Geppetto: Enabling Semantic Design of Expressive Robot Behaviors *Supplementary Material*

Ruta Desai
Carnegie Mellon University
Pittsburgh, PA, USA
rutad@cmu.edu

Fraser Anderson
Autodesk Research
Toronto, ON, Canada
fraser.anderson@autodesk.com

Justin Matejka
Autodesk Research
Toronto, ON, Canada
justin.matejka@autodesk.com

Stelian Coros
ETH
Zurich, Switzerland
scoros@inf.ethz.ch

James McCann
Carnegie Mellon University
Pittsburgh, PA, USA
jmccann@cs.cmu.edu

George Fitzmaurice
Autodesk Research
Toronto, ON, Canada
george.fitzmaurice@autodesk.com

Tovi Grossman
Autodesk Research
Toronto, ON, Canada
tovi.grossman@autodesk.com

ACM Reference Format:
Ruta Desai, Fraser Anderson, Justin Matejka, Stelian Coros, James McCann, George Fitzmaurice, and Tovi Grossman. 2019. Geppetto: Enabling Semantic Design of Expressive Robot Behaviors *Supplementary Material*. In *CHI Conference on Human Factors in Computing Systems Proceedings (CHI 2019), May 4–9, 2019, Glasgow, Scotland UK.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3290605.3300599

1 MOTION SYNTHESIS FOR WALKING ROBOT

Parameterization and optimization framework

The quadruped robot's motion consists of periodic coordinated limb movements, repeating in cycles of time T (henceforth referred as gait cycle). Such a motion $\mathbf{m}$ is defined by time-indexed sequence of vectors P_i, as $\mathbf{m}\;\; P_1, \ldots\ldots\ldots, P_T$. P_i represents joint and limb end-effector positions at i^{th} time-step (robot pose), as well as information about whether a limb is on the ground or moving in the air at that instant. Based on the prior work, we obtain the robot motion $\mathbf{m}$ by solving a constrained trajectory optimization problem described below [4].

$$\begin{aligned} &\min_{\mathbf{m}} \;{}_{i} E_i \mathbf{m} \\ &s.t. \quad \mathbf{Cm}\;\; 0, \end{aligned} \tag{1}$$

where $E_i\mathbf{m}$ represent quadratic penalty terms for achieving desired speed and motion style, while $\mathbf{Cm}\;\; 0$ are various constraints necessary for a valid motion. These constraints include kinematic and limb collision avoidance constraints that ensure motion's physical validity, and friction constraints to avoid foot slipping. Constraints corresponding to hardware such as motor joint and torque limits are also included. A detailed explanation of constraints and objectives can be found in the work of Megaro et al. [4]. An example quadratic objective for achieving desired walking speed is described below.

$$E_{\text{speed}} \;\; \frac{1}{2}||\mathbf{x}_T - \mathbf{x}_1 - \mathbf{d}^D||^2, \tag{2}$$

where $\mathbf{x_i}$ is the robot's COM, and $\mathbf{d}^D$ is desired travel distance to be achieved in time T so as to maintain a user-defined walking speed. The goal of motion optimization is thus to make the robot move forward with a desired speed and motion style, while satisfying various physics-based and feasibility constraints. We use gradient-based solvers to find such a valid and desired motion [4].Various parameters effecting the motion $\mathbf{m}$ are summarized in Table 1. Megaro et al. have also

ACM ISBN 978-1-4503-5970-2/19/05. . . $15.00
https://doi.org/10.1145/3290605.3300599

Table 1: Walking robot motion parameters

Parameter name	Parameter type	Parameter dimension
Gait pattern	Discrete	1
Gait time	Continuous	1
Speed	Continuous	1
Foot height	Continuous	1
Robot pose	Continuous	7

tested motions generated using this framework on three types of walking robot hardware prototypes [4].

Implementation

The motion simulation and optimization is implemented in C++. The implementation allows using different robotic platforms, given their kinematic information and 3D part files. Note that 3D parts are needed for visualization in the UI as well as for collision avoidance. Various mesh formats such as .obj are supported for loading 3D parts, while kinematic information can be loaded as text files. This could easily be extended to support other formats such as Universal Robot Description Format (URDF) used by Robot Operation System (ROS) [9]. Motor angle trajectories of all joints for the final motions can be exported as text files, or can be directly sent to micro-controllers such as Arduino [1], Dynamixel's OpenCM [8] etc. for execution on hardware. The system also allows saving robot motions in an internal format, so as to support robot behavior design over multiple sessions.

2 MOTION SYNTHESIS FOR ROBOTIC ARM

Boids simulation framework

We provide a detailed overview of our parameterization and extension of Boids simulation framework [7] for driving the robot arm.

Parameterization. The Boids simulation is parameterized by interaction rules specific parameters such as the weights w_s, w_a, w_c, w_g, w_e corresponding to our five interaction rules (separation, alignment, cohesion, goal-seeking, and exploration), and the neighborhood n. A maximum speed v_{max} is defined for the overall motion of the boids. v_{max} serves two purposes. First, it enables us to create flock motions (and thereby robot arm motions) with different overall speeds ranging from very slow to fast motions. Secondly, it ensures that the simulation doesn't become unstable by applying an unreasonably high steering force on the boids. Other additional parameters include – parameters for exploration rule, and those corresponding to flock initialization procedure (explained later). These parameters are summarized in Table 2. Together, they form at 11-dimensional parameter space.

Table 2: Boids simulation parameters

Parameter name	Parameter type
Neighborhood n	Continuous
Separation rule weight w_s	Continuous
Alignment rule weight w_a	Continuous
Cohesion rule weight w_c	Continuous
Seek rule weight w_g	Continuous
Explore rule weight w_e	Continuous
Explore radius r_e	Continuous
Explore frequency f_e	Continuous
Maximum flock speed v_{max}	Continuous
Initial boid velocity direction $\vec{v}_{init}$	Discrete
Initialization time t_{init}	Continuous

We extend the basic Boids simulation framework in three ways. First, we define an initialization procedure for the boids to increase the diversity of flock behavior. Second, we define a custom parameterized interaction rule called exploration. Lastly, we develop a heuristics to smoothly blend the exploration rule with other interaction rules of the basic boids model.

Boid flock initialization procedure. This procedure was added to increase the variations of generated paths such as paths that go backwards or sideways before converging on the desired goal. We use two parameters for this purpose – an initialization time t_{init}, and an initialization direction $\vec{v}_{init}$. Each boid's velocity is initialized to be in $\vec{v}_{init}$ direction before the simulation. To create paths that go backwards, sideways etc., $\vec{v}_{init}$ is chosen from a set of six direction basis corresponding to *+X, -X, +Y, -Y, +Z, -Z* axis in the task space. The initialization time t_{init}, which is typically less than the overall motion time, represents a buffer time in the beginning of simulation when the flocking behavior is dominated in the $\vec{v}_{init}$ direction. This is to ensure that the initialization procedure effects the flock's behavior, and is achieved by gradual ramping of certain interaction rule weights.

Exploration rule. This rule steers the boid towards a random goal intermittently to enforce randomness in the flock behavior. This ensures interesting non-straight paths for the robot that eventually converge at the desired goal in our case. It is defined using two parameters – exploration radius r_e, and exploration frequency f_e. These parameters are used to decide where and when to set a new random goal position for the flock. This position is sampled at random from within a sphere with radius r_e centered at the average flock position, at every f_e^{th} time-step of the simulation. Explore rule then steers each boid to move towards the sampled goal position using the seek rule.

Ramping-damping strategy for rule blending. Seeking behavior that aims to steer the boids towards a goal contradicts the other flocking behaviors that typically steer the boids locally, towards more random movements. To prevent these opposing behaviors from fighting against each other, which may create jerky flock movements, we adopt a ramping-damping strategy. We ramp up some parameters gradually over the course of the motion, while we damp down some parameters as the flock nears the goal. In particular, we linearly ramp up the seeking behavior weight w_g, and damp explore behavior parameters (weight w_e, and explore radius r_e), as well as boid neighborhood n. Damping of neighborhood ensures that boids converge at the goal without locally reacting to each other due to flocking behaviors such as separation and alignment. We use an initial buffer time t_{init} to ramp up the explore behavior parameters before damping damp down. This is to ensure that explore behavior doesn't interfere with the velocity initialization, and that the flock moves in the direction of initial velocity $\vec{v}_{init}$ in the beginning. Figure 1 outlines our ramping-damping strategy.

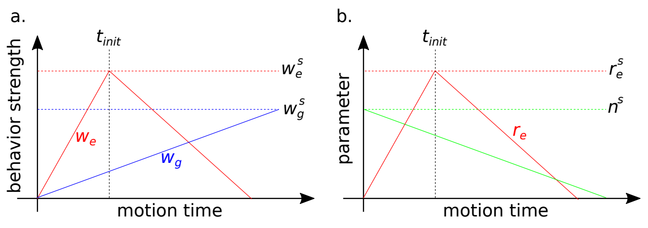

Figure 1: (a) Strength of various behaviors is controlled by ramping and damping their corresponding weights from their sampled values represented by ^{s} superscript. Explore behavior parameters (weight w_e, and explore radius r_e) are color-coded with red, and those of seeking behavior (weight w_g) are color-coded with blue. (b) We also ramp and damp other parameters such as explore radius r_e, and the boid neighborhood n. t_{init} is the initialization time.

3 CROWDSOURCING

We use pairwise comparisons between motion samples to estimate their emotional expression quality. A crowd worker is shown videos of motion sample pairs, and is asked to choose the motion that better conveys an emotion relatively. Such pairwise comparison is then converted into a Gaussian score estimate for each motion sample using Trueskill [5].

Filtering during crowdsourcing

Each worker is employed with a *task* of doing 50 comparisons. We also allow turkers to complete only one such task every 24 hours, in order to prevent a single turker from biasing the results. To further ensure the quality of the ratings, we filter out unreliable turkers. In particular, we use two methods for such filtering, inspired by [11]. First, we expect most turkers to complete a task within a similar period of time. We therefore filter out turkers who do not spend sufficient time on the task, and use less than half of the median task completion time. Secondly, to ensure consistency, we also filter out responses from turkers who either choose the same answer option, or swap their answer choices too frequently over multiple tasks. Specifically, we keep an account of the answer swapping frequency between tasks, and filter out turkers if their swapping frequency is below 35%, or above 75%.

Tournament structure

To efficiently compute emotion ratings of motion samples using TrueSkill, we use an elimination-based tournament setup instead of exhaustively competing each sample against every other. Figure 2 illustrates our tournament structure in comparison to the standard round-robin style tournament on a toy dataset for scoring the designs based on the expressed sadness. Such tournaments are repeated for each emotion category in our crowdsourcing experiments. Elimination of ambiguous, low-ranking samples in earlier rounds allows expressive, high-ranking samples to have a higher number of comparisons against other highly-ranked designs in later rounds. This strategy thus provides more accuracy for the high-ranking samples, while minimizing resources spent on ambiguous, low-ranking samples.

4 REGRESSION EXPERIMENTS

We now explain our data preprocessing for regression analysis, which was used to fit a function between between robot's motion parameters and corresponding emotional expression scores. We also elaborate on our experiments with Artificial Neural Networks (ANN) for such regression. Linear regression was implemented using Python's scikit-learn library [6], while ANN-based fit was obtained using Tensorflow(r1.3) [2].

Pre-processing

We convert the discrete parameters in a n dimensional motion parameter set ϕ_n using one-hot encoding, and standardize the continuous parameters to be amicable to the regression analysis [6]. Further, to prevent over-fitting of functions and to enable generalization beyond our sampled dataset, we fit the functions using 80% of the dataset (train set), and withheld the remaining data for testing the accuracy of the fitted functions. Finally, to deal with the noise in emotional score estimates μ (represented by σ, and obtained from Trueskill), we downweight the ambiguous samples with higher uncertainty σ, while computing the mean squared error cost for function fitting. Thus, a weighted mean squared error (MSE) is used for training. Using weights that are inversely proportional to σ in error computation allows us to focus on fitting the well-expressive samples better.

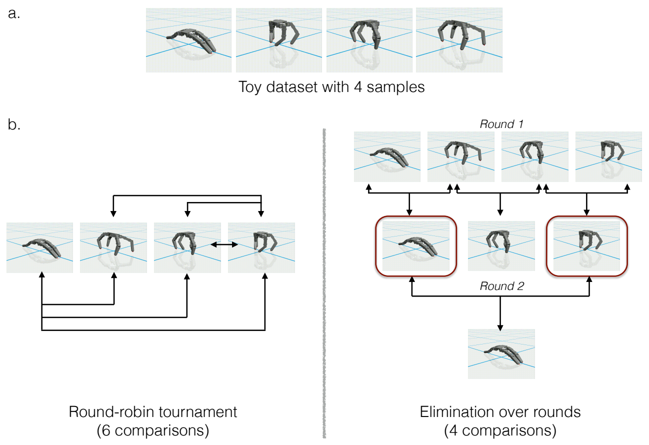

Figure 2: (a) A toy dataset consisting of 4 motion samples is shown. (b) The evaluation of these samples for scoring the sadness expressed by them is illustrated with round-robin and our elimination-based tournament structures. The round-robin tournament structure requires exhaustive comparisons between all pairs of samples, resulting in 6 comparisons for the toy dataset samples. On the other hand, our tournament structure with elimination over rounds, wherein only top half samples (highlighted in red) are considered for evaluation in each consecutive rounds, requires only 4 comparisons. This sample efficiency is very important for dealing with large datasets. Note that in our actual crowdsourcing experiments, we use 3 such rounds in the tournament. Each sample is also compared 5 times (against 5 different designs, by 5 different people) in every round to deal with unreliable crowdworkers.

Regression using ANN

We use separate fully-connected neural networks to fit individual function f for each of our emotion categories. Tensorflow's built-in *Adam* optimizer is used for training [3]. To find the best network for each emotion category, we tune various parameters for the network's architecture such as number of hidden layers, activation functions etc. using standard grid-search [6]. We also use recommended techniques such as drop-out to regularize our networks and to prevent over-fitting.

To evaluate the goodness of fit, we use a statistical measure called R-squared (R^2) or coefficient of determination. R^2 is the proportion of the variance in the dependent variable that is predictable from the independent variable(s) [10]. The weighted R^2 scores for all the predictor functions using the held out 20% of our dataset (test set) are summarized in Table 3 for both the walking robot and robotic arm dataset. We used the best found NN architecture through grid-search in each case for this purpose. Corresponding values for a Linear Regression (LR) fit are also provided for comparison.

Predictor name	NN R^2 (Quadruped)	LR R^2 (Quadruped)	NN R^2 (Arm)	LR R^2 (Arm)
Happy	0.512	0.499	0.327	0.325
Sad	0.40	0.396	0.363	0.358
Angry	0.268	0.244	0.397	0.408
Scared	0.245	0.252	0.408	0.405
Disgusted	0.206	0.177	0.225	0.204
Surprised	0.10	0.124	0.20	0.207

Table 3: Accuracies of various perception predictor functions f that are learned as neural networks or with linear regression, for the quadruped and robotic arm dataset are shown using the weighted R^2 score on test data.

As seen in Table 3 linear regression provided a similar fit and was much faster to execute. So we used linear regression to support design within our system.

5 USER-STUDY SURVEY

The post-study survey consisted of 5-pt Likert scale questions to capture user-experience of designing with the semantic interface. The questions elicited participant feedback about individual UI features (Figure 3) as well as the overall UI (Figure 4).

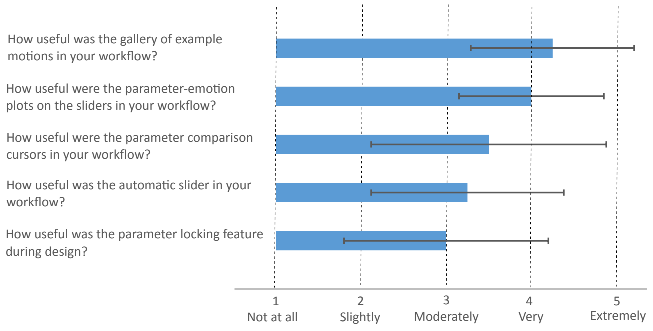

Figure 3: Participants provided feedback about individual UI features using 5-pt Likert scale questions. Average scores and standard deviation are shown for 12 participants.

We also asked the participants about how satisfied they were with their behavior designs (Figure 5). Their satisfaction was a function of the quality of guidance provided by the semantic system. Consequently, the participants were most satisfied with happy motion designs and least satisfied with surprised motion designs.

REFERENCES

[1] 2017. *Arduino*. https://www.arduino.cc/.
[2] Google Inc. 2017. *tensorflow*. Google Inc. https://www.tensorflow.org/.
[3] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

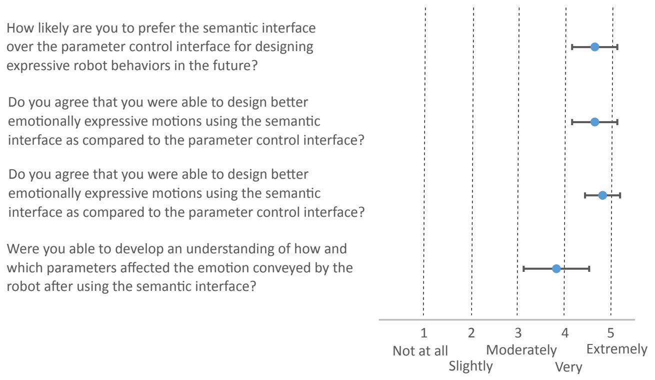

Figure 4: Survey questions and responses highlighting the participants' reception of the semantic interface. Average scores and standard deviation are shown for 12 participants.

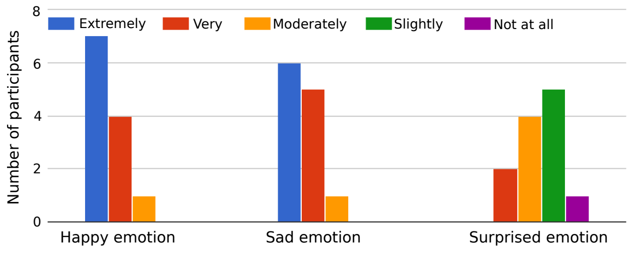

Figure 5: A histogram of number of participants vs. the participants' amount of satisfaction for their robot behavior designs is shown.

[4] Vittorio Megaro, Bernhard Thomaszewski, Maurizio Nitti, Otmar Hilliges, Markus Gross, and Stelian Coros. 2015. Interactive design of 3D-printable robotic creatures. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 216.

[5] Microsoft Research 2017. *TrueSkill*. Microsoft Research. http://trueskill.org/.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[7] Craig W Reynolds. 1987. Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH computer graphics* 21, 4 (1987), 25–34.

[8] Robotis 2015. *Dynamixel X Series*. Robotis. http://www.robotis.us/dynamixel-xm430-w210-r/.

[9] ROS 2018. *Unified Robot Description Format (URDF)*. ROS. http://wiki.ros.org/urdf.

[10] Wikipedia 2017. *The Free Encyclopedia*. Wikipedia. https://en.wikipedia.org/.

[11] Mehmet Ersin Yumer, Siddhartha Chaudhuri, Jessica K Hodgins, and Levent Burak Kara. 2015. Semantic shape editing using deformation handles. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 86.