In International Conference on Development and Learning (ICDL),
Bloomington,IN,May 2006.

Toward a Vocabulary of Primitive Task Programs for Humanoid Robots

Evan Drumwright
USC Robotics Research Labs
University of Southern California
Los Angeles, CA 90089-0781
drumwrig@robotics.usc.edu

Victor Ng-Thow-Hing
Honda Research Institute USA
Mountain View, CA 94041
vng@honda-ri.com

Maja Matarić
USC Robotics Research Labs
University of Southern California
Los Angeles, CA 90089-0781
mataric@robotics.usc.edu

***Abstract*— Researchers and engineers have used primitive actions to facilitate programming of tasks since the days of *Shakey* [1]. *Task-level programming*, which requires the user to specify only subgoals of a task to be accomplished, depends on such a set of primitive task programs to perform these subgoals. Past research in this area has used the commands from *robot programming languages* as the vocabulary of primitive tasks for robotic manipulators. We propose drawing from *work measurement systems* to construct the vocabulary of primitive task programs. We describe one such work measurement system, present several primitive task programs for humanoid robots inspired from this system, and show how these primitive programs can be used to construct complex behaviors.**

***Index Terms*— robot programming, task-level programming, humanoid robots**

I. Introduction

Roboticists have used primitive actions to facilitate programming of tasks from the days of the mobile robot *Shakey* [1]. Primitive actions allow users to program in the domain of human semantics, rather than requiring programming in the motor command space of the robot. Additionally, primitive actions are used in imitation learning methods [2] and in classification of observed human movement [3]. However, the development of a standard vocabulary of primitive actions, or *primitive task programs*, for humanoid robots has eluded researchers.

This paper addresses the manual programming of humanoid robots using such a vocabulary. Research into manual programming of manipulator robots has tended to concentrate on *task-level programming* [4], a method that operates by specifying task subgoals to be accomplished rather than robot commands to be performed. Task-level programming systems tend to use either an *ad hoc* set of primitive actions or the commands of a specific robot programming language (e.g., AML [5]). In contrast, we draw from research into human occupational task performance to develop a library of robot-independent primitive task programs for humanoids.

II. Task-level programming

Task-level programming advanced the idea of decomposing a goal into subtasks that can be resolved using primitive actions. Much of the research into this area has focused on planning. Specifically, such research has attempted to circumvent the EXP-hard or PSPACE-complete complexity of planning [6], ascertain how to ignore extraneous data in constructing plans from observed task executions [7], [8], or learning policies for executing tasks with performance criteria [9]. Subsequently, research into "good" sets of primitive robot actions has been relatively overlooked; researchers [4], [7] tend to propose new primitives in an *ad hoc* manner. In contrast, we propose a set of task primitives inspired by research into human occupational tasks. Additionally, our work focuses on robot-independent task-level programming.

It must be noted that the task primitives discussed here share little with the concept of motor primitives [10], [11] investigated elsewhere. Task primitives focus on the atomic work elements necessary to accomplish a given task. In contrast, motor primitives aim to find a lower dimensional subspace of the thirty or more degrees-of-freedom employed by most humanoid robots for the purpose of making learning algorithms tractable.

III. Work measurement systems

Work measurement systems are methods developed to allow human observers to compute the mean time required to perform a given occupational task. These systems function by decomposing a task into primitive elements, each of which is assigned some mean time for execution; work measurement is proven at this decomposition. Consequently, these systems are a cogent choice for inspiring the design of a primitive task program vocabulary for humanoid robots.

The most prominent work measurement system is currently the proprietary Motion-Time Measurement (MTM) system [12]. MTM is available both in multiple primary variants (MTM-1, MTM-2, MTM-3, and associated subvariants) and additional versions that specialize in particular fields (e.g., clerical work, micro-miniature operations, metal work operations, etc.). The primary variants of MTM differ in the amount of detail that the system can capture.

This section describes the MTM-1 system [13], the first version of MTM. The subsequent variants (MTM-2, MTM-

3, etc.), were developed to ease the burden on the work measurement observer at the expense of accuracy [14]; to facilitate this goal, these systems combine the primitive elements of MTM-1 into coarser elements. For example, the *reach*, *move*, *grasp*, and *release* elements in MTM-1 are merged into the MTM-2 tasks *get* and *put*; MTM-3 merges these two elements into a single element, *handle*. We feel that MTM-1 is the most suitable work measurement system to exploit for the creation of a vocabulary of humanoid task programs; more coarse vocabularies, such as those used in MTM-2 and MTM-3 would lead to more complex primitive actions.

MTM-1 elements are the result of frame-by-frame analysis of video involving diverse areas of work [14]. Unfortunately, the system can seem subjective and presumptive in some ways. There are fuzzy boundaries between some elements in MTM-1 (e.g., *move* and *position*, described below). Some subcategories of elements (e.g., move to an indefinite location) require intelligence on the part of the observer. And, a few elements make certain assumptions about the type of objects being handled; for example, the *grasp* element supposes that the geometry of the object being handled is roughly regular, or even cylindrical or spherical. These apparently arbitrary decisions are likely a result of the development of MTM-1 for work measurement, rather than task classification. Subsequently, MTM-1 is used only as inspiration, rather than as a canonical reference, for the construction of the vocabulary of primitive task programs introduced in this paper.

The remainder of this section describes the nine atomic motion elements of the MTM-1 system: *reach, move, turn and apply pressure, grasp, position, release, disengage, eye movements,* and *body, leg, and foot motions*.

A. Reach

Reach is consistent with the standard definition of the word, that of stretching the arm in order to touch or grasp an object. The time units assigned to a particular case is dependent upon the distance that the hand moves and whether the dominant or secondary hand is used to perform the task. Additionally, MTM-1 subcategorizes reaching into five cases, listed below:

- reaching to an object in a fixed location, or to an object in the other hand or on which the other hand rests
- reaching to a single object in a location which may vary slightly from cycle to cycle
- reaching to an object jumbled with other objects in a group so that search and select occur (see Figure 1a)
- reaching to a very small object or where accurate grasping is required
- reaching to an indefinite location to get the hand out of the way or in position for body balancing or the next movement

B. Move

Move indicates that an object is moved with some imprecision (contrast with *position* below). The timing of the movement is modified by the distance that the object is moved, as well as the weight of the object. MTM-1 divides *move* into three subcategories:

- moving an object to the other hand or against a "stop"
- moving an object to an approximate or indefinite location (see Figure 1b)
- moving an object to an exact location

C. Turn and apply pressure

Turn and apply pressure is a movement that describes applying forces or torques to objects (see Figure 1c). The MTM Manual. [13] defines *turn* as "the basic motion employed to rotate the hand about the long axis of the forearm". MTM-1 combines these somewhat dissimilar actions together due to the prevalence of their simultaneous application. This element's timing is modified by the approximate quantity of force applied and the degrees turned about the arm.

D. Grasp

The semantics of *grasp*, like *reach*, is consistent with the accepted definition of the word. MTM-1 identifies five types of grasps:

- the pick up grasp (further subcategorized by whether the object is floating in space or lying close against a flat surface and the size of the object)
- regrasping (i.e., grasping the object in a different manner)
- transfer grasping (i.e., transferring the object to a different hand)
- grasping among a jumbled collection of objects
- "contact", "slide", or "hook" grasping (indicates grasping with the finger tips, sliding the object along a surface, or scooping the object into the hand, respectively)

E. Position

Position, as seen in Figure 1d, is used to indicate a more refined movement than *move*; *position* is typically used in the context of mating parts. The timing of the *position* element is a function of the symmetry of the handled object (i.e., "symmetrical", "semi-symmetrical", or "non-symmetrical"), and the difficulty of the object to handle (i.e., "easy" or "difficulty"). These object characteristics are completely subjective; the MTM manual [13] leaves the determination of object symmetry and handling difficulty to the observer. Additionally, *position* is subdivided into three categories, based on the "class of fit": "loose" (no pressure required), "close" (light pressure required), and "exact" (heavy pressure required).

F. Release

The *release* element is complementary to the *grasp* element; it consists of only two cases: a "normal" release performed by opening the fingers as an independent motion and a "contact release".

G. Disengage

Disengage is complementary to *position*, indicating that the operator is separating two objects. Similarly to *position*, *disengage* is subdivided into three categories, based on the "class of fit": "loose" (very slight effort required, the movement blends with the subsequent move), "close" (normal effort required, the hand recoils slightly), and "tight" (considerable effort required, the hand recoils markedly). Also like *position*, the timing of the *disengage* movement is dependent upon the handling difficulty of the object (either "easy" or "difficult").

H. Eye movements

Timing measurement for eye movements is also measured by the MTM-1 system. This element considers both the "eye travel time" (a function of the distance between points from and to which the eye travels and perpendicular distance from the eye to the line of travel) and the "eye focus time" (the time required to focus upon an object). Note that head movement is not considered.

I. Body, leg, and foot motions

MTM-1 groups several disparate motions in five subcategories under the single element *body, leg, and foot motions*. The individual motion times are determined by the distance that the body, leg, or foot moves. The subcategories of this element are: foot, leg, or foreleg motion; sidestep; bend, stoop, kneel, or arise; sit; and walk.

IV. The Task Matrix

The previous section inspired the development of our vocabulary of humanoid task programs. The selection of the set of primitive tasks is a critical step; however, significant issues remain before these primitives can be implemented programmatically on a humanoid robot and used for performing arbitrary complex tasks. First, some mechanism should be available for "porting" primitive task programs to diverse humanoid robot platforms. Second, facilities must be provided for executing primitive task programs concurrently and sequentially.

The Task Matrix [15] is a viable framework for addressing the above issues. The Task Matrix is a data structure composed of a diverse collection: an assortment of heterogeneous, robot-independent task programs, a set of sensory conditions used for testing the state of the environment, and a specification for robot-dependent skill modules. Additionally, the Task Matrix enables the creation of complex task programs that consist of primitive task programs (and even other complex task programs) executing sequentially and concurrently; the complex task programs are formulated using a simple state-machine representation.

The Task Matrix is an ideal substrate for implementing a vocabulary of primitive task programs. The task programs within the matrix are robot independent, allowing for execution over a variety of humanoid robot platforms. Robot independence promotes code reuse; the individual task programs can be made more robust over time. The allowance of sequential and concurrent execution of task programs within the Task Matrix enables users to compose complex behaviors from primitive task programs.

The remainder of this section discusses the interdependent, components of the Task Matrix: *conditions*, *skills*, and *task programs*. The section concludes with a description of the connectivity between the various matrix constituents.

A. Conditions

A condition is a Boolean function of state (typically percepts). Conditions are used both to determine whether a task program is capable of executing (*precondition*) and to determine whether a task program can continue executing (*incondition*).

1) Postural: The *Postural* condition is satisfied if all degrees-of-freedom for a specified kinematic chain of the robot are equal (to within floating point tolerance) to predetermined values. These values are stored externally so that porting to new robot platforms is simplified.

2) Near: The *near* condition determines whether an object x is sufficiently near (the distance is a user-specified parameter) some object y. *Near* can operate upon either the bounding box or a reference point (e.g., center-of-mass, task frame, etc.) of an object.

3) Above: The *above* condition determines whether the bounding box for an object x is completely above the bounding box for an object y (i.e., whether the bottom of x's bounding box is above the top of y's bounding box.)

4) Graspable: The *graspable* condition is satisfied if one of the robot's hands is in a location such that the robot is able to grasp the specified object.

5) Grasping: The *grasping* condition is satisfied if one of the robot's hands is currently grasping the specified object.

B. Skills

The Task Matrix relies upon a set of common (across robot platforms) skills to perform tasks in the matrix. A task program that simply follows a trajectory, for example, does not operate directly upon the robot. Instead, the program uses the *trajectory following skill*. The interface to skills is independent of the underlying controllers; the task does not need to know whether the robot uses computed torque control, feedback control, etc.

Note that we make a critical distinction between tasks and skills in this work. A *task* is a function to be performed;

(a) *reach (with search and select)* (b) *move* (c) *turn and apply pressure* (d) *position*

Fig. 1. Several examples of MTM-1 elements in action.

in contrast, *skill* refers to a developed ability. A task is an objective to be accomplished and is robot independent. Skills are diverse methods used to achieve that objective and are robot-specific.

The common skill set for robots currently consists of trajectory tracking (following a trajectory), motion planning (with collision avoidance), trajectory rescaling (slowing the timing of a trajectory so that it may be followed using the robot's dynamics limitations), forward and inverse kinematics, and a method for determining a viable humanoid hand configuration for grasping a given object. A user that wants to perform the programs in the Task Matrix on a humanoid robot need only implement the common skill set; the task programs should work approximately the same whether the underlying controller is proportional-derivative (PD) or computed-torque.

C. Task programs

A task program is a function of time and state that runs for some duration (possibly unlimited), performing robot skills. Task programs may run interactively (e.g., reactively) or may require considerable computation for planning. Additionally, users (or other task programs) can send parameters to a task program that influences its execution. Note that task programs neither query nor drive robots directly.

The remainder of this section discusses the primitive programs in the Task Matrix that correspond directly to the atomic elements in MTM-1. The position-controlled task programs *reach*, *position*, *grasp*, *release*, and *fixate* have been developed; these correspond to the MTM-1 elements *reach*, *position / move*, *grasp*, *release*, and *fixate*.

1) Reach: The *reach* task program utilizes motion planning to formulate a collision-free plan for driving the humanoid from its current configuration to one that allows grasping of a specified object with a "hand" of the robot. *Reach* handles three of the five subcategories of the MTM-1 *reach* element described in Section III-A; it does not handle reaching to get the hand out of the way or in position for the next movement (this is accomplished as needed by other tasks) or reaching that requires "search and select".

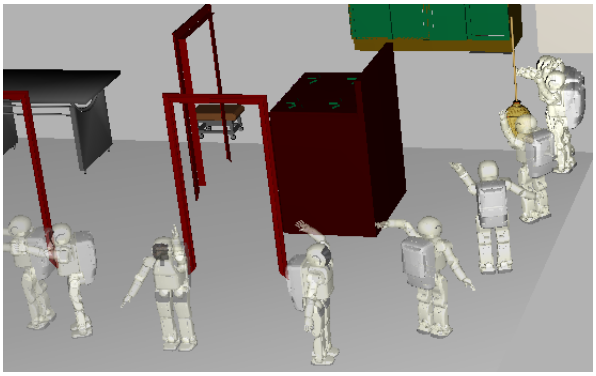

Fig. 2. A depiction of the *reach* task program in action. This figure shows the process of reaching to a broom in a simulated real-world environment in a collision-free manner.

2) Position: Position is analogous to *reach* with a tool or object used as the end-effector of the robot, rather than the hand. The *position* program corresponds to the MTM-1 elements *move* and *position*. We make no distinction between the precision required to move an object; thus, we are able to combine the two MTM-1 elements into a single task program. The Task Matrix *position* program is currently unable to move objects in ways that require interaction with the environment. For example, *position* is unfit for using a tool to compress a spring. Subsequently, *position* does not handle all cases of the MTM-1 *position* element (i.e., it cannot handle cases that require "light" or "heavy" pressure [see Section III-E]). However, the Task Matrix *position* program does not need to consider the symmetry of the object or its ease of handling; the Task Matrix manages this by using grasp configurations in the skill set (see Section IV-B). Similarly, the task program need not consider moving the object to an indefinite location, as is specified in the MTM-1 *move* element, because the target location is a requisite parameter for the *position* program.

3) Grasp: The *grasp* task program is used for grasping objects for manipulation. *Grasp* utilizes collision detection to move the fingers as much toward a clenched fist configuration

as possible; each segment of each finger is moved independently in simulation until contact is made. Like *reach*, *grasp* exits immediately if the condition *grasping*(object) is met, precluding unnecessary motion planning.

Grasp currently implements only one of the five subcategories of the MTM-1 *grasp* element, that of the "pick up" grasp. "Regrasping" and "transfer grasping", though not currently handled, could be added to the *grasp* program without much difficulty. However, grasping among a jumbled collection of objects and grasps that require complex interactions with the environment (i.e., the "contact", "slide", and "hook" grasps) present significant obstacles to implementation. Thus, the *reach* program is currently far less expressive than its corresponding MTM-1 element.

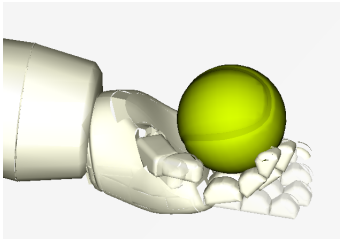

Fig. 3. Depiction of using the *grasp* program to clutch a tennis ball

4) Release: Release is used to release the grasp on an object. It utilizes a "rest" posture for the robot hand and generates joint-space trajectories to drive the fingers from the current grasping configuration to the rest posture. Thus, it is equivalent to the MTM-1 "normal" release (see Section III-F).

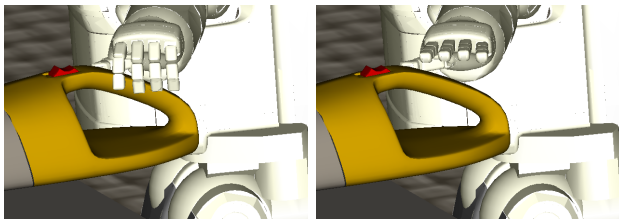

Fig. 4. Using *release* to relinquish the grip on the vacuum

5) Fixate: The *fixate* program focuses the robot's "view" (the front of the head) on both moving and non-moving objects. *Fixate* was developed for two purposes. First, it aims to make the appearance of executed tasks more human-like by directing the robot to look at objects that it is manipulating. However, the primary objective of *fixate* is to facilitate updating of the robot's model of the environment where it is changing (i.e., at the locus of manipulation).

Fixate corresponds only roughly to MTM-1's *eye movements* element; the former specifies head and base movement, while the latter specifies only eye movement. However, both the *fixate* program and the *eye movements* element accomplish the same task, that of looking at a specific location.

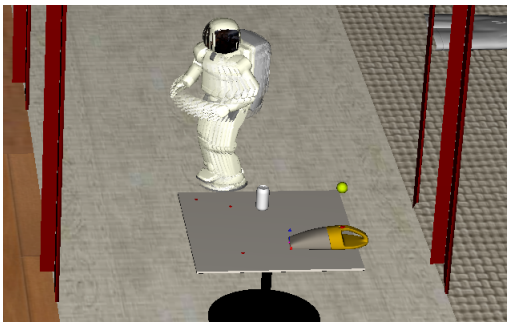

Fig. 5. Depiction of the action of the *fixate* program; the simulated robot is commanded to focus on the vacuum. The robot turns on its base while simultaneously orienting its head.

D. Task Matrix connectivity

The Task Matrix is not a database; the collection of components within it is interdependent. Indeed, the term *matrix* does not refer to a mathematical array of quantities, but rather a medium in which task programs can reside and be interconnected. We utilize *Message-Driven Machines* (MDMs) to perform *macro task* programs, which are task programs composed of other task programs (including possibly other macro task programs).

MDMs operate using a message passing mechanism. Task programs are executed or terminated based on messages from other task programs. Typical messages include *task-complete* (indicating the task has completed execution), *planning-started* (indicating the planning phase of the task has begun), and *force-quit* (indicating that the task was terminated prematurely).

MDMs are composed of a set of states, each state corresponding to a task program, and transitions. There is a many-to-one mapping from states to task programs (i.e., multiple states may utilize the same task program) within a MDM; the task programs in this mapping may be primitive programs or other MDMs. A transition within a MDM indicates that a task program is to be executed or terminated, depending on the transition type, and is followed if the appropriate message is received.

It is natural to wonder what happens if one of the subtasks in a macro task fails. We strive to make the primitive tasks in the Task Matrix very robust, but failure is always possible. MDMs can catch and recover from failures in execution using an alternate path of execution, transitioned to by receiving the *task-failed* message. By default, if a *task-failed* message is not "caught", execution of the MDM terminates.

V. Results

We have utilized combinations of the primitive tasks presented above to express complex behaviors, including

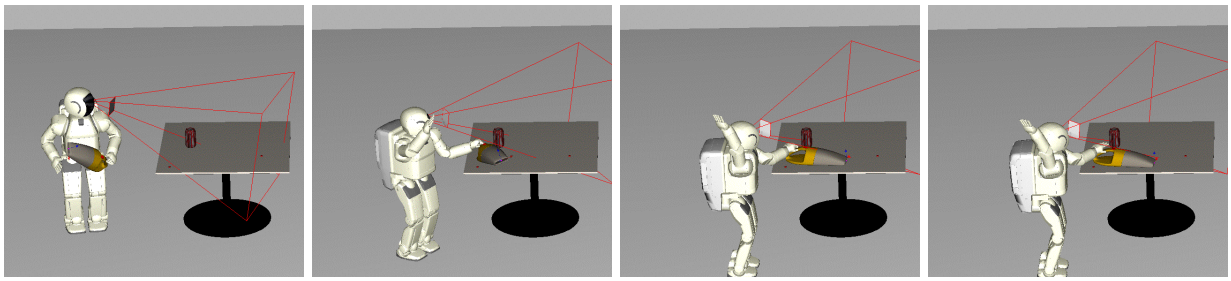

Fig. 6. Using a complex behavior composed of *fixate*, *position*, and *release* to put down an object

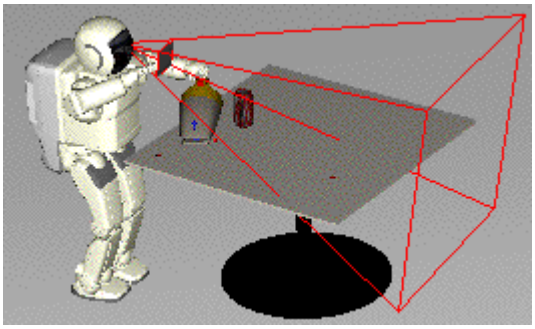

Fig. 7. Using a complex behavior composed of *position* and *fixate* to simulated vacuuming debris

picking up objects, putting down objects (see Figure 6), and performing a vacuuming task (see Figure 7) on a kinematic simulation of the humanoid robot *Asimo* [16]. Previous work [15] has demonstrated the viability of the Task Matrix framework toward executing multiple primitive tasks concurrently; for example, one such complex behavior consisted of a simulated humanoid robot facing and waving to a moving humanoid simultaneously.

VI. Conclusion

The Task Matrix and MTM-1 have served as media for implementing a set of primitive task programs. Though the set of primitive task programs does not yet fully correspond to the primitive actions identified by MTM-1, these primitive programs have already proven successful in composing several complex position-controlled tasks. Future work will address utilizing this vocabulary of primitive tasks for imitation learning; we hope to endow humanoid robots with the ability to acquire complex abilities from human demonstration.

Acknowledgment

The authors wish to thank Suzanne Drumwright for providing photos of the MTM-1 elements.

References

[1] N. J. Nilsson, "A mobile automaton: An application of artificial intelligence techniques," in *Proc. of Intl. Joint Conf. on Artificial Intelligence*, Washington, D.C., May 1969.

[2] S. Schaal, "Is imitation learning the route to humanoid robotics?" *Trends in Cognitive Sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[3] E. Drumwright and M. Matarić, "Generating and recognizing free-space movements in humanoid robots," in *2003 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Las Vegas, NV, October 2003, pp. 1672–1678.

[4] T. Lozano-Pérez, "Task planning," in *Robot motion: planning and control*, M. Brady, J. M. Hollerbach, T. L. Johnson, T. Lozano-Perez, and M. T. Mason, Eds. MIT Press, 1982, pp. 474–498.

[5] R. Taylor, P. Summers, and J. Meyer, "Aml: A manufacturing language," *Intl. Journal of Robotics Research*, vol. 1, no. 3, 1982.

[6] S. Narasimhan, "Task level strategies for robots," Ph.D. dissertation, Massachusetts Institute of Technology, 1994.

[7] A. M. Segre, *Machine learning of robot assembly plans*. Kluwer Academic Publishers, 1988.

[8] J. R. Chen, "Constructing task-level assembly strategies in robot programming by demonstration," *Intl. Journal of Robotics Research*, vol. 24, no. 12, pp. 1073–1085, Dec 2005.

[9] D. C. Bentivegna, "Learning from observation using primitives," Ph.D. dissertation, Georgia Institute of Technology, 2004.

[10] A. Fod, M. Matarić, and O. Jenkins, "Automated derivation of primitives for movement classification," *Autonomous Robots*, vol. 12, no. 1, pp. 39–54, January 2002.

[11] A. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," in *Advances in Neural Information Processing Systems 15*, S. Becker, S. Thrun, and K. Obermayer, Eds., 2002, pp. 1547–1554.

[12] D. W. Karger and W. M. Hancock, *Advanced Work Measurement*. New York, NY: Industrial Press, Inc., 1982.

[13] W. Antis, J. John M. Honeycutt, and E. N. Koch, *The Basic Motions of MTM*. The Maynard Foundation, 1973.

[14] B. W. Niebel, *Motion and time study, 8th ed.* Richard D. Irwin, Inc., 1988.

[15] E. Drumwright and V. Ng-Thow-Hing, "The task matrix: An extensible framework for creating versatile humanoid robots," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Orlando, FL, USA, 2006.

[16] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka, "The development of honda humanoid robot," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Leuven, Belgium, May 1998, pp. 1321–1326.