Expressive, Interactive Robots: Tools, Techniques, and Insights based on Collaborations

Jesse Gray, Guy Hoffman, Sigurdur Orn Adalgeirsson, Matt Berlin, and Cynthia Breazeal
Robotic Life Group
MIT Media Laboratory
20 Ames Street E15-468
Cambridge, MA 02139
Email: {jg,guy,siggi,mattb,cynthiab}@media.mit.edu

***Abstract*—In our experience, a robot designer, behavior architect, and animator must work closely together to create an interactive robot with expressive, dynamic behavior. This paper describes lessons learned from these collaborations, as well as a set of tools and techniques developed to help facilitate the collaboration. The guiding principles of these tools and techniques are to allow each collaborator maximum flexibility with their role and shield them from distracting complexities, while facilitating the integration of their efforts, propagating important constraints to all parties, and minimizing redundant or automatable tasks. We focus on three areas: (1) how the animator shares their creations with the behavior architect, (2) how the behavior architect integrates artistic content into dynamic behavior, and (3) how that behavior is performed on the physical robot.**

I. Introduction

Creating a robot to expressively interact with humans poses a novel set of challenges: designing a physical robot capable of compelling motion; animating expressive physical motion for that complex interactive robot; and combining this expressive motion with the robot's functional control to produce interactive behavior.

Traditionally, motion generation for robots has fallen into one of two extremes. On the one hand, mobile and industrial robots have been controlled by strictly *functional* approaches, such as Inverse Kinematics (IK) and self-collision avoidance. Some of these motion generation systems have been transferred to control interactive robots, a decision resulting in stiff, unnatural, and often slow or clumsy motion. On the other hand, animatronic robots have been *scripted* using a variety of input techniques, including direct motor commands from a sequencer or even a 3D animation tool. These techniques either constrain the animator to animate in the unintuitive space defined by the physical motor layout, or constrain the robot design to follow restrictions of the 3D animation tool, or both. Moreover, just as functional control lacks the expressiveness of authored gestures, pre-scripted animatronic control can be devoid of a functional relation to the robot's surroundings, prohibiting the adaptive and reactive behavior required of robots designed for human interaction.

Human-interactive robots must combine these approaches, and embrace the reality that experts with different backgrounds must collaborate together to make these robots possible. Like animatronic robots, they need to move in a reactive and life-like manner, employing gestures and nonverbal behavior fit for human interaction. But due to their existence in real-world environments, they must also relate to their environment and interact functionally with the world around them.

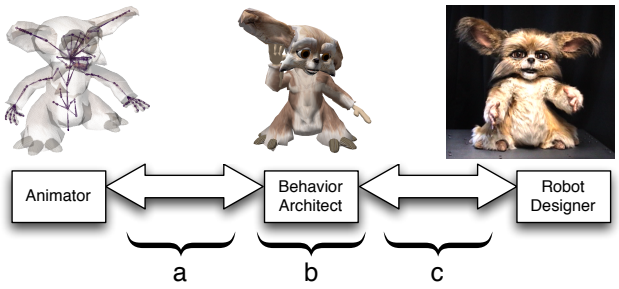

Fig. 1. Experts from three fields collaborate to produce an interactive robot with expressive behavior. This paper covers lessons learned and tools developed from our experiences with these collaborations, focusing on the parts shown: *a)* interface between animator and behavior architect, *b)* mechanisms to produce dynamic behavior using animated content, and *c)* the interface between behavior architect and physical robot.

To produce such a robot requires collaboration between people with a wide range of areas of expertise. This paper describes our own method for breaking down this process among experts, and the lessons we have learned about how to allow them to best work together to produce a compelling, expressive system.

The roles we will describe here are the animator, behavior architect, and designer of the physical robot. Our motivation is to enable each of these collaborators to have the most flexibility in their work and to limit the constraints placed on them to ones that productively define the capabilities of the system. We want the animator to be able to use the tool they are most familiar with. We want the behavior architect to have access to any mechanisms helpful for creating dynamic, interactive behavior. We want the designer of the physical robot to employ any complex mechanisms that are necessary, free of worry that complicated mechanisms will frustrate the animator or behavior architect. The goal is not to hide the limitations of the robotic system from the other collaborators - quite the opposite, we believe making these limits as visible

as possible will aid the creation of the best, usable content. We do, however, seek to shield each from complexities of implementation and control that would complicate their work without benefit. This interaction is pictured in figure 1.

Across all of the mechanisms, we have a consistent set of goals:

- *Best Tool/Technique for the Job*: Allow each collaborator to employ whatever tools/techniques they need. Excellent animation tools exist, and we should be able to leverage these tools and the artist's familiarity with them. The behavior architect should be provided with the best mechanisms possible to mix, blend, and combine animations to get the behavior they want. The robot designer should build the robot without constraining their creativity based on a certain control structure.
- *Playback Consistency*: The correlation between the robot motion as viewed in the animation authoring tools, the tools used by the behavior architect, and the performance on the actual robot should be clear and predictable.
- *Manage complexity*: Each collaborator should have access to as many useful constraints, data, and meta-information as possible, but not be burdened with arbitrary complexities.
- *Safety*: Animation/behaviors should be safe when played out on the robot. The authoring tools and—more critically—the execution systems should take into account the robot's physical limits: self-collision, joint limits, cable limits, workspace collision, and safe velocity and acceleration bounds.
- *Scalability*: Provide scalability (both in allowing high degrees-of-freedom robots, and in the capability to transfer the system between robots) by automating processes and facilitating the sharing of information/data among collaborators.

Covering our approaches to these requirements brings us through sections III to V, where we follow, step by step, the progression the animated content takes on the way to the robot: starting with the animation tool, making its way into the behavior engine to be re-mixed and blended, then finally off to be transformed into data necessary for the physical robot.

The system we will describe here is not the only way to accomplish these goals, and there is always room for improvement (see section VI). However, we feel we have assembled a set of tools/techniques that hits an important "sweet spot", greatly advancing possible collaboration between people in these three roles.

In section III, we describe the interface between the animator and behavior architect, which is designed to eliminate any redundant setup work, allow them to share an intuitive view of the joints of the robot, and allow the animator to prototype/author animations while also providing appropriate behavioral hints.

In section IV, we describe the tools available to the behavior architect which relate to authoring motor behaviors through combining animation data in different ways.

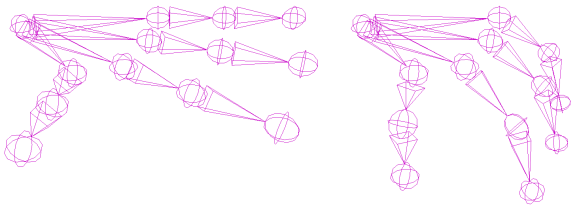

Fig. 2. In order to achieve an evenly bending finger, the model contains multiple joints linked down the center of the finger. These joints bend simultaneously causing the finger to curve.

In section V, we describe the interface between the behavior architect and the physical robot, which is designed to abstract complex linkages, real-time concerns, and calibration issues away from the day to day work of the behavior architect.

II. Physical Platform

While these collaboration techniques were first attempted with a 13 Degree of Freedom (DoF) robot called "Public Anemone" [1], the first robot to push the development of much of the automation and abstraction was the much more complicated 65 DoF Leonardo [2]. These tools have also been used in robotic projects such as Aida, Aur [3], the Huggable project [4], the Operabots project, and Nexi.

III. Connecting Animator to Behavior Designer

In this section, we describe insights and techniques based on our work collaborating with animators to create expressive yet interactive behavior for robots. At the minimum, it is necessary to share 3D models and animations between the animator and the behavior architect. However, we have found that sharing additional information such as simplified DoF abstractions, joint constraints, and meta information about the animations enhances the collaboration, without increasing our commitment to a particular animation software package.

A. Abstract File Formats

To create a clean interface to any authoring tool a professional animator might want to use, we created file formats for representing 3D models and animations. The only restriction on authoring tools that can be integrated into our pipeline is that they provide plug-in capability to access and export the 3D models and animations. This provides the flexibility to switch to new authoring tools as they become available. Currently, such plug-ins have been written for Maya and 3D Studio Max.

B. Abstracting DoFs from Skeleton

We use the "skeleton" modeling technique, where a robot is represented as a hierarchical skeleton of joints connected by bones, with the visible surfaces of the 3D bot driven by the motion of these underlying joints. Because of the way skeleton modeling functions, an animator might be forced to model certain DoFs in a fairly complicated way, e.g. figure 2 and 3. These methods both use multiple joints in the animation tool to model what the animator and behavior architect would

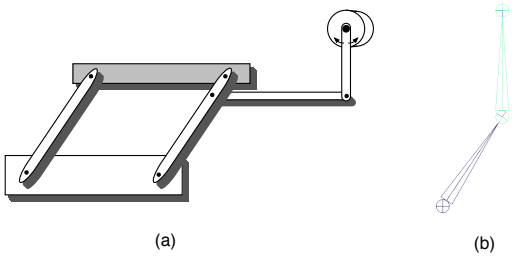

Fig. 3. (a) In a four-bar linkage the effector (shaded) stays parallel through its motion. (b) This is approximated in the 3D authoring environment by using two joints, where the child joint is programmed to compensate for the rotation of the parent.

prefer to think of as a single DoF. Luckily, the animation tools all include mechanisms for the animator to make an interface to move those joints as a single element.

Unfortunately, this wouldn't help our behavior architect, because whatever interface is added in the animation tool to facilitate this process won't exist in the raw 3D model exported to the behavior engine, and so any procedural moving of the DoF in question would involve keeping track of all its component parts.

We want the same simplified controls the animator created in the authoring tool to become available for manipulation of degrees of freedom programatically by the behavior architect. Since the animator has already done the work of defining these controls, instead of having our behavior architect redefine them we can export this information from the animation authoring tool.

There are many different ways the animator might accomplish tying multiple joints into one scalar control. We wish to remain agnostic to the specifics of the animation tool and to the method used by the animator to tie the joints' motions together. So, instead of attempting to process and export the animator's custom interfaces directly, we resort to using a "calibration animation."

C. Calibration Animation

It is important for the animator's model to include the correct axes of rotation and joint limits for the joints of the robot. The animation authoring tools tend to have a good UI for manipulating these joint parameters, so we use the animator's 3D model of the robot as the canonical repository of this information. Keeping the animator's model as the canonical repository of this information ensures that the animator has access to all the known information about joint restrictions, decreasing the chance of creating animations that will not run correctly on the robot.

However, these parameters are also required for the operation of the behavior engine (and we wish to avoid error-prone manual replication of information). In an effort to stay agnostic towards any specific authoring tools, our architecture uses a calibration animation to obtain specific attributes about the robot's configuration instead of deeply inspecting the 3D model within the animation tool. The authoring tool simply

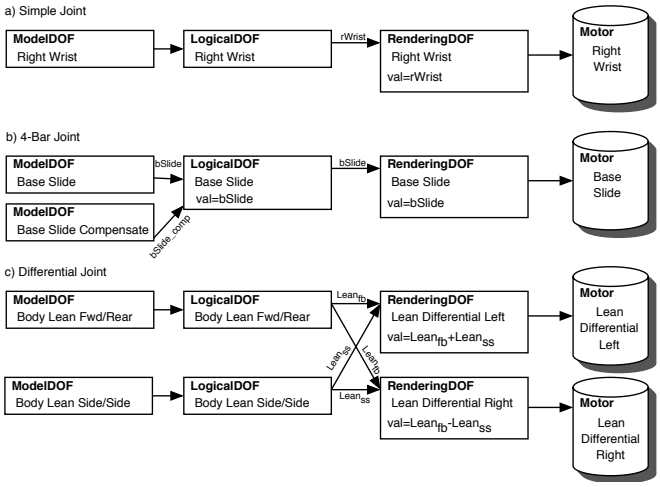

Fig. 4. `LogicalDoF`s interface between the—sometimes unintuitive—3D model of a degree of freedom and the—also possibly unintuitive—mechanical linkage that moves that DoF, providing a simple scalar value controlling a "logical" degree of freedom.

outputs an animation where every DoF is moved individually to its limits (which for some could represent movement of multiple model joints, if the animator has set up a complex DoF as described in the previous section). Our system reads this calibration animation and uses the motion contained within to define the joint axes and joint limits, as well as to discover which joints are tied together as one DoF (and in those cases, figure out the mapping of how they are correlated).

D. Logical DoF Representation

Once the behavior system is able to pull DoF information out of the calibration animation, it can store all this information in a *LogicalDoF*. The role of the *LogicalDoF* is to store all the relevant information that the animation tool had about the DoF (including limits, axes, and any DoF simplification controls added by the animator to abstract multiple joints as one DoF), and present it as an intuitive interface to the behavior architect. In this way, instead of being faced with potentially messy joint setups necessitated by skeleton modeling, the behavior architect is presented with a similar interface to the one the animator had created for themselves: a single scalar value per DoF. This process is shown in the left-hand-side of figure 4 (the right hand side will be covered in section V).

E. Animating Joint Priorities

Our robots frequently have to perform multiple different motions at the same time: for example, a robot might be running an animation that extends its right arm for a handshake while maintaining eye-contact with a person. In this example, one part of the robot is controlled via animation while the other is controlled using functional control with sensor feedback. In many cases these situations are handled by the blending systems in section IV without intervention of the animator.

However, we find that since our robots are employing a procedural orient behavior at almost all times, it can be helpful

to provide the animator a mechanism to specify when certain joints normally overridden for orienting the robot are required for the expressive purpose of the animation.

For example, if an animation includes an "eye roll", it is imperative that at that moment the animation have full control over the eyes (the robot must momentary cease any eye orientation behavior it is performing). To address this need, we provide a mechanism for the animator to specify the importance of certain DoFs to the success of the gesture. This mechanism is implemented as a set of special, invisible joints whose value, instead of indicating a rotation or translation, indicates the animation's desire for full control over a particular DoF. This implementation allows the animator to vary the ownership of a DoF over the duration of an animation, so the joint is only seized for the short time it is required. Also, this strategy means that the ownership data will be automatically included in any exported animation file without any authoring tool modifications.

IV. Tools For the Behavior Architect

The role of the behavior architect, as it fits into the structure we are proposing here, is to create the system which will drive the real-time behavior of the robot. This could take many forms, with varying levels of autonomy, but here we're focusing in particular on how the work of the animator can be used to create expressive behavior for the robot, while allowing for the flexibility of control required for an interactive robot. This section covers the common types of motion we have had our robots perform, and the tools we provide the architect to accomplish these motions utilizing the animations from the animator. Many of these techniques were developed for graphical characters, and are adapted from [5].

A. Kinds of Motion

We have found that a robot interacting with a human counterpart needs to be able to move in four distinct ways:

1) Gestural: First, a robot is expected to express its internal state through understandable social gestures and communicate ideas in verbal and non-verbal ways. This calls for a system that enables an animator to author natural looking *gestures* and behaviors to be played out on the robot. These motions are typically iconic gestures like thumbs-up, shoulder shrug, nod, and eye-roll.

2) Functional: As a physically embodied agent, the robot needs to be able to engage in *functional* motion relating to objects in the robot's workspace and human counterparts. Touching and manipulating objects and IK gaze fixation are examples of such motion.

3) Procedural Expressive: A third motion requirement may be called *procedural expressive* motion. These are motions that are mostly expressive in their function, i.e. not related to an external object, but are too variable to be authored as complete fixed gestures. An intermittent blink behavior, an ear twitch in response to a new sound, and an overall body posture indicating an emotional state are some examples of procedural expressive motion.

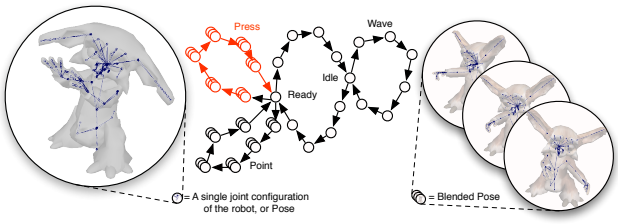

Fig. 5. Posegraph representation of imported animation data. Each node either represents a single pose of the robot, or a set of example poses that can be blended together at runtime based on input parameters.

4) Parameterized Gestures: Finally, *parameterized gestures* are required when an action may be performed in a number of ways, and it is desirable to form a continuous, parameterized space of actions rather than rely on choosing amongst a few discrete examples. We have used this method, for example, to create a continuous space of pointing gestures which allow the robot to point to an arbitrary location while maintaining the expressiveness of the original, discrete set of hand animated motions.

B. Realizing these Kinds of Motion

This section covers mechanisms to produce each type of the above motions individually. However, to create interesting behaviors, several motions may need to occur simultaneously which will be explained in the next section.

Our basic representation for positioning the robot is through a posegraph [6] (figure 5). In this graph, each node represents a pose of the robot, and the (directed) edges represent allowable transitions between poses. An animation, then, is loaded into this representation as a series of connected poses.

Gestures can be performed by simply traversing the appropriate series of poses in the posegraph. The connections between an animation and the rest of the posegraph are accomplished manually - this gives the behavior architect control of what transitions between animations are allowed based on their appropriateness to the behavior of the creature.

Parameterized gestures are created by authoring multiple versions of the same gesture, and blending them together based on a set of parameters provided in real-time (in the manner of verb/adverb actions [7]). This blending happens within the nodes of the posegraph. These blended nodes, instead of representing a single static pose, each contain multiple poses that define a blend space. Whenever a blended node is traversed, external parameters indicate a position in this blend space, and the node reads these parameters to produce the resulting blended pose.

Functional motion can be determined by direct IK calculations given the kinematics of the robot and the goals of the action. This often results in undesirably robotic motion, so to realize a functional goal (such as touching an object) we combine the IK calculation with *parameterized gestures*. These gestures are used to get as close as possible to a goal condition, then IK can take over to fulfill the goal with minimal

disruption to the expressive behavior of the robot.

Procedural expressive motion, while it describes different motion scenarios than the above categories, can be implemented as special cases of the above techniques. Many situations in this category can be described as a need to blend immediately between two static poses, for example, a blinking eye, or a slight "perk up" in response to an audio signal. These cases can be seen as a trivial case of *parameterized gestures*, where the change in parameter controls the motion and the example animations themselves contain no motion. For example, a blend parameter might control the blend space from example animation "eyes open" to example animation "eye closed", allowing for procedural blinking.

C. Combining Simultaneous Motions

The above section covered each category of desired motions individually. In general, however, the robot's behavior will call for executing a number of motions simultaneously. We have found that there are several ways in which we find ourselves routinely combining different motor behaviors.

1) Multiple Gestures: In a simple posegraph, the robot has one "play-head" which represents its current position as it traverses the graph. This can be limiting, because the robot may well wish to perform two gestures (parameterized or simple) simultaneously on different body parts (e.g., nodding while pointing). For this reason, we allow multiple "play-heads" to simultaneously traverse the posegraph. However, each gesture, or path through the graph, will have an associated set of preferences over what joints it requires. This allows the robot to play two compatible gestures simultaneously, and prevents it from initiating a gesture which requires joints that are currently in use. The set of required joints can be specified manually, however usually it can be assumed to be the set of joints that move during that animation.

2) Postural Overlay: As opposed to the gestures above, a postural overlay is an animation (or blended space of animations) which is designed to be applied to the DoFs of the robot all the time, even when gestures are happening. The posture often is used to reflect the emotional state of the robot. Just as with "multiple gestures" blending, a new "play-head" is required which will play this overlay animation at the same time as gestures or other activity is taking place. However, instead of taking full control over specific DoFs, the overlay is designed to be applied with a very light weight to all the DoFs of the robot, thereby slightly changing any gesture the robot performs. This is done by simply blending the postural animation with the gesture, using a very light blend weight. Gestures also have the option of locking out this postural overlay for specific joints, if their absolute position is critical (e.g., when reaching for an object, ancestors of the robot's hand must maintain their exact position).

3) Idle Overlay: An idle overlay can give the creature an appearance of life even when it is not actively executing a gesture. This type of overlay keeps track of which joints have not been claimed by any active gestures, and applies the current pose from an idle animation to those joints. We typically use an idle animation which simulates gentle breathing motions.

4) Procedural Overlay: Finally, any desired procedural overlay can be applied. One type of procedural overlay is the IK system for fine control of the robot's hand position. However, our most used procedural overlay is the orient system which the robot uses to look at a target. This system determines which joints it currently has access to based on the preference set of the current gestures, and then uses the joints it can access to orient the eyes and body as best it can towards the target. Because the robot is constantly looking around, we provide a special channel here to give the animator direct control over this behavior (section III). This allows the animator to control this important aspect of the animation's performance; instead of animating just the position of the eyes, they can also animate the transitions between procedural and animated content for them.

D. Types of Blending

There is an important distinction between additive and weighted-average blending, we have found it is important to provide both as options to the behavior architect. Weighted average blending is useful for combining multiple animations into a resulting animation that has aspects of all of the inputs, where each joint position will lie somewhere between the example positions.

Additive blending, on the other hand, is useful for offsetting animations so they are fully performed but from a new starting position. This is particularly useful to apply to the torso and neck areas of the look-at behavior of the robot. For example, when performing a nod and looking at a person, using additive blending the robot will perform the full nod in its current orientation. In a weighted average blend system, it would have to either fully center itself to perform the full nod (turn look-at off), or do a blend and get something halfway between a centered nod and an unmoving "look to the right" pose (a half-height nod oriented half-way to target).

V. Connection to Physical Robot

Just as section III discussed 3D model implementation issues that produce unnecessary complications, the physical mechanisms of the robot can also give rise to certain complexities that could burden the behavior architect or the animator. In this section, we describe some of the requirements of controlling a physical robot, and how we use the interface between the behavior system and the robot to model these complexities and shield the behavior architect and animator from needing to consider them in their daily work. This means that the robot designer need not be constrained by the preferred tools of the other collaborators, nor are the other collaborators inconvenienced by the complexities introduced by the robot designer.

A. Model / Motor Discrepancies

The most common DoF is a single motor controlling a single rotation, mirroring the simple representation in the

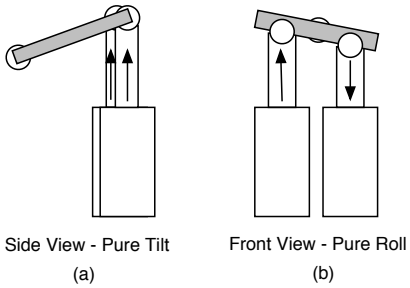

Fig. 6. A differential linkage prohibits mapping a single motor to a single DoF's movement. Both motors need to move in symmetry to achieve tilt motion (a) and in anti-symmetry to roll (b).

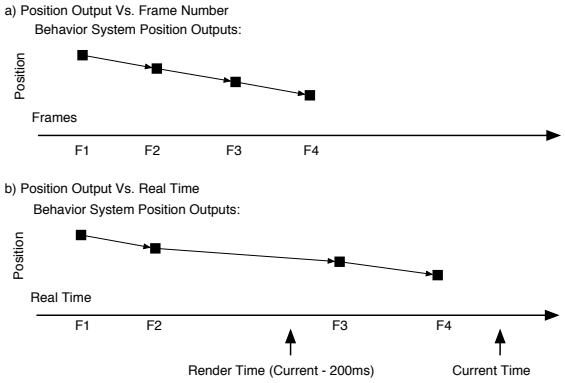

Fig. 7. A) shows the position output of a DoF from the behavior system plotted against frame number. We can see that there the DoF has a constant velocity B) shows the position output of a DoF from the behavior system plotted against real world time. We can see that although the DoF has constant velocity with respect to frame number, the real DoF velocity changes when there are fluctuations in duration of a frame.

behavior generation system. However, other linkages are more complicated.

There are many interesting linkages that occur throughout the robots discussed here, but a good example is a differential linkage (Fig. 6), which is often used to control two DoFs in a torso or neck where movement both forward/back and side-to-side is required. As shown in the figure, the *LogicalDoF*s of forward/back and side-to-side do not map cleanly onto individual motors - each direction requires motion from both of the motors. Explicitly representing the motors of the differential is not useful, and is somewhat confusing, for both the animator and the behavior architect. For this reason, we have the *RenderingDoFs* on the right hand side of figure 4. Each physical motor has a corresponding *RenderingDoF*, and its job is to acquire and transform data from one or more *LogicalDoFs* into the form needed by that motor.

Thus, in the behavior system, the data is represented in its most intuitive "logical" format, with one *LogicalDoF* for each of the logical degrees of freedom of the robot. The *RenderingDoF*s serve as the mechanism to transform the data into the format used by the actual motors of the robot. These complete the three stage system from figure 4, where each stage serves to abstract unnecessary complexities away from the collaborators.

B. Real-time Control

Updating the target position for a motor must happen at a regular, high frequency to produce smooth motion. Updating as slow as 30hz, a reasonable update rate for computer graphics, can introduce visible and audible jittering in a motor's performance. However, it is undesirable to insist on a precise, 60hz or greater update from the behavior engine. Depending on the interaction, it may have a large amount of processing to do, which could put a cap on its maximum frame-rate. Further, without a real-time operating system there is no guarantee that the updates will come at precise intervals.

We address this problem with a "Motor Rendering" layer that buffers data from the behavior engine. This layer introduces a 200 ms delay with its buffer, but it allows the data to be read at whatever frame-rate the motors require by upsampling with spline interpolation.

An additional problem we face with our system is based on an internal assumption that the time between each update is precisely 1/30th of a second. Using virtual-time in the system greatly simplifies certain calculations (as well as aiding in debugging), but it can introduce velocity discontinuities if the updates happen somewhat irregularly on a taxed computer (figure 7). To allow the behavior architect to continue to work in this simplified virtual-time, but to preserve joint velocities, we introduce the *timewarp renderer*.

The *timewarp renderer* takes in position samples from the behavior engine and places them in the interpolation buffer (just as described above). However, instead of placing them in the buffer at the current time, it places them at even 1/30th of a second intervals (despite the fact that they arrive irregularly). Then, by the time the read head travels through that buffer pulling out upsampled data for the motors, the behavior engine's samples are neatly arranged as if they came in at precisely 1/30th of a second intervals, and constant velocities are preserved.

The timewarp renderer must perform one more critical step: keep the read head from catching up to the write head (underbuffer), or falling too far behind (overflow), as would happen if the average speed of the behavior engine changes from the declared 30hz. This can be achieved by applying a scaling factor to the $\Delta real - time$ value used to advance the read-head between reads. This factor can be used to keep the buffer close to a desired size, but care should be taken to keep this value well filtered - if it changes too quickly, it creates the same velocity discontinuities we are trying to prevent.

C. Model/Robot Calibration

We have found it crucial to maintain an animated model that is as true to the physical incorporation of the robot as possible, in *structure*, *dimension*, and *movement*. While this seems obvious, we stress that in order to bridge the inherent differences between the virtual model and the robot, we have gone to great lengths to create a feasible mapping between the

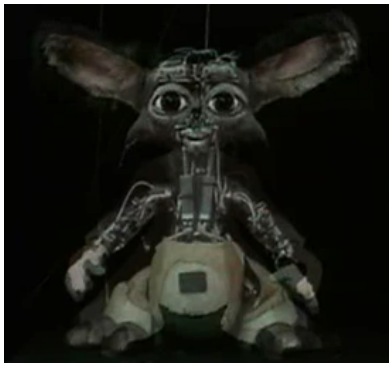

Fig. 8. Transparent rendering of 3D model of robot overlayed on live video feed. Useful for calibration.

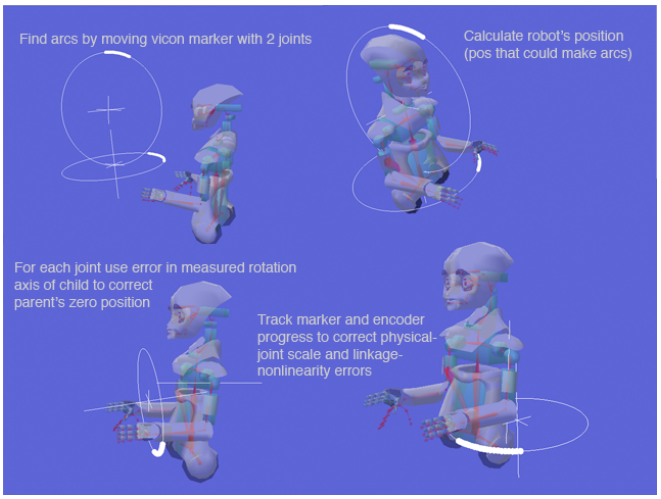

Fig. 9. Joint mappings and zeros calibrated by optically tracking a single marker.

euclidean joints representing the virtual model and the various physical controls that drive the robot.

In the past, we have found that a poorly matching model results in a severely hampered workflow, due to the misrepresentation in software of the actual result of the robot's physical motion. This violated not only the *playback consistency* authoring requirement but also affected safety calculations, as it was hard to evaluate from the virtual models when the robot would self-collide or reach other physical limits. In addition, a poorly calibrated model results in a highly iterative authoring process, requiring manual adjustment of uncalibrated animations until the desired physical result is reached. This also imposes unnecessary wear and tear on the robot.

The minimal parameters needed for calibration are offset (offset from zero position in the 3D model to the zero position on the physical robot's encoder) and scale (encoder ticks per radian). For complex linkages, the relation of radians traversed at the end effector to encoder ticks traveled on the rotation sensor may be nonlinear - in this case, a linear scale may not be sufficient. For many joints, a combination of calculation (eg. known encoder/gearbox parameters) and observation (visually lining up zero positions) may be enough. However, this can be become tiresome for a robot with many DoFs, or it may not provide the necessary accuracy.

1) Video Calibration: For joints that cannot be clearly described in terms of radians (e.g., a paddle that moves skin on a robot's cheek), visual scale and offset calibration are required. One technique that can facilitate this process is a video overlay (see figure 8). In this strategy, the camera parameters of the virtual camera are aligned with those of a real camera, and the two images are overlaid using transparency. This allows for straightforward tuning of calibration parameters with less reliance on subjective assessments.

2) Motion Capture based Calibration: For joints that can be clearly described in terms of rotation, optical motion capture can be used to automatically calibrate offset and scale (or a more complicated non-linear relation, represented as an interpolated map of example correspondence points). To minimize requirements on the physical robot, our technique requires a single trackable marker to be placed on an effector, and no access is required at pivots or joint axes. (See figure 9).

First, the two joints closest to the model's hierarchical root are used to precisely calculate the position of the physical robot (allowing the 3D model to be aligned for the next steps). This can be done by rotating these two joints through their range, with a marker mounted somewhere on their descendants. This will create two arcs that will define a single valid position for the robot.

For each joint that needs calibration, rotating the joint through its range will cause the marker, mounted on the final end effector, to traverse through a set of points. Those points will define a circle, and if each point is recorded along with the encoder reading at that time, the radians traversed can be matched against encoder ticks traveled, and these can generate an encoders-to-radians scale, or even a non-linear mapping useful for that joint's calibration.

Further, the line passing through the center of the circle (perpendicular to its plane) is the axis of rotation for that joint. The difference between this observed axis and the expected axis (calculated from the 3D model) indicates the zero offset necessary to correct the zero position of the joint's parent (assuming grandparents and further ancestors are already calibrated)

D. Flexibility

Robots and their behaviors in the context of HRI research can often be a moving target. As projects evolve requirements can change, hardware failures can occur, and mechanisms can be redesigned. We have found that maintaining an abstraction layer between the representation of the robot model used in the behavior system, and the mechanism for rendering motion data out to the physical robot through *RenderingDoF*s can be quite useful. Calibration, joint interdependencies, and non-linear linkages all are handled here, and thus many changes to the physical structure of the robot will not affect the behavior engine or animator. Of course, if there are significant changes

that actually change the morphology, they will need to be carried all the way through so that each collaborator has a correct kinematic model.

E. Safety

Experimental robots actively used for research are subject to damage through standard wear-and-tear as well as unsafe usage. Also, the possibility of causing such damage can slow development, as users of the robot will have to be extra cautious about every new change. Therefore, integrating a safety layer protecting the robot from harm not only will save time and money in robot repair but also allow the animator and behavior architect to work faster and more freely.

We currently do not have a foolproof system to prevent all kinds of damage to the robot, and this is definitely an area where we could benefit from more work to achieve our safety goals outlined in section I. However, we do employ a number of heuristics that help keep the robot safe.

1) Simulator: The first line of defense is the correspondence between the 3D model and the physical robot. This allows the animator and the behavior architect to prototype new animations and behaviors before ever sending them to the robot.

2) Constrained Generation: In some cases, the mechanism we use for *parameterized gestures* can provide a measure of safety. As opposed to IK solutions that might have unpredictable results when given incorrect input, if we are generating motions by blending within a set of example animations which define a continuous, safe space of gestures, no incorrect input can generate damaging output. For example, if we are generating a pointing motion by blending example pointing animations, flawed target parameters can at most generate extreme pointing examples, but will not be able to cause self-collision.

3) Output Sanity Check: Although we do not yet check for possible self-collisions, we do check individual joint limits, and at the lowest level each joint is prevented from moving past the extremes of its safe range. Finally, any accelerations that are out of the acceptable range can signal a fault, and thus halt the robot.

4) Self Report Watchdog: This is a high-level watchdog system, designed to detect errors that aren't caught by other, more specific checks. Each joint is queried for its current target position, as well as its current measured position (via a potentiometer or encoder). If a joint's measured position deviates from the target by more than the allowed latency and noise parameters permit, it may have encountered a hardware problem or experienced a collision. This system will generate an error message, and, depending on the configuration, disable that joint or the entire robot.

VI. Discussion

In this paper we have explained a system that was iteratively designed through many years of collaborations with artists and engineers to control at least seven different robots of different levels of interactivity and physical complexity. Our system strives to empower each participant of the collaboration as much as possible by allowing them freedom, making them aware of important constraints, and shielding them from unnecessary complexities.

A. Future Work

We have found these techniques and tools to be quite useful, and though many have seen use across a variety of robots, new challenges arise with every new project and there is always room for improvement.

Safety is an important feature for a system that enables non-roboticists to author content to be played out on delicate, one of a kind robotic platforms. Our system has several levels of checking for safety while executing animations and performing functional control of the robot but there is certainly more to be done in this area. For example, we currently don't check for self-collisions on a model level.

Full confidence in a comprehensive safety system would allow for very fast iteration and development, with less time spent double-checking new content and procedures.

Another area for improvement is for better preview tools for the animator. We currently integrate joint limits into the model, but integrating velocity and acceleration limits would eliminate another possible source of error. Even better would be to provide a preview tool that incorporated physics, so the animator could quickly view a very realistic rendition of how an animation would affect the robot.

Finally, we would like to see the animator gain the ability to, early in the authoring process, view how their animations will be later blended together (as in section IV). Each animation is currently viewed independently in the authoring tool, yet they will be combined in different ways during the robot's behavior - it might be interesting, for example, for an animator developing a postural overlay animation to be able to watch it affect existing gestures as they author it.

References

[1] C. Breazeal, A. Brooks, J. Gray, M. Hancher, J. McBean, D. Stiehl, and J. Strickon, "Interactive robot theatre," *Communications of the ACM*, vol. 46, no. 7, p. 85, 2003.

[2] C. Breazeal, A. Brooks, J. Gray, G. Hoffman, J. Lieberman, H. Lee, A. Lockerd, and D. Mulanda, "Tutelage and collaboration for humanoid robots," *International Journal of Humanoid Robotics*, vol. 1, no. 2, pp. 315–348, 2004.

[3] G. Hoffman and C. Breazeal, "Effects of anticipatory perceptual simulation on practiced human-robot tasks," *Autonomous Robots*, pp. 1–21, 2009.

[4] J. Lee, R. Toscano, W. Stiehl, and C. Breazeal, "The design of a semi-autonomous robot avatar for family communication and education," in *Robot and Human Interactive Communication, 2008. RO-MAN 2008. The 17th IEEE International Symposium on*, 2008, pp. 166–173.

[5] B. Blumberg, M. Downie, Y. Ivanov, M. Berlin, M. P. Johnson, and B. Tomlinson, "Integrated learning for interactive synthetic characters," *ACM Transactions on Graphics*, vol. 21, no. 3: Proceedings of ACM SIGGRAPH 2002, 2002.

[6] M. Downie, "Behavior, animation, and music: The music and movemnt of synthetic characters," Master's thesis, MIT, 2000.

[7] C. Rose, M. F. Cohen, and B. Bodenheimer, "Verbs and adverbs: Multidimensional motion interpolation," *IEEE Computer Graphics and Applications*, vol. 18, no. 5, pp. 32–40, 1998.