Supplementary Method S1. Creation of a prism-based forward kinematic animation rig

Whereas the construction of digital marionettes for both ball and socket (hip) joints followed the procedure outlined by Manafzadeh & Padian (2018), here we developed a new method for creating digital marionettes of hinge (knee and ankle) joints using Maya 2020 (Autodesk, San Rafael, CA, USA). The terms below are specific to Maya but the general logic of this procedure can be translated to other software (e.g., Blender) if need be. Our method exploits the new translation measurement system outlined in this paper, allowing direct input of these translation measurements into the forward kinematic rig. An animation-expression-based input approach that allows automation of the incorporation of translations in interpenetration detection is outlined as *Supplementary Method S2*; here we first cover basic creation and operation of the rig.

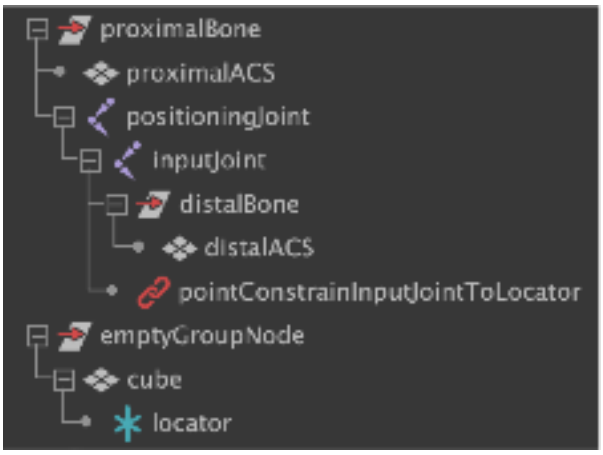
We began by constructing a separate basic digital marionette for each joint following Manafzadeh & Padian (2018). In short, beginning with a single pair of bones in their reference pose (with anatomical coordinate systems [ACSs] fully aligned such that joint coordinate system rotations and translations equal zero), a character animation joint hierarchy was created. Although Manafzadeh & Padian (2018) only explicitly refer to the creation of a single animation joint, here we suggest creating two joints, one parented to the other, with the top "positioning" joint used for positioning and orientation and the bottom "input" joint used for animation. We point and orient constrained the "positioning" joint to the proximal ACS. The constraints can be deleted if desired if the proximal bone (and thus the proximal ACS) will be held in one position; however, if the proximal bone will be repositioned for any reason, these constraints should be left intact. We then parent constrained the distal bone to the newly created "input" joint.

Next, we created a polygonal cube object of dimensions 1 × 1 × 1 cm centered at the global origin (0, 0, 0), with one division per dimension. We moved the scale and rotational pivots of this cube to a position of (-0.5, -0.5, -0.5), and then translated the cube itself to a position of (0.5, 0.5, 0.5) such that its pivot was placed at the world origin. A locator was created and placed at the vertex of this cube diagonal from its vertex at the origin (at the local coordinate [1,1,1]), and parented to the cube. The cube was then parented to an empty group node.

We point and orient constrained this group node to the proximal ACS, causing the cube to be placed with one of its vertices at the position of the proximal ACS. We then used the Maya Connection Editor to connect the Z rotation of the "input" character animation joint to the Z rotation of the cube, ensuring that the cube would rotate along with the primary hinge rotation. Finally, we point constrained the "input" joint to the cube's locator, causing the distal ACS (and rotational pivot) to be located at the cube vertex diagonal from the proximal ACS. A schematic of the resulting Outliner hierarchy from this process is shown below (Supplementary Methods Figure SM1). Note that whereas the point and orient constraints for the group node can be deleted if desired, the point constraint for the input joint **must** be left intact for proper functioning of the rig.

Under this setup, changing the X, Y, and Z scale values of the 1 x 1 x 1 cube causes it to morph into a rectangular prism, moving the distal bone along with it. The scale of this prism in each dimension reflects the corresponding translation measurement of the hinge joint under our new

convention -- for example, a prism of XYZ scale = (0.2, 0.3, 0.5) cm corresponds to hinge joint translations of XYZ = (0.2, 0.3, 0.5) cm.



Supplementary Methods Figure SM1. Example Maya hierarchy resulting from the creation of a prism-based forward kinematic rig.

Supplementary Method S2. Inclusion of translations in automatic interpenetration detection

Here we offer a protocol for automating six degree of freedom ROM analyses in Maya 2020 (Autodesk, San Rafael, CA, USA), building from the foundation laid for three degree of freedom ROM analyses by Manafzadeh & Padian (2018). The terms and code syntax below are specific to Maya but the general logic of this procedure can be translated to other software (e.g., Blender) if need be. First, create a forward kinematic rig following Manafzadeh & Padian (2018) or our new prism-based rigging approach (*Supplementary Method S1*). Add a Boolean-type attribute called "viable" to the character animation joint.

Then, animate the joint through a systematic sample of rotational poses. The Maya Embedded Language (MEL) code to accomplish this was originally presented by Manafzadeh & Padian (2018). We reproduce it here for sampling of a full joint pose space at 5 degree resolution, but the bounds on each degree of freedom and the sampling resolution can easily be modified. Edit and paste the following code into the Script Editor and run once. Red text represents regions of code that are study-specific and must be edited: the name of the character animation joint used in the forward kinematic rig (*jointName*).

```
{
int $i;
int $j;
int $k;
int $frame = 1;
currentTime -edit 1;

//Z rotation ranges and sampling resolution can be adjusted here
for ($i = -180; $i < 181; $i = $i + 5)
        {
```

```
        //Y rotation ranges and sampling resolution can be adjusted here
        for ($j = -90; $j < 91; $j = $j + 5)
                {
                //X rotation ranges and sampling resolution can be adjusted here
                for ($k = -180; $k < 181; $k = $k + 5)
                        {
                        setKeyframe -at rotateZ -v $i -t $frame jointName;
                        setKeyframe -at rotateY -v $j -t $frame jointName;
                        setKeyframe -at rotateX -v $k -t $frame jointName;
                        $frame = $frame + 1;
                        }
                }
        }
}
```

Next, select the two bone mesh models and run a boolean intersection operation. Our goal will be to query the surface area of the resulting boolean mesh at each rotational pose. If the boolean mesh has a surface area of zero, the bone mesh models do not interpenetrate and the pose is viable. However, if the boolean mesh has a surface area of greater than zero, the bone mesh models interpenetrate and the pose is not viable. **To ensure reliable boolean behavior, make sure to clean polygonal meshes before running; all objects should be manifold surfaces, face normals should be uniform, and co-planar or self-intersecting faces must be removed.**

The following MEL animation expression code replaces Manafzadeh & Padian's animation expression for querying boolean surface area and keyframing pose viability. In contrast to the original expression, this approach now allows the researcher to input three float arrays containing values to test in each translational degree of freedom. This expression will evaluate at each frame of the animation (i.e., each rotational pose) when it is played. As the expression loops through the float arrays provided by the user, the joint will assume the corresponding translational values. The expression evaluates the interpenetration of bone mesh models in each of these configurations. Once any single combination of translations results in a boolean mesh surface area of zero, the loop is broken, the pose is marked as viable, and the animation proceeds to the next frame. By breaking the loop at this point, simulation redundancy is reduced and overall run time is decreased. (If the research question at hand requires knowledge of which specific translation combinations yielded a viable outcome, the code provided here will need to be modified.) Under this approach, scene playback can be paused at any time to save results along the way. Note that because translation values are controlled by the animation expression at each frame, the animation does not need to be baked before running the expression (*contra* Manafzadeh & Padian [2018]).

We suggest that in the future, as articulation criteria are ground-truthed and formalized, they can easily be layered onto this framework by adding lines of code to these animation expressions.

<u>If using a prism-based forward kinematic rig (see *Supplementary Method S1*):</u> Modify and paste the following code into the Expression Editor, click "Edit," and then play the animation to automatically test pose viability. The viability (1) or inviability (0) of each pose will be keyframed at each frame of the animation as it plays. Red text represents regions of code that are study-specific and must be edited: translation values to test, the name of the character animation joint

(*jointName*), the name of the prism-based rigging cube (*cubeName*), and the name of the boolean mesh object (*booleanName*). Note that to use this approach, a boolean-type attribute called "viable" must have been added to the character animation joint. To test only a single translation value in a given degree of freedom, simply enter only one value into its array rather than a comma-separated list.

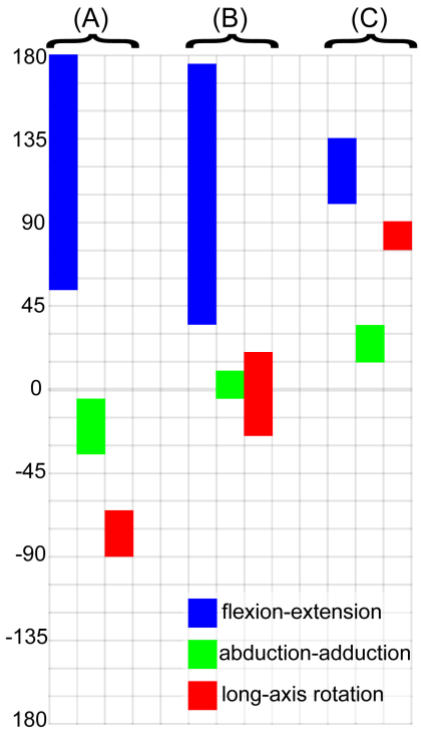
```
//define translation values to be tested
float $xTransArray[] ={insert comma-separated X translation values here};
float $yTransArray[] ={insert comma-separated Y translation values here};
float $zTransArray[] ={insert comma-separated Z translation values here};

//set pose viability to 0 (inviable) to start
int $viable = 0;
setKeyframe -at viable -v $viable jointName;

//loop through translation combinations and mark pose as viable (1) as soon as any translation combination results in a boolean mesh surface area of 0
int $i;
int $j;
int $k;

int $xSize = size($xTransArray);
int $ySize = size($yTransArray);
int $zSize = size($zTransArray);

for ($i=0;$i<$xSize;$i++)
	{
	for ($j=0;$j<$ySize;$j++)
		{
		for ($k=0;$k<$zSize;$k++)
			{
			if ($viable==0)
				{
				setAttr "cubeName.scaleX" $xTransArray[$i];
				setAttr "cubeName.scaleY" $yTransArray[$j];
				setAttr "cubeName.scaleZ" $zTransArray[$k];

				float $area[] = `polyEvaluate -area booleanName`;
				if ($area[0]>0)
					{
					$viable = 0;
					}
				else
					{
					$viable = 1;
					setKeyframe -at viable -v $viable jointName;
```

```
                                        }
                                    }
                                }
                            }
                        }
```

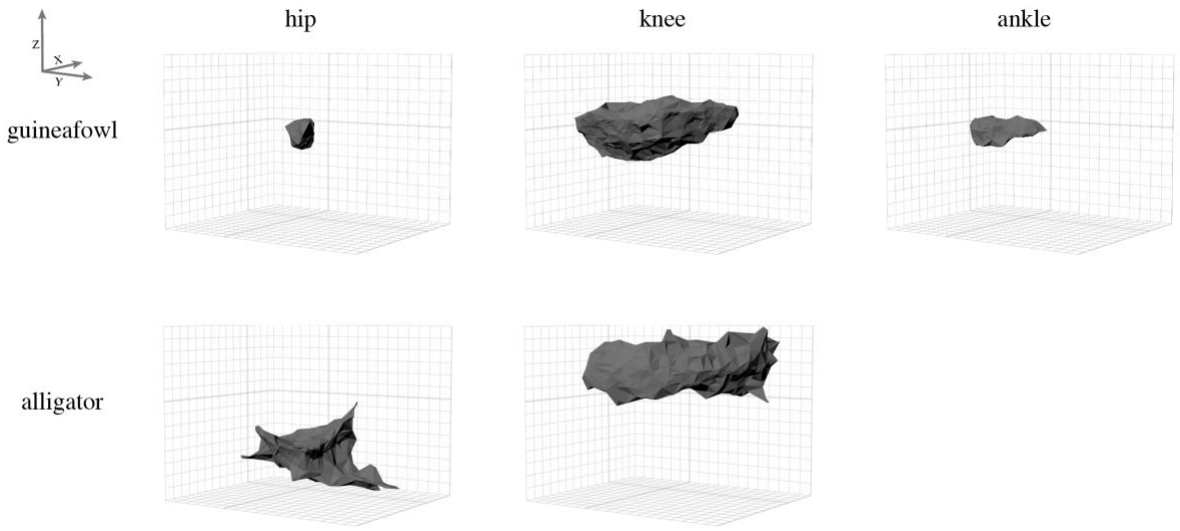
If using a non-prism-based forward kinematic rig (see Manafzadeh & Padian, 2018): Modify and paste the following code into the Expression Editor, click "Edit," and then play the animation to automatically test pose viability. The viability (1) or inviability (0) of each pose will be keyframed at each frame of the animation as it plays. Red text represents regions of code that are study-specific and must be edited: translation values to test, the name of the character animation joint (*jointName*), and the name of the boolean mesh object (*booleanName*). Note that to use this approach, a boolean-type attribute called "viable" must have been added to the character animation joint. To test only a single translation value in a given degree of freedom, simply enter only one value into its array rather than a comma-separated list.

```
//define translation values to be tested
float $xTransArray[] ={insert comma-separated X translation values here};
float $yTransArray[] ={insert comma-separated Y translation values here};
float $zTransArray[] ={insert comma-separated Z translation values here};

//set pose viability to 0 (inviable) to start
int $viable = 0;
setKeyframe -at viable -v $viable jointName;

//loop through translation combinations and mark pose as viable (1) as soon as any translation combination results in a boolean mesh surface area of 0
int $i;
int $j;
int $k;

int $xSize = size($xTransArray);
int $ySize = size($yTransArray);
int $zSize = size($zTransArray);

for ($i=0;$i<$xSize;$i++)
        {
        for ($j=0;$j<$ySize;$j++)
                {
                for ($k=0;$k<$zSize;$k++)
                        {
                        if ($viable==0)
                                {
                                setAttr "jointName.translateX" $xTransArray[$i];
                                setAttr "jointName.translateY" $yTransArray[$j];
                                setAttr "jointName.translateZ" $zTransArray[$k];
```

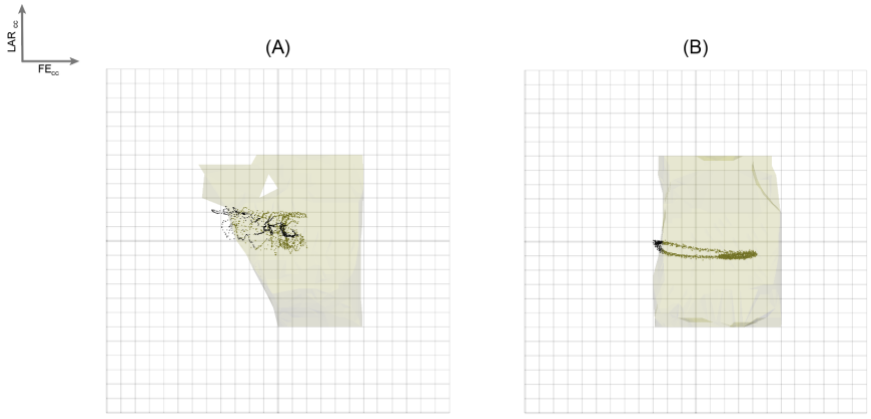
```
                    float $area[] = `polyEvaluate -area booleanName`;
                    if ($area[0]>0)
                        {
                        $viable = 0;
                        }
                    else
                        {
                        $viable = 1;
                        setKeyframe -at viable -v $viable jointName;
                        }
                    }
                }
            }
}
```



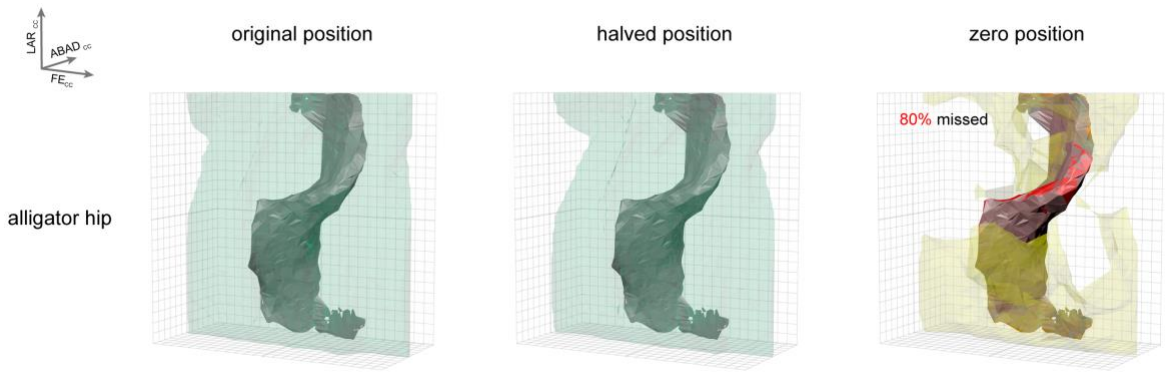
Supplementary Figure S1. Selection of a rotational starting, 'neutral,' or 'reference' pose strongly influences single-axis measurements of mobility. Estimating guineafowl knee ROM from example starting poses of (flexion-extension, abduction-adduction, long-axis rotation) degrees = (A) (60, -10, -80), (B) (90, 5, 0), and (C) (120, 35, 75), without allowing combinations of motions in all three rotational degrees of freedom, results in three very different measurements of joint mobility.



Supplementary Figure S2. All five joints studied demonstrate interactions among translational degrees of freedom. Simultaneous excursions measured in centimeters from all three translational degrees of freedom are plotted as alpha shapes generated from (X, Y, Z) points with an alpha radius of 0.1. Measurements for hip joints follow previous XROMM studies, whereas measurements for knee and ankle joints follow the prism-based hinge joint convention presented in this study. All values reflect right-sided conventions for positive measurements. Bolded grid lines represent 0; grid lines are 0.1 cm apart.



Supplementary Figure S3. Two examples of cases in which osteological simulations fail to capture *in vivo* locomotor poses. (A) Alligator knee no-translation simulation and (B) guineafowl ankle single-translation simulation displayed as yellow polygons with a selection of *in vivo* locomotor poses measured by Manafzadeh et al. (2021) displayed as black spheres. See Manafzadeh et al. (2021) for additional *in vivo* poses.



Supplementary Figure S4. Sensitivity of alligator hip no-translation osteological simulation to placement of rotational pivot. No-translation simulation results are displayed for the rotational pivot in its original position (middle of the cadaveric distraction-compression range), in a position with half the original excursion in distraction-compression translation, and in a position with zero translation (placed based on the spheres used to create the joint coordinate system [see Kambic et al., 2014]). See also *Supplementary Movie S7*.

Supplementary Figure S5. Guineafowl ankle all-translation osteological simulation result with additional joint extension allowed. All-translation simulation results are displayed for the original flexion-extension range of [0,180] (corresponding to cosine-corrected flexion-extension [FE_{CC}] of [-90,90]) and for an expanded flexion-extension range of [0,185] (corresponding to FE_{CC} of [-90,95]). See also *Supplementary Movie S8*.

Supplementary Movie Captions

Supplementary Movie S1. Example series of motions in all six degrees of freedom at the guineafowl knee (hinge) joint, demonstrating our new prism-based translation measurement convention. See also *Figure 2C-D*.

Supplementary Movie S2. Alligator hip cadaveric, no-translation, single-translation, and all-translation alpha shapes, rotated. See also *Figure 3A*.

Supplementary Movie S3. Guineafowl hip cadaveric, no-translation, single-translation, and all-translation alpha shapes, rotated. See also *Figure 3B*.

Supplementary Movie S4. Guineafowl knee cadaveric, no-translation, single-translation, and all-translation alpha shapes, rotated. See also *Figure 3C*.

Supplementary Movie S5. Guineafowl ankle cadaveric, no-translation, single-translation, and all-translation alpha shapes, rotated. See also *Figure 3D*.

Supplementary Movie S6. Alligator knee cadaveric, no-translation, single-translation, and all-translation alpha shapes, rotated. See also *Figure 3E*.

Supplementary Movie S7. Alligator hip sensitivity analysis alpha shapes, rotated. See also *Supplementary Figure S4*.

Supplementary Movie S8. Guineafowl ankle original and expanded all-translation alpha shapes, rotated. See also *Supplementary Figure S5*.