Tradeoffs in Neuroevolutionary Learning-Based Real-Time Robotic Task Design in the Imprecise Computation Framework

PEI-CHI HUANG, University of Nebraska at Omaha
LUIS SENTIS, University of Texas at Austin, USA
JOEL LEHMAN, IT University of Copenhagen, Denmark
CHIEN-LIANG FOK, ALOYSIUS K. MOK, and RISTO MIIKKULAINEN,
University of Texas at Austin, USA

A cyberphysical avatar is a semi-autonomous robot that adjusts to an unstructured environment and performs physical tasks subject to critical timing constraints while under human supervision. This article first realizes a cyberphysical avatar that integrates three key technologies: body-compliant control, neuroevolution, and real-time constraints. Body-compliant control is essential for operator safety, because avatars perform cooperative tasks in close proximity to humans; neuroevolution (NEAT) enables "programming" avatars such that they can be used by non-experts for a large array of tasks, some unforeseen, in an unstructured environment; and real-time constraints are indispensable to provide predictable, bounded-time response in human-avatar interaction. Then, we present a study on the tradeoffs between three design parameters for robotic task systems that must incorporate at least three dimensions: (1) the amount of training effort for robot to perform the task, (2) the time available to complete the task when the command is given, and (3) the quality of the result of the performed task. A tradeoff study in this design space by using the imprecise computation as a framework is to perform a common robotic task, specifically, grasping of unknown objects. The results were validated with a real robot and contribute to the development of a systematic approach for designing robotic task systems that must function in environments like flexible manufacturing systems of the future.

CCS Concepts: • **Computing methodologies → Machine learning**; **Evolutionary robotics**; • **Computer systems organization → Robotics**;

Additional Key Words and Phrases: Cyber-physical system, neural networks, real-time performance, evolutionary computation, robotics

ACM Reference format:
Pei-Chi Huang, Luis Sentis, Joel Lehman, Chien-Liang Fok, Aloysius K. Mok, and Risto Miikkulainen. 2018. Tradeoffs in Neuroevolutionary Learning-Based Real-Time Robotic Task Design in the Imprecise Computation Framework. *ACM Trans. Cyber-Phys. Syst.* 3, 2, Article 14 (October 2018), 29 pages.
https://doi.org/10.1145/3178903

Authors' addresses: P.-C. Huang, Department of Computer Science, The University of Nebraska at Omaha, Omaha, NE 68182; email: phuang@unomaha.edu; L. Sentis, Department of Aerospace Engineering and Engineering Mechanics, The University of Texas at Austin, Austin, TX 78712; email: lsentis@austin.utexas.edu; J. Lehman, 710 8th Ave, San Francisco, CA 94118; email: jleh@itu.dk; C.-L. Fok, 464 Common St. #146, Belmont, MA 02478; email: liangfok@gmail.com; A. K. Mok, Department of Computer Science, The University of Texas at Austin, Austin, TX 78712; email: mok@cs.utexas.edu; R. Miikkulainen, Department of Computer Science, The University of Texas at Austin, Austin, TX 78712; email: risto@cs.utexas.edu.

2378-962X/2018/10-ART14 $15.00
https://doi.org/10.1145/3178903

1 INTRODUCTION

Although modern robots can perform complex tasks competently through hand-designed algorithms, it remains challenging [38] to create robots capable of completing mission-critical tasks in unstructured environments without complete dependence on a human (e.g., through teleoperation). Our framework to tackle this challenge is based on the concept of a "cyberphysical avatar," defined to be a semi-autonomous remote robotic system that adjusts to an unstructured environment and performs physical tasks subject to real-time constraints under human supervision [5].

The key scientific question of interest is what form and minimum degree of human supervision is required to enable a robot to perform a particular type of task. To answer this question, it is important to recognize the fact that the quality of robotic task performance is a function of at least two parameters: the amount of training the robot has had through machine-learning algorithms, as well as the tightness of the real-time task deadline that the robot is to meet. For example, if we give the robot one second to grasp an unknown object, the it is likely that the grasp will not be firm and reliable. The goal of our research is to perform a systematic investigation of the tradeoffs between the training effort, the resulting quality of the robotic task and the time the robot takes to perform the task. Understanding this tradeoff is essential to design robots that can function effectively in real time.

The specific robotic task we use for this investigation is the *grasping* of an unknown object by the robot. While robotic grasping has received significant research attention, the type of tradeoff investigation mentioned above is not. We use *Dreamer*, a humanoid torque-controlled mobile robot as our experimental platform. For training *Dreamer* to perform the grasping task, we adopt the NeuroEvolution of Augmenting Topologies (NEAT) machine-learning method. The design space of the robotic grasping task has three dimensions as follows: (1) the training effort, measured by the time used in running the NEAT algorithm to train *Dreamer* to perform the grasp; (2) the task completion time, defined by the time *Dreamer* has to perform the grasping task (to enforce the completion time constraint, a trajectory planner is used to compute the way-points for the trajectory that connects the initial and final configuration of *Dreamer* within the target completion time; the actual physical trajectory is realized by *Dreamer*'s on-board controller); and (3) the quality of the grasp is evaluated both in simulation and on a physical robot. Our training method relies on simulations modeled with GraspIt! [27], which is an open-source grasp simulation environment that models the targeted robotic hand and can evaluate grasp quality. Many future robotic applications require much faster training. For example, for flexible manufacturing a robot may be trained to assemble only a small number of a particular product (i.e., rapid small-lot manufacturing) before being retrained to make a different product. It is therefore important to understand how much training is enough for the robot to satisfactorily perform a task. Our investigation is relevant in light of recent successes in deep learning [18, 42]. Such successful application of deep learning involves multi-layer neural networks and depends on a well-chosen network topology and a sufficient number of training examples. Thus, an important question is how task performance improves as a function of the number of training cases. A plausible conjecture is that with a properly connected multi-layer network, the performance curve may exhibit fast improvement once past some critical number of input training cases. For example, a juggling robot may "suddenly" acquire the juggling skill once some basic hand-eye coordination "invariant" has been captured by the evolving neural network. To answer this type of question, the tradeoff study in this article should be useful. In particular, we use a framework from the area of real-time systems research called the imprecise computation model [23, 24] to help explore the boundary region of tolerance and find best effort techniques. Our work can be viewed as providing a realistic basis for some of the scheduling work done by the real-time systems community in the past two decades.

The remainder of this article is organized as follows. Section 2 reviews related work and describes neuroevolution, imprecise computation, and grasp quality measurement. Section 3 describes system integration and its architecture. Section 4 introduces our model of the whole-body compliant grasp of *Dreamer* and its hierarchical control structure. The learning approach is presented in Section 5, while Section 6 describes experimental results and their evaluation. Section 7 measures the grasping performance by applying imprecise computation. Finally, Section 8 and Section 9 conclude by reviewing remaining problems and future work.

2 BACKGROUND AND RELATED WORK

This section reviews previous machine-learning approaches to robotic grasping in Section 2.1, the neuroevolution method applied in the experiments in Section 2.2, the imprecise computation technique in Section 2.3, and grasp quality measurement in Section 2.4.

2.1 Robotic Grasping through Machine Learning

Impressive progress has been made in learning to grasp novel objects [17, 28, 32, 33, 37, 38]. To the best of our knowledge, previous methods use only simple hand models [28, 38] and are not directly applicable for the target hand in this work (i.e., *Dreamer* robot's hand, *Mekahand*). Also, transferring controllers from simulation to reality remains challenging [13, 21].

Related to the approach described here are previous artificial neural networks (ANNs) approaches that simulate arm kinematics [29, 31, 34]. Other approaches use reinforcement learning techniques [15, 47] to explore search spaces optimally for control strategies, learning from demonstration (LfD) [1] to improve grasping capability, and partially observable Markov decision processes (POMDP) [7, 8] to choose optimal control policies. Yet only a few of these methods were tested in the real world.

2.2 Neuroevolution

Neuroevolution (NE) is an approach where an evolutionary algorithm is applied to learn the structure of an ANN, its connection weights, or both [44]. Compared with other machine-learning methods, neuroevolution is unique in two main ways.

First, most other learning methods are supervised, i.e., they learn behavior that approximates a given set of examples [6]. It is important that such examples are carefully chosen to ensure that the training process results in learning a function that smoothly interpolates between them. For instance, in robotic grasping, a training set consists of grasping situations paired with the corresponding optimal grasping behavior. Because optimal behavior is often not known, it is unclear how such examples can be produced to cover representative situations well. In contrast, neuroevolution is a reinforcement learning method, and as such it does not require training examples where ideal behavior is known. Second, neuroevolution does not rely on complete state information. Other methods that are designed to learn under sparse reinforcement, such as Q-learning (or value function learning in general) often assume that the current state of the system is completely known. However, if objects are occluded or situation varies dynamically, then it is difficult for such methods to differentiate between possible situations, because the observed values of actions cannot be associated with the correct underlying state. Neuroevalution solves the problem by evolving recurrent connectivity; recurrence establishes memory that make it possible to distinguish between states.

One complication in applying neuroevolution to a complex domain like robotic grasping is that the ideal network topology (i.e., how many neurons compose the network and how are they interconnected) is not a known *a priori*. Because the depth image input contains many low-level features (i.e., pixels), a fully connected network with many hidden neurons may have an intractable number of parameters to tune. This motivates the NEAT [44] method which is a popular method for evolving both network topology and connection weights. With NEAT, the ideal network topology needs not be known *a priori* but is discovered automatically as part of evolution.

2.3 Imprecise Computation

In real-time applications (e.g., safety-critical applications), it is difficult for every critical task to meet its deadline. Imprecise computation is a scheduling technique that reduces the amount of time used on a job by means of sacrificing levels of quality of service (QoS) [23, 24]. If the best desired quality of results cannot be obtained, then imprecise computation decreases the QoS to make it possible to meet timing constraints of real-time tasks while still keeping the quality within an acceptable range. When the system cannot produce accurate results in a timely manner, the graceful degradation can be achieved by providing users with an approximate quality of acceptable results to prevent timing failures. For the real-time robotic task design in the imprecise computation framework, consider a grasping task, implementing two parts: a mandatory part that the task must complete before its deadline to achieve the minimum expected quality and an optional part that can be discarded at any time to improve the quality results by associating a reward with their execution.

2.4 Grasp Quality Measurement

Given an object, finding a suitable grasp configuration among the infinite set of candidates has been studied extensively in the robotics community over the two decades [28, 36, 41, 43]. The algorithms of producing a feasible grasp require determining proper dexterous hand configurations (i.e., *Mekahand*) as well as contact points on the objects. Much previous grasping quality research focuses only on contact types and positions, ignoring hand geometry and kinematics. Other measures assume simple grippers. Roa and Suárez [36] reviewed and analyzed the performances of evaluation methodologies of grasp quality in the literature. One of the most popular quality measure approaches is to consider the force constraints, a grasping quality measure is defined as the externally largest perturbation wrench where the grasp must be capable of resisting in any direction of forces [4, 14]; the frictional grasps of mathematical basis has been proposed [26, 46] and is also used to predict grasping work [12, 27, 30]. A grasp is in balance when the summation of forces and torques is null and the fingers is applied in the object (external disturbances) [2, 22, 25]. The method of grasping quality is based on an analytic formulation to compute how friction occurring grasping acting on those contacts affects the space of forces and torques that can be applied to an object and further rank/decide what set of contacts are appropriate to grasp. This method was adopted in GraspIt! to measure grasp quality of the *Mekahand*, which is used in this article.

3 THE SEMI-AUTONOMOUS ROBOTIC SYSTEM

Having summarized the motivation for designing cyberphysical avatars, emphasizing the important contributions that they could make, we turn to present the actual architecture of a cyberphysical avatar, also called a semi-autonomous robotics system, used interchangeably in this article.

This semi-autonomous robotic system comprises a mobile dexterous humanoid robot *Dreamer* with its whole body control system, and devised machine-learning algorithms including awareness of the environment complexity and sensing unpredictable world, a real-time physical distribution network, and a series of cost-effective, real-time, and vision systems. The specific task explored in this work is controlling the *Dreamer* robot to approach and pick up a designated target object under remote human supervision in a real-time environment. The physical realization of the cyberphysical avatar has been implemented in the Human Centered Robotics Laboratory at the

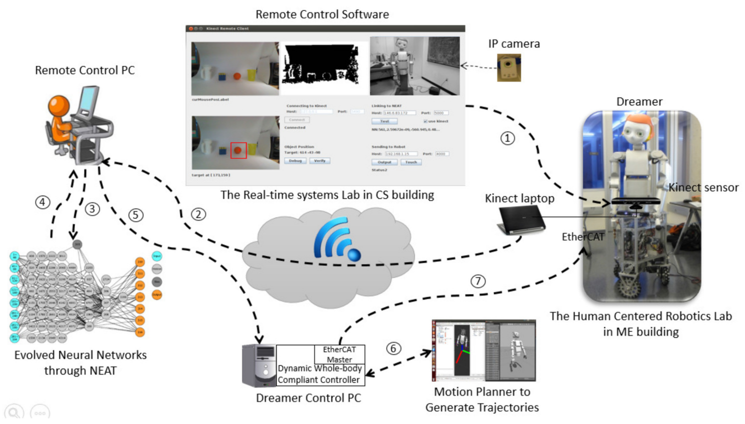

Remote Control Software
IP camera
Remote Control PC
Dreamer
Kinect sensor
The Real-time systems Lab in CS building
Kinect laptop
EtherCAT
Evolved Neural Networks through NEAT
EtherCAT Master
Dynamic Whole-body Compliant Controller
Dreamer Control PC
Motion Planner to Generate Trajectories
The Human Centered Robotics Lab in ME building
①
②
③
④
⑤
⑥
⑦

Fig. 1. The semi-autonomous robotic system. (1) A human supervisor connects to the Kinect laptop, (2) captures a depth image, and (3) parses the depth array to serve as input to an evolved ANN. (4) The neural network's output is interpreted as directions to control *Mekahand*'s position and orientation, and is sent to the supervisor. (5) The supervisor sends commands to manupulate *Dreamer* robot. (6) Motion planner generates a trajector from the initial state to the final state. (7) The controller of the wheeled humanoid avatar controls its body and arm to destination in unstructured environments. The system integrates real-time vision, neuroevolution as a training method, and control manipulator while skillfully reaching an object through the human–machine interface.

University of Texas at Austin, and the portable remote control user interface is located in another building nearby.

Figure 1 illustrates an overview of the semi-autonomous robotic system. *Dreamer* consists of a torso, two arms, two hands, an anthropomorphic head [40]. The *Dreamer* is equipped with torque and sensors to provide force compliant capabilities. A desktop PC running Ubuntu Linux with the RTAI Real-time Kernel executes the models and control infrastructure to govern *Dreamer*'s behavior via EtherCAT serial ports. Two types of cameras are installed in the system. A Kinect camera connects to a laptop and is installed in front of the robot to capture images and depth information, and an IP camera is installed at the ceiling to capture *Dreamer*'s surrounding environment. The Kinect laptop connects to the avatar and sends images to the remote supervisor.

A grasping experiment is achieved as follows. First, the human supervisor directs the *Dreamer* robot with a command to grasp the desired object. The cyberphysical avatar communication software relays the human input and depth information to a neural network that has been evolved with NEAT. Recall that NEAT's role is to train a neural network in a simulator to produce the appropriate outputs for *Dreamer* to act on. To apply NEAT to learn where and how to grasp an object requires both training scenarios and a measure for evaluating performance. GraspIt! [27] provides the interactive simulation, planning, analysis, and visualization. The neural network (trained offline) outputs the appropriate positions and orientations to *Dreamer*, which then moves towards the destination and grasps the targeted object with its *Mekahand*.

4 DYNAMIC CONTROL OF HUMANOID ROBOTS IN UNSTRUCTURED ENVIRONMENTS

Dreamer's upper body consists of 3-dof torso, 7-dof arms, and a 12-dof *Mekahand*, as shown in Figure 2. The 3-dof torso has one unactuated joint that is coupled with the waist joint. The hand also has five actuated joints and seven coupled unactuated joints, shown in Figure 2. To simplify

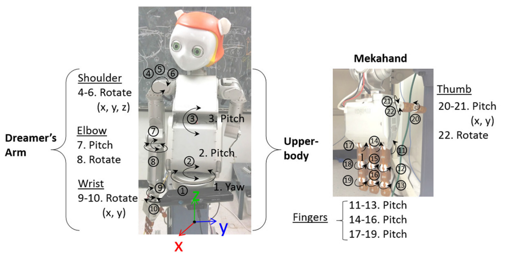


Fig. 2. *Dreamer*'s upper body and the *Mekahand*. *Dreamer* contains 3-dof torso (1-3), a 7-dof arm (4-10), and a 12-dof *Mekahand* (11-22). Each unactuated/actuated joint is coupled with another joint. Since many DOFs increase in difficulty of *Dreamer*'s balance control while grasping, it is necessary to design a skill modeling and dynamic control of *Dreamer*.

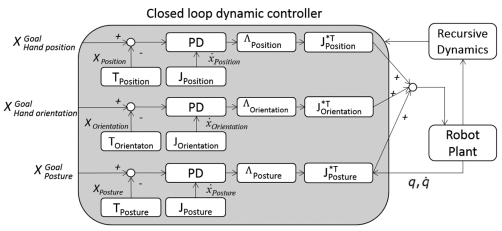


Fig. 3. The designed whole-body compliant controller (WBC). The tasks of hand position/orientation and the posture of *Dreamer*'s upper-body are combined to perform a grasping skill. The feedback/feedforward control policies contribute to the closed-loop dynamic controller. The design can effectively utilize dynamic and contact models in unstructured environment.

the controller, we divided the controller into one for controlling the body and the arm and the other for controlling the hand.

To control the body and the arm together, skill modeling and dynamic control of the robot are necessary. The prioritized whole-body compliant controller (WBC) is used for our purpose [40]. In WBC, first an objective is set and then a task is defined by a Jacobian [39] to derive the relations between the robot's 10-dimensional joint spaces and the M-dimensional operational space. The controller is derived from the following constrained system dynamics equations:

$$A\ddot{q} + b(q, \dot{q}) + g(q) + J_c^T \lambda = U^T T, \tag{1}$$

where A is the mass matrix of the system, q is the joint coordinate vector, b is the torque caused by Coriolis and Centrifugal effects, g is the torque caused by gravity, J_c is the constrained Jacobian, λ is the Lagrangian multiplier that describes the constrained joints, U is the actuation matrix, and T is the torque input to the system. The reason why the constrained Jacobian and the Lagrangian multiplies are shown in the system is to model the underactuated torso and the transmission constraint. The body joints 1 and 2 are coupled together. Therefore, we can specify the constraint as follows:

$$\dot{q}_1 - \dot{q}_2 = 0, \tag{2}$$

$$J_c \dot{q} = 0, \tag{3}$$

$$J_c = \begin{bmatrix} 0 & 1 & -1 & 0 & \cdots & 0 \end{bmatrix} \in \mathbf{R}^{1\times 10}. \tag{4}$$

We can take the constrained mass matrix Λ_c, the dynamically consistent generalized inverse of J_c, and the constrained null space N_c to derive the constrained dynamic equation as follows:

$$\Lambda_c \triangleq \left(J_c A^{-1} J_c^T\right)^+, \tag{5}$$

$$\overline{J_c} \triangleq A^{-1} J_c^T \Lambda_c, \tag{6}$$

$$N_c \triangleq I - \overline{J_c} J_c, \tag{7}$$

$$\ddot{q} = A^{-1} N_c^T U^T T. \tag{8}$$

Then, we can define task space specifications to derive the desired forces in the constrained dynamic systems. In the case of the position task that makes the end-effector (hand) approach the object, the task Jacobian is defined as

$$\dot{x} = J_{\text{position}}\dot{q}, \tag{9}$$

where x is the end-effector coordinate. The task Jacobian can describe the relation between the joint velocities and the coordinate system that a supervisor expects to control. The task Jacobian does not include the constrained dynamics, so we need to project this Jacobian to the constrained space and then generate the constrained task Jacobian,

$$J^*_{\text{position}} \triangleq J_{\text{position}}\overline{UN_c}. \tag{10}$$

The grasping skill, including posture, position, and orientation, is defined as a juxtaposition of multiple operational tasks to help translate between high-level goals, such as those provided by the planning algorithms, and the operational tasks. In the robot's environment, a skill is composed of the three tasks in Figure 3: hand position, hand orientation, and the posture of *Dreamer*'s whole body. The control structure can be expressed as

$$\tau_{\text{control}} = J^{*T}_{\text{position}}F_{\text{position}} + J^{*T}_{\text{ori}}F_{\text{ori}} + J^{*T}_{\text{posture}}F_{\text{posture}}, \tag{11}$$

where F_{position}, F_{ori}, and F_{posture} are the force or impedance commands to control the hand, and J^{*T}_{position}, J^{*T}_{ori}, and J^{*T}_{posture} are the whole-body task Jacobians [39]. The grasping process consists of three behaviors, among which the *Mekahand* position and orientation have higher priority than the whole-body posture task, because the latter is of secondary concern in the context of grasping.

The feedback control policies for the entire controller are shown in Figure 3, which depicts the closed-loop dynamic controller. In the figure, the Goal Hand Position/Orientation is a compliant hand position/orientation that enables the robot to reach a designated position/orientation. The Goal Posture control exploits the remaining DOFs to stabilize self-motions. The proposed feedback/feedforward control laws are

$$\begin{aligned} F_{\text{position}} &= \Lambda^*_{\text{position}}(-k_{p,\text{ position}}e^{\text{goal}}_{\text{position}} - k_{v,\text{ position}}\dot{x}_{\text{position}}) \\ &\quad + p_{\text{position}}, \qquad (12) \\ F_{\text{orientation}} &= \Lambda^*_{\text{ori}}(-k_{p,\text{ ori}}e^{\text{goal}}_{\text{ori}} - k_{v,\text{ ori}}\dot{x}_{\text{ori}}) + p_{\text{ori}}, \qquad (13) \\ F_{\text{posture}} &= \Lambda^*_{\text{posture}}(-k_{p,\text{ posture}}e^{\text{goal}}_{\text{posture}} - k_{v,\text{ posture}}\dot{x}_{\text{posture}}) \\ &\quad + p_{\text{posture}}, \qquad (14) \end{aligned}$$

where $\Lambda^*_{\text{position}}$, Λ^*_{ori}, and $\Lambda^*_{\text{posture}}$ are the inertial matrices projected in the manifold of the constraints; $e^{\text{goal}}_{\text{position}}$, $e^{\text{goal}}_{\text{ori}}$, and $e^{\text{goal}}_{\text{posture}}$ are feedback error functions; k_p, k_v are gain matrices; and p_{position}, p_{ori}, and p_{posture} are gravitational terms. This structure is a derivation of the previous work on compliant whole-body control [40].

Since our designed control structure can effectively use dynamic and contact models of the physical robot in its environments, it is able to optimize the process of approaching and grasping objects simultaneously and to achieve precise tracking of forces and trajectories within the contact conditions. Thus, the grasping skill is acquired through neural network described next.

5 ACQUIRING GRASPING SKILLS THROUGH NEUROEVOLUTION

The approach in the article applies reinforcement learning to facilitate learning high-level behaviors that can be then invoked by a human operator. In particular, neuroevolution algorithms have proven effective in domains with low-level continuous features that are characteristics of the problem here, i.e., learning to grip objects given depth sensor information. Thus, this section introduces our approach [9], which is based on applying NEAT to the GraspIt! simulation environment. Section 5.1 introduces the grasping learning approach; Section 5.2 then describes the learning process. Finally, Section 5.3 presents the tradeoffs evaluation in the imprecise computation framework.

5.1 Grasp Learning Approach

Our approach takes inspiration from Kohl et al. [16], who showed that neuroevolution can develop effective automobile warning systems from only low-level sensor input (i.e., pixels) taken from a digital camera. A similar vision-based feature extraction approach is applied here, where through neuroevolution the *Mekahand* robotic arm learns appropriate hand positions and orientations for grasping. Such learning is enabled by interacting with objects in the GraspIt! simulation environment, which is described next followed by the approach to measure grasping quality and determine a visual bounding box for grasping.

5.1.1 GraspIt! Simulation Implementation. To apply neuroevolution to learn where and how to grasp an object requires both training scenarios and a metric for evaluating performance. GraspIt! [3] facilitates simulating the *Mekahand* robot in representative grasping tasks and aids in measuring the quality of resulting grips.

GraspIt! only provides a rough *Mekahand* model, so we extended the simulator to better model it. In GraspIt!, the *Mekahand* is defined by one DOF for each knuckle in each finger, with an additional DOF for the thumb's rotator. The mechanics of this model are modified here to augment two aspects of the simulation. First, controlling the wrist is not modeled by default, but is an important DOF. Therefore, a wrist component was added to the *Mekahand* model supplied by GraspIt!. Second, most of the DOFs in the real *Mekahand* are not actuated, although they are modeled as actuated in the GraspIt! simulation. Each finger of the real *Mekahand* consists of three joints that are all connected by a single rubber tendon. Thus when the finger curls, all three knuckles curl in unison. Therefore, the torques in GraspIt! were adjusted such that the set of torques given to a single finger are equivalent to the torques initiated by stretching the rubber tendon in the real robot.

GraspIt! uses a quaternion to represent the rotation of a three-dimensional (3D) object. Since our learning output applies axis-angle representation in a 3D Euclidean space. Our implementation automatically translates the quaternion into the axis-angle representation in a 3D Euclidean space for the output.

5.1.2 Measuring Grasp Quality. An evolutionary search optimizes a fitness function that measures the quality of candidate solutions. Because robust grasping behaviors are desired in this experiment, an important consideration is how to measure the quality of grasps appropriately. Recall that the approach for grasp measure [2, 22, 25, 26] was applied in GraspIt! to measure grasp quality of the *Mekahand*. Given a 3D object and posture of the *Mekahand*, their measure can accurately identify the types of contact points between the links of the hand and the object and compute the grasp's quality. Figure 4 illustrates the score for each grasp when applied to different objects (a single cylinder, sphere, cube, and mug) from the different positions and orientations in ascending order. The grasp quality is -1 if the *Mekahand* just touches the object; otherwise, the quality is larger than zero. This grasp quality metric can yield different scores, which can be used to score each grasp for machine learning.

5.1.3 Visual Bounding Box. In the experiment, ANNs through exploration learn how to grasp objects by integrating information from a high-dimensional depth image provided by a Kinect sensor. To better focus on the most important features of the depth image, a bounding box strategy was implemented, thereby lessening the dimensionality of the robot's computer vision processing.

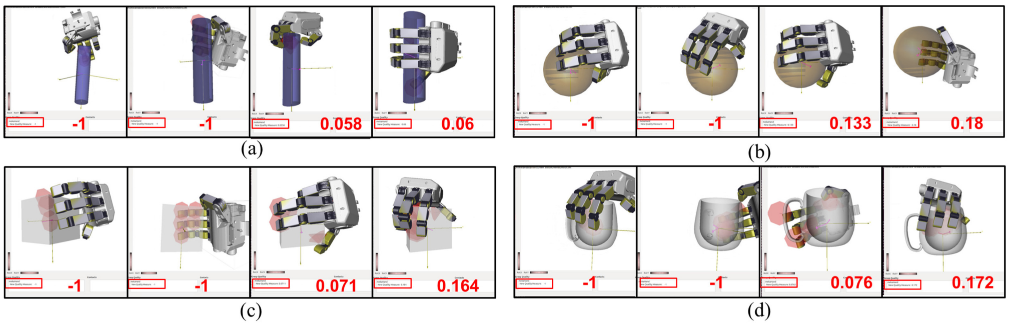


Fig. 4. Measuring grasp quality. One grasp acting on a single (a) cylinder, (b) sphere, (c) cube, (d) and mug is represented by different scores based on the *Mekahand*'s position and orientation. The grasp quality is -1 if the *Mekahand* just touches the object; otherwise, the quality is larger than zero. This quality metric can be utilized to score each grasp for machine learning.

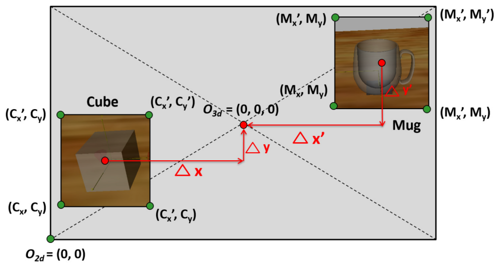


Fig. 5. Bounding boxes of a cube and mug, and the output shift offsets Δx and Δy ($\Delta x'$ and $\Delta y'$). Because all relative 2D coordinates of each object are known, an encompassing bounding box is generated centered on the desired object. This figure shows that the boundary range can be mapped to four coordinates. To simplify implementation, the position of the camera sensor is always set such that the origin O_{3d} $(0, 0, 0)$ in the GraspIt! scene is always in the center of 2D plane. A bounding box strategy can focus on the most important features of the depth image.

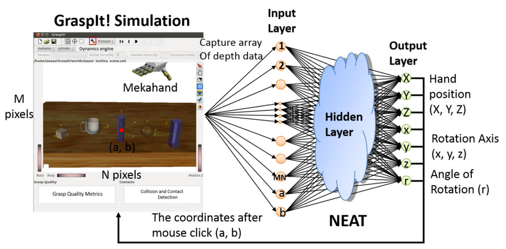


Fig. 6. Representation of the designed grasp controller network. The left side of the figure shows GraspIt! simulation environment; the right side of the figure shows a neural network receiving input consisting of depth data and the goal coordinate (a, b) on the GraspIt! visual input scene. The network has seven output nodes: hand position (X, Y, Z), rotation axis (x, y, z) and rotation angle (r). Note that NEAT can add internal hidden nodes as evolution progresses. The figure shows how to implement grasping experiments with NEAT in GraspIt!.

For each object extracted from the original scene, image data was considered only from within a supervisor-specified bounding box. The bounding box thus serves to minimize the number of irrelevant pixels considered and then simplifies the learning problem.

The training process with the bounding box method proceeds as follows. GraspIt! loads a scene, and then two mouse clicks from the user specify a rectangular bounding box that encompasses the object. In the simulated implementation, because all relative 2D coordinates of each object can be determined, an encompassing bounding box is automatically generated and centered on the desired object. For simplicity, all the computed bounding boxes have the same size. The boundary range can be mapped to four coordinates. For example, in Figure 5, a cube is chosen, so the bounding box is (C_x, C_y), (C_x', C_y), (C_x, C_y'), (C_x', C_y'). The depth array of the bounding box is then divided into $M \times N$ pixels that are given to the ANN being evaluated as input data.

To simplify the implementation, the position of the camera sensor is always set such that the origin O_{3d} $(0, 0, 0)$ in the GraspIt! scene is in the center of the 2D plane, as shown in Figure 5. Because the input is reduced to a small part of the depth image, after the ANN produces the output, the position of each object must be offset relative to the bounding box. For example, in Figure 5, for the cube, Δx and Δy should be added to the position of the output for mapping to the normalized origin position.

5.2 Learning Process

In learning process, we elaborate the NEAT method first. Then, combining neuroevolution with the grasping task requires specifying the input and output layers of the neural network, as well as a fitness function to evaluate grasps. A schematic description of the general framework combining GraspIt! and NEAT is depicted in Figure 6. The algorithm learns from reinforcement feedback based on only the measured quality of attempted grasps. In this way, evolution can discover solutions that work well even when the optimal behaviors are unknown.

5.2.1 The NEAT Method. Behaviors are evolved for robots that are controlled by ANNs. Thus, the NEAT method is suitable to underpin our experiments, because it is broadly utilized [19, 20, 35, 44, 45]. NEAT initially evolved by a small and simple population of networks, and *complexifies* the network topology into various species over generations, contributing to increasingly sophisticated and complex behavior. Here, a brief review of the NEAT methodology was provided; for comprehensive introductions see, e.g., Referenes [19, 44, 45]. The key features of the method described below: "To keep track of which gene is being added to new genes, a *historical marking* is uniquely assigned to each new structural component. During crossover, genes with the same historical markings are aligned, effectively producing meaningful offspring. *Speciation* in NEAT protects structural innovations by decreasing competition among different structures and network complexities, allowing newer and more complex structures to be adjusted. Networks are assigned to species according to the extent to which they share historical markers." In addition, the ability of NEAT to evolve increasingly complex ANNs is well suited to robotic grasping behaviors, which need potentially complex evolved structure.

5.2.2 Input and Output Layers of Neural Network. Each ANN evaluated by NEAT receives input data denoting the current state of the robot in its environment. It is thus necessary to encode such state information, which includes the position of the target object as well as information about the object's shape. To eliminate dependency on high-level human-provided features of the grasped object, the object's state is described by general low-level features provided by a depth map. In particular, each pixel in the depth information array is assigned a unique input node, as shown in Figure 6. In this way, the network can potentially learn to associate the state of an arbitrary object in an arbitrary environment with an appropriate grasping strategy.

Each ANN predicts where the object is and in what direction to grasp the object by outputting 3D hand positions and orientations. Note that each dimensional coordinate of the *Mekahand*'s position and orientation maps to one output neuron. Because the orientation is expressed in an axis-angle format (e.g., a 3D axis vector and one angle), the total dimensionality is seven, i.e., the ANN has seven output neurons. Evolution is initialized with ANNs with input nodes that are fully connected to at least a single hidden neuron and with the hidden node fully connected to the output neurons. Recall that during evolution, ANNs can accumulate additional connections and nodes through structural mutations that augment network topology.

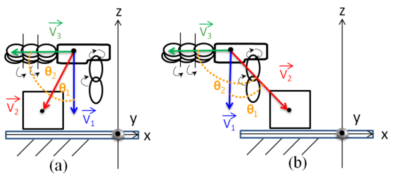


Fig. 7. An angle θ between the *Mekahand* and grasping object. $\overrightarrow{V_1}$ is a vector from the center of palm to the fingertip of the thumb; $\overrightarrow{V_2}$ is a vector from the center of palm to the center-of-gravity of the cube; $\overrightarrow{V_3}$ is a vector from the *Mekahand*'s rotation axis. (a) A good case where the palm's center is facing the target object; the sum of θ_1 and θ_2 is almost 90°. (b) A bad case where the palm's center is not facing the target object; the sum of θ_1 and θ_2 is larger than 90°. The conclusion is that because the center of plam facing towards an object can increase the grasping opportunity, the component was added to reward the fitness function.

5.2.3 *Grasping Fitness Function.* A key element of the experimental design is to construct a fitness function to guide the search process for an appropriate ANN grasp controller. The design of a fitness function is a critical factor for guiding successful evolution.

In particular, in this experiment, the fitness of a network n with respect to an object O has four components:

- f_1: Grasp quality metric Q, described in Section 5.1.2.
- f_2: The reciprocal of Euclidean distance $d(\overrightarrow{P_i}, \overrightarrow{O_i})$ between the hand position computed by the neural network $(\overrightarrow{P_i})$ and a desired object $(\overrightarrow{O_i})$. Note that $\overrightarrow{P_i}$ and $\overrightarrow{O_i}$ are vectors.
- f_3: The reciprocal of Euclidean distance $d(\overrightarrow{P_i}, \overrightarrow{S_i})$ between the hand position computed by the neural network $(\overrightarrow{P_i})$ and the actual hand coordinate after interacting with the environment $(\overrightarrow{S_i})$. Note that $\overrightarrow{P_i}$ and $\overrightarrow{S_i}$ are vectors.
- f_4: An angle θ between the *Mekahand* and grasping object. Let $\overrightarrow{V_1}$ be one vector from the center of the palm to the fingertip of the thumb; let $\overrightarrow{V_2}$ be the vector from the hand position to the center-of-gravity of the desired object; let $\overrightarrow{V_3}$ be the vector indicating the direction of the hand's axis of rotation. Let θ_1 (θ_2, respectively) be an angle between $\overrightarrow{V_1}$ and $\overrightarrow{V_2}$ ($\overrightarrow{V_2}$ and $\overrightarrow{V_3}$, respectively). To ensure that the center of palm always turns toward the object, the sum of θ_1 and θ_2 must be roughly around 90°. Figure 7(a) is one good case where the hand axis-angle is almost perpendicular to the object. Figure 7(b) is one bad case where the palm of hand is not orientated toward the object. Here, $\overrightarrow{V_1}$ and $\overrightarrow{V_2}$ ($\overrightarrow{V_2}$ and $\overrightarrow{V_3}$, respectively) are normalized so that $||\overrightarrow{V_1}|| = ||\overrightarrow{V_2}|| = 1$ ($||\overrightarrow{V_2}|| = ||\overrightarrow{V_3}|| = 1$, respectively). The angle θ is the sum of θ_1 and θ_2 as follows:

$$\begin{aligned}\theta &= \theta_1 + \theta_2 = \theta(\overrightarrow{V_1}, \overrightarrow{V_2}) + \theta(\overrightarrow{V_2}, \overrightarrow{V_3}) \\ &= \frac{\arccos(\overrightarrow{V_1} \cdot \overrightarrow{V_2})}{||\ \overrightarrow{V_1}\ ||||\ \overrightarrow{V_2}\ ||} + \frac{\arccos(\overrightarrow{V_2} \cdot \overrightarrow{V_3})}{||\ \overrightarrow{V_2}\ ||||\ \overrightarrow{V_3}\ ||}.\end{aligned} \tag{15}$$

Thus, the fitness function f of a network n is defined as follows:

$$\begin{aligned}f &= f_1 + f_2 + f_3 + f_4 \\ &= \gamma Q + \frac{\beta}{d(\overrightarrow{P_i}, \overrightarrow{O_i}) + \alpha} + \frac{\lambda}{d(\overrightarrow{P_i}, \overrightarrow{S_i}) + \epsilon} + f(\theta).\end{aligned} \tag{16}$$

where α, β, γ, λ, and ϵ are constants chosen to balance the various parameters. Note that

$$f(\theta) = f(\theta_1 + \theta_2) = \begin{cases} \omega, & \text{if } 85^\circ \leq \theta \leq 95^\circ \\ 0, & \text{otherwise.} \end{cases}$$

During the initial phases of evolution, when the neural networks are mostly untrained, all networks may direct the *Mekahand* to grasp at positions where it cannot even touch the object. As a result, in early generation f_1 is often effectively zero. Thus in this stage, f_2, which rewards approaching the target object, is important for differentiating the fitness. After further evolution, when the hand can grasp the object, f_1 begins to dominate and the neural networks are ranked mostly by grasp quality. In addition, the third term f_3 is large if the *Mekahand* is not blocked by obstacles (e.g., objects other than the target object). Finally, the fourth term (f_4) rewards facing the palm of the robotic hand towards the target object. Parameters α, β, γ, λ, ϵ, and ω adjust the relative effects of those four terms. In this way, the described fitness function rewards ANNs first to learn to approach the object and then to grasp the object in an increasingly appropriate way. Algorithm 1 shows the fitness function in detail.

ALGORITHM 1: Computation of the Fitness Function

```
1: Input: $Q$ is the grasp quality after the execution of a single grasp, $\theta$ is the summation of $\theta_1$ and $\theta_2$, $\overrightarrow{P_i}$ is
   the predicted position of hand for grasping by the network, $\overrightarrow{O_i}$ is the coordinate of the selected object
   after the mouse click, $\overrightarrow{S_i}$ is the actual hand coordinate after interacting with the environment.
2: Output: A fitness evaluation of a single grasp.
3: Let $A_j$ be a set of 3D coordinates of objects in the environment, where $1 \leq j \leq n$;
4: for $j = 1$ to $n$ do
5:     $Dist_o = \min(Dist_o, \sqrt{\sum_{i \in x,y,z} (\overrightarrow{A_{j,i}} - \overrightarrow{S_i})^2});$
6: end for
7: $Dist_t = \sqrt{\sum_{i \in x,y,z} (\overrightarrow{O_i} - \overrightarrow{S_i})^2});$
8: if $(Q = 0) \parallel (Dist_o < Dist_t)$ then
9:     {*No graspquality or Mekahand is closer to other objects.*}
10:    $f_1 = 0;$
11:    $f_2 = \frac{\beta}{d(\overrightarrow{P_i}, \overrightarrow{O_i}) + \alpha};$
12: else
13:    $f_1 = \gamma Q$, where $\gamma \geq 10000;$
14:    $f_2 = k$, where $k \leq 1000;$
15: end if
16: $f_3 = \frac{\lambda}{d(\overrightarrow{P_i}, \overrightarrow{S_i}) + \epsilon};$
17: if $(85 \leq \theta)$ && $(\theta \leq 95)$ then
18:    if $Dist_t < 50$ then
19:        $f_4 = \omega;$
20:    else
21:        $f_4 = w, w < \omega;$
22:    end if
23: else
24:    $f_4 = 0;$
25: end if
26: return $sum = \sum_{i=1}^{4} fit_i;$
```

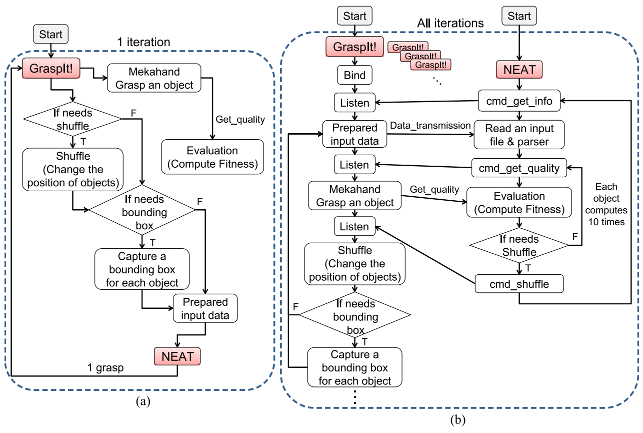


Fig. 8. The same computers were used to compare the sequential and parallel comparison methods. (a) The original sequential method. (b) The faster parallel method. The results show that with the original sequential implementation, the program only utilizes a single core, but after parallelizing the algorithm, the program can fully utilize four cores, and the experiment's runtime is shortened by a factor of three.

5.2.4 Reducing Training Time through Parallelization. The computational cost incurred by the *sequential* implementation of the fitness function computation is as follows. For one experiment, each generation consists of $\hat{o}$ ANNs, and each ANN is evaluated over $\hat{s}$ object combinations. Each object combination contains $\hat{b}$ objects, and each object is selected as $\hat{k}$ candidates to be an input. If one experiment runs for $\hat{g}$ generations, then the total number of independent training simulations in GraspIt! T is $\hat{o} \times \hat{s} \times \hat{b} \times \hat{k} \times \hat{g}$. In our experiments, $\hat{o} = 200, \hat{s} = 5, \hat{b} = 4, \hat{k} = 10, \hat{g} = 150$. Thus, $T = 6{,}000{,}000$. Therefore, a parallel strategy that dispatches different trials to all available computer cores is implemented to encourage computational efficiency. In particular, work is dispatched over a network to multiple GraspIt! processes that run on different computers. In this way, each CPU core in different computers can be fully employed, and the resulting multi-threaded implementation speeds up the evolution process.

Figure 8(a) illustrates the sequential method for each generation. To reduce execution time, the following computational steps were parallelized, as shown in Figure 8(b). The sequential method was redesigned as a producer-consumer pattern for parallel programming, which consists of two components: GraspIt! simulator and NEAT training. NEAT as producer generates the postures and enqueues into the queue for further processing. GraspIt! simulator as consumer dequeues the postures and evaluates grasping quality and then enqueues the measurements into the second queue. After all postures were generated and NEAT obtained all grasping qualities, the information would be fed into an ANN as an input for the next generation. First, three commands are defined: *cmd_get_info* is to get the depth array, *cmd_get_quality* is to get the quality for each grasp, and *cmd_shuffle* is to change the position and orientation of each object. Here, assume that four instances of GraspIt! are run and waiting for commands. Two kinds of threads are created: *Organism tasks* that use ANNs from NEAT's main process to generate grasping tasks and collect the resulting fitness score; and *GraspIt! tasks* that communicate with a GraspIt! process to send the output from an ANN in GraspIt! and receive the resulting grasp quality. The speedup achieved by such parallization depends on how many GraspIt! instances are running. To gain more computing power and speed up the training time, the producer-consumer was implemented as client/server architecture, where command and data are exchanged. Our results show that the runtime is accelerated by at least a factor of three.

5.3 Robotic Control in the Imprecise Computation

Our specific robotic grasping task has time constraint, so how to maximize the reward associated with the optional part of execution while satisfying all mandatory deadlines can be considered it as an imprecise scheduling problem. For example, objects localization from images processing, rough estimate of location from low-resolution images produces in time, whereas accurate location from high-resolution images takes longer time. In the robotic control, the mandatory part guarantees an approximate solution, so can be viewed as mandatory. However, the optional part depends on the precision of the solution and must be non-decreasing, so it can be varied by adjusting robotic arm speed and neural network training time. Ideally, each task would finish running its optional part, but these computations can be canceled when out of time. Therefore, this model is still not sufficient to provide a well-defined scheduling problem, because some mechanisms are required to decide which optional parts to carry out. This article fulfills the fundamental step to explore the boundary region of tolerance and find best-effort techniques to satisfy the minimum QoS requirement. With an increased number of subsequent deadline constraints, the objective is to derive the relationship between execution time and grasping trajectory accuracy. The tradeoffs analysis lays the foundation for the scheduling in the imprecise computation framework. For example, when we remotely operate *Dreamer* in the real world, the choices of completion time associated with each data point might be different in the experiments, as described in Section 7. In this case, the tradeoff mapping could assist the robot to make a decision on choosing different combinations to complete the task without exceeding the time constraint.

6 EXPERIMENTAL EVALUATION

This section describes the training and testing experiments. The design and parameters are presented in Sections 6.1 and 6.2. The first set of training experiments combines the four fitness components in different ways, as described in Section 6.3. The best combination is applied in the second and third sets of training experiments, which evaluate the benefit of applying a bounding box to focus the ANN's attention in Section 6.4. Fully trained ANNs are tested in simulation (Section 6.5) and also transferred to the real robot (Section 6.6).

6.1 Experimental Design

Because the raw depth data from the Kinect sensor is of high dimensionality, for practical purposes the array is first down-scaled. Before the input data are supplied to an ANN, the 640×480 pixel array was sampled to form a reduced 20×15 array. A larger scale was also tried, such as 40×30 and 80×60, but not only was the evolution process time-consuming but also the improvement over the results was not obvious at all. Therefore, we decided to shrink back 20×15 array. This smaller array was converted to gray-scale intensity values and then normalized between zero and 1; an example is shown in Figure 9. The input data also include a coordinate that represents the mouse click input from the user that specifies the target object. In the grasping experiments, the coordinate is chosen by randomly picking a different point on the target object in each trial. To increase accuracy in evaluating each network, they are each evaluated five times over different trials. That is, the robot attempts to grasp each target object five times, and the fitness value is the average over all the attempts. To preserve generality, the position and orientation of the objects for each evaluation are randomized.

The experiments are divided into two parts: training and testing. A collection of objects are divided into N separate classes, and for each class, ANNs are trained by NEAT to grasp objects from that class. For testing, the best neural network generated from training is further tested in

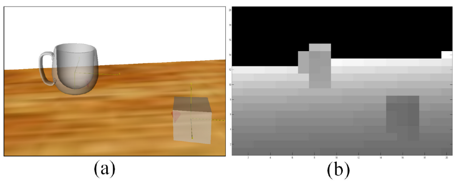


Fig. 9. Sample input data for training neural networks. (a) The RGB pixel data of the scene from the camera within GraspIt!. (b) The 20×15 depth data array supplied to the neural network as input. The depth data is normalized to a floating point number between [0, 1]. The purpose is that the original raw pixel data are high-dimensional, so a down-scaled data of the same data can be easily performed.

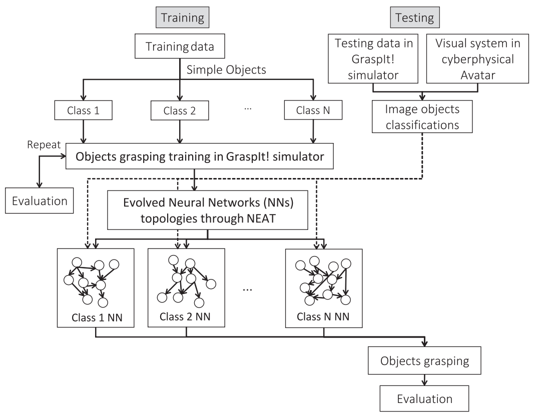

Training
Training data
Simple Objects
Class 1
Class 2
...
Class N
Repeat
Objects grasping training in GraspIt! simulator
Evaluation
Evolved Neural Networks (NNs) topologies through NEAT
Class 1 NN
Class 2 NN
...
Class N NN
Testing
Testing data in GraspIt! simulator
Visual system in cyberphysical Avatar
Image objects classifications
Objects grasping
Evaluation

Fig. 10. The flowchart of the training and testing processes for the experiments. In the training process, a set of objects are grouped into N separate classes, and each class produces a neural network through NEAT; in the testing process, the best neural networks can be applied in simulations and tested in a real scenario. These processes can examine if our approach can work.

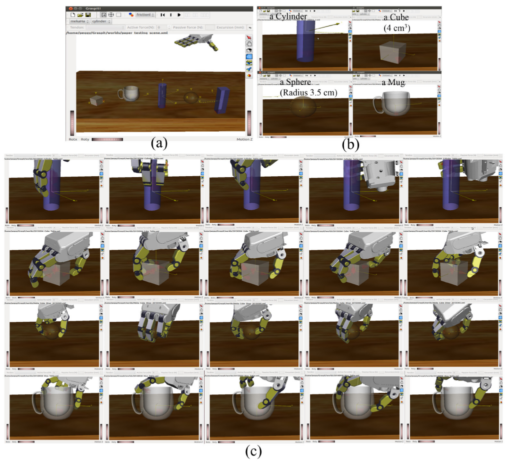


Fig. 11. Experimental scenarios. (a) A single cylinder, cube, sphere, mug, and cuboid with a dining table and the *Mekahand*. (b) Focus on a single target object each time. (c) The five results for each object during training. The conclusion is that the fitness function can guide *Mekahand* to grasp four different objects.

simulations over objects placed in different locations. A final test applies a real scenario from *Dreamer* to the evolved neural networks. The flowchart for training and testing is shown in Figure 10. All experimental parameters are described in Section 6.2.

6.2 Experimental Parameters of Neural Network

In the experiments, the population size was set to 150–200. Different values of the three parameters α, β, and γ of the fitness function (Equation (16)) were tried and tuned to guide evolution. The number of generations was 100. The coefficients for measuring compatibility for NEAT were $c_1 = 1.0$, $c_2 = 2.0$, $c_3 = 2.0$. The survival threshold was set to 0.2–0.3. The drop-off age was set to 10–20. Recurrent connections were disabled, because the task is not dependent on history. The probability of adding nodes and adding new connections to evolved ANNs were set to 0.2 and 0.3, respectively. Detailed description of these parameters are given in Reference [44].

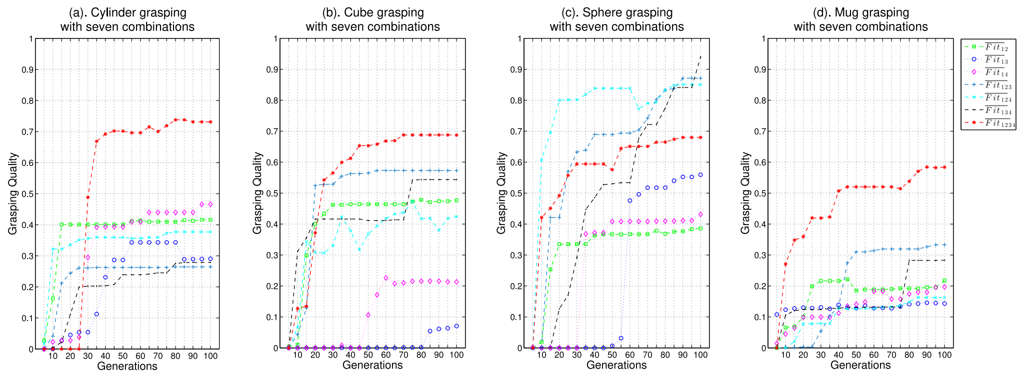


Fig. 12. Training performance with combinations of fitness components. The training scenario includes a cylinder, a cube, a sphere, and a mug on a dinner table, but the depth sensor focuses only on a single object for each experiment. The x-axis represents the number of generations while the y-axis represents the normalized grasping quality. These figures show how grasping quality increases over the course of evolution. To evaluate whether each of the four fitness component helps improve performance, (a)–(d) compare seven combinations of fitness components: (a) the results for grasping the cylinder, (b) for the cube, (c) for the sphere, and (d) for the mug. The conclusion is that $\overline{Fit_{1234}}$ produces the best grasping quality for (a), (b), and (d), while $\overline{Fit_{134}}$ provides the best one for (c).

6.3 Testing Combinations of Fitness Function Components

Training experiments are performed with four target objects plus a dining table to vary the scene distribution as shown in Figure 11(a) and (b). The goal is to gauge which combination of fitness function components (from Section 5.2.3) will yield the best performance. Figure 11(c) shows the five results of the fitness function for four scenarios through iterative training experiments.

Because grasp quality (f_1) is the most important performance metric, each case must contain f_1, so the combination of total cases is $C_3^3 + C_2^3 + C_1^3 = 7$. The following notation is used to refer to the section: $\{ \overline{Fit_i} \mid i \in \{12, 13, 14, 123, 124, 134, 1234\} \}$. As an example, $\overline{Fit_{134}}$ denotes the case with f_1, f_3, and f_4. The simulation environment performs a series of simulated grasps on one object on a dinner table for grasping evaluation. Figure 12(a)–(d) shows training results for grasping a single cylinder, cube, sphere, and mug, respectively. For the cylinder, cube, and mug, the maximum grasping quality f_1 is achieved through $\overline{Fit_{1234}}$ (i.e., each fitness component is helpful). However, Figure 12(c) shows that the maximum grasping quality f_1 for a sphere is achieved through $\overline{Fit_{134}}$, which suggests that f_2 does not contribute to better performance. Because the sphere is relatively small, placed between other objects it is sometimes blocked by other objects. Because its color is similar to the table, it is hard to distinguish it from the other objects. As a result, NEAT will be mislead by the simple f_2 distance metric.

6.4 Bounding Box Experiments

In the second set of experiments, four different training scenarios (without a bounding box) are performed with different target objects, similar to Section 6.3. Figure 13(a)–(d) shows training results for networks trained to grasp a single cylinder, cube, sphere, and mug. These figures show how fitness values increase over the course of evolution. Note that a larger fitness value implies better grasping quality; also, to differentiate the contributions of f_1, f_2, f_3, and f_4, each of these terms is normalized.

According to the best combination of the four fitness components from Figure 12(a)–(d), Figure 13(a), (b), and (d) differentiate the contributions of f_1, f_2, f_3, and f_4, and Figure 13(c)

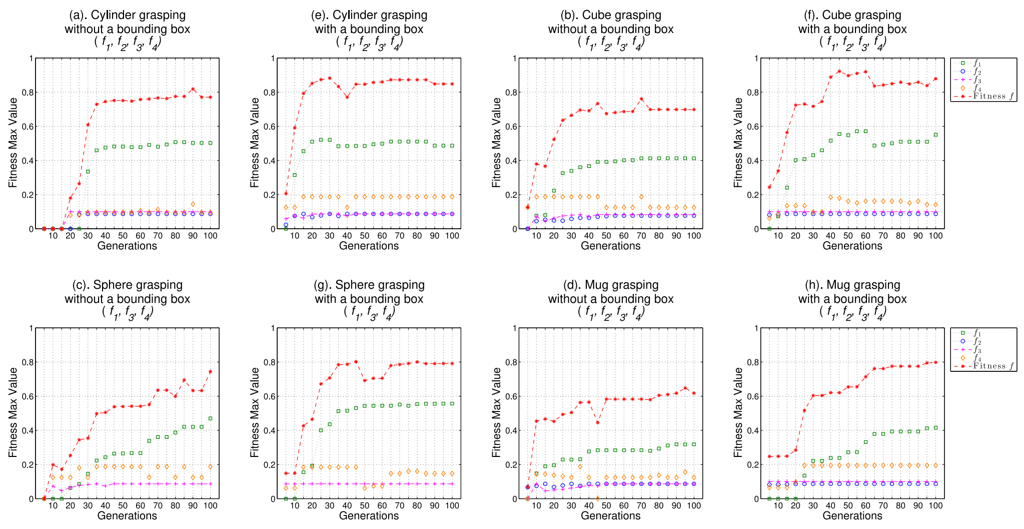


Fig. 13. Training performance with and without a bounding box. How fitness values increase over generations is shown for each experiment. Plots (a) and (e) show a scenario with a single cylinder on a table, (b) and (f) a single cube on a table, (c) and (g) a single sphere on a table, and (d) and (h) a single mug on a table. To evaluate whether a bounding box benefits performance, (a)–(d) have no bounding box, while (e)–(h) include the bounding box technique. The total fitness value is shown, as are the contributions from the three or four underlying normalized terms. Therefore, the bounding box increases performance, and all experiments eventually evolve ANNs able to grasp the objects.

differentiates the contributions of f_1, f_3, and f_4. Note that the maximum score f_1 can attain is 0.6, the maximum for both f_2 and f_3 is 0.1, and the maximum for f_4 is 0.2. Because f_2 and f_3 encourage approaching objects and avoiding obstacles, f_4 rewards orienting the palm toward objects that can serve as secondary objectives. These terms are therefore given lower weights than f_1, which measures the grasping quality itself and is thus the most important performance metric.

Because in practice only the best controller would be used, overall best-case results are presented here. To start evolution, individuals in the population are initialized with random weights and a simple topology (i.e., input nodes fully connected to one hidden node, and this hidden node fully connected to the outputs). Because randomly generated policies generally do not cause the robot hand to approach the target objects, low fitness scores are expected. In this stage, f_1 for all the networks is low, so the fitness scores of the networks are mainly determined by f_2 and f_3. These two terms guide evolution to produce networks that approach the objects without being blocked by obstacles. The f_4 component leads the *Mekahand* to the right orientation toward the object. In accordance with this explanation, Figure 13(a) shows that initially f_1 is smaller than f_2 and f_3. However, after 25 generations, f_1 becomes dominant. Then, after 90 generations, f_1 reaches its maximum value of 0.5, which means the *Mekahand* can grasp the object more accurately with proper position and orientation. Similar results appear in the other three experiments (Figure 13(b)–(d)). In Figure 13(b), after approximately 15 generations, f_1 sharply increases, and the total fitness value steadily increases to reach a maximum value of 0.7. In Figure 13(c), only f_1, f_3, f_4 are considered, but the fitness value remains around 0.7. In Figure 13(d), the fitness value only achieves 0.6. The reason is that it is difficult for the neural network to distinguish the mug object from the other objects. Comparing the four figures, it can be seen that the fitness scores of neural networks trained on the simple objects (Figure 13(a)—(c)) were larger than those trained

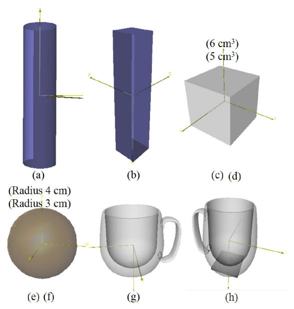


Fig. 14. Testing different sizes and textures of objects across novel locations and orientation. Shown in the figure are a cylinder, a cuboid, a cube, a sphere, a mug, and a plated mug. Note that the letters labeling each object correspond to similar labels in Table 1.

Table 1. Generalization Results of Grasping Objects at Novel Positions with Evolved Networks

#	**Objects**	**Class**	**without a Bounding Box**	**with a Bounding Box**
(a)	cylinder	N_A	52%	89%
(b)	cuboid	N_A	65%	81%
(c)	cube (6 cm^3)	N_B	69%	76%
(d)	cube (5 cm^3)	N_B	73%	82%
(e)	sphere (radius 4 cm)	N_C	71%	88%
(f)	sphere (radius 3 cm)	N_C	68%	80%
(g)	mug	N_D	71%	85%
(h)	plated mug	N_D	62%	74%
	Mean/Std		66.38% (±6.80%)	81.88% (±5.33%)

The results with a bounding box outperform the ones without a bounding box, which indicates that a bounding box is an effective way of increasing grasping performance.

on the more complicated one (Figure 13(d)). However, even in the more complicated scenario the networks all learned to approach the target objects and grasp them.

The third set of experiments tests evolution in the same four scenarios, but adds a visual bounding box that can focus the ANN on the most relevant information. The first experiment is shown in Figure 13(e). The fitness value gradually increases, and after five generations, the values are better than Figure 13(a), achieving a value of 0.9 after 45 generations. Similar results are seen in Figure 13(f)–(h). In Figure 13(f), the maximum fitness value is 0.92. Figure 13(h) illustrates that with a bounding box, more complex object configurations can still produce consistent results around 0.8.

This experiments suggest that the more complex the training scenario (i.e., the number of different kinds of objects in the scene), the more difficult it is to train the neural network. Furthermore, if a facet is obscured or the depth array values of an object are similar to the background, then even if the object to be grasped is simple, the training results are poor. However, applying the bounding box significantly improves the results in such cases.

6.5 Validating the Generality of Evolved Neural Networks

The training methodology results in neural networks evolved to grasp objects in simulation. To validate such networks, they were further tested in a variety of novel situations through GraspIt! (i.e., situating for which object was not explicitly trained). Most objects in the scenes were not seen at all during evolution or not placed in the same location, and their arrangement is new. The experiment thus measures how general the evolved solutions are. A successful case is recorded if the *Mekahand* can grasp the object; otherwise, it is recorded as a failure.

For this generality test, each object was tested 100 times. The grasping procedures were implemented under test conditions randomly placing the different sizes and textures of a cylinder, a cube, a sphere, and a mug, as shown in Figure 14, at different positions and orientations on the table. The evolved neural networks in 6.4 were labeled as Cylinder (N_A), Cube (N_B), Sphere (N_C), and Mug (N_D) and based on similar classification of objects, the most appropriate neural network was chosen for testing. The success rate in Table 1 compares the neural networks with the different objects. These results show that despite its simplicity, the proposed bounding box method still performs reasonably well in grasping novel objects. However, if the target object is too far from the center of the image frame, then the neural networks often perform unreliably, indicating the training process may need further refinement to deal with such boundary cases. Table 1 shows the best results from among all the experiments. Also, in some cases the *Mekahand* collides with objects while grasping, because many objects are placed on the table. A potential remedy is to decompose the movement into more steps to avoid such collisions. One way to do so would be to rely on additional input from the human supervisor.

6.6 Validating with Dreamer

Beyond simulated results, learned policies were also transferred to the physical world. A physical (i.e., not simulated) Kinect sensor was applied to capture object depth array information. This information was provided as input to an evolved neural network to guide *Dreamer* robot's grasp.

6.6.1 Kinect Sensor Implementation. To retrieve the Kinect sensor data and feed it into the system, the sample program regview provided by OpenKinect project[1] was modified. This program was enhanced to be run as a server that waits for the connection from the remote-control PC over the TCP/IP. Besides, it was tweaked to register the video format as FREENECT_DEPTH_REGISTERED. The reason is that in the Kinect sensor, the depth camera and the color camera are two separate sensors, which means their views are different. Only by doing so, the depth data will be projected to the view of the color camera. In this video mode, the depth data is in millimeters, and the pixel coordinates can be translated from (i, j, z) to (x, y, z) as follows:

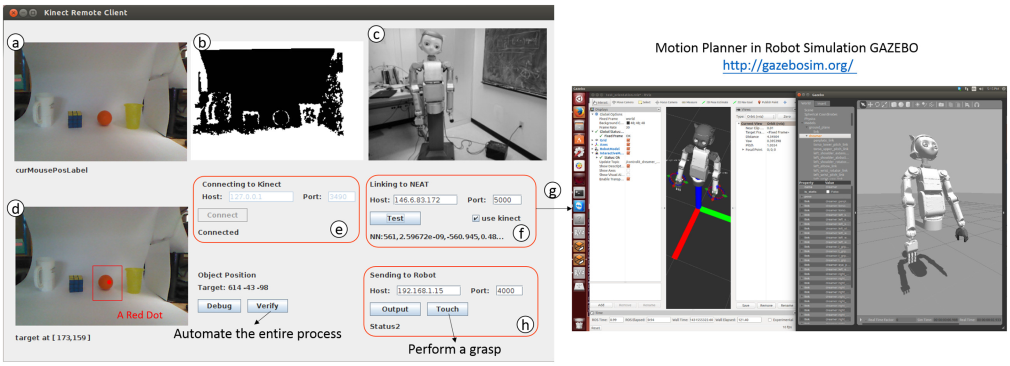
$$\begin{aligned} x =&\ (i - \text{width}/2) * (z + \text{minDistance}) \\ &* \text{scaleFactor} * (\text{width}/\text{height}), && (17) \\ y =&\ (j - \text{height}/2) * (z + \text{minDistance}) * \text{scaleFactor}, && (18) \\ z =&\ z, && (19) \end{aligned}$$

where weight and height are the images size. The x, y, z is a right-handed Cartesian system: With z-axis perpendicular to the Kinect image towards the image, x-axis points to the left, and y-axis points up. Before sending commands, the coordinates are transformed again to match *Dreamer*'s coordinates.

6.6.2 Remote Control Panel. Figure 15 shows a screen capture of the remote-control application for supervising *Dreamer*. The remote-control user interface shows the color images and depth images from the Kinect camera; the images from the IP camera are displayed in the third image panel at the top of the user interface.

To automate the high-level supervision of the grasping experiment, six commands was implemented on the remote control panel: Connect, Test, Output, Touch, Verify, and Debug, as shown in Figure 15. When the supervisor clicks on the Connect button, the computer connects to the Kinect sensor to capture depth information. Then, when the Test button is pressed, the depth array is provided to the evolved neural network as an input. After executing the neural network, its outputs are interpreted as coordinates and orientation of the hand for grasping the object. When the Output button is pressed, the results are sent to *Dreamer* and the robot is directed to approach the object. Finally, when the Touch button is pressed, *Dreamer* will grasp the object using the grasping information provided by the neural network. After *Dreamer* obtains this information, i.e., the grip orientation and position, the controller PC computes the distance between the *Mekahand* and the object, predicts the hand's trajectory, and approaches the object. Once the *Mekahand* is near the target object, the thumb and the three finger motors are synchronized to perform the grasp. A

[1] http://openkinect.org.



target at [173,159]

Connecting to Kinect
Host: 127.0.0.1 Port: 3490
Connected

Linking to NEAT
Host: 146.6.83.172 Port: 5000
use kinect [✓]
NN:561, 2.59672e-09, -560.945, 0.48...

Object Position
Target: 614 -43 -98

Sending to Robot
Host: 192.168.1.15 Port: 4000
Status2

Fig. 15. A screen capture of the remote-control software application for supervising the *Dreamer* robot. (a) Color and (b) depth images from the Kinect sensor. (c) The image from the IP camera. (d) An image snapshot taken when the user clicks on the color image. (e) A dialog for connecting to *Dreamer* through a computer network. (f) A dialog for inputting the captured depth array into an evolved ANN. (g) Use motion planner to obtain a trajectory. (h) A dialog for sending the orientations and positions from the ANN to *Dreamer* to control its grasp. The conclusion is that the grasping experiment can be implemented throught the remote control panel.

Verify button is provided to automate the entire process for convenience; the Debug button serves to aid in system debugging, providing coding logging information.

6.6.3 Transitioning to Physical Controller. An automated grasping platform was built to demonstrate this process. The networks evolved in simulation are transferred to this platform to evaluate them in a physical environment. To carry out an experiment, a human experimenter uses the control panel to choose a target either without or with a bounding box in the color image from the laptop screen with the Kinect sensor by clicking on it. After designating the target, a copy of the color image is copied to the target object panel, and a red dot is added on the image, indicating the position of the click. The depth data at that point is used to calculate the approximate position of the object to be grasped. This results specifies the grasping task for the robot to perform. Here, the grasping behavior was not evolved on the actual robot but was transferred from simulation. The video http://www.cs.utexas.edu/~peggy/rtss2015.html demonstrates grasping of novel objects from the simulation to the real *Dreamer* robot. In addition, Figure 16 shows screen captures taken from a proof-of-concept demonstration of grasping a tennis ball, a bottle, ball, a Rubik's cube, and a cup. *Dreamer* can successfully approach and grasp target objects when controlled by an evolved neural network.

Since these objects were not seen during evolution, the experiment demonstrates two achievements: (1) Learning transfers from simulation to the real world, and (2) it generalizes to grasp objects. Quantifying how well grasping works needs a metric for the assessment of the quality of a real grasp, so further work to incorporate real sensor data on the *Mekahand* (e.g., touch pressure) is ongoing.

7 THE REAL-TIME PERFORMANCE MEASUREMENTS AMONG TASK COMPLETION TIME, TRAINING EFFORT, AND GRASPING QUALITY

This section discusses whether string task completion deadlines can be met by applying imprecise computation to trade increased speed for decreased accuracy. That is, in some situations a faster yet less precise grasp may better satisfy the use case. When considering tradeoffs, the design space of the grasping task has three main dimensions: (1) the training effort, (2) the task completion time,

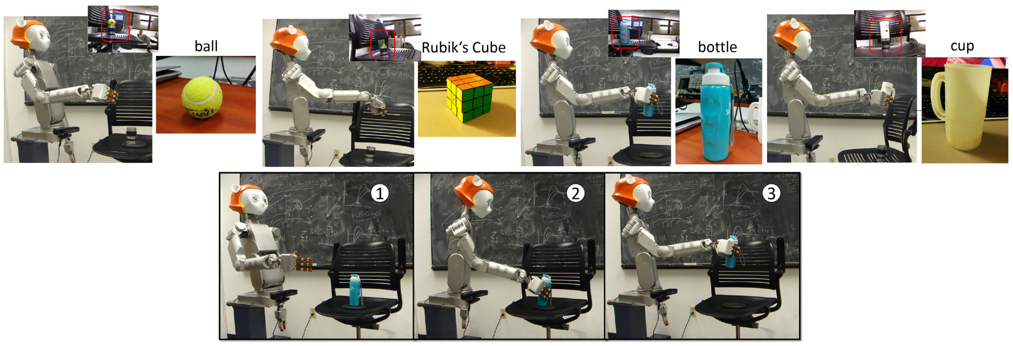


Fig. 16. Screen captures from videos (http://www.cs.utexas.edu/~peggy/rtss2015.html) demonstrating *Dreamer* grasping a ball, a bottle, a cube, and a cup through an evolved controller. Note that the small picture with red dots are the snapshots from Kinect sensor panel. The bottom snapshots labeled with (1)–(3) represent the object grasping process, from the initial, approach to grasp a bottle. The figures confirm that transferring results from simulation to reality is possible, and applying the approach generalizes to novel objects.

and (3) the grasp quality [10]. In Section 7.1, we first investigate how increasingly stringent time limits on computation reduce the accuracy of the robotic hand's approach trajectory. We then discuss the performance tradeoff between grasp quality and task completion time. In Section 7.2, we evaluate the tradeoff between training effort and grasp quality, and in Section 7.3, we measure the tradeoff between training effort and task completion time with the successful grasp.

7.1 Grasp Quality vs. Task Completion Time Tradeoff Evaluation

For the grasping task, trajectories may be denoted by the point-to-point positions and orientations of the end-effector as long as no collision occurs. This section focuses on the actual interaction between the *Mekahand* and its environment assuming that there is no collision.

In a grasping experiment, the initial starting point S^* is the current position and orientation of *Dreamer's* end effector. The human supervisor assigns an object to be grasped from the user interface panel; the evolved neural network automatically determines the final destination D^* and orientation of *Mekahand* and sends it to *Dreamer's* main controller. On command, *Dreamer* moves along the designated trajectory to approach and grasp the object, and then returns to the start position S^*. The actual trajectory of *Dreamer* is acquired by recording the position of the end-effector from forward kinematics calculations with joint positions. In controlling the movement of *Dreamer's* arm, we use the proportional-derivative (PD) controller in the Whole Body Control (WBC) algorithm. The position and orientation data are transmitted with a wireless system from the sensors to the control computer.

The first set of experiments measures the quality of the grasping trajectories versus various task completion times. In each experiment, *Mekahand* moves from S^* to D^* within a specified time interval of length ranging from 8s down to 0.5s. Each configuration was measured five times over different trials to obtain an accurate trajectory error estimation. An ideal trajectory was designed by a trajectory generation algorithm, and all experiments attempted to follow this trajectory, subject to different completion time deadlines. Each execution time was separately conducted five times and averaged the five trajectories. Due to space limitations, the entire experimental results are shown in Reference [11]. Figure 17 shows the difference between the ideal and actual trajectory over various task completion times. As expected, the trajectory closest to the ideal one is the one given the most time (i.e., 8s). Figure 17 shows that in general, the shorter the completion time

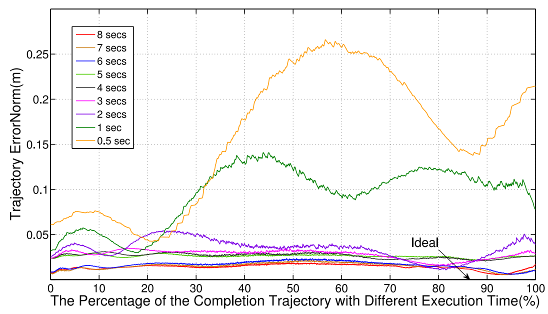


Fig. 17. Tracking trajectories by varying execution times ranging from 8 to 0.5s. The x-axis represents the completion of the trajectory while the y-axis represents the normalized trajectory error compared to the ideal trajectory. The highest error is found in the 0.5s trials, while the lowest error is found in the 8s trials. The trajectory error increases as the allowed time for execution decreases.

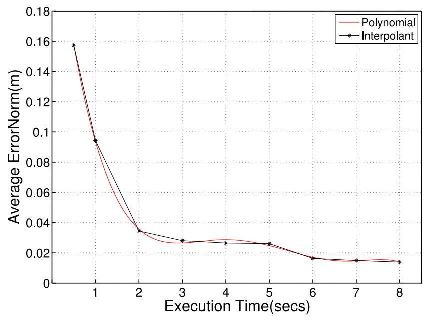


Fig. 18. Fitting the RMS data with linear interpolant and 5-degree polynomials. This shows inaccuracy is maximal when execution time is shortest (0.5s) but rapidly improves as the budget increases to 3s. Error decreases slightly between 3 and 6s and plateaus thereafter. This derivation line shows stability after 6s and can predict the error after 9s (nearly stable) below 0.02m.

deadline, the higher the trajectory error. It should be noted that the design goal is to contain the trajectory error so that the grasp action can succeed at the end of the trajectory.

To predict the probability of a success grasp, we can fit a statistical model to characterize the tradeoff between average trajectory error and task completion time. First, it is necessary to evaluate whether the five experiments are sufficient to represent ground truth. The standard deviation (STD) (root-mean-squared (RMS)) error for each task completion time respectively range from 0.00231(m) to 0.00009(m) (from 0.15742(m) to 0.01395(m)). Such a low STD indicates that the results do not vary much and can therefore serve as a reasonable basis to derive models of RMS error.

To find a well-fitting regression model, Table 2 shows the results of approximated RMS error and adjusted R^2 of power, Weibull, rational, Gaussian, and polynomial distributions. The trajectory errors are best modeled by a polynomial distribution of order five, as demonstrated by its lowest RMS errors and highest coefficient of determination. The best fit polynomial model function $f_{\text{model}}(x)$ is:

$$\begin{aligned} f_{\text{model}}(x) = &-0.000714 * x^5 + 0.002011 * x^4 \\ &+ - 0.02155 * x^3 + 0.1088 * x^2 \\ &+ - 0.26 * x + 0.2635. \end{aligned} \tag{20}$$

Figure 18 shows the RMS errors fit to a linear interpolation and polynomial distributions. The model shows the results approaches stability after 6s, even the experiments after 9s still can predict the error may be below 0.02m, which indicates the 9 results are sufficient to proceed with the following experiment. To investigate how the reduction of task completion time may jeopardize sufficient accuracy for effecting a grasp, 6s was chosen as the time-constraint for the second experiment.

The second set of experiments focuses on trajectory accuracy and latency delay within a given time constraint, i.e., 6s. The controller operation is the most time-consuming part of the practice, because many DOFs are considered in each step. In contrast, sending data and NEAT operation are relatively fast, only taking around 1 to 2s each. Therefore, a round-trip is given 14s. The

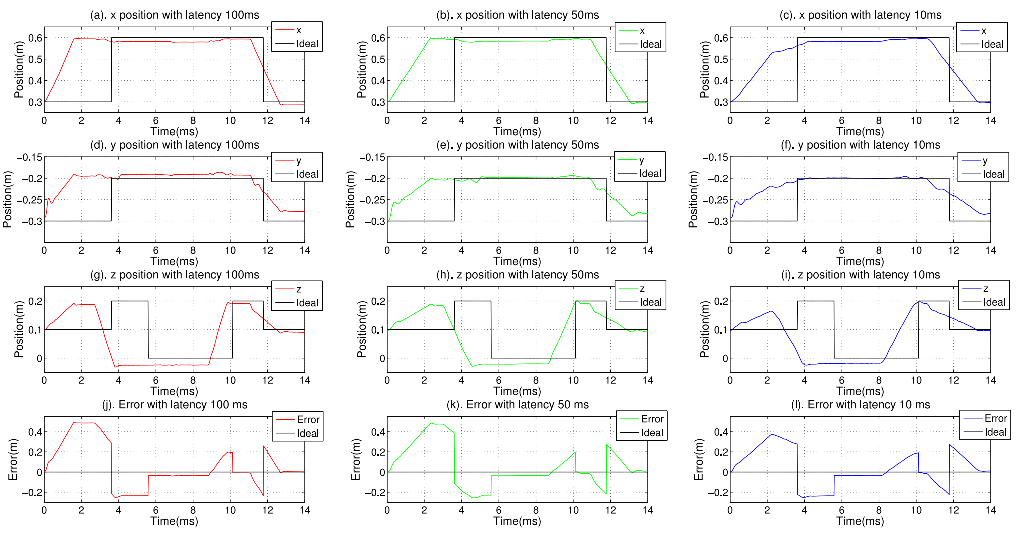


Fig. 19. ((a), (d), and (g)), ((b), (e), and (h)), and ((c), (f), and (i)) display the relationship between trajectory and latency delay (100ms, 50ms, and 10ms); (a)–(c), (d)–(f), and (g)–(i) show x, y, and z positions in 3D space. Figures (j)–(l) depict the relationship between the error and latency delay. The x-axis represents the given time; the y-axis in (a)-(i) denotes the actual and ideal positions and in (j)–(l) denotes the error. The error was calculated as the actual minus ideal trajectory. As expected, the variation with 10ms was the best.

latency delays are 100ms, 50ms, and 10ms. The longer the latency delay, the worse the performance. To complete the task within the limited time frame, it is hard to control *Dreamer* very well. Figure 19(a)–(i) depict the relationship between the position and time with 100ms, 50ms, and 10ms; Figure 19(j)–(l) display the relationship between error and time. Note that the error was computed as the actual minus the ideal desired trajectory. Figure 19, as expected, shows that the lowest latency delay (10ms) performs best.

7.2 Training Effort vs. Grasp Quality Tradeoff Evaluation

We first describe the experimental setup and then present a set of grasping results that relate the quality of grasping to the training effort (defined to be the time spent on searching for the best *Mekahand* configuration for effecting the grasp by the NEAT algorithm). To speed up the training computation, we apply a parallelization strategy and run the NEAT algorithm with four multi-core computers.

We evaluate the effectiveness of our learning approach by conducting the two following sets of experiments. For the first experiment, the computational cost incurred by the *sequential* implementation is described under Section 5.2.4. The *parallel* strategy that dispatches different trials to all available computer cores is implemented to increase computational efficiency. In particular, work is dispatched over the network to multiple GraspIt! processes (36 threads) that run on four computers, whose specifications are detailed in Table 3.

The first set of simulated experiments involve scenarios with different target objects, as explained in Section 6.3. Specifically, Figure 20(a)–(d) show the training results for networks trained to grasp a single cylinder, a single cube, a single sphere, and a single mug. Because the goal is to use only the best controller for the actual grasp, only overall best-case results are presented here. For each 10 minutes, the best-case value was recorded, and we repeated the same experiments three times, and the average values (the dots) and the fitting models are shown in Figure 20(a)–(d). These

Table 2. A Comparison Is Shown of Five Well-fitting Statistical Models of How RMS Error Varies with Execution Time

Fitting model	**RMS errors**	**adjusted R^2**
Power (2 terms)	0.006670	0.9812
Weibull	0.006179	0.9838
Rational (degree 5)	0.003164	0.9958
Gaussian (2 terms)	0.002755	0.9968
Polynomial (degree 5)	0.002308	0.9977

The best fitting model is a polynomial of degree five, which predict the expected RMS error at 9s.

Table 3. The Experimental Machine Specifications

CPU	**GPU (NVIDIA GeForce)**	**Memory (DDR3)**
Intel Core i7 @ 3.6GHz 4 Core/ 8 Threads	GTX 760	16GB @ 1,333MHz
Intel Core i7@ 3.0GHz 8 Core/16 Threads	GTX 980	32GB @ 2,133MHz
Intel Xeon@ 2.67GHz 4 Core/8 Threads	GTX 285	8GB @ 1,066MHz
Intel Core i7@ 2.8GHz 2 Core/4 Threads	GT 520M	8GB @ 1,333MHz

All of the CPU cores among the four computers can be fully employed by using the multi-threaded implementation, which significantly speeds up the evolution process.

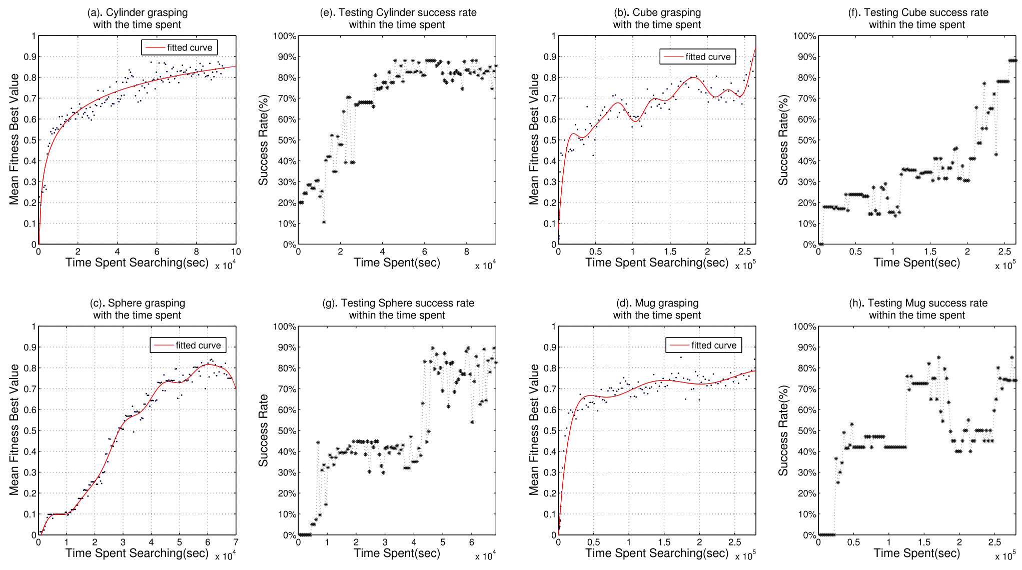


Fig. 20. Training and testing performance with the different time spent searching. How fitness values increase over time is shown for each experiment. Plots (a) and (e) show a scenario with a single cylinder on a table, (b) and (f) a single cube on a table, (c) and (g) a single sphere on a table, (d) and (h) a single mug on a table. To evaluate whether Algorithm 1 benefits performance, (a)–(d) are training results, while (e)–(h) are testing outcomes. Fitness generally improves as training progresses. The conclusion is that although accuracy generally benefits from increased time, increased time may sometimes also there is risk overfitting to the training cases, as shown by the intermediate trough in test performance in (h).

figures show how fitness values increase over evolutionary search. The underlying assumption is that larger fitness values implies better quality grasping.

To start the neuroevolution simulation, individuals in the population are initialized with random weights and a simple topology (i.e., in our case input nodes fully connected to four hidden nodes that are fully connected to the outputs), as shown in Figure 6. Because randomly generated policies generally do not cause the robot hand to approach the target objects, low fitness scores

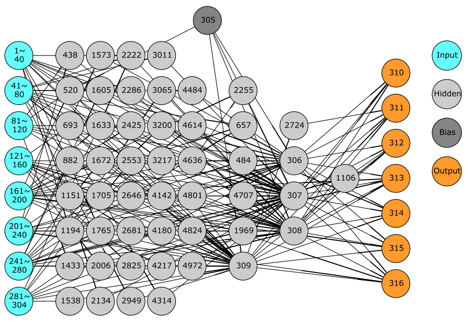


Fig. 21. An example of a grasping network evolved by NEAT after 100, 000s. The 304 Input nodes (20 × 15 pixels, plus a set of coordinate inputs, and object scale inputs) are located on the left, while the seven output nodes are located at the right side. The light gray nodes in between are the evolved hidden nodes. The conclusion is that as time goes by, NEAT searches through increasingly complex networks to find one able to match the complexity of the grasping problem.

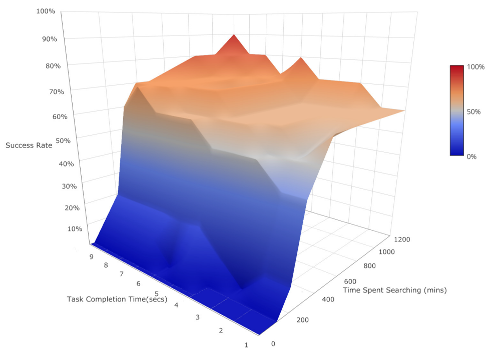


Fig. 22. The grasp success rate vs. task completion time vs. training time. The x-axis represents the task completion time (1-10s), the y-axis indicates the time spent searching (0-1,000 minutes), and the z-axis shows the resulting grasp success rate. The results indicate that a 70% successful grasp rate could be achieved after around 600 minutes of training effort, which helps highlight what task completion time is sufficient to successfully grasp given enough training effort.

are expected. As time passes, Algorithm 1 leads the *Mekahand* to the right position/orientation toward the object. Accordingly, Figure 20(a), a cylinder, shows that initially the fitness value is low, and then after 80,000s, it reaches 0.8. Similar results appear in the other three experiments in Figure 20(b)–(d) (the cube, the sphere, and the mug). However, in Figure 20(b) and (d), (the cube and the mug) the ANNs took 170,000s to get 0.8. In Figure 20(c), the sphere, the ANNs used 50,000s to achieve 0.8. Two interesting results are noteworthy here. First, in the case of the sphere (Figure 20(c)), fitness dropped to 0.7 after 60,000s of training; one explanation is that sometimes overfitting occurs, which reduces accuracy. Second, in the case of the mug (Figure 20(d)), the fitness value plateaus after 100,000s. Such plateaus in evolutionary search often reflect a local optimum; the mug is likely difficult to distinguish from the other objects. Comparing the four figures, the ANNs trained on the "simple" objects (Figure 20(a) and (c), the cylinder and the sphere) were much easier trained than networks trained on objects of more complicated geometry (Figure 20(b) and (d), the cube and the mug). Figure 21 shows an example of an ANN evolved for 100,000s to grasp the cylinder.

The second set of experiments applies NEAT on the same four target objects but in novel situations unseen in training, as discussed in Section 6.5. The one-hundred best-trained neural networks that result from different search times were picked and tested with the same input. Each ANN was tested 200 times. The success rates in Figures 20(e)–(h) compare generalization of neural networks versus time spent in training. The first experiment in Figure 20(e), the cylinder, shows that as the search time increases, the success rate improves and after 45,000s, the success rate reaches and stabilizes at around 80%. Similar results are seen in Figure 20(f) and (g) (the cube and the sphere). In Figure 20(f) and (g), the maximum success rate is 88%. However, in Figure 20(g), the sphere, the success rate continues to oscillate after 40,000, even after the training fitness value has stabilized around 0.8. This may reflect the difficulty in finding a robust and general grasp for an entirely-curved surface; maybe more training time is required. Figure 20(h), the mug, illustrates that sometimes grasps for complex objects can be consistent, although there is some oscillation in test performance later in evolution.

7.3 Grasp Quality vs. Training Effort vs. Task Completion Time Tradeoff Evaluation

This set of experiments measures the success rate of grasps across a two-dimensional surface of tradeoffs, examining how success varies across both training effort and task completion time. The idea is to explore what minimum amount of training effort and task completion time is necessary to successfully grasp an object. The grasping procedures were implemented under the designed motion planner, whose sequence consists of five states and state transitions: (1) starting from initial to grasp coordinates, (2) grasp, (3) go to dropoff location, (4) place, and (5) go back to initial state. The different training stage of the evolved neural network were used to generate the grasping position and orientation of the *Mekahand* at step (1). A grasp was rated as successful if an object was grasped, lifted, and placed to another location from the initial position.

Figure 22 gives the complete grasping success rate with different task completion times and different training efforts. Each object was tested 10 times per task completion time and per training effort. For the implementation used, the grasping task completion time was measured ranging from 1s to 10s (the interval is 1s) and the training effort was chosen ranging from 0 minutes to 1,200 minutes (the interval is 120 minutes). Therefore, the total number of object tests is $10 \times 10 \times 10$. Overall, these results indicate that when accumulated training effort is higher than around 600 minutes, we could achieve 70% successful grasps rate. Interestingly, the task completion time has only a negligible impact on the success rate; even if only 1s is allocated to task completion, the grasp quality remains nearly the same, indicating the controller has a stable design. The best match is depicted in Figure 22. If the success rate is set 70%, then best performance results when the ANN is trained for 600minutes, and the task completion time is 8s. Figure 22 was meant to be used as a reference to indicate the expected success rate when pre-conditions are given, which is derived from the experiments, allowing decision-makers to make optimal tradeoffs between the grasping quality, the task completion time, and the training effort in robotic control.

8 LESSONS LEARNED AND OBSERVATIONS FOR FUTURE WORK

From this tradeoff study, we have learned and observed the following interesting results. As shown in Figure 12, different fitness functions can be more robust to sensor error. For instance, the sphere object is too small to allow robust sensing, so $\overline{Fit_{134}}$ is more robust than $\overline{Fit_{1234}}$. Figure 12 shows a series of fitness function combinations tested against various objects to check grasp quality, supporting above observation. The difficulty in using ANNs is how to tune the parameters. What Section 6.4 shows is that the success rate of grasping novel objects is due highly to the bounding box method. This method often performs well except when the target object is too far from the image center. The solution is to have human input during the training process.

An especially interesting observation is that the quality of grasping may exhibit a steep improvement after some critical number of training cases. As shown in Figure 20, the efficacy of learning depends on the shape of the object being grasped, and the fitness function does not seem to be a reliable predictor of the success of a grasp, especially when training time is limited. A possible explanation is that for objects of shapes that generate a search space with sharp local maxima, NEAT does not perform well. From that point onward, we should pay attention to searching for the right topology.

A practical result, as shown in Figure 22, is that the imprecise computation framework is a good match for robotic skill acquisition. Inasmuch as the goal is to grasp an object well enough so that the object cannot be dropped easily by minor perturbation on the controller, it is important to delineate the boundary between a good enough grasp from one that may drop the grasped object.

This framework can be extended to carry out more complex human-like behaviors in more realistic scenarios.

9 CONCLUSION

Future robotic systems will need to function in unknown and unstructured environments, under demanding timing constraints. This article realizes a semi-autonomous remote robotic system that integrates body-compliant control, neuroevolution, and real-time constraints. An important contribution of the article is a tradeoff study in the design space of a representative robotic task, specifically, grasping of unknown objects in unstructured environments through imprecise computation. Such a design space typically includes three key dimensions: (1) the amount of training effort to teach the robot to perform the task, (2) the time available to complete the task once a command is given to perform it, and (3) the quality of the results of completing the task. The tradeoff results in this study indicate that training effort is a critical factor for attaining high-quality grasping. The trained model, after training, can then be reused without further modification. The work here can be viewed as offering a realistic basis for the scheduling work done by the real-time systems community in the past two decades. Importantly, beyond simulated tests, the results were also validated with a real robot (*Dreamer*) and contribute to the development of a systematic approach for designing robotic systems that can function in the challenging environments of the future, such as flexible manufacturing factories.

REFERENCES

[1] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. 2009. A survey of robot learning from demonstration. *Int. J. Robot. Auton. Syst.* 57, 5 (2009), 469–483.

[2] Martin Buss, Hideki Hashimoto, and John B. Moore. 1996. Dextrous hand grasping force optimization. *IEEE Trans. Robot. Autom.* 12, 3 (1996), 406–418.

[3] Columbia University Robotics Groups. 2017. GraspIt! Simulator. Retrieved from http://graspit-simulator.github.io/#.

[4] Carlo Ferrari and John Canny. 1992. Planning optimal grasps. In *Proceedings of the IEEE International Conference on Robotics and Automation,* Vol. 3. 2290–2295.

[5] Song Han, Aloysius K. Mok, Jianyong Meng, Yi-Hung Wei, Pei-Chi Huang, Quan Leng, Xiuming Zhu, Luis Sentis, Kwan Suk Kim, and Risto Miikkulainen. 2013. Architecture of a cyberphysical avatar. In *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems.* 189–198.

[6] Simon S. Haykin. 2009. *Neural Networks and Learning Machines* (3 ed.). Prentice Hall, Upper Saddle River, NJ.

[7] Kaijen Hsiao, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. 2007. Grasping POMDPs. In *Proceedings of the IEEE International Conference on Robotics and Automation.* 4685–4692.

[8] Kaijen Hsiao and Tomás Lozano-Pérez. 2006. Imitation learning of whole-body grasps. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.* 5657–5662.

[9] Pei-Chi Huang, Joel Lehman, Aloysius K Mok, Risto Miikkulainen, and Luis Sentis. 2014. Grasping novel objects with a dexterous robotic hand through neuroevolution. In *Proceedings of the 2014 IEEE Symposium on Computational Intelligence in Control and Automation.* 1–8.

[10] Pei-Chi Huang, Luis Sentis, Joel Lehman, Chien-Liang Fok, Aloysius K. Mok, and Risto Miikkulainen. 2015. Tradeoffs in real-time robotic task design with neuroevolution learning for imprecise computation. In *Proceedings of the Real-Time Systems Symposium.* IEEE, 206–215.

[11] Pei-Chi Huang, Luis Sentis, Joel Lehman, Chien-Liang Fok, Aloysius K. Mok, and Risto Miikkulainen. 2017. *Tradeoffs in Neuroevolutionary Learning-Based Real-Time Robotic Task Design in the Imprecise Computation Framework.* Technical Report. Department of Computer Science, The University of Texas at Austin. Retrieved from https://www.cs.utexas.edu/ peggy/techReport201701.pdf.

[12] Joshua M. Inouye and Francisco J. Valero-Cuevas. 2014. Anthropomorphic tendon-driven robotic hands can exceed human grasping capabilities following optimization. *Int. J. Robot. Res.* 33, 5 (2014), 694–705.

[13] Nick Jakobi. 1998. *Minimal Simulations for Evolutionary Robotics.* Ph.D. Dissertation. University of Sussex, School of Cognitive and Computing Sciences.

[14] David Kirkpatrick, Bhubaneswar Mishra, and Chee-Keng Yap. 1992. Quantitative steinitz's theorems with applications to multifingered grasping. *Discr. Comput. Geom.* 7, 3 (1992), 295–318.

[15] Jens Kober, J. Andrew Bagnell, and Jan Peters. 2013. Reinforcement learning in robotics: A survey. *Int. J. Robot. Res.* 32, 11 (Jul. 2013), 1238–1274.

[16] Nate Kohl, Kenneth Stanley, Risto Miikkulainen, and Rini Sherony. 2006. Evolving a real-world vehicle warning system. In *Proceedings of the Genetic and Evolutionary Computation Conference.* 1681–1688.

[17] Quoc V. Le, David Kamm, Arda F. Kara, and Andrew Y. Ng. 2010. Learning to grasp objects with multiple contact points. In *Proceedings of the IEEE International Conference on Robotics and Automation.* 5062–5069.

[18] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[19] Joel Lehman and Risto Miikkulainen. 2014. Overcoming deception in evolution of cognitive behaviors. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation.* ACM, 185–192.

[20] Joel Lehman and Kenneth O. Stanley. 2011. Abandoning objectives: Evolution through the search for novelty alone. *Evol. Comput.* 19, 2 (2011), 189–223.

[21] Hod Lipson, Josh C. Bongard, Victor Zykov, and Evan Malone. 2006. Evolutionary robotics for legged machines: From simulation to physical reality. In *Proceedings of the 9th International Conference on Intelligent Autonomous Systems.* 11–18.

[22] Guanfeng Liu, Jijie Xu, and Zexiang Li. 2004. On geometric algorithms for real-time grasping force optimization. *IEEE Trans. Contr. Syst. Technol.* 12, 6 (2004), 843–859.

[23] Jane W.-S. Liu, Kwei-Jay Lin, C. L. Liu, and C. W. Gear. 1992. Imprecise computations. *Mission Critical Operating Systems.* IOS Press, 159–169.

[24] Jane W.-S. Liu, Wei-Kuan Shih, Kwei-Jay Lin, Riccardo Bettati, and Jen-Yao Chung. 1994. Imprecise computations. In *Proceedings of the IEEE* 82, 1 (Jan. 1994), 83–94.

[25] Yun-Hui Liu. 1999. Qualitative test and force optimization of 3-D frictional form-closure grasps using linear programming. *IEEE Trans. Robot. Autom.* 15, 1 (1999), 163–173.

[26] Andrew T. Miller and Peter K. Allen. 1999. Examples of 3D grasp quality computations. In *Proceedings of the IEEE International Conference on Robotics and Automation,* Vol. 8. 1240–1246.

[27] Andrew T. Miller and Peter K. Allen. 2004. GraspIt!: A versatile simulator for robotic grasping. *IEEE Robot. Autom. Mag.* 11, 4 (Dec. 2004), 110–122.

[28] Andrew T. Miller, Steffen Knoop, Henrik I. Christensen, and Peter K. Allen. 2003. Automatic grasp planning using shape primitives. In *Proceedings of the IEEE International Conference on Robotics and Automation.* 1824–1829.

[29] Javier Molina-Vilaplana, J. Feliu-Batlle, and J. López-Coronado. 2007. A modular neural network architecture for step-wise learning of grasping tasks. *Int. J. Neur. Netw.* 20, 5 (2007), 631–645.

[30] Saigopal Nelaturi, Arvind Rangarajan, Christian Fritz, and Tolga Kurtoglu. 2014. Automated fixture configuration for rapid manufacturing planning. *Comput.-Aid. Des.* 46, 1 (2014), 160–169.

[31] Leonardo M. Pedro, Valdinei L. Belini, and Glauco A. P. Caurin. 2013. Learning how to grasp based on neural network retraining. *Adv. Robot.* 27, 10 (2013), 785–797.

[32] Raphael Pelossof, Andrew T. Miller, Peter K. Allen, and Tony Jebara. 2004. An SVM learning approach to robotic grasping. In *Proceedings of the IEEE International Conference on Robotics and Automation.* 3512–3518.

[33] Deepak Rao, Quoc V. Le, Thanathorn Phoka, Morgan Quigley, Attawith Sudsang, and Andrew Y. Ng. 2010. Grasping novel objects with depth segmentation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.* 2578–2585.

[34] Nasser Rezzoug and Philippe Gorce. 2006. *Robotic Grasping: A Generic Neural Network Architecture.* Mobile Robots Towards New Applications. Pro Literatur Verlag Robert MayerScholz, Berlin, Germany.

[35] Sebastian Risi, Charles E. Hughes, and Kenneth O. Stanley. 2010. Evolving plastic neural networks with novelty search. *Adapt. Behav.* 18, 6 (2010), 470–491.

[36] Máximo A. Roa and Raúl Suárez. 2015. Grasp quality measures: Review and performance. *Auton. Robots* 38, 1 (2015), 65–88.

[37] Ashutosh Saxena. 2009. *Monocular Depth Perception and Robotic Grasping of Novel Objects.* Ph.D. Dissertation. Stanford University.

[38] Ashutosh Saxena, Justin Driemeyer, and Andrew Y. Ng. 2008. Robotic grasping of novel objects using vision. *Int. J. Robot. Res.* 27, 2 (2008), 157–173.

[39] Luis Sentis, Jaeheung Park, and Oussama Khatib. 2010. Compliant control of multi-contact and center of mass behaviors in humanoid robots. *IEEE Trans. Robot.* 26, 3 (Jun. 2010), 483–501.

[40] Luis Sentis, Josh Petersen, and Roland Philippsen. 2013. Implementation and stability analysis of prioritized whole-body compliant controllers on a wheeled humanoid robot in uneven terrains. *Auton Robot* 35, 4 (Nov. 2013), 301–319.

[41] Karun B. Shimoga. 1996. Robot grasp synthesis algorithms: A survey. *Int. J. Robot. Res.* 15, 3 (1996), 230–266.

[42] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[43] Avishai Sintov and Amir Shapiro. 2017. An analysis of grasp quality measures for the application of sheet metal parts grasping. *Auton. Robots* 41, 1 (2017), 145–161.

[44] Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evol. Comput.* 10, 2 (2002), 99–127.

[45] Kenneth O. Stanley and Risto Miikkulainen. 2004. Competitive coevolution through evolutionary complexification. *Int. J. Artif. Intell. Res.* 21 (2004), 63–100.

[46] Marek Teichmann and Bud Mishra. 1997. The power of friction: Quantifying the "goodness" of frictional grasps. *Algorithms for Robotic Motion and Manipulation.* 311–320.

[47] Jianwei Zhang and Bernd Rössler. 2004. Self-valuing learning and generalization with application in visually guided grasping of complex objects. *Robot. Auton. Syst.* 47, 2 (2004), 117–127.

Received February 2017; revised January 2018; accepted January 2018