Mirror Puppeteering: Animating Toy Robots in Front of a Webcam

Ronit Slyper
Media Innovation Lab
The Interdisciplinary Center
Herzliya, Israel
rys@cs.cmu.edu

Guy Hoffman
Media Innovation Lab
The Interdisciplinary Center
Herzliya, Israel
hoffman@idc.ac.il

Ariel Shamir
Computer Science
The Interdisciplinary Center
Herzliya, Israel
arik@idc.ac.il

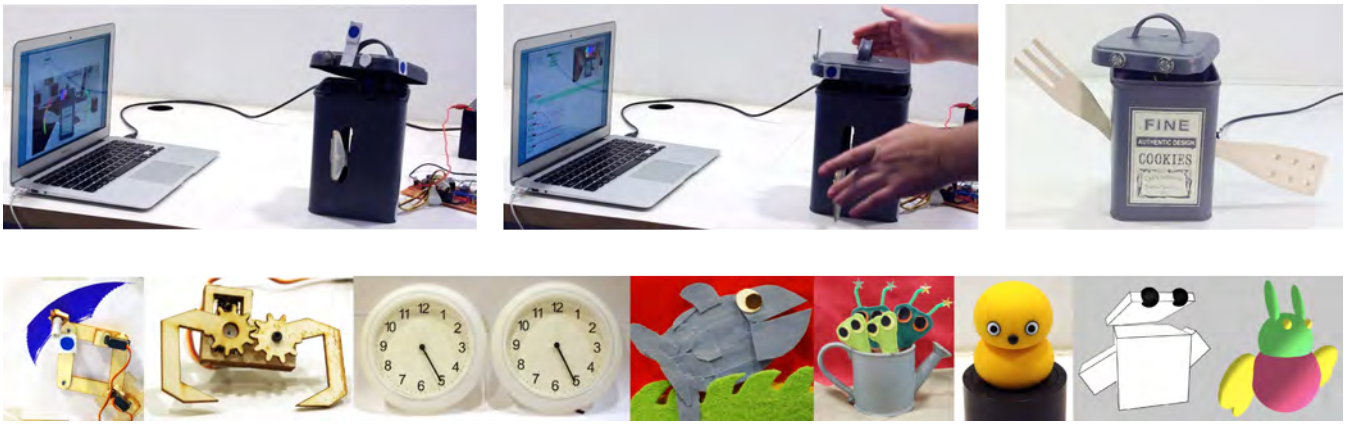

Figure 1. Mirror Puppeteering is a system for creating gestures for a robot by moving its limbs in front of a webcam. First, a robot tagged with markers self-calibrates in front of a webcam, building up a map between motor angles and camera-space marker locations (*top left*). The user then creates animations by puppeteering the robot's limbs in realtime (*top middle*). Animations are played back by the stand-alone robot (*top right*), or can be transferred to control animated characters. We demonstrate our system on a variety of homemade robots, the commercial toy My Keepon, and two virtual characters (*bottom row*).

ABSTRACT

Mirror Puppeteering is a system for easily creating gestures ("animations") for robotic toys, custom robots, and virtual characters. Lay users can record animations by simply moving a robot's limbs in front of a webcam. Makers and hobbyists can use the system to easily set up their custom-built robots for animation. Gamers and amateur animators can real-time control or save animations for virtual characters. Our system works by tracking circular markers on the robot's surface and translating these into motor commands, using a calibration map between marker locations in camera space and motor angles. New robots can be quickly set up for Mirror Puppeteering without knowledge of the robot's 3D structure, as we demonstrate on several robots. In a user study, participants found our method more enjoyable, usable, easy to learn, and successful than traditional animation methods.

TEI '15, January 16 - 19 2015, Stanford, CA, USA
Copyright 2015 ACM 978-1-4503-3305-4/15/01...$15.00
http://dx.doi.org/10.1145/2677199.2680548

Author Keywords

robotics; puppeteering; animation; gesture authoring

ACM Classification Keywords

H.5.1 Information interfaces and presentation (e.g., HCI): Multimedia Information Systems—*Animations*

INTRODUCTION

With the increasing accessibility of robot kits, microcontroller platforms, home manufacturing methods, and cheap actuators, more people are building and buying simple low-cost robots than ever before. Some of these are commercial, such as the Karotz robotic companion, My Keepon, or the Jibo family robot, while others come in the form of assembly kits, such as the CrustCrawler construction set or the Robotis Bioloid series. In addition, hobbyists and "makers" are increasingly building their own robots using 3D-printed and laser-cut parts and sharing their creations online.

A major element in the appeal of these robots is the expressivity of their movement and gestures, which need to be carefully designed to evince emotion and intention. However, designing expressive robot motion is challenging. It lies mostly in the realm of professionals, depending on meticulous programming, or techniques from 3D animation, which come with a significant learning curve. In contrast, we are

interested in low-end amateur motion generation: animation of toy robots by lay users with no technical background.

This paper presents a novel approach and system we call *Mirror Puppeteering*, which allows makers and lay users to create expressive animations for simple robots at home, by directly puppeteering them in front of a webcam. Our system does not depend on knowing the 3D model of the robot, nor does it require high-end hardware on the robot or the user's PC. Instead it is a software system that only needs a webcam and cheap actuators with no position feedback.

We make use of the fact that many home-made or store-bought desktop robots are small, have a stationary base, and have few degrees of freedom. This makes such a robot structurally simple enough that if it could merely "take a look at itself in a mirror", it should be able to figure out what motion it is undergoing. We provide such a mirror, and the computation to translate the mirror image into a direct puppeteering system. From a technical perspective, our method works by converting markers captured in a webcam image to motor angles using a calibration map either provided with the robot or generated by the user.

DESIGN GOALS

Our aim in designing "Mirror Puppeteering" was to bring the expressive power and direct manipulation of puppeteering to low-cost robotic toys and lay users.

Puppeteering—the process of animating virtual characters and robots by moving a real-world physical model—has long been a useful technique in the toolbox of both animators and robot motion designers. Using the body and gestures to explore the animation space is an intuitive and rich way to express movement, allowing the quick and easy creation of complex motions that would be tedious to keyframe or to describe numerically.

However, today's puppeteering systems rely on high-end motion capture systems, custom waldos, mechanical proxies, or sensor-equipped actuators, making these methods more appropriate for professional and semi-professional settings.

In contrast, we want to allow the use of out-of-the-box low-cost puppets for which anyone can quickly create expressive motions. This should ideally be as easy as playing with the robot itself.

This leads us to the following design goals:

- **Easy to Learn and Use —** The system is intended for lay users, with no technical background, and for hobbyist makers. It should be easy to learn and use. It should work out-of-the box with direct manipulation and one-click controls.
- **Fits Low-cost Robotic Toys —** To work with commercial robotic toys and home-made robots, the system should not rely on motor sensors, position feedback channels, or sophisticated robot hardware.
- **Makes Use of Existing Hardware —** For wide acceptance, we aim for a software-only system that does not require high-end computing hardware on the user's side either.
- **Robust —** Given the low-cost hardware and consumer-grade computing requirement, the method should be robust enough to be feasible for home use.
- **Customizable for new robots and toys —** To make this appealing for makers, the system should be easily adapted to new custom-made robots. Makers should be able to quickly set it up with no more complexity than building and programming their simple robot.

USER SCENARIOS

We illustrate how "Mirror Puppeteering" addresses these goals by presenting scenarios of three kinds of users (Fig. 2): The novice user generates animations for the robot by puppeteering it out of the box in a set position in front of the camera. A more sophisticated user can set the robot in a different robot-camera configuration and run the robot's self-calibration procedure with the press of a button. Finally, "makers" can build their own robots and quickly set them up to be animated with Mirror Puppeteering.

Novice (Puppeteering only)

In the simplest use-case, the novice user buys a toy robot, which comes with a number of preset gestures and expressions, but also with the capability of authoring new gestures using "Mirror Puppeteering". To do so, the user places the robot in front of a webcam, the robot's "mirror", in one of a choice of preset positions indicated by outlines drawn on the camera video view. The user then presses "Record" and starts puppeteering the robot by moving the desired limbs in realtime. The user can also layer movements one limb at a time (i.e., animating one part, rewinding the timeline and then animating others). When puppeteering is completed, the user presses the "Stop" button, and the robot can save and play back the animation without the need for a camera.

Advanced User (Calibration + Puppeteering)

As mentioned above, the robot would ship with a number of pre-set camera positions for animation. However, an advanced user might want to set up the robot in a different position, for example to use two robots at once in a robot theater scene. To do so, the advanced user places the robot in the desired position and runs a firmware self-calibration routine with a single press of the "Calibrate" button. The robot moves its motors through all degrees of freedom to generate an up-to-date calibration for the new position. The new configuration is stored, and used for as many puppeteering sessions as the animator likes.

Maker (Design + Calibration + Puppeteering)

For "makers", Mirror Puppeteering allows rapid prototyping of new home-made robots. The maker builds a robot using robotics kits, 3D-printing, or laser cutting, in combination with hobby servos or stepper motors controlled through a micro-controller board. Once the robot is built, the maker

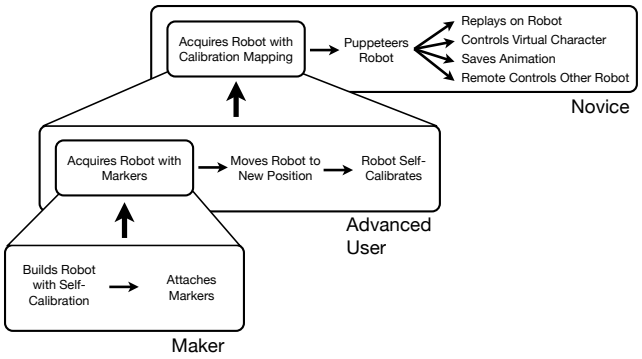

Figure 2. Three uses of Mirror Puppeteering. Novice users can simply puppeteer and playback (top row); advanced users can create new calibration maps in different scenarios (middle expansion); makers can quickly prototype new robots (bottom expansion).

simply attaches temporary markers to the robot's exterior and includes a short self-calibration routine as part of the robot's firmware.

Relation to Design Goals

These use-cases, in combination with our user study, demonstrate the way "Mirror Puppeteering" addresses our design goals. We evaluate ease of learning and use in a user study involving novices with no prior gesture animation experience. We find that participants rate continuous puppeteering more usable, easier to learn, more successful, and more enjoyable to use than traditional keyframing.

To demonstrate the feasibility of our system for consumer-grade robotic and PC hardware, we apply it to a low-cost robotic toy, My Keepon. We also run the software for all our demo robots using a standard MacBook laptop.

Our computer vision algorithm is designed for robustness, with the home user in mind. Marker detection is done per-frame, not tracked over time, thus vision errors do not accumulate and markers can pass in and out of occlusion. Our formulation of the matching algorithm is robust to detection failure and ignores excess markers in the background.

Finally, the self-calibration routine means that new robots can immediately use our system; the user need not import or specify the joint hierarchy or 3D model. In fact, we have been able to build five different custom robots, each in a few hours, and use our system to animate them without altering the software in any way. We also used the system with a store-bought robot toy, demonstrating the use case of a hacker retrofitting a commercial product.

Figure 1 and the accompanying video clip show the generalizability of Mirror Puppeteering through the variety of robots we animated using our system. The robots include, among others, a 2-axis robotic drawing machine, a robotic gripper, four dancing felt snakes, and a soft commercial robotic toy. We also used our system to control and animate two virtual characters.

RELATED WORK

Methods for animating robots present tradeoffs between the level of control over the final animation, and the ease and naturalness of the animation process. At the highest level of control, robot motion is programmed numerically in code. A step forward in usability is designing the motion in 3D animation software and then exporting the generated motion into motor space. The process is time-consuming but allows a fine level of control, making it mostly appropriate for highly trained professionals.

In contrast, using human motion to animate a robot is accessible even to non-professionals, and enables a high level of naturalness by taking advantage of the embodied coordination of the human body. In previous work, data from motion capture systems has been retargeted to animate complex robots and characters [14, 19]. Human motion for animation can be captured more cheaply, but with fewer degrees of freedom, using video [7], accelerometers [16], depth cameras [9], force sensors [10], and multitouch input devices [17]. Proxy objects, such as the sensing stuffed toy used in Sympathetic Interfaces, can also be used [12]. Motion retargeting, however, comes with a price: robots can have markedly different morphologies than humans, and in the transfer of the motion to the robot physiology, small-scale motion fidelity is lost. Our work, instead, works by directly puppeteering the robot.

Direct puppeteering takes advantage of the identity of input and output, while still allowing direct embodied control over the motor trajectories. This method generally relies on actuator proprioceptive sensors. In previous puppeteering work, a sensing robot spine for dancing stuffed animals [3] uses the Robotis Dynamixel line of servos with serial-readable encoders. Other actuated sensing platforms include an active puppet [20]; Curlybot, a toy that can replay motion [8]; and Topobo, a construction set with kinetic memory [15]. In contrast to our system, they all come with position-readable actuator hardware, usually not found in low-end robotic toys. Although it is possible to hack the latter to get encoding feedback, the servo must be powered (thus non-backdrivable) to read the encoder, so this trick is not viable for puppeteering.

A related field, also using sensor-equipped robots, is "learning by demonstration" (see [4] for a survey). Training examples are created for the robot, e.g. by teleoperation, human motion capture, or kinesthetic teaching, where a human guides the robot to show it how to point to an object [13], perform a task [2], or gesture [6]. In contrast to ours, the application domain for these is mostly industrial and concerned with movement generalization for high-end robots.

Using an external camera to record motion presents a tempting alternative to sensors, as cameras are cheap, reusable, and noninvasive. Puppeteering systems can track 2D paper cutouts using an overhead camera [5], and 3D objects using a depth sensor [11]. Motion capture systems have been used to calibrate robot motors [18]. None of these, however, have used a webcam to directly puppeteer a robot.

SYSTEM DESCRIPTION

"Mirror Puppeteering" works by converting markers to motor angles using a calibration map either provided with the robot or generated by the advanced user. This mapping is a lookup table from markers on the robot viewed in camera space, to motor angles; it is thus generated once per robot and camera position.

The method consists of three stages: self-calibration, puppeteering, and playback. In self-calibration, the robot moves each of its limbs through its full range, streaming the limb's current motor commands to the computer. Simultaneously, the moving markers are detected in the webcam frames, allowing the software to build up a calibration map between motor angles and camera-space markers. This stage needs to be done only once per camera-robot configuration, and can be done ahead of time as part of the robot's factory settings. In puppeteering, the animator moves the robot through its desired trajectories, and our software detects markers in the camera image. For each frame, the system scans through the limb's map, to find the entry with markers in a similar position as the current camera frame. Once a match is found, the corresponding motor angles are added to the animation timeline. In playback, the robot uses the stored motor commands to recreate the movement. This stage does not require markers or a camera. Replay can be duplicated and transferred to a different robot, or to a virtual character.

Note that, at no stage, did we need to give the software any knowledge of the robot's geometry, joint hierarchies, input limits, or self-collisions. Instead, we maintain the paradigm that it is the robot that embodies self-knowledge; our "mirror" is a hardware-agnostic tool for the robot to look at itself. Any new robot with the right calibration routine can be used out-of-the-box, without altering the mirror software.

We split the description of the system into four parts:

1. **Self-Calibration** we describe the automatic calibration stage, including marker placement and tracking.
2. **Puppeteering** we describe puppeteering in realtime continuous mode and tracked keyframing mode, and detail the matching algorithm and graphical user interface.
3. **Playback and Beyond** we describe ways in which the data collected during Mirror Puppeteering can be used, beyond direct playback.
4. **Prototype** we describe the hardware setup we used in our Mirror Puppeteering prototype.

Self-Calibration

In the self-calibration stage, our software tracks markers placed on each of the robot's limbs. We use markers consisting of dark circles on a light background. We chose these markers not only for their traceability, but also because they can be easily incorporated into a robotic puppet, e.g. as eyes.

We identify markers in realtime using OpenCV; with a webcam resolution of 1280x720, median blur and marker detection are actually the speed-limiting part of our software, running at circa 18 fps on a 2GHz quadcore processor. We experimented with using QR codes and various marker shapes and colors to provide identification and orientation information, and found circles to be the only shape whose tracking is robust to the significant amount of motion blur.

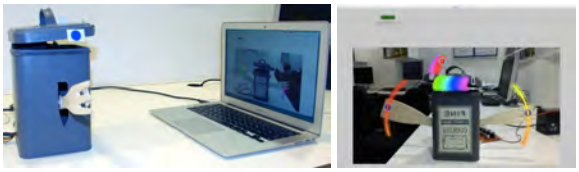

Figure 3. The cookie jar robot calibrating its head, with the calibration map visualized as a colored overlay onscreen.

The markers need to be placed such that in any given robot pose, at least one marker will be visible at the end of every motor chain. For this purpose, we define a *motor chain* as an ordered set of motors from the robot base to an end effector. When referring to motors, we use the notation

$$chain_i \equiv (motor_{i_1}, motor_{i_2}, \ldots, motor_{i_N})$$

Motors can belong to more than one chain; for instance, if motors are arranged in a "Y" configuration, the base motor belongs to two chains, and is therefore both $motor_{1_1}$ and $motor_{2_1}$. The angle of $motor_{i_j}$ is denoted θ_{i_j}.

One marker visible in the 2D camera view is sufficient to characterize a two-motor chain. In a chain of length $N > 2$, each additional $motor_{i_j}, j \leq N-1$, added to the base of the chain, requires a marker to always be visible on the robot limb between $motor_{i_j}$ and $motor_{i_{j+1}}$.

We do not track marker trajectories over time, nor label the markers; each frame provides only a set of marker locations and sizes. Our philosophy is to build a stable system by making use of the parts of computer vision that are very accurate: blob (marker) finding in a single frame is mostly repeatable and mistakes do not accumulate; in contrast, attempting to label markers over time as they come in and out of occlusion is more error prone.

We will use $\{(x_k, y_k, r_k)\}$ to refer to the set of markers, with x and y coordinates *in camera frame space* and radii r (as a proxy for z position), detected in a given frame k. During the calibration phase we compute which markers are in motion. We maintain a one-second buffer of marker sets; to compute whether a marker in the current frame is moving, we look one second earlier and check if there was a marker in the same location and size as the current marker. If not, the marker is defined as moving.

To calibrate the robot, it is positioned in front of the webcam as shown in Figure 3. By clicking on the "Calibrate" button in the interface, a signal is sent to the microcontroller to begin the calibration routine. In this routine, the microcontroller takes each chain in turn, and moves the chain through every combination of angles it can assume. With each change

in angle, the microcontroller sends the list of motors in the chain, and their current angles, to the software for recording. As the software receives the angles over serial for each $chain_i$, it simultaneously finds the markers $\{(x_k, y_k, r_k)\}$ that are moving in the current camera frame. The software then stores this combination as one calibration map entry in $calib_i$, matching the angles and the moving markers:

$$calib_i[(\theta_{i_1}, \theta_{i_2}, \ldots)] = \{(x_k, y_k, r_k)\} \quad (1)$$

Calibration for the cookie jar robot seen in Figure 1 takes roughly 5 minutes. The arm is moved from its lowest position to its highest, in increments of $\sim 1°$. For each $3°$ yaw of the head, the head does a full up-down sweep. For chains of length $N > 2$, it is important to work backwards from $motor_{i_N}$ in the outer loop to the base motor, $motor_{i_1}$, in the innermost loop. With the base motor always moving, *all* markers on the chain will be moving, and they can thus be isolated from other chains.

Along with the calibration map, we save a sample frame and its motor angles; when the user reloads a map, we present the frame overlayed on the current live video stream so that the robot can be positioned accurately.

Puppeteering

After the calibration step, or by loading previous calibration data and aligning the robot, the animator can start animating.

Matching Algorithm

The matching algorithm selects the motor signals that would create the robot pose currently seen by the webcam. Marker locations are captured by the webcam and matched against entries in the calibration map. A chain can have many calibration poses with similar marker sets; our matching function will choose between them by checking for continuity with previously computed angles. Thus, we track the robot throughout a user's session, whether or not the data is being recorded. We initialize the matching algorithm with the angles from the sample calibration frame.

In our matching algorithm we look at each motor chain independently. For each $chain_i$, we run through all entries in $calib_i$. The entry we are seeking will be an accurate subset of the markers (moving or not) in the current camera frame. A distance function uses this metric, looping over each marker (x_k^c, y_k^c, r_k^c) in the calibration entry and adding the distance to the nearest marker in the camera frame (from the set $\{(x_k, y_k, r_k)\}$). By matching from the calibration set to the current frame, spurious markers in the background, and markers in other chains, do not hinder matching.

When calculating the distance function, we penalize motor jumps greater than 10 percent. This enforcement of motor continuity is key to our being able to use indistinguishable circular markers. For example, the cookie jar looking 90 degrees right with a marker on the left side of its lid, and looking straight with a marker in front center of its lid, both present identical (x_k, y_k, r_k) marker locations; however, the penalty ensures only the pose closest to the previously computed one is chosen.

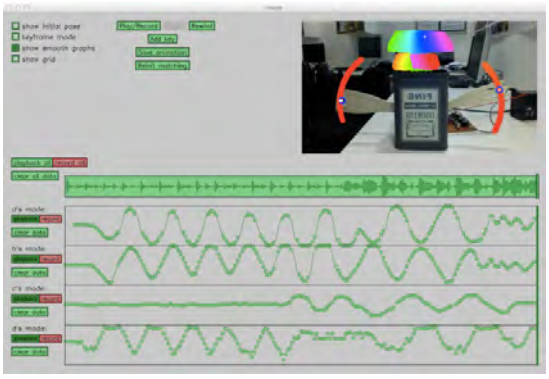

Figure 4. The interface showing a puppeteered animation.

Graphical User Interface

Our GUI, shown in Figure 4, presents a timeline with audio visualization and motor graphs along the bottom of the screen, and button controls in the upper left. A live view of the camera is in the upper right. The view is overlayed with the calibration data and a realtime view of the matching results (white lines between detected markers and the calibration markers that matched to them).

We support two Mirror Puppeteering techniques in the GUI: *continuous puppeteering* and *tracked keyframing*. During *continuous puppeteering*, the robot's movements are recorded in realtime with the timeline playhead advancing continuously. The animator toggles motors back and forth between "record" and "playback" mode. When the animator hits "play" to puppeteer the robot, motors set to "record" will continuously record, and motors set to "playback" will move of their own accord. Thus the animator can layer animation, animating one limb, and subsequently replaying the motion at the same time he is animating another limb. In *tracked keyframing*, the software tracks the robot as the animator moves it; but only discrete poses are recorded. The animator moves the playhead to a given position, poses the robot, and clicks "add keyframe" to add the pose to the graph. For the purpose of evaluating our system with respect to traditional methods (see below), we also include an option for *traditional keyframing*, where no tracking is used and the animator clicks on graphs representing each limb's position to create, delete, and adjust keypoints.

Playback and Beyond

Playback can be done away from the camera; at this point, the systems knows all the motor positions in the animation. Markers are usually part of the robot's "costume", but if temporary markers were attached to the robot, they can be removed at this stage.

The "mirror" we have created produces a fairly generic mapping: markers visible in the camera to motor positions. Mirror Puppeteering thus has capabilities in addition to the recording and playback of a single robot.

Without any modification, our system supports realtime show control. A user with two physically identical robots can pup-

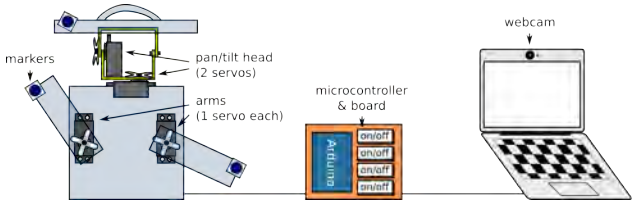

Figure 5. Our prototypes consist of a robot made up of servos, a microcontroller with a circuit board, and a laptop with a webcam.

peteer one robot "behind the curtain" while simultaneously playing back the animation on the second robot; we show this technique in the accompanying video using the clock robots. The control robot can also be a smaller proxy of a larger animated robot on stage. Our software can also function as a more general motion capture tool. For example, after calibrating the cookie jar robot's arm, one could attach a marker to one's own hand, and animate the robot by waving one's hand along a similar path.

Finally, an animator can use the system to drive a virtual character by puppeteering a physical robot. Our software streams out motor angles simultaneously to USB and to a named pipe, allowing both custom hardware and animation software to link easily into our system.

Hardware Prototype

Mirror Puppeteering works on any robot with position controlled actuators which can be back-driven when switched off. In our prototypes, we tested the system on My Keepon, a toy robot with DC gear motors, and several robots we built ourselves, actuated with hobby servos.

In our prototyping setup for homemade servo robots, the robot's custom calibration routine is loaded on the Arduino microcontroller acting as the robot's firmware. The circuit board attached to the microcontroller cuts/gives power to the servos to switch them between "record" and "playback" modes (Figure 5). We build two development boards. The manual board contains flip switches to control power to each servo individually. The automatic board contains relays which are controlled by our software "record" and "play" toggles.

The Arduino and computer communicate over USB. During calibration, the Arduino streams the current motor angles to the computer. During animation playback, the computer sends power control signals and motor angles in realtime to the Arduino and the Arduino moves the motors.

EVALUATION

We evaluated our system using the 4-DoF cookie jar robot shown throughout this paper. We measured the system's accuracy by comparing original motor positions to reconstructed motor positions for several motion trajectories. We measured user experience in a user study comparing continuous puppeteering, tracked keyframing, and traditional keyframing.

Accuracy

To evaluate the accuracy of our system we took puppeteered dance animations from participants in the study described below, and played them back on the robot. Simultaneously, we used our matching algorithm to reconstruct the motor positions.

We measured the mean error in degrees for each motor over three 15-second animations, sampled at 1000Hz. Mean errors were 3.2° for the left arm, 3.0° for the right arm, 2.7° for the head pan, and 3.1° for the head tilt. Fast movements were the greatest source of error, as they create motion blur which can cause tracking failure.

User Study

To evaluate the usability of our system as compared to traditional robot animation, we ran a user study comparing three methods: continuous puppeteering, tracked keyframing, and traditional keyframing. Participants used the cookie jar robot in a preset configuration setup, simulating the use-case of a novice user animating a commercial robot.

Participants were given fifteen seconds of a piece of dance music, and asked to design a dance for the robot. The dance could be on-beat or off-beat, detailed or not. For each of the three methods, used in randomized order, the experimenter explained the method, the participant worked on a dance "until they were satisfied" (usually around 10 minutes), and then the participant completed a survey.

The survey contained four Likert scales, each the mean of two items: quality of results ("I was satisfied with the quality of the animation I created.", "The result of this method was close to what I originally had in mind."); enjoyment ("I had fun animating using this method.","This method was annoying to use (reversed scale)."); usability ("I could easily get the robot to move like I wanted it to.","I found this method inefficient (reversed scale)."); and learning curve ("I understood how to use this method easily.","I became productive quickly using this method."). These correspond to our design goals. Each item was rated from 0 (strongly disagree) to 7 (strongly agree). At the end of the survey were two optional open-ended questions asking the participant to write one good and one bad thing about the method.

Eighteen participants, 11 male, 10 female (3 participants were pairs), ages 10 to 39, participated in the study. Four had experience with an animation software such as Maya, but none were expert animators. All participants seemed entertained by the experience of programming a robot, no matter which method.

Results

From subjective observation, we note that each of the three methods requires a different mental approach to animating; the experimenter observed that participants would often "get" one method most intuitively.

We found support for this observation in the participants' open question remarks. Most participants used the word

	mean (std dev)		
	continuous puppeteering	tracked keyframing	traditional keyframing
Results	**5.7 (1.3)**	5.2 (1.1)	**4.3 (1.9)**
Enjoyability	**6.4 (0.9)**	*5.6 (1.5)*	***4.0 (2.0)***
Usability	**5.6 (1.4)**	5.1 (1.3)	**4.2 (1.7)**
Learning Curve	**6.2 (1.1)**	5.4 (1.2)	**5.2 (1.5)**

Table 1. Participants rated continuous puppeteering and traditional keyframing significantly different along all four scales (bold). Tracked keyframing was significantly different (italics) only in "enjoyability" against traditional keyframing.

"fun" to describe the puppeteering interface; but while some found it "easy to use" and "more interactive", and one liked the "unmediated feel to the animation process", several others complained that it required too much coordination. Some found that tracked keyframing, "allows you to easily stay on beat" and "be more exact", but others had trouble with the thinking process, because "you need to remember what you already made the robot do in order to make a good dance. You have to focus, which makes it more difficult", and "you don't really know how it will all come together". When using the traditional keyframing interface, some participants happily created an interesting curve on the graph, and extensively fine-tuned it, with little regard for what it would actually look like on the robot; others focused intensely on mentally mapping locations on a motor graph to corresponding motor angles. The traditional keyframing interface was thus both "straightforward and pretty simple" and "quicker", as well as "far less intuitive" and "exhausting".

Likert Scales

Table 1 shows the results of the four Likert scale measures. One-way ANOVA revealed significant differences between groups in all four scales: scale Results, $F(2, 51) = 3.97, p < .05$; Enjoyment, $F(2, 51) = 12.36, p < .001$; Usability, $F(2, 51) = 4.03, p < .05$; and Learning Curve, $F(2, 51) = 3.68, p < .05$. Post-hoc Bonferroni-corrected pairwise comparisons revealed that puppeteering was significantly different from traditional keyframing on all four metrics ($p < .05$). Tracked keyframing was significantly different only in the "enjoyment" scale, compared with traditional keyframing ($p < .05$).

LIMITATIONS

The primary limitation of our algorithm, which restricts us to low-DoF robots, is that the number of combinations of angles in the calibration sequence must be tractable. As each motor chain is calibrated independently, the robot can have a large number of limbs, but to be computationally feasible, each limb must be restricted to two degrees of freedom. A limb can have more degrees of freedom if each motor covers a small range. For example, in the accompanying video we add ears to the lid of the cookie jar robot, creating three DoF chains; however, we restrict the range of the ears to three discrete positions (left-middle-right).

Robots must have a stationary base. While this does not make our system appropriate for mobile robots, in practice, many of the expressive robots designed in the last decade do actually have a stationary base. Some examples include MIT's Leonardo and AUR; iCat; Jibo; Georgia Tech's Travis and Shimon; Keepon; and Karotz.

Very fast puppeteering can cause markers to motion blur in the camera frame. We overcome the ambiguity among similar marker sets in calibration entries by looking for temporal motor angle continuity. However, if—for example—-we rotate the cookie jar head too rapidly from left to right, causing motion blur during the transition, our tracking algorithm will fail, as both sides look similar. We can alleviate this problem by offsetting the markers so they do not end up in identical camera frame locations. Note in Figure 1 that the center marker on the cookie jar's head is raised relative to the side markers. Future work could include additional disambiguation cues such as marker color.

Robots must be designed so that one limb cannot fully occlude all the markers in another limb. A possible solution in case of occlusion is to place the markers on an extension for calibration and puppeteering. In the snakes robot in Figure 1, for example, the front snakes fully occlude the rear snakes; we added antennae to hold the rear markers.

The relationship between the camera and the 3D marker movement affects the precision of tracking. Marker movements that are co-planar or near co-planar to the image plane are most readily tracked. Movements which are largely towards or away from the camera are not as easily distinguished, but this limitation is ameliorated by factoring in the radius of the markers. Naturally, the higher the resolution of the camera, the less sensitive the system is to this issue.

The robot must be in precisely the same camera alignment during calibration and animation. In practice we found it often faster to take the role of the "advanced user" and let the robot run self-calibration whenever placing it in front of the camera.

CONCLUSION AND FUTURE WORK

This paper presented Mirror Puppeteering, a direct puppeteering system that enables lay users to create expressive animations on toy robots. Robots in this class tend to be small and structurally simple, an ideal opportunity for a vision-based system. We use markers on the robot, which can be embedded in the robot's design, and track these markers to recover motor positions. Our system is agnostic to the 3D structure of the robot, allowing new robots to quickly use the software as-is. This makes our system a candidate to be web-deployed and generic enough for a large number of toys.

Mirror Puppeteering can also be used to control robots in realtime, control virtual characters, and record animations for them. In this paper and the accompanying video, we demonstrate these possibilities on a range of small robots and virtual characters.

Our user study showed that puppeteering at this scale is understandable by the naive user, useful, and, moreover, enjoy-

able. Although the optimality of continuous recording versus poses is task dependent [1], participants' preferences in even our single task were divided among all three methods, encouraging their inclusion in future puppeteering systems. Moreover, several participants suggested that the ideal animation software would be a combination of puppeteering and traditional keyframing: puppeteering to lay down the rough animation, and keyframing to fine-tune it. This workflow is possible in our current software, although it would be tedious to adjust each point in the dense curves created by puppeteering. An intuitive way to modify these curves at different scales would be a useful addition to our software.

Future work could make the system simpler still, eliminating the need for even the robot's firmware to know its joint hierarchy and motor limits in advance. Similar to the self-calibration step, the robot could look at itself in the mirror, tracking marker locations, to figure out what its structure and limits are.

Given the increasing popularity of both toy robotics and amateur 3D animation, the fact that Mirror Puppeteering is specifically designed for consumer-grade webcams and low-cost motors, and is designed to work easily out of the box, makes this a promising candidate for real-world usage of puppeteering as a home animation technique. Novices can thus use both hands, and the richness and fluency of their embodied motion, to bring their robots and 3D characters to life.

REFERENCES

1. B. Akgun, M. Cakmak, K. Jiang, and A. Thomaz. Keyframe-based learning from demonstration. *International Journal of Social Robotics*, 4(4):343–355, 2012.
2. B. Akgun, M. Cakmak, J. W. Yoo, and A. Thomaz. Trajectories and keyframes for kinesthetic teaching: a human-robot interaction perspective. In *ACM/IEEE International Conf. on Human-Robot Interaction*, pages 391–398, 2012.
3. J. Allen, J. Young, D. Sakamoto, and T. Igarashi. Style by demonstration for interactive robot motion. In *ACM Designing Interactive Systems Conf.*, pages 592–601, 2012.
4. B. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.
5. C. Barnes, D. Jacobs, J. Sanders, D. Goldman, S. Rusinkiewicz, A. Finkelstein, and M. Agrawala. Video puppetry: a performative interface for cutout animation. *ACM Transactions on Graphics*, 27(5):124:1–124:9, 2008.
6. S. Calinon and A. Billard. Incremental learning of gestures by imitation in a humanoid robot. In *ACM/IEEE International Conf. on Human-Robot Interaction*, pages 255–262, 2007.
7. J. Cole, D. Grimes, and R. Rao. Learning full-body motions from monocular vision: dynamic imitation in a humanoid robot. In *IEEE/RSJ International Conf. on Intelligent Robots and Systems*, pages 240–246, 2007.
8. P. Frei, V. Su, B. Mikhak, and H. Ishii. Curlybot: Designing a new class of computational toys. In *ACM Conf. on Human Factors in Computing Systems*, pages 129–136, 2000.
9. P. Goebel. Skeleton tracker teleoperation package for mobile robot, 2013. `www.ros.org`.
10. S. Ha, Y. Bai, and K. Liu. Human motion reconstruction from force sensors. In *ACM SIGGRAPH/Eurographics Symp. on Computer Animation*, pages 129–138, 2011.
11. R. Held, A. Gupta, B. Curless, and M. Agrawala. 3d puppetry: a kinect-based interface for 3d animation. In *ACM Symp. on User Interface Software and Technology*, pages 423–434, 2012.
12. M. Johnson, A. Wilson, B. Blumberg, C. Kline, and A. Bobick. Sympathetic interfaces: Using a plush toy to direct synthetic characters. In *ACM Conf. on Human Factors in Computing Systems*, pages 152–158, 1999.
13. A. Lemme, A. Freire, G. Barreto, and J. Steil. Kinesthetic teaching of visuomotor coordination for pointing by the humanoid robot icub. *Neurocomputing*, 2013.
14. N. Pollard and J. Hodgins. Optimizing human motion for the control of a humanoid robot. In *International Symp. on Adaptive Motion of Animals and Machines*, March 2003.
15. H. Raffle, A. Parkes, and H. Ishii. Topobo: a constructive assembly system with kinetic memory. In *ACM Conf. on Human Factors in Computing Systems*, pages 647–654, 2004.
16. R. Slyper and J. Hodgins. Action capture with accelerometers. In *ACM SIGGRAPH/Eurographics Symp. on Computer Animation*, pages 193–199, 2008.
17. Y. Sugiura, G. Kakehi, A. Withana, C. Fernando, D. Sakamoto, M. Inami, and T. Igarashi. Walky: An operating method for a bipedal walking robot for entertainment. In *ACM SIGGRAPH Asia Emerging Technologies*, pages 79–79, 2009.
18. Y. Takase, H. Mitake, Y. Yamashita, and S. Hasegawa. Motion generation for the stuffed-toy robot. In *SICE Annual Conf.*, pages 213–217, 2013.
19. K. Yamane, Y. Ariki, and J. Hodgins. Animating non-humanoid characters with human motion data. In *ACM SIGGRAPH/Eurographics Symp. on Computer Animation*, pages 169–178, 2010.
20. W. Yoshizaki, Y. Sugiura, A. Chiou, S. Hashimoto, M. Inami, T. Igarashi, Y. Akazawa, K. Kawachi, S. Kagami, and M. Mochimaru. An actuated physical puppet as an input device for controlling a digital manikin. In *ACM Conf. on Human Factors in Computing Systems*, pages 637–646, 2011.