckle} - P_{hand}$$

$$k = j \; x \; \vec{v}/\|\vec{v}\|, \text{ where } \vec{v} = P_i - P_{hand}$$

$$i = j \; x \; k$$

For spatial capture,

$$j = \vec{u}/\|\vec{u}\|, \text{ where } \vec{u} = P_{middleProximal} - P_{middleKnuckle}$$

$$k = j \; x \; \vec{v}/\|\vec{v}\|, \text{ where } \vec{v} = P_{indexKnuckle} - P_{middleKnuckle}$$

$$i = j \; x \; k$$

Using the values calculated above we can define a 4x4 homogenous matrix representing the moving frame (M) with respect to the fixed frame (F) located at the camera as shown in Fig 3.2 as follows:

$$T = \begin{bmatrix} (-1 * I) \cdot i & (-1 * I) \cdot j & (-1 * I) \cdot k & d_1 \\ J \cdot i & J \cdot j & J \cdot k & d_2 \\ K \cdot i & K \cdot j & K \cdot k & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.1)$$

where I, and J are the unit basis vector of F, K is the unit basis vector of the reference frame attached to the hip, and i, j, and k are the unit bases vectors of M. As seen in Fig 3.2, a RGB webcam uses a left-handed coordinate system. In the framework developed by Rumit Desai, the Kinect reported joint information using a right-hand coordinate system. To replicate the same algorithms used by Desai, the unit basis of vector of F, I, was rotated 180 degrees by multiplying the vector by -1 to achieve a right-hand coordinate system. Furthermore, the coordinate attached to the hand was drawn with the y-axis pointing up at no rotation, a choice we made since users are more familiar with a coordinate system orientated this way, as seen in Fig. 3.4.

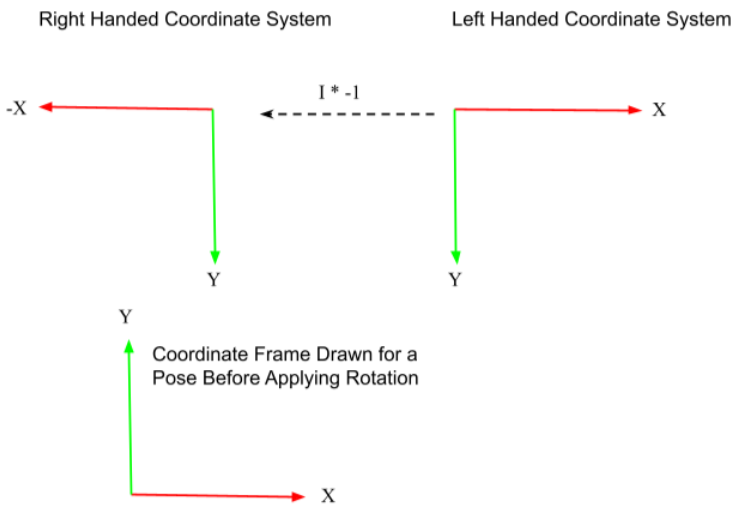


Fig 3.4: Conversion of coordinate system from a left-handed to a right-handed coordinate system

The translation vector from F to M, $d = (d_1, d_2, d_3)$ is given by P_{hand} for planar capture and $P_{middleKnuckle}$ for spatial capture which are values obtained from the Holistic model. In eqn. (3.1) the upper left 3×3 matrix is the direction cosine matrix which gives the change in orientation between F and M as we move our hand. The homogenous matrix is converted to a dual quaternion, to avoid the loss of orthonormality of the rotation matrix during interpolation or approximation, by computing $Q = (q_1, q_2, q_3, q_4)$ representing the rotation part of a dual quaternion by solving eqn. (2.4). The dual part $Q^0 = (q_1^0, q_2^0, q_3^0, q_4^0)$ is obtained from eqn. (2.5). A dual quaternion $\hat{Q} = (Q, Q^0)$ is obtained for every task position we capture.

To generate and visualize planar motions and trajectories for synthesizing planar mechanisms we make the z-translation to be constant by setting $d_3 = 0$ in eqn. (3.1). The upper left 2x2 matrix becomes a rotation matrix describing rotation strictly the X-Y plane instead of the X-Y, Y-Z, and X-Z planes for spatial displacement. The homogenous matrix in eqn. (3.1) reduces to

$$[H] = \begin{bmatrix} cos\theta & -sin\theta & 0 & d_1 \\ sin\theta & cos\theta & 0 & d_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

where θ is the angle of rotation in the X-Y plane. The above equation can be given in terms of eqn. (3.1) as

$$T = \begin{bmatrix} (-1 * I) \cdot i & (-1 * I) \cdot j & 0 & d_1 \\ J \cdot i & J \cdot j & 0 & d_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

where I, J and i, j are the unit basis vectors of F and M respectively, while $d = (d_1, d_2)$ represents planar displacement in X-Y plane.

For each position, the corresponding homogenous transformation matrix can be obtained by solving for rotation matrix [R] from eqn. (2.4) and translation vector d by solving eqn. (2.6).

3.2 Key Position Recognition

The user can capture key positions through which the motion will interpolate or approximate based on the interpolation or approximation motion the user wishes to generate. A continuous motion capture will generate poses in every frame, whereas for a kay position capture the user can give specific positions to generate motions.

We use the same algorithm by Desai, Algorithm 1, to distinguish a key position by measuring the deviation of a particular position over time. A position is recorded as a key position if it does not change over a certain number of frames. A numerical difference between two successive positions in two successive frames is obtained from the dual quaternion $\hat{Q}$ of each position. If the difference is below the *Threshold* and *frameBuffer* is equal to *frameCount* then the position is recorded as a key position.

Algorithm 1 Position Capture Algorithm

1: **for** i = 1 **do**

2: **if** *body* → *tracked* **then**

3: **if** Δ *handPosition* < *Threshold* then

4: **if** *frameBuffer* == *frameCount* **then**

5: *KeyPosition* ← *Position*

6: **else**

7: *frameBuffer*++

9: **else**

10: *frameBuffer* = 0

11: **return** *KeyPosition*

A metric for a planar or spatial displacement should be invariant with respect to change in moving or fixed reference frames and meaningful so that the units of translation and rotation can be combined [20]. The method by which a numerical difference between two successive positions is measured is by converting a dual quaternion into a biquaternion that represents a four-dimensional rotation [22]. We briefly review this method for the computation of a deviation in hand position, as presented in Desai's framework for human motion capture. A pair of unit quaternions, **G** and **H**, are called biquaternions and are represented as follows

$$G = DQ, H = D^{*}Q \tag{3.4}$$

where **D** is a unit quaternion given by

$$D = \frac{2R}{\sqrt{4R^2 + d^2}} + \bar{d}\frac{d}{\sqrt{4R^2 + d^2}} \tag{3.5}$$

and its conjugate

$$D^* = \frac{2R}{\sqrt{4R^2 + d^2}} - \bar{d}\frac{d}{\sqrt{4R^2 + d^2}} \tag{3.6}$$

In eqn. 3.5 and 3.6, R is chosen to be 10000, a sufficiently large number, to achieve better approximation [21], while $\bar{d}$ is a unit vector along translation vector **d** and *d* is the magnitude of the translation vector. Two poses in two successive frames, represented by two dual quaternions Q_1 and Q_2, can be defined by two sets of biquaternions, (G_1, H_1) and (G_2, H_2). The distance between them is given by

$$\Delta_{handPosition} = \sqrt{(G_1 - G_2) \cdot (G_1 - G_2) + (H_1 - H_2) \cdot (H_1 - H_2)} \tag{3.7}$$

In algorithm 1, we have used a threshold value of 0.3 and *frameCount* was set to four times the current frames per second reading of the Holistic model, since the frame count will vary depending on the computer specifications, which is equivalent to four seconds of pause after which a position is declared to be a key position. These values vary slightly from those chosen by Desai but achieve the same goal of allowing the user to pause for three seconds before which a position is stored as a key position. For continuous motion capture, all frames are recorded, and the resulting displacements are stored as a set of dual quaternions.

3.3 Motion Generation and Visualization

Joint data is obtained from the holistic model at approximately 5 frames per second, which varies by the processing speed of a computer, as (x, y, z) coordinates in pixels. The x and y values are normalized by the width and height of the video element to be between 0 and 1, and the z-coordinate has the same scaling as the x-coordinate. For example, a right wrist joint that is being tracked in the camera frame can be at a position of (400, 300, 500) pixels. The video element's width and height in our tool are 640 and 480 pixels, respectively. So, the joint information returned by the holistic model for the right wrist in this frame would be (0.625, 0.625, 0.781), where the x and z coordinates are divided by 640, the width of the video element, and the y-coordinate is divided by 480, the height of the video element. This framework, which uses a pose detection model trained on a dataset of pictures, and a camera to locate joint positions is inspired by Desai's Kinect framework. So, once the user interactively records key poses, they can visualize and generate motions using various motion design algorithm. This is an important feature of the web application because the joint information is bound to be noisy and somewhat inaccurate in detailing the trajectory of a joint. The reason being that a machine learning model is trained to use an input to produce an output. In this case, given an RGB image the model tries to predict the location of joints on a human body in the frame if it detects a human, and since a web camera does not include a sensor, the depth estimation will only be inconsistent. Computer Aided Geometric Design (CAGD) algorithms (see section 2.3) in dual quaternion space are

applied on input positions, represented as dual quaternions, to obtain a representation for rational motions in the Cartesian space [16, 14, 9].

The visualization of motion is done using the 3D graphics library, P5.js, which requires the orientation as Euler angles and position as (x, y, z) coordinates, or a 4x4 homogenous transformation matrix describing both the orientation and position of an object. We are mainly using dual quaternion representation, which is required to apply CAGD algorithms, since the homogenous transform violated the orthonormality condition. The dual quaternions are converted to 4x4 homogenous transformation matrices to visualize the generated motion using P5.js. The complete flow of data from the Holistic model to the final output of motion visualization for spatial displacement, which is almost identical to Desai's for Kinect, can be given as follows:

1. Compute unit vectors of moving frame (M) as follows

$$X_{unit} = P_{indexKnuckle} - P_{middleKnuckle} \tag{3.8}$$

$$Y_{unit} = P_{indexProximal} - P_{middleKnuckle} \tag{3.9}$$

$$Z_{unit} = X_{unit} \times Y_{unit} \tag{3.10}$$

2. Compute homogenous matrix [T] as follows:

$$T = \begin{bmatrix} x \cdot X_{unit} & x \cdot Y_{unit} & x \cdot Z_{unit} & d_1 \\ y \cdot X_{unit} & y \cdot Y_{unit} & y \cdot Z_{unit} & d_2 \\ z \cdot X_{unit} & z \cdot Y_{unit} & z \cdot Z_{unit} & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

3. After computing [T] convert it to dual quaternion by solving eqn. 2.4 and obtaining the dual part by solving eqn. 2.5.
4. Use this dual quaternion as an input to various motion design algorithms discussed in sec. 2.3.
5. Compute homogenous matrix for interpolated positions which are in dual quaternion format, by solving for rotation matrix [R] in eqn. 2.4 and translation vector $P_{middleKnuckle}$ can be obtained by solving eqn. 2.6.

The x and y values of the translation vector $P_{middleKnuckle}$ are mapped from the range of 0 to 1, to the range of -width/2 to width/2 and -height/2 to height/2. The motion is visualized using P5.js WEGL mode which allows for 3D rendering with the origin in the center, thus values must be remapped to return a motion with the same position as that recorded in the video element. In addition, the z value is multiplied by width/2 since it follows the same scaling as x and can be greater than 1. This mapping as well as casting negative position values to be positive so that they can be mapped from the range of 0 to 1 are adjustments that were made during the development of the framework to allows for the visualization of motion recorded using the Holistic model. So, when trying to import the dual quaternion values reported for a continuous motion reported by Desai, they orientation at each position was the same, but the positions were incorrect yielding an unsimilar motion. This would be attributed to the way

the points were chosen to be mapped after solving eqn. 2.6 for the translation vector. This visualization process can be summarized schematically as shown in Fig. 3.5.

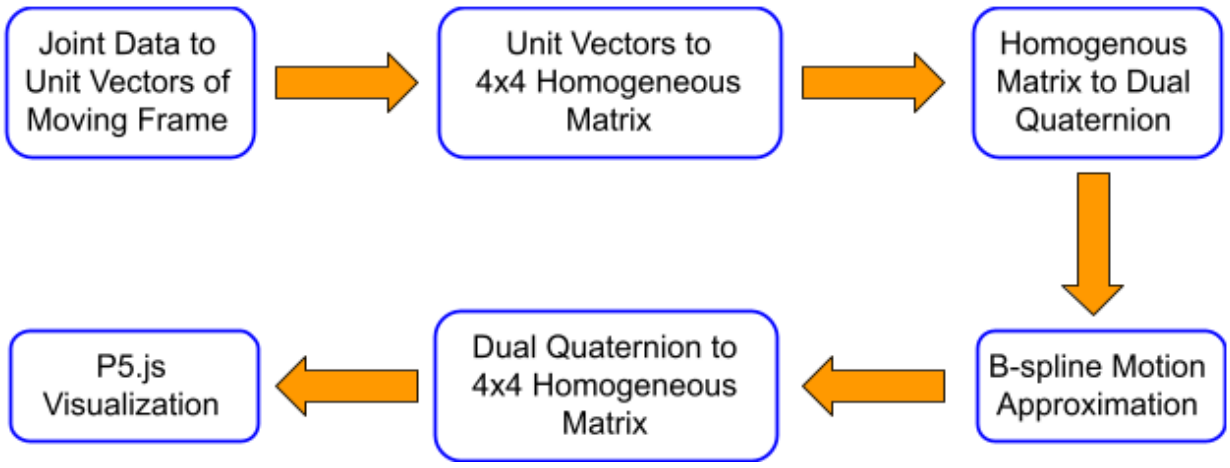


Figure 3.5: Data Flow from Holistic model to the final output on a canvas using P5.js

3.4 Getting Started with Camera Based Task Position Capture

In this chapter, key position capture and continuous motion capture were discussed. Key position capture records a set of poses, according to an algorithm discussed in sec. 3.2, which are classified as key positions and can be used as input to a motion design problem. Continuous motion capture records a pose for every frame for the duration of the recording. A user can choose to record a motion by overlaying the tip of his index finger to be within a green circle on the video element for about two seconds and end the motion by placing it in the same circle, which appears in red during the recording, again for about one second. Selection for continuous motion or key position capture is provided in the UI shown in Fig. 5.2. The tool allows for four types of motion capture:

1. Planar Continuous Motion Capture
2. Spatial Continuous Motion Capture
3. Planar Key Position Capture
4. Spatial Key Position Capture

Once the user selects one of the above combinations and toggles the start recording checkbox on, then a green circle will appear on camera for the user to interact with. Placing the user's index finger in the circle for 10 frames prepares the recording to be started, whereby a countdown of five seconds occurs to allows the user to get in position for recording. The user can stop the recording by placing their index finger in the circle for 5 frames. The last five poses are removed from the capture to account for the five frames of unwanted pose data. Furthermore, the user can delete positions from a continuous motion using the interface. This method of starting and ending a recording differs from Desai's method of closing and opening the left fist to indicate starting and ending the recording, respectively. This is because Kinect's API provides a gesture

recognition feature that can detect and open or closed fist, a feature that MediaPipe's API does not have [8].

Figure 3.6: A user placing the tip of their index finger on the green circle to start recording

Figure 3.7: A user placing the tip of their finger on the red circle to end recording after eleven seconds

3.4.1: Planar Continuous Motion Capture

In a continuous motion capture, the position in every frame is recorded as task positions. Out of these captured positions a user can choose a group of positions to be key positions used for motion generation with motion design algorithms discussed in sec. 2.3. We have selected 8 key positions to use as input for generating rational Screw, rational Bezier, rational B-spline approximated and rational B-spline interpolated motions.

C_i	q_0	q_1	q_2	q_3	q_0^0	q_1^0	q_2^0	q_3^0
C_1	0	0	0.402379	0.915473	0.237071	0.282481	0	0
C_2	0	0	0.349517	0.936929	0.204133	0.246651	0	0
C_3	0	0	0.228704	0.973495	0.156804	0.191990	0	0
C_4	0	0	0.008728	0.999961	0.151006	0.195935	0	0
C_5	0	0	0.248519	0.968626	0.286599	0.222063	0	0
C_6	0	0	0.270123	0.962825	0.373573	0.261267	0	0
C_7	0	0	-0.322453	0.946585	0.254696	0.352208	0	0
C_8	0	0	-0.028048	0.999606	0.333613	0.154582	0	0

Table 3.1: Dual Quaternion key positions selected from planar continuous capture in the form $(\boldsymbol{q},\boldsymbol{q^0})$

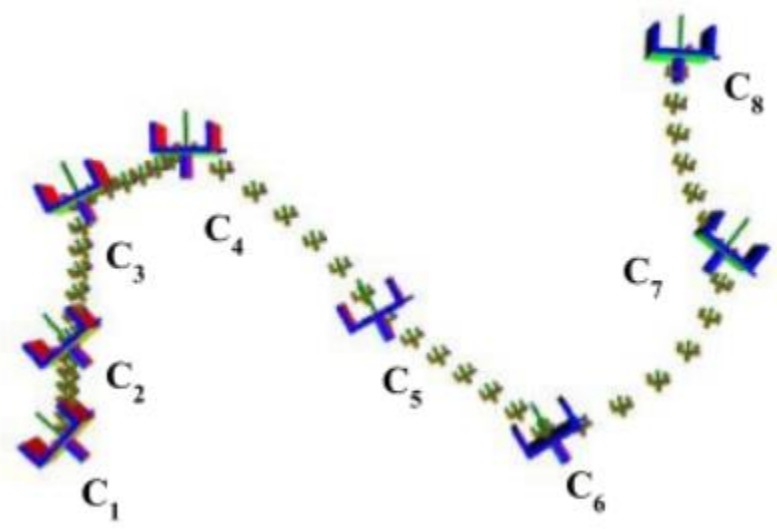


Figure 3.8: A rational screw motion between eight control positions $(C_i; i = 1, \dots, 8)$

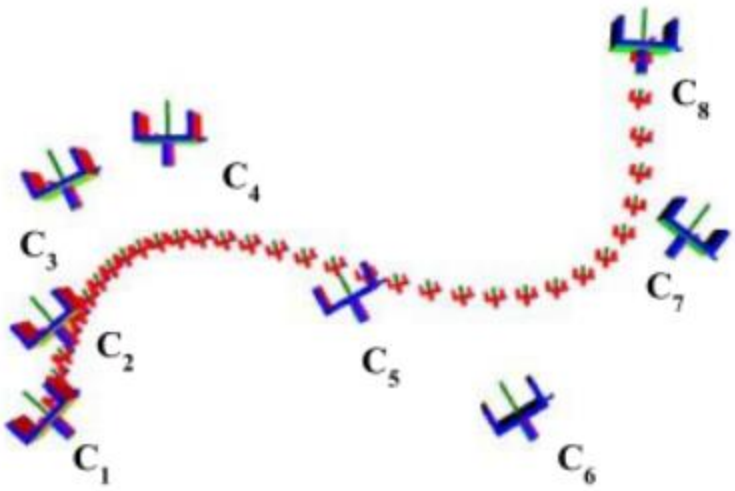


Figure 3.9: A rational bézier motion between eight control positions $(C_i; i = 1, \dots, 8)$

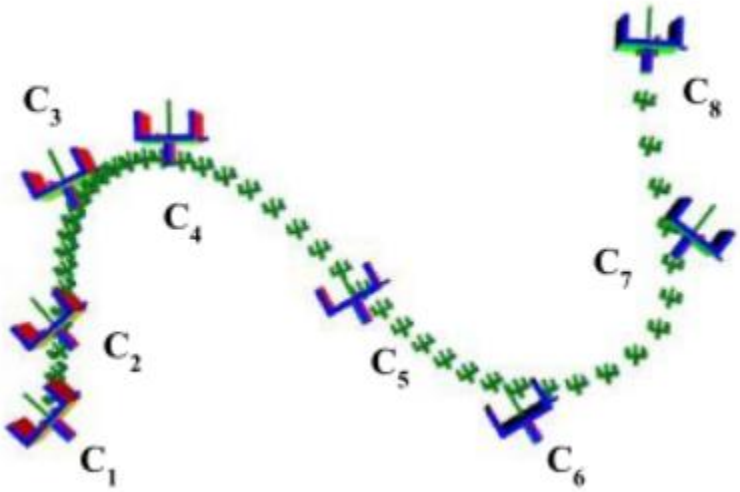


Figure 3.10: A rational B-spline motion between eight control positions $(C_i; i = 1, \dots, 8)$

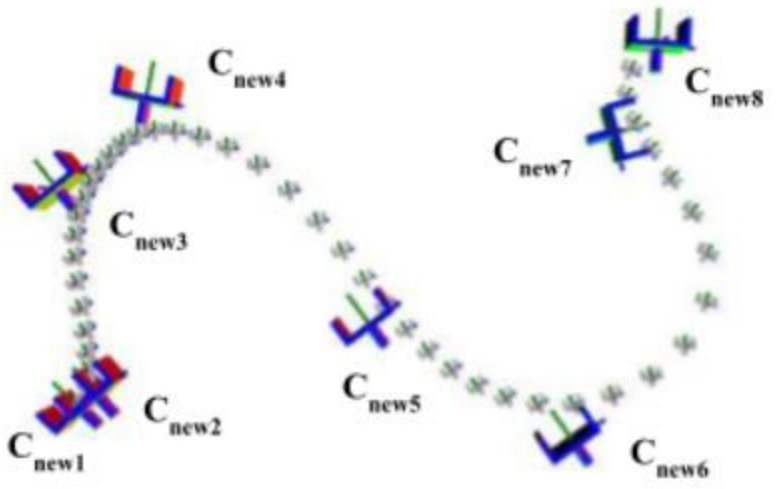


Figure 3.11: A rational B-spline motion interpolating between eight control positions ($C_{newi}; i = 1, \dots, 8$) *calculated using the global motion interpolation algorithm*

3.4.2: Spatial Continuous Motion Capture

Using a continuous capture, we have captured task positions for a spatial motion from which we extracted 8 key positions, shown in table 3.2, to use as input to motion design algorithms.

C_i	q_0	q_1	q_2	q_3	q_0^0	q_1^0	q_2^0	q_3^0
C_1	0.0319	-0.0585	-0.1711	0.9829	0.0016	0.1689	-0.4651	-0.0709
C_2	-0.0737	-0.0887	-0.1381	0.9812	0.0069	0.1942	-0.4570	-0.0462
C_3	-0.2580	-0.1359	-0.1153	0.9495	0.0071	0.2740	-0.4180	-0.0096
C_4	-0.4803	-0.0400	0.0623	0.5727	0.0631	0.3254	-0.1504	0.0921
C_5	0.0021	0.6043	0.5320	0.1828	0.4107	-0.0286	0.0289	0.0052
C_6	-0.0667	0.6737	0.4483	0.2448	0.4063	0.0053	0.0941	-0.0762
C_7	-0.0843	0.7026	0.2610	0.1503	0.3016	-0.0124	0.2028	-0.1248
C_8	-0.1368	0.8761	0.0692	0.2822	0.4118	0.0773	0.1861	-0.0862

Table 3.2: Dual Quaternion key positions selected from spatial continuous capture in the form ($\boldsymbol{q}$,$\boldsymbol{q^0}$)

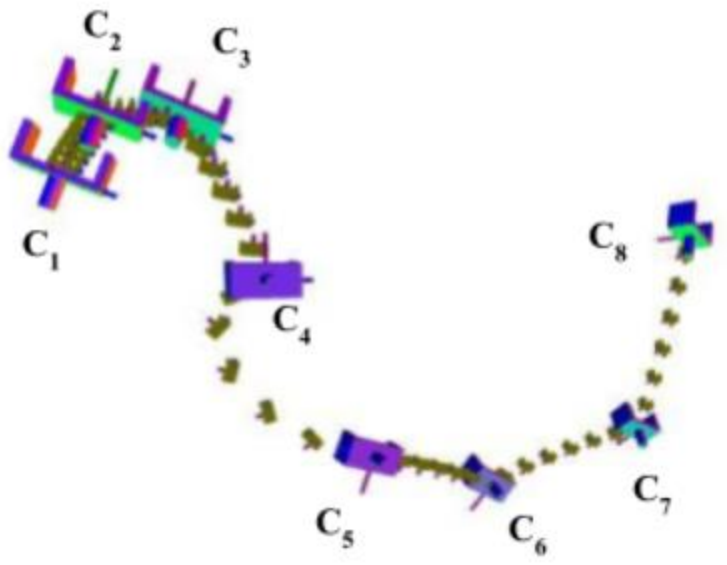


Figure 3.12: A rational screw motion between eight spatial control positions $(C_i; i = 1, \dots, 8)$

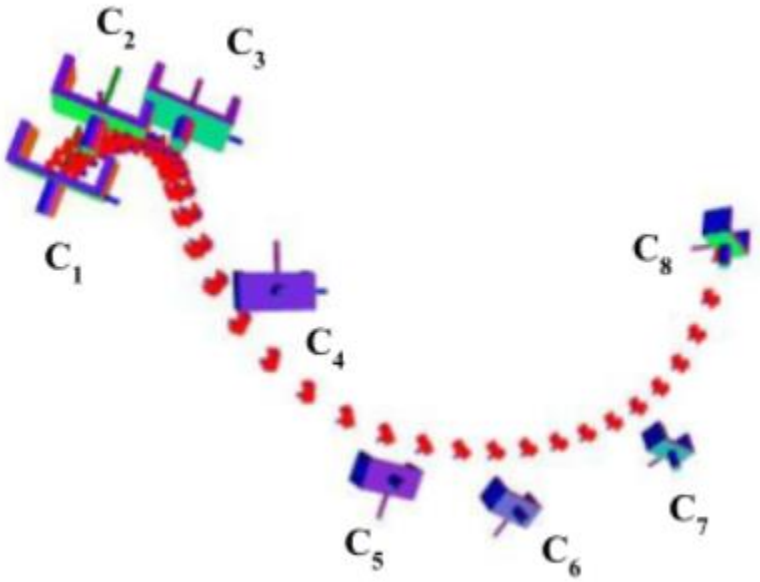


Figure 3.13: A rational bézier motion between spatial eight control positions $(C_i; i = 1, \dots, 8)$

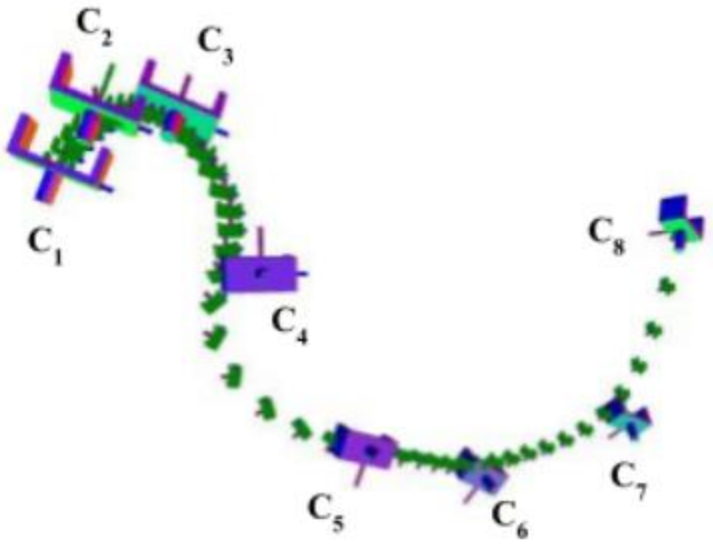


Figure 3.14: A rational B-spline motion between eight spatial control positions $(C_i; i = 1, \dots, 8)$

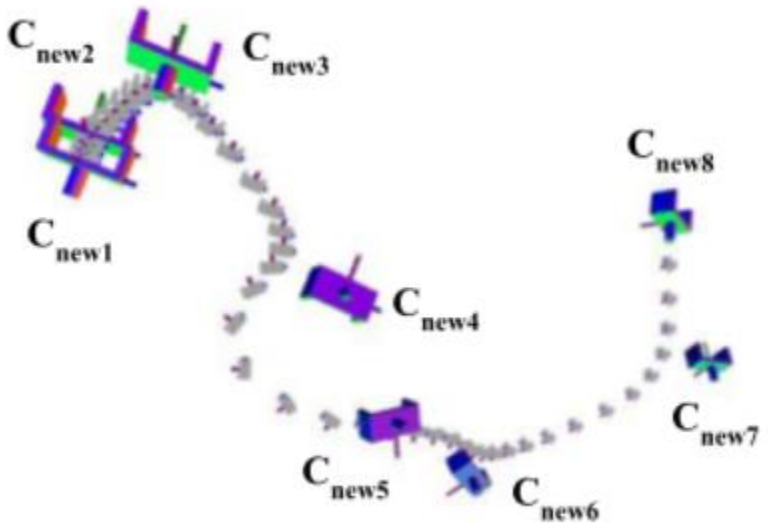


Figure 3.15: A rational B-spline motion interpolating between eight spatial control positions $(C_{newi}; i = 1, \dots, 8)$ *calculated using the global motion interpolation algorithm*

Chapter 4: Human Limb Data and Joint Data Capture for Motion Visualization

In this chapter we will discuss the human limb data capture using a RGB camera. We will cover how to capture joint trajectories and explain how to obtain axis angle representation of a limb

from available joint data. As a demonstration of the human limb capture capability of the framework, a human limb capture of a sit to stand motion will be presented.

4.1: Joint Trajectory Capture

A user may choose to track a joint's displacement to obtain the trajectory of that joint to visualize the motion or filter the noise from the motion using one of the motion design algorithms detailed in chapter 2. In the case of joint motion capture, we follow Desai's consideration that there is no relative rotation between fixed frame (F), the camera's reference frame, and moving frame (M), the frame attached to the joint being tracked. Thus, the spatial displacement of a particular joint can be given as

$$[T] = \begin{bmatrix} 1 & 0 & 0 & d_1 \\ 0 & 1 & 0 & d_2 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

where d_1, d_2, and d_3 represent joint coordinates in the camera's reference frame, as shown in Fig. 3.1. In our framework, a joint is visualized as a point in 3D space. Therefore, a captured joint position at a singular frame is stored as a translation vector, $d = (d_1, d_2, d_3)$, for visualization in P5.js. In the case of a planar capture, the vector becomes $d = (d_1, d_2, 0)$.

4.2: Human Limb Data Capture

This section covers the method by which axis-angle representation for a limb is obtained and how a 4x4 homogenous matrix is derived to represent a spatial displacement of a limb.

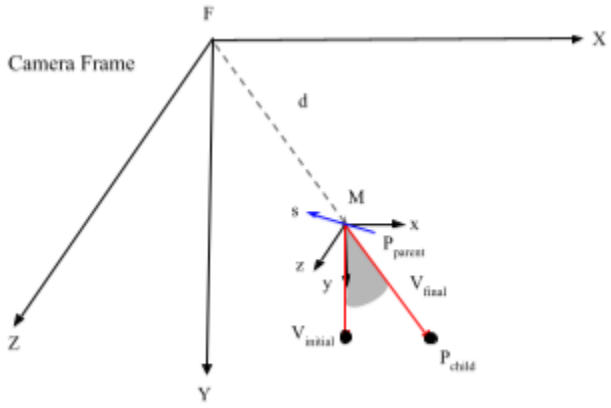


Figure 4.1: Two vectors $\hat{V}_{initial}$ and $\hat{V}_{final}$ as shown in moving frame (M) at a distance d from fixed reference frame (F)

4.2.1 Axis-Angle Representation

In order to capture human limb data, we need the orientation of the limb and translation of the limb in the fixed reference frame to represent its spatial displacement. As shown in Fig. 4.2, we can denote the original position of a stationary limb by $\hat{V}_{initial}$ which is a unit vector pointing downward in the positive direction. This assumes that a limb at rest is oriented downwards. The final position of a limb, $\hat{V}_{final}$, is a unit vector calculated using the top joint of a limb, P_{parent}, and bottom joint of a limb, P_{child}. The initial and final position of a limb can be defined mathematically as:

$$\hat{V}_{initial} = (0,1,0) \tag{4.2}$$

$$\hat{V}_{final} = P_{child} - P_{parent} \tag{4.3}$$

where a moving frame (M) is attached to the parent joint, representing the limb, and the fixed reference frame (F) is attached to the camera as shown in Fig. 3.1.

To obtain the axis-angle representing the orientation of a limb, the angle of rotation θ between $\hat{V}_{initial}$ and $\hat{V}_{final}$, and the unit axis of rotation s $(s_1,\ s_2,\ s_3)$ along which it rotates $\hat{V}_{initial}$ to $\hat{V}_{final}$ are required, which can be given by

$$s = \hat{V}_{initial} \times \hat{V}_{final} \tag{4.4}$$

$$\theta = cos^{-1}\,(\hat{V}_{initial} \cdot \hat{V}_{final}) \tag{4.5}$$

In every frame $\hat{V}_{final}$ can be computed from eqn. 4.3 but $\hat{V}_{initial}$ remains the same for every limb capture. The axis-angle representation of a limb is used to define Euler-Rodrigues parameters

$$q_1 = s_1 sin\frac{\theta}{2}, q_2 = s_2 sin\frac{\theta}{2}, q_3 = s_3 sin\frac{\theta}{2}, q_4 = cos\frac{\theta}{2} \tag{4.6}$$

The rotation matrix [R] representing the orientation of the limb can be computed from Euler parameters using eqn. 2.4. The coordinates of translation vector *d* describing the position P_{parent} in fixed reference frame (F) can be combined with rotation matrix [R] to write a 4x4 homogeneous matrix.

The procedure for limb data capture van be summarized as follows:

1. Compute the unit vector $\hat{V}_{final}$ from eqn. 4.3. $\hat{V}_{initial}$ is known from eqn. 4.2 from the assumption that a capture starts with a limb pointing downwards. Make sure the computed $\hat{V}_{final}$ is a unit vector in the direction of that limb.
2. Compute rotation axis *s* from eqn. 4.4, and angle of rotation θ from eqn. 4.5.
3. Using the rotation angle and axis, compute the Euler Rodrigues parameters representing the rotation of $\hat{V}_{initial}$ to $\hat{V}_{final}$.

4. Compute the rotation matrix [R] from the Euler Rodrigues parameters as shown in eqn. 2.4. The position vector *d* describing the position of P_{parent} joint is readily available from the pose detection model.
5. Combine the rotation matrix [R] and translation vector *d* to form a 4x4 homogeneous matrix from eqn. 2.1 representing spatial displacement of the tracked limb.

A dual quaternion representation can be obtained from the 4x4 homogenous matrix from eqn. 2.7 and eqn. 2.13. The spatial displacement of a particular limb is represented using a dual quaternion because of its application in motion interpolation algorithms where the upper 3x3 rotation matrix should be orthonormal [18].

4.3: Examples

In this section we will show an example of captured limb data and the joint trajectories for the same capture.

4.3.1: Sit-to-Stand Motion

We captured a sit to stand motion, similar to Desai, using our framework to capture the motion of the spine and femur, as shown in Fig. 4.2.

Figure 4.2: Continuous Limb Capture. Brown marked limbs represent poses for the spine and orange marked limbs represent poses for left femur (thigh)

Out of these captured poses, we select eight intermediate poses uniformly spaced to apply our motion filtration algorithms, discussed in section 2.3, to obtain smooth trajectories and eliminate the noise seen in Fig. 4.2. The position for each key position is provided in tables 4.1 and 4.2.

Figure 4.3: Selected key positions for global motion interpolation

C_i	q_0	q_1	q_2	q_3	q_0^0	q_1^0	q_2^0	q_3^0
C_1	0	0	0.37321	0.07507	0	0	0.64211	0.76660
C_2	0	0	0.36960	0.07945	0	0	0.63963	0.76867
C_3	0	0	0.34179	0.12268	0	0	0.57543	0.81785
C_4	0	0	0.31380	0.15966	0	0	0.49050	0.87143
C_5	0	0	0.28330	0.17633	0	0	0.40317	0.91512
C_6	0	0	0.24663	0.17534	0	0	0.27227	0.96221
C_7	0	0	0.14852	0.06731	0	0	-0.08201	0.99663
C_8	0	0	0.19513	0.17153	0	0	0.11666	0.99317

Table 4.1: Dual Quaternion key positions of femur limb in the form ($\boldsymbol{q}$,$\boldsymbol{q^0}$)

C_i	q_0	q_1	q_2	q_3	q_0^0	q_1^0	q_2^0	q_3^0
C_1	0	0	0.22943	0.16251	0	0	0.00288	0.99999
C_2	0	0	0.22887	0.15844	0	0	0.01593	0.99987
C_3	0	0	0.19353	0.16775	0	0	-0.01883	0.99982
C_4	0	0	0.14100	0.18349	0	0	-0.10525	0.99444
C_5	0	0	0.13422	0.15021	0	0	-0.11733	0.99309
C_6	0	0	0.14043	0.10694	0	0	-0.11366	0.99351
C_7	0	0	0.14852	0.06731	0	0	-0.08201	0.99663
C_8	0	0	0.15040	0.04056	0	0	-0.04646	0.99891

Table 4.2: Dual Quaternion key positions of spine limb in the form ($\boldsymbol{q}$,$\boldsymbol{q^0}$)

Fig 4.4: B-spline interpolated motion for femur and spine limb

Chapter 5: Framework Architecture

5.1: Introduction

The pose detection-based framework is a web-based application that uses MediaPipe's Holistic model for pose detection. It is developed using JavaScript (JS) language and the React.js framework. Front-end development is done using React and the P5.js library which allows for manipulation of DOM elements. Furthermore, P5.js is used to for graphical-rendering and text file import export of position information. Node.js is used to install necessary libraries in our project and allow for import of necessary libraries for use. The code is split into two files, one JS file that contains all three P5.js sketch functions along with html elements rendered in React, and a second JS file that contains a class of helper functions that we wrote to aid in the implementation of algorithms discussed in chapter 2-4.

5.2: Pose Detection using MediaPipe's Holistic Model

MediaPipe provides a JS Solution API that supports the Chrome browser on the following platforms: Android, Windows, Mac, and IOS, and supports the Safari browser on iPad, iPhone, and Mac. MediaPipe also provides a usage example in JS for the full body detection model [1]. The example is provided in vanilla JS and provides utilities for pose detection using a camera and drawing on a JS canvas. The drawing utilities, however, have not been used and we have developed our own functions for drawing a skeleton and visualizing motion.

5.3: P5.js Library

There is no available P5.js module to install using Node.js and use in React.js, but there is a ReactP5Wrapper library that allows for a P5 sketch to be developed in the form of a function that includes P5's setup() and draw() functions. This function can then be called using the ReactP5Wrapper inside a *div* element in React's DOM. There are three boxes in the application where canvases are used, and P5.js was used to develop the visuals in these canvases. P5.js was also used to develop the UI since using React hooks to update variables in the DOM causes the page to refresh. This would in turn reload the P5.js sketches which is avoided using P5.js sketches to manipulate the values of global variables throughout the React application.

5.4: User-Interface

As shown in Fig. 5.1 Desai's UI has a color frame where the video feed of the camera is displayed, a skeleton frame where a moving skeleton highlighting the joints and limbs on the user's body is drawn, an OpenGL widget where the motion is visualized, and a control box where the user can interact with the interface. Our user interface, shown in Fig. 5.2, is structured similarly with a skeleton that can move in 3D space, a canvas for visualizing motion, and a control box where the user can make selections and edit the motion captured.

Figure 5.1: Kinect Framework User Interface

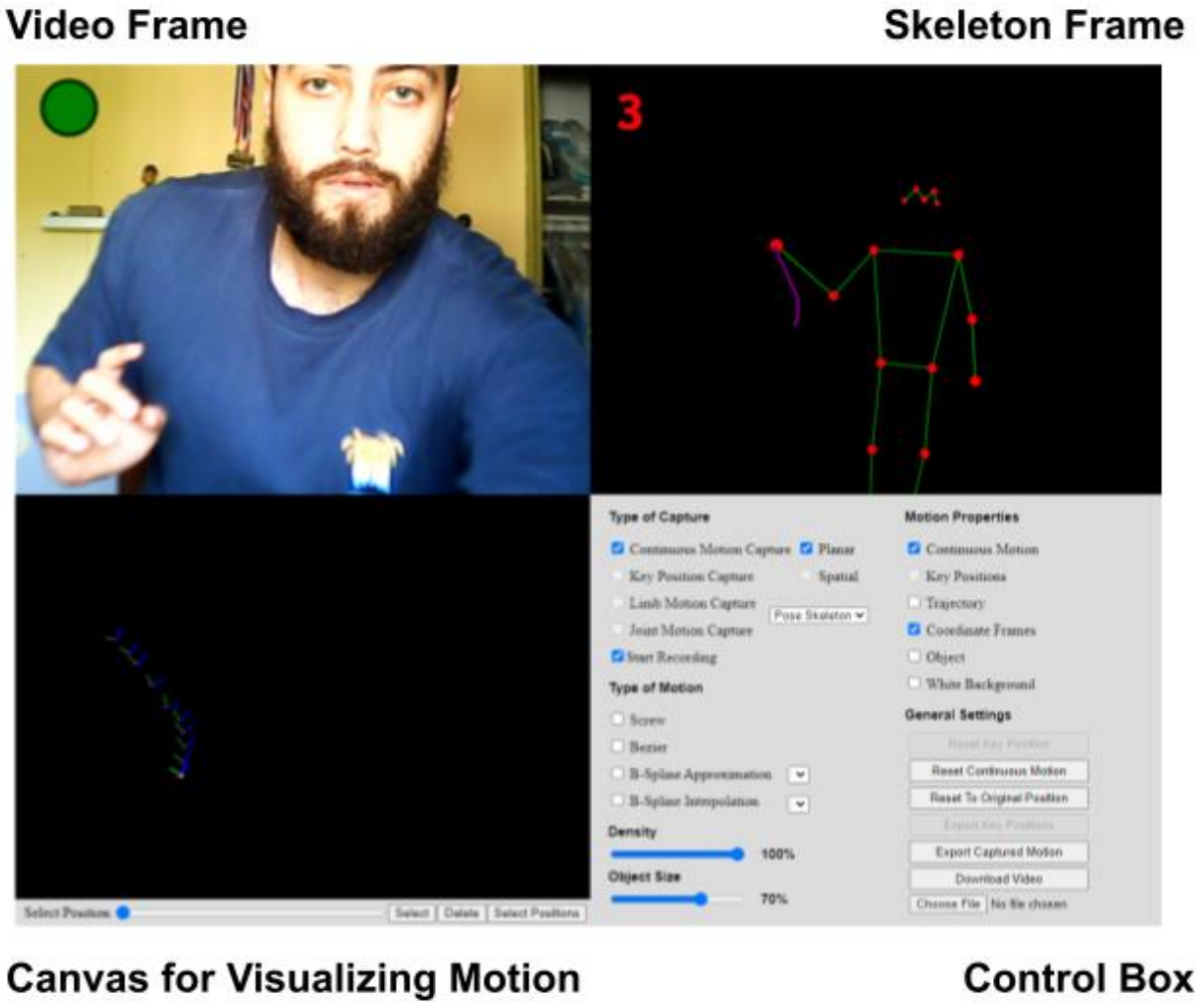


Figure 5.2: Our Pose Detection User Interface

5.4.1: Video Frame

We have a 640x480 pixels video element in which the user can be seen by a camera and detected. The video frame is overlayed with a canvas where both elements are in React's DOM. On the overlayed canvas, which has the same size as the video element, a green circle is draw on the top left, see Fig. 3.6, when start recording is chosen by the user. If the user holds the tip of their index finger in the circle for 10 frames, then the recording will stop, and if the user holds the tip of the finger in the red circle, see Fig. 3.7, for 5 frames, then the recording will stop.

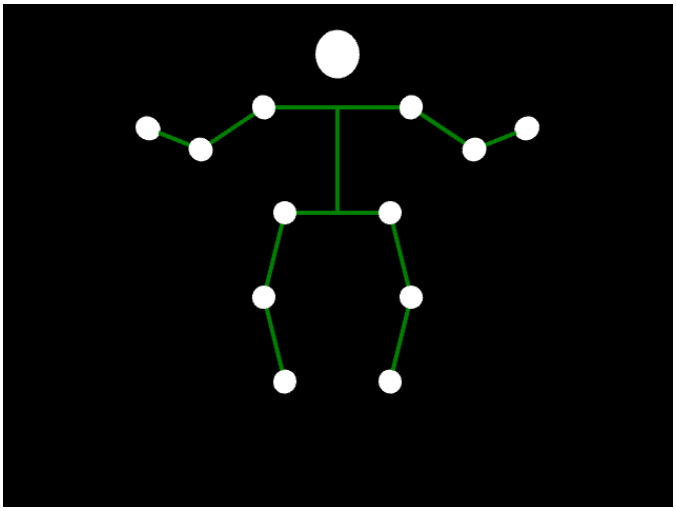


Figure 5.3: Skeleton Frame while not recording

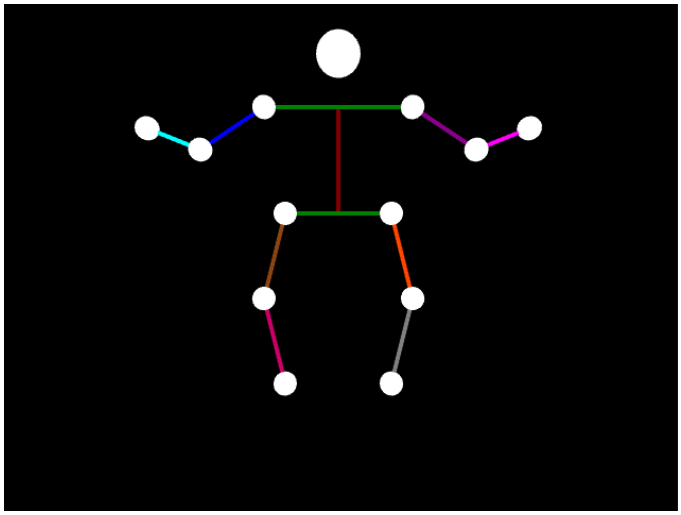


Figure 5.4: Skeleton Frame while not recording with all limbs selected

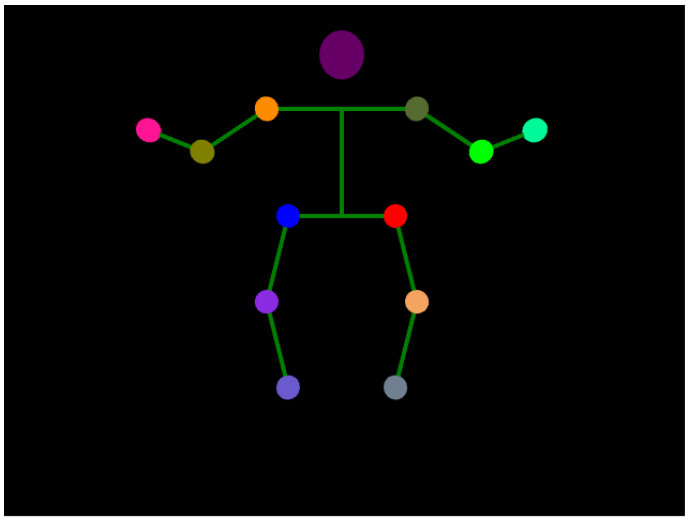


Figure 5.5: Skeleton Frame while not recording with all joint selected

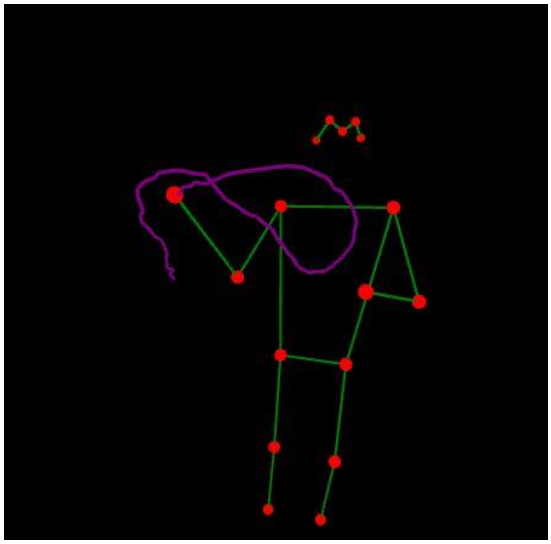


Figure 5.6: Moving skeleton in 3D space showing trajectory

5.4.2: Skeleton Frame

A canvas with the same size as the video element, 640x480 pixels, is created in a P5 sketch. While not recording, the canvas displays a skeleton figure that is used as a joint/limb selection dialogue. A user can select a limb or joint for tracking if they choose limb motion capture or joint motion capture, respectively. The following is a list of selectable left/right joints: shoulder, elbow, wrist, hip, knee, ankle, and the spine. This is the list of selectable left/right limbs: arm, forearm, femur, tibia, and the spine. Fig. 5.4 and 5.5, show the different joints/limbs selected and the color assigned for each one to allow the user to distinguish between the representing objects drawn on the visualization canvas. Fig 5.6 shows a skeleton in 3D space formed using the joint locations obtained from the model, as well as the trajectory of the tracked joint.

5.4.3: Visualization Canvas

This canvas is a P5 canvas that is also 640x480 pixels to allow for the mapping of recorded joint coordinates to the same position a joint was detected at in the video frame. The canvas is created in WEBGL mode to allow for 3D rendering and orbit control by the user. Captured limbs are drawn as cylinders, captured joints are drawn as points, and captured motion from a continuous capture can be represented with coordinate frames or wrist objects to use as input for a mechanism synthesis problem.

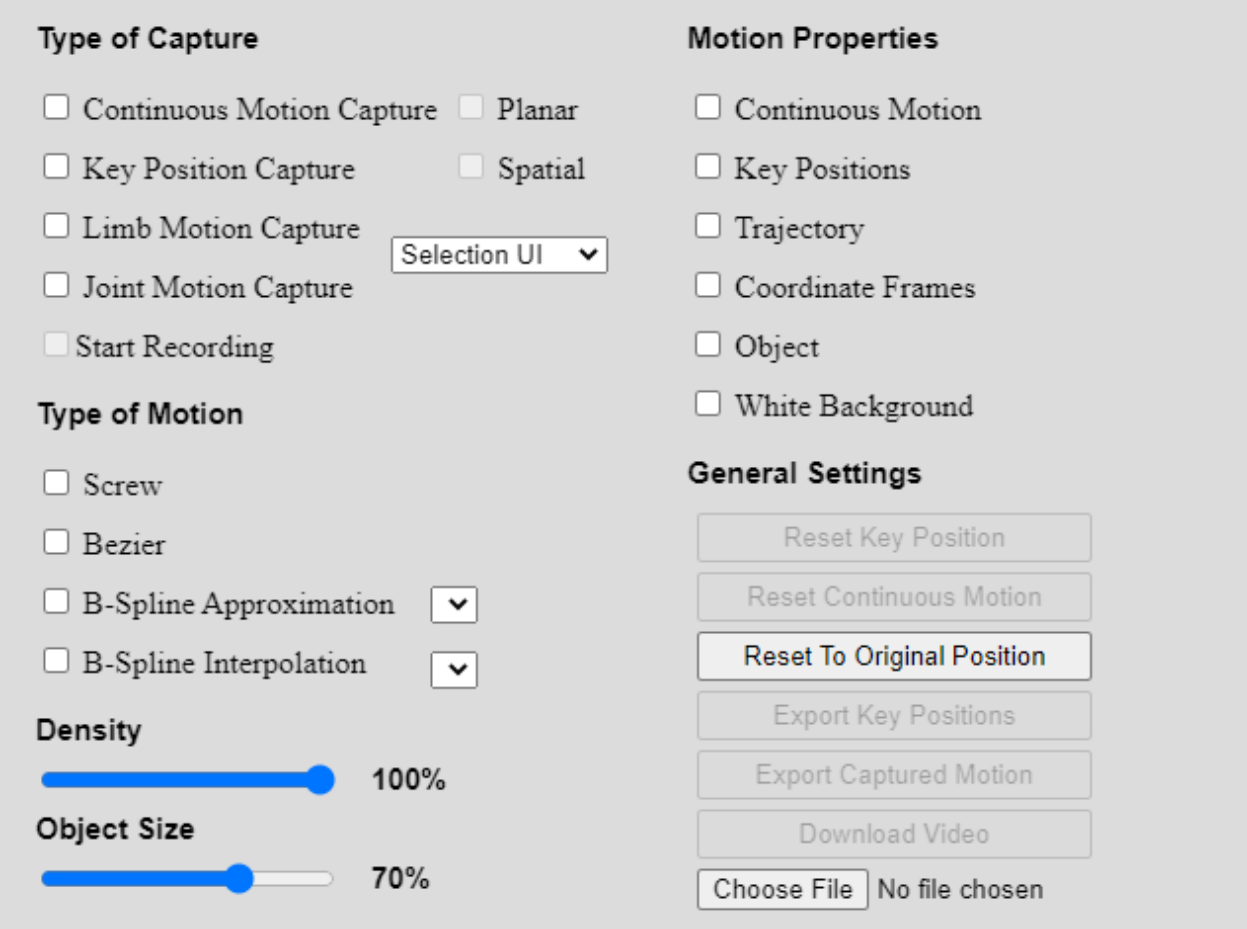


Figure 5.7: Control Box

5.4.4: Control Box

As shown in Fig. 5.7, a user can select the type of capture they want to do. Before starting a motion capture, the user needs to select the type of capture they want to do and specify whether the capture should be planar or spatial. Other options under *Type of Motion*, *Motion Properties*, and *General Settings* should be used once the capture is completed. We have added a feature that allows a user to download a video file of their recording as well as that of the moving skeleton. This would be useful to compare the movement of the user in real life and the movement captured by the model to spot inaccurate frames. The selection box next to Lim Motion Capture

and Joint Motion Capture has two options that allow the user to navigate between the Joint/Limb selection interface and the moving skeleton view where they can see the traced trajectory as well. The moving skeleton view allows for orbit control and the position of the view can be reset using the Reset to Original Position button.

5.5: Application Download and Run Instructions

To download and run the application, the user must have Visual Studio (VS) Code installed. A user must also have node.js installed. Our web application can be downloaded or cloned from our repository; a link for it can be found in the appendix. Open a terminal in VS and change the directory to the program folder. This can be achieved by using the *ls* command to view folders and files in the current directory, and then the *cd* command followed by the folder name to access it. For example, given the following folder tree Research-main => Research-main => program, we would start in Research-main after opening the folder using VS code then proceeding to open a terminal. Using the *ls* command would show that there is only one folder, Research-main, so we use the command *cd Research-main* to get into it. Next, we would use the *ls* command again to see what folders are inside. Among the shown files and folders would be program, so we would use *cd program* to get into it.

Once we have changed our directory to that of the folder program, we run the command*: npm install* to install the necessary modules. After the necessary packages are installed, the program can be started using the command: *npm start*. This will launch the application in a web browser. Note that the installation of modules is done only once after downloading the respository.The appendix also includes a few videos demonstrating some features of the program to familiarize the user with the application.

Chapter 6: Conclusions and Future Work

6.1: Conclusion

The development of this tool is inspired by the Kinect framework developed by Rumit Desai for human limb and joint capture. The algorithms used by Desai in his tool have been employed in the development of our application with slight modification where we thought necessary because of the difference in the device used for capture of joint information. A simple RGB camera, like a webcam that is built into modern laptops, cannot be as accurate as the Kinect device given the absence of a sensor device that can detect a joint’s depth. However, the advantage of creating a framework that uses a camera to capture human joint and limb motion is that a webcam is cheap device to purchase if not readily installed in a laptop. If everyone with a laptop also has access to a webcam and internet, then our tool would be accessible and easy to use for anyone looking to visualize the unique joint or limb motion of a human user. Furthermore, this tool has an important application in robotics as it allows for the exporting of collected data, after filtering if the user wishes, to use as input for a mechanism synthesis problem. This tool can find use in the

biomedical for capturing human motion without an emphasis on 100% accurate capture of a joint's positions.

6.2: Future Work

Our framework currently exports positions in the form of quaternions coupled with translation vectors to describe the orientation and position of an object. A translation vector composed of x, y, and z coordinates can be adjusted by the user, but given that this tool is designed to be accessible to the public, as it is a web application accessible to anyone with internet access, not many people would understand how to change a quaternion's parameters to affect the orientation. Therefore, outputting Euler angles instead of a quaternion would make it easier for many users to manipulate a pose's orientation and position by manually changing angles and coordinates.

The pose detection's frame rate during a recording is affected by the computer's performance, where a slower processor will result in a frame rate of around 5 or less and at higher processing speeds the frame rate can exceed 10 frames per second. It is also known that lighting and occlusion joints can affect the model's inference of joint positions, sometimes resulting in very inaccurate information. For this reason, we provide the user with the ability to download a recording of themselves as well as that of the skeleton, which is a representation of how the model sees the user's joints, to examine when the model may be very inaccurate. In the future, the pose detection model might receive updates that improve its frame rate or depth perception which can improve the spatial capture feature of the framework. Also, if a better pose detection model that allows for depth detection, with a JS API, is released in the future, it can replace MediaPipe's Holistic model in our framework.

Bibliography

[1] Google, “MeidaPipe Holistic”, URL https://google.github.io/MediaPipe/solutions/holistic.html.

[2] Google, “Model Card: MediaPipe Hands”, URL https://drive.google.com/file/d/1-rmIgTfuCbBPW_IFHkh3f0-U_lnGrWpg/preview.

[3] Google, “Model Card: MediaPipe BlazePose GHUM 3D”, URL https://drive.google.com/file/d/10WlcTvrQnR_R2TdTmKw0nkyRLqrwNkWU/preview.

[4] Google, “MeidaPipe Pose”, URL https://google.github.io/MediaPipe/solutions/pose.html.

[5] Google, “MeidaPipe Hands”, URL https://google.github.io/MediaPipe/solutions/hands.html.

[6] Xu Hongyi, Zanfir Andrei, Freeman T. William, Bazava Eduard, Sukthankar Rahul, Sminchisescu Cristian, “GHUM & GHUML: Generative 3D Human Shape and Articulated Pose Models”, URL https://github.com/google-research/google-research/tree/master/ghum.

[7] Bazarevsky Valentin, Grishchenko Ivan, 2020, “On-device, Real-time Body Pose Tracking with MediaPipe BlazePose”, URL https://ai.googleblog.com/2020/08/on-device-real-time-body-pose-tracking.html.

[8] Desai Rumit, 2016, “A Kinect-based Framework for Human Motion Capture for Mechanism Synthesis, Motion Generation and Visualization”.

[9] Purwar, A. and Ge, Q. J., 2005, “On the Effect of Dual Weights in Computer Aided Design of Rational Motions”, ASME Journal of Mechanical Design, 127(5), pp. 967–972.

[10] Motion, M., “Gypsy 7 Motion capture system”, URL http:// metamotion.com/gypsy/gypsy-motion-capture-system.htm.

[11] Microsoft, “Developing with Kinect for Windows”, URL https://dev. windows.com/en-us/kinect/develop.

[12] Microsoft, “Skeletal Joint Smoothing White Paper”.

[13] Bottema, O. and Roth, B., 1979, Theoretical Kinematics, Dover Publication Inc., New York.

[14] Juttler, B. and Wagner, M., 1996, “Computer Aided Design With Spatial Rational B-Spline Motions”, ASME Journal of Mechanical Design, 119(2), pp. 193–201.

[15] McCarthy, J. M., 1990, Introduction to Theoretical Kinematics, The MIT Press, Cambridge, MA.

[16] Ge, Q. and Ravani, B., 1991, “Computer aided geometric design of motion interpolant”, Proceedings of 17th ASME Design Automation Conference, pp. 33–41.

[17] Fillmore, J. P., 1984, “A note on rotation matrices”, IEEE Computer Graphics & Application, 4(2), pp. 30–33.

[18] Röschel, O., 1998, “Rational motion design - a survey”, Computer-Aided Design, 30(3), pp. 169–178.

[19] Shoemake, K., “Animating rotation with quaternion curves”, SIGGRAPH’85, ACM Computer Graphics, volume 19, pp. 245–254.

[20] Park, F. C., 1995, “Distance Metrics on the Rigid-Body Motions with Applications to Mechanism Design”, ASME Journal of Mechanical Design, 117(1), pp. 48–54.
[21] Larochelle, P., Murray, A., and Angeles, J., 2007, “A distance metric for finite sets of rigid-body displacements via the polar decomposition”, ASME Journal of Mechanical Design, 129(8), pp. 883–886.
[22] Purwar, A. and Ge, Q. J., 2013, “Polar Decomposition of Unit Dual Quaternions”, ASME Journal of Mechanisms and Robotics, 5(3), pp. 031001–031001, 10.1115/1.4024236.
[23] Piegl, L. and Tiller, W., 1995, The NURBS Book, Springer, Berlin.

Appendix A

A.1 GitHub Repository

https://github.com/ahalwah/Research

A.2 Demonstration Videos for Motion Capture

https://www.youtube.com/playlist?list=PLIq8Oomy8HEhCFy-FuaNh7p72LjOI_xq-