Synthesis of Concurrent Object Manipulation Tasks

Yunfei Bai* Kristin Siu† C. Karen Liu‡

Georgia Institute of Technology

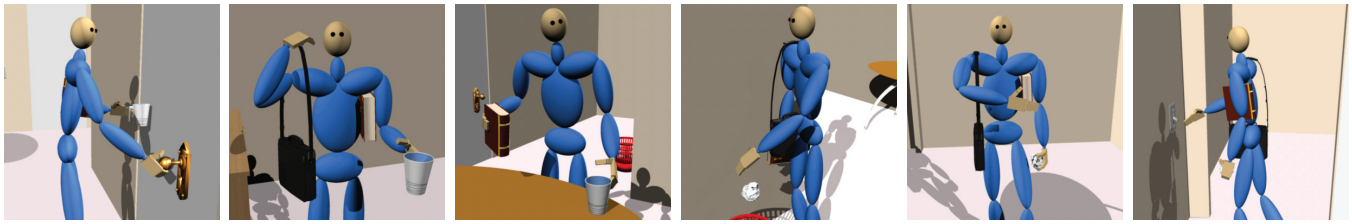

Figure 1: *A simulated character manipulating multiple objects concurrently in different scenarios.*

Abstract

We introduce a physics-based method to synthesize concurrent object manipulation using a variety of manipulation strategies provided by different body parts, such as grasping objects with the hands, carrying objects on the shoulders, or pushing objects with the elbows or the torso. We design dynamic controllers to physically simulate upper-body manipulation and integrate it with procedurally generated locomotion and hand grasping motion. The output of the algorithm is a continuous animation of the character manipulating multiple objects and environment features concurrently at various locations in a constrained environment. To capture how humans deftly exploit different properties of body parts and objects for multitasking, we need to solve challenging planning and execution problems. We introduce a graph structure, a *manipulation graph*, to describe how each object can be manipulated using different strategies. The problem of manipulation planning can then be transformed to a standard graph traversal. To achieve the manipulation plan, our control algorithm optimally schedules and executes multiple tasks based on the dynamic space of the tasks and the state of the character. We introduce a "task consistency" metric to measure the physical feasibility of multitasking. Furthermore, we exploit the redundancy of control space to improve the character's ability to multitask. As a result, the character will try its best to achieve the current tasks while adjusting its motion continuously to improve the multitasking consistency for future tasks.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: human simulation; physically based animation; motion planning; optimization; physically based animation

*e-mail: ybai30@mail.gatech.edu

†e-mail:kasiu@gatech.edu

‡e-mail:karenliu@cc.gatech.edu

Links: DL PDF

1 Introduction

Performing multiple object manipulation tasks concurrently is an essential human activity in everyday environments. A mundane morning routine before going to work can involve numerous consecutive and concurrent tasks: picking up the briefcase on the floor, opening the refrigerator to fetch a lunch box, using the elbow to close the refrigerator door, tucking the lunch box under the arm so the hand can search for keys in the pocket, and pushing the front door open by leaning on it. This sequence of tasks, which humans perform effortlessly, requires sophisticated planning and dynamic motion control, which have not been broadly explored in physics-based computer animation or robotics. Unlike existing robots, humans can employ a variety of *manipulation strategies* to interact with objects, such as using their hands, shoulders, elbows, torso, or even their head. Consequently, synthesizing full-body manipulation requires not only simulating physically realistic joint motion, but also capturing how humans deftly exploit different properties of body parts and objects for multitasking.

We introduce a physics-based technique to synthesize human activities involving concurrent full-body manipulation of multiple objects. We view full-body manipulation as three interrelated layers of motor control: locomotion, upper-body manipulation, and detailed hand manipulation. This paper focuses on the second layer – we design dynamic controllers to physically simulate upper-body manipulation and integrate it with procedurally generated locomotion and hand manipulation. The main algorithm must overcome major challenges in both planning and execution.

Planning a valid sequence of manipulation strategies for a character is difficult because humans have abundant choices for manipulating an object. To circumvent this issue, our key insight is that, instead

ACM Reference Format
Bai, Y., Siu, K., Liu, C. 2012. Synthesis of Concurrent Object Manipulation Tasks.
ACM Trans. Graph. 31 6, Article 156 (November 2012), 9 pages. DOI = 10.1145/2366145.2366175
http://doi.acm.org/10.1145/2366145.2366175.

of making plans for each end-effector or body part of the character, we make plans for each object. We introduce a graph structure, a *manipulation graph*, to describe how each object can be manipulated by different strategies. An object's manipulation graph is based on its properties and contains a set of nodes, each of which represents a manipulation strategy (e.g. left hand, left shoulder, etc). An edge between two nodes represents a valid transition between two manipulation strategies (e.g. transporting the object from the left hand to the left shoulder). With the manipulation graph representation, the problem of manipulation planning can be conveniently transformed to a standard graph traversal.

Executing the manipulation plan has its own challenges. Because humans tend to act on tasks concurrently to save time or minimize traveling distance, a successful control algorithm must appropriately schedule multiple tasks, i.e. when to overlap tasks and when to execute them in succession. Given multiple tasks, we introduce a "task consistency" metric to measure the physical feasibility of multitasking based on the task spaces and the state of the character. Using this metric, we formulate a convex optimization to determine when and how to optimally overlap tasks. Furthermore, the dynamic controller must be general and robust to execute arbitrary multiple tasks simultaneously without interfering with each other. Inspired by the framework of operational space control [Khatib 1987], we exploit the redundancy of control space to improve the character's ability to multitask. Specifically, our algorithm actively solves for an ideal next pose such that the task space in the next time step is most consistent with desired tasks. As a result, the character will try its best to achieve the current tasks while adjusting its motion continuously to improve the "multitasking consistency" for future tasks.

We present our results as an animation of a character successfully navigating a cluttered environment, while concurrently manipulating multiple objects, such as a bag that can be slung over the shoulder and books that can be tucked under the arms. Our algorithm can compute a new plan efficiently according to the changes in any property of the environment or the objects.

2 Related Work

Effective methods for synthesizing full-body manipulation are crucial for animating everyday human behaviors. Most previous work exploited inverse kinematics and motion planning techniques to generate motion that satisfies desired manipulation tasks. To achieve collision-free motion, these methods applied path planning algorithms in workspace [Liu and Badler 2003; Yamane et al. 2004], configuration space [Koga et al. 1994; Kallmann et al. 2003; Kallmann 2005], or configuration space with an additional timing dimension [Shapiro et al. 2007]. Using the search result as guidance, natural-looking, full-body animation can be synthesized based on heuristics or mocap data. The general problem of manipulating objects with locomotion has been studied previously [Feng et al. 2012; Huang et al. 2011; Stilman et al. 2007]. Our work also produces full-body manipulation, but we focus on human multitasking that exploits various manipulation strategies concurrently. In addition, we take the approach of physical simulation instead of kinematic procedures or motion capture. The simulated motion exhibits more physical realism for dynamically demanding tasks, such as holding a cup of water while lifting a heavy handbag. Full-body manipulation is also extensively studied in robotics [Harada et al. 2003; Takubo et al. 2005; Yoshida et al. 2005; Nishiwaki et al. 2006]. However, due to the physical design and hardware constraints, existing robots only use predetermined manipulation strategies with intended manipulators. Consequently, concurrent manipulation scenarios described in this work have not been broadly explored in robotics.

The idea of associating interaction information with objects rather than characters has been proposed previously [Rijpkema and Girard 1991; Kallmann 2004]. The *smart object* approach stores manipulation information, such as grasping animation or approaching direction, as part of the object description. The interaction information is also useful for behavior modeling. Lamarche and Donikian [Lamarche and Donikian 2001] introduced a behavior representation that considers the involved body parts using concurrent state machines. As a result, the character can mix different behaviors without the need of creating specific behaviors exhaustively. The manipulation graph in our work is inspired by this similar idea. However, our object representation does not contain any keyframes or animation sequences because the output motion is entirely physically simulated. The object representation, instead, stores the information of manipulation strategies and their transitions.

Our work also builds upon operational space control in robotics [Khatib 1987; Nakamura et al. 1987; Khatib et al. 2004; Sentis and Khatib 2005; Mistry and Righetti 2011]. Operational space control exploits kinematic and motor redundancy to achieve prioritized task goals. These works demonstrated that humanoid robots could accurately track lower priority movement postures without interfering with the higher priority manipulation tasks. In computer animation, Abe and Popović [2006] extended this framework to handle closed-loop joint structures. de Lasa *et al.* [de Lasa and Hertzmann 2009; de Lasa et al. 2010] introduced an optimization scheme for task-space control, in which a nested sequence of objectives are optimized so as not to conflict with higher-priority objectives. The problem addressed in this paper also depends on prioritizing multiple tasks, but we introduce two new ideas to handle concurrent manipulation tasks. By analyzing the subspace of task-equivalent control forces, we define a metric to measure the consistency of multiple tasks and schedule the tasks accordingly. Moreover, we actively optimize the character's next state so that the character is not only aiming to achieve the current tasks, but also adjusting its pose in preparation for future tasks.

Generating continuous locomotion is one of the most important applications in computer animation. Various kinematic techniques have been proposed and applied to gaming or virtual world applications [Bruderlin and Williams 1995; Rose et al. 1998; Kovar et al. 2002; Choi et al. 2003; Lau and Kuffner 2005]. To generate a long, continuous motion sequence from short motion clips, an effective motion blending technique must be able to handle walk cycles with different gait speeds, turning directions, stride lengths, and contact phases. We apply a similar idea as described by Kovar *et al.* [Kovar et al. 2002] to interpolate similar motion clips while maintaining the correct contact states.

3 Overview

We introduce a physics-based method, which utilizes different manipulation strategies, to synthesize concurrent object manipulation. The input to our algorithm includes an environment map along with the information about the objects and features in the environment, and manipulation graphs that describe all possible strategies to hold, move, push, or release an object (Figure 2). The output of the algorithm is a continuous animation of the character manipulating multiple objects and environment features (e.g. doors or light switches) concurrently at various locations in a constrained environment.

The problem involves two stages: planning and execution (Figure 3). Given the user-specified input, the planning process produces a spatial path for locomotion and manipulation plans for all the

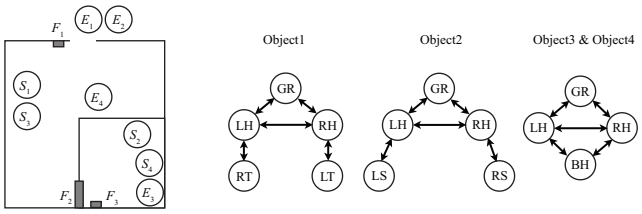

Figure 2: *The input to our algorithm. Left: An environment map is a 2D illustration of a virtual environment. The user can specify an initial configuration (S_i) and a goal configuration (E_i) for each object i, as well as manipulatable features (F_i), such as doors or light switches. In this example, the user specified an environment map with nested spaces and three features, along with the tasks of transporting four objects. Right: A manipulation graph describes all possible strategies to manipulate an object. Here we show the manipulation graphs for four objects. A node in the graph represents one of the allowed manipulation strategies for this object. If two manipulation strategies can be executed in succession, we add an edge between their corresponding nodes. All nodes can transition to themselves but the edges are ignored for clarity. In this example, Object 1 is a book which can be picked up with either hand (LH/RH) from the ground (GR), and tucked under the left or right arm (LT/RT).*

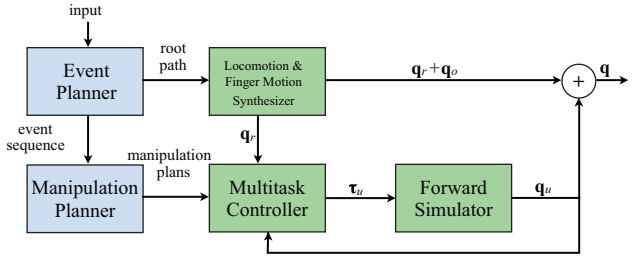

Figure 3: *Algorithm overview. The problem involves two stages: planning (shown in blue) and execution (shown in green).*

objects and features in the scene. A manipulation plan indicates which manipulation strategies should be used according to the manipulation graph of each object (Figure 5). During each execution step, the multitask controller determines a set of concurrent tasks and computes control forces τ_u such that the concurrent tasks have minimal interference with each other. Finally, the forward simulator uses the control forces to simulate the next upper body pose $\mathbf{q}_u$ while the root motion $\mathbf{q}_r$, lower body and finger motion $\mathbf{q}_o$ are produced by a kinematic-based motion synthesizer.

4 Planning

The entry point of the algorithm is a two-step planning process, which consists of an event planner and a manipulation planner. Before we describe the planning algorithms, we first define a few terminologies in detail.

4.1 Definitions

An *environment map* is a 2D illustration of a virtual environment including walls, furniture, and manipulatable features (F), such as door knobs or light switches (Figure 2, Left). Each manipulatable feature F_i comes with a set of allowable manipulation strategies. For example, a door knob can only be manipulated using the left or right hand. On the environment map, the user can specify initial configuration (S_i) and goal configuration (E_i) for each object i to indicate the location and approaching orientation for pick-up and release of the object. We define an *event* as one of three actions: the pick-up of an object S_i, the release of an object E_i, or the interaction with a feature F_i, such as turning a light switch on or off.

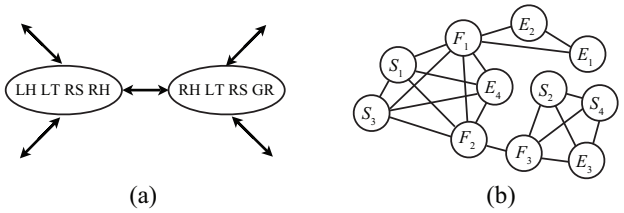

Figure 4: *(a) Partial aggregate manipulation graph constructed from the four manipulation graphs described in Figure 2. (b) The event graph for the scenario in Figure 2.*

In addition, a *manipulation graph* for each object needs to be specified by the user. A node in the graph represents one of the allowed manipulation strategies for this object. We define nine different types of strategies in this paper: LH/RH/BH (use left/right/both hands to grasp the object), LS/RS (use left/right shoulders to carry the object), LT/RT (tuck the object between torso and left/right arm), and LE/RE (use left/right elbow to push the object). We also define a special node, GR, to indicate the "ground state" when the object is not manipulated by the character. If two manipulation strategies can be executed in succession, we add an edge between their corresponding nodes. For example, a book can be picked up by the left hand and tucked under the right arm. In this case, we add an edge between LH and RT (Figure 2, Right).

An *aggregate manipulation graph* combines all the input manipulation graphs into one (Figure 4(a)). Given n manipulation graphs, we construct each node of the aggregate graph by taking an n-tuple consisting of one node from each manipulation graph. Initially, the aggregate graph nodes consist of all possible n-tuples of object configurations. We prune nodes with configurations that cannot be achieved due to the mutual exclusivity of the manipulation strategies. For example, an aggregate node containing LH (left hand) and BH (both hands) is invalid if the character is not allowed to manipulate one object using the left hand and the other using both hands [1]. Once we define a valid set of aggregate nodes, we determine the connection of each pair of nodes based on the connectivity of the original manipulation graphs. Consider two aggregate nodes s_1 and s_2, where $s[i]$ denotes the manipulation strategy for object i in node s. We add an edge between s_1 and s_2 if and only if, for every object i, there exists an edge between $s_1[i]$ and $s_2[i]$ in its original manipulation graph.

4.2 Event Planner

The goal of event planner is to search for a valid event sequence which achieves all the required tasks on the environment map based on user-specified object configurations and initial feature states (e.g. a light switch is on or off). Our algorithm casts the search problem as a graph traversal. The first step is to construct a *event graph*, of which each node corresponds to an event (S_i, E_i, or F_i) on the environment map. If there is a collision-free path between two nodes on the environment map, we add an edge between them and assign the Euclidean distance as the cost of the edge. For example,

[1] In this paper, every manipulation strategy is mutually excluded with itself. In addition, BH is mutually excluded with LH and RH.

Event Sequence: S1 S3 F2 E3 S4 S2 F3 E4 F1 E2 E1

Event Constraints:

	S1	S3	F2	E3	S4	S2	F3	E4	F1	E2	E1
Obj1	LH/RH										LH/RH
Obj2						LH/RH				LH/RH	
Obj3		LH/RH		LH/RH							
Obj4					LH/RH			LH/RH			
Fea1									LH/RH		
Fea2			LH/RH								
Fea3							LH/RH				

Manipulation Plans:

	S1	S3	F2		E3		S4	S2		F3		E4		F1	E2		E1
Obj1	RH	RH	LT	LT	RH	RH	RH	LT	LT	LT	LT	RH	RH	RH	LH	LH	LH
Obj2									RH	RS	RS	RS	RS	RS	RH		
Obj3		LH	LH	LH	LH												
Obj4							LH	LH	LH	LH	LH	LH					
Fea1														LH			
Fea2				RH													
Fea3											RH						

Figure 5: *The optimal event sequence, the event constraints and the derived manipulation plans for the scenario in Figure 2. Each row of the manipulation plans corresponds to an object and indicates the manipulation strategies planned for achieving the optimal event sequence.*

Figure 4(b) shows that (S_1) and (S_2) cannot be connected directly because there is a wall between them.

Before we can traverse the event graph, we need to identify two types of constraints. *Precedence constraints* enforce that all S_i nodes appear prior to their corresponding E_i nodes in the event sequence, because an object cannot be released before it is picked up. *Capacity constraints* make sure that currently "active" objects only employ manipulation strategies affordable by the character. We consider object i active if S_i is in the current path but E_i is not. To satisfy capacity constraints, the set of manipulation strategies employed by currently active objects must match one of the nodes in the aggregate manipulation graph.

We can now apply a standard depth-first-search (DFS) on the event graph to find a shortest path that visits every S node and E node exactly once, subject to precedence and capacity constraints. The output path is the optimal event sequence, $P = \{p_1, \cdots, p_n\}$ that achieves the required manipulation tasks. Based on a feature's state and description, we can remove its corresponding event from P if it does not require manipulation. For example, if a light switch is already on, (e.g. F_3 in Figure 2, Left) when the character enters the room, this feature event can be removed from the event sequence.

Since we know the location of each event in P, we can compute a spatial path for the root trajectory that visits every event in a sequence using P as a guide. In addition to reaching the event locations, the path must approach each event at the desired angle and avoid obstacles in the environment. Our algorithm first converts the given environment map to a distance map based on the locations of obstacles (e.g. walls and furniture). We then connect each pair of consecutive event locations with a Hermite curve, such that incoming tangents meet the desired approaching angles. If the curve intersects with an obstacle on the distance map, we move the point of deepest penetration to a collision-free location and subdivide the curve at that point. This step is repeated recursively until the curve is collision-free.

4.3 Manipulation Planner

From the optimal event sequence, we can derive manipulation plans (Figure 5) by searching for a valid path, $Q = \{q_1, \cdots, q_n\}$, on the aggregate manipulation graph, where q_i represents the manipulation strategy associated with event p_i. Each event in the event sequence imposes certain constraints on the search problem. For example, if the event corresponds to pick-up or release of an object ($p_i = S_j$ or $p_i = E_j$), the manipulation strategy is constrained to left or right hand ($q_i[j]$ =LH or RH). If the event corresponds to a feature ($p_i = F_j$), the manipulation strategy must match one of the strategies allowed by that feature. The top table in Figure 5 shows the event constraints for each object-event pair. The goal of the search algorithm is to fill in the manipulation strategies between S_i and E_i for each row O_i.

We apply DFS on the aggregate manipulation graph to find a valid path Q, subject to the event constraints. However, the manipulation plans resulting from Q can be unrealistic because they might contain too many "ground states" between the pick-up and release of an object, or transition between different strategies too frequently. As a result, the character's behavior may appear unnatural and unintelligent. To solve these issues, we prioritize the edges at each node during DFS, such that the adjacent nodes with no ground state and with the same manipulation strategies as the current node will be selected first. In addition, we allow the character more flexibility to rearrange the active objects before executing each event. For example, the character can tuck the book under its arm before opening the door. To achieve this relaxation, we allow the solution path to take an extra node before each q_i, i.e. adding another column before each S or E event and resulting a path $Q = \{q'_1, q_1, \cdots, q'_n, q_n\}$. Once a valid Q is found, we remove redundant nodes q'_i if $q'_i = q_i$.

4.4 From events to tasks

Before we can execute the actions specified by the manipulation plan, we need to translate the events in the manipulation plan to a set of concrete motor tasks. For example, a transition from LH to LS (Figure 2, Object 2) requires a task to transport the object from the left hand to the left shoulder, followed by another task to move the left hand back to a neutral position. For the examples shown in this paper, four different tasks are defined: a tracking task, a holding task, a transporting task, and an attention task. We describe these tasks in further detail in Section 5.2.

The tasks associated with each event transition can be stored at the edge of a manipulation graph. We group the edges into three categories:

- L/RH $\rightarrow *$: A transition, from a hand to any manipulation strategy, associated with a *transporting* task followed by a *tracking* task.
- $* \rightarrow$ L/RH: A transition, from any manipulation strategy to a hand, associated with a *tracking* task followed by a *transporting* task.
- $q \rightarrow q$: A self transition associated with a *holding* task.

We can now map the event transitions in the manipulation plan to a set of consecutive and concurrent motor tasks. In the next section, we will introduce an algorithm to execute these tasks efficiently.

5 Execution

The manipulation plans and the root path provide guidance for executing the character's motion. Our algorithm uses a forward simulator to physically simulate upper-body motion, while the root,

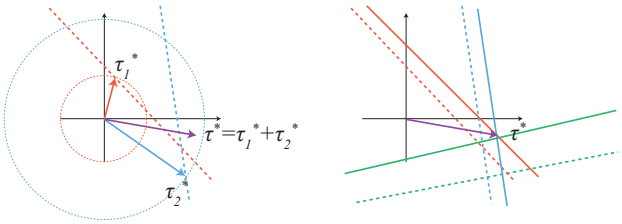

Figure 6: *Left: Computation of optimal torque τ^*. Assuming there are two active tasks shown in red and blue, the dashed line indicates the torques that satisfy the task and the dashed circle indicates the torque bounds for the task. The optimal torque for the "red" task, shown as the red arrow, must lie on the red dashed line within the red circle, and be as parallel as possible to the blue dashed line. Similarly, the blue arrow indicates the torque that achieves the "blue" task while having the least interference with the red task. The final torque is the sum of these two individual torques shown as the purple arrow. Right: The optimal next pose. Changing the pose for the next time step effectively changes the future task spaces. Consider the two active tasks (red and blue dashed line) and a currently inactive task shown as a green dashed line. The optimal next pose will create new task spaces (solid lines) such that their intersection is closer to the current optimal torque (purple arrow).*

lower-body, and finger motions are generated by kinematic procedures.

Our main focus in this section is the simulation of upper-body multitasking. Because the character might not be able to execute all the tasks concurrently, we need an efficient task scheduler and an effective control algorithm for multitasking.

5.1 Multitask Controller

We first review the formulation of task-space dynamics and control. Let $\mathbf{q} \in \mathbf{R}^n$ be the independent degrees of freedom (DOFs) in the upper body of the character. The equations of motion in generalized coordinates can be expressed as follows:

$$\tau = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) - \mathbf{J}_p^T\mathbf{F} \tag{1}$$

where $\mathbf{M}$ denotes the mass matrix, $\mathbf{C}$ includes Coriolis, centrifugal, and gravitational forces, $\mathbf{J}_p$ is the Jacobian matrix that projects external force $\mathbf{F}$ applied at a body point $\mathbf{p}$ from Cartesian to generalized coordinates, and τ represents the generalized control forces applied internally by the character. If a task is to control the acceleration of a Cartesian point $\mathbf{x}$ on the character, it is more convenient to express the equations of motion also in the Cartesian space as follows:

$$\mathbf{J}\mathbf{M}^{-1}\tau = \ddot{\mathbf{x}} + \mathbf{J}\mathbf{M}^{-1}\mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) - \dot{\mathbf{J}}\dot{\mathbf{q}} - \mathbf{J}\mathbf{M}^{-1}\mathbf{J}_p^T\mathbf{F} \tag{2}$$

Note that $\mathbf{J}$ is the Jacobian matrix evaluated at the point $\mathbf{x}$ and is in general different from $\mathbf{J}_p$. Equation 2 represents a simple linear relation between the commanding force $\ddot{\mathbf{x}}$ for the task and the required joint torques τ.

$$\mathbf{A}\tau + \mathbf{b} = \ddot{\mathbf{x}} \tag{3}$$

where $\mathbf{A} = \mathbf{J}\mathbf{M}^{-1}$ and $\mathbf{b} = -\mathbf{A}\mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) + \dot{\mathbf{J}}\dot{\mathbf{q}} + \mathbf{A}\mathbf{J}_p^T\mathbf{F}$.

Task scheduling. Using the linear relationship in Equation 3, we can define a task-inconsistency metric to measure the interference among multiple tasks. Let $\mathbf{P} \in \mathbf{R}^{n\times m}$ $(n > m)$ be the matrix whose range is the nullspace of $\mathbf{A}$. The torques that satisfy Equation 3 can be expressed as $\tau = \tau' + \mathbf{P}\mathbf{z}$, where τ' is a particular solution of Equation 3 and $\mathbf{z}$ is an arbitrary vector in $\mathbf{R}^m$. Given two tasks T_i and T_j, a torque that satisfies T_i is considered consistent with T_j if it is in the range of $\mathbf{P}_j$. Therefore, we can define the most "multitaskable" torque for T_i as $\tau_i^* = \tau_i' + \mathbf{P}_i\mathbf{z}_i^*$, where $\mathbf{z}_i^*$ is the minimizer of the following convex optimization.

$$\mathbf{z}_i^* = \underset{\mathbf{z}_i}{\operatorname{argmin}}\ g(\mathbf{z}_i;\mathbf{P}_j) = \|\mathbb{P}_j(\tau_i' + \mathbf{P}_i\mathbf{z}_i) - (\tau_i' + \mathbf{P}_i\mathbf{z}_i)\|^2$$
$$\text{subject to}\quad \|\tau_i' + \mathbf{P}_i\mathbf{z}_i\|_2 \le d_i \tag{4}$$

$\mathbb{P}_j = \mathbf{P}_j(\mathbf{P}_j^T\mathbf{P}_j)^{-1}\mathbf{P}_j^T$ denotes the projection matrix onto $\mathbf{P}_j$, and d_i defines the torque bounds for T_i. It is critical to enforce bounds on the resulting torque so that the character does not use excessive torque to multitask. The residual of the optimization $r_i = g(\mathbf{z}_i^*;\mathbf{P}_j)$ measures how inconsistent T_i is with respect to T_j (Figure 6, Left).

When the character is dealing with a larger set of tasks, we simply replace $g(\mathbf{z}_i;\mathbf{P}_j)$ in Equation 4 with $g(\mathbf{z}_i;\bigcap_j \mathcal{R}(\mathbf{P}_j))$, where $\mathcal{R}(\mathbf{P})$ denotes the range of $\mathbf{P}$ and $\bigcap_j \mathcal{R}(\mathbf{P}_j)$ is the intersection of ranges of all tasks in the set except for task i. The residual of this optimization, $r_i = g(\mathbf{z}^*)$, indicates the inconsistency between T_i and the rest of the tasks. At each time step, the multitask controller computes the inconsistency metric r_i for every candidate task according to the manipulation plan. If the sum, $\bar{r} = \sum_i r_i$, is greater than a certain threshold, tasks are removed one by one until $\bar{r}$ is below the threshold. The order used to remove tasks is predefined based on task types: 1. Attention, 2. Tracking, 3. Transporting, 4. Holding. The remaining tasks constitute the active task set $\mathcal{A}^{(t)}$ for the current time step. The final optimal torque is computed as $\tau^* = \sum_{i\in\mathcal{A}^{(t)}} \tau_i^*$.

Optimal next pose. The algorithm described so far computes the optimal torque at each time step to best achieve currently activated tasks ($\mathcal{A}^{(t)}$), but it does not have any effect on the task space in the future. A more efficient way of multitasking requires the character to not only achieve the currently active tasks, but to anticipate other inactive candidate tasks. Because the task space parameters depend on the character's pose, i.e. both $\mathbf{A}$ and $\mathbf{b}$ are functions of $\mathbf{q}$, we can search for an ideal next pose which defines a task space more consistent with currently inactivate candidate tasks. In addition, the task space at the next time step should be similar to the current one so that the optimal torque computed by Equation 4 can be continuous over time.

To this end, our algorithm optimizes the pose $\mathbf{q}^{(t+1)}$ at the next time step such that the intersection of all the candidate task spaces, which depends on $\mathbf{q}^{(t+1)}$, is brought closer to the currently optimal torque τ^* (Figure 6, Right). We use a similar formulation as described in Section 5.1; the torques that achieve a task at the next time step must satisfy Equation 3, with $\mathbf{A}$ and $\mathbf{b}$ evaluated at $\mathbf{q}^{(t+1)}$. If there are multiple tasks at the next time step, we simply stack all the linear equations to obtain aggregate $\mathbf{A}$ and $\mathbf{b}$. The general solution for a torque that achieves all the tasks at next time step can be expressed as $\tau(\mathbf{q}^{(t+1)}) = \tau'(\mathbf{q}^{(t+1)}) + \mathbf{P}(\mathbf{q}^{(t+1)})\mathbf{z}$. Our algorithm optimizes the task space by finding a $\mathbf{q}^{(t+1)}$ such that the future intersection of task spaces is closer to the current optimal torque τ^*.

$$\underset{\mathbf{q}^{(t+1)},\,\mathbf{z}}{\operatorname{argmin}}\ \|(\tau'(\mathbf{q}^{(t+1)}) + \mathbf{P}(\mathbf{q}^{(t+1)})\mathbf{z}) - \tau^*\|^2 \tag{5}$$

Because the optimal value for $\mathbf{z}$ can be expressed analytically as $\mathbf{z}^* = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T(\tau^* - \tau')$, Equation 5 can be rewritten as

$$\underset{\mathbf{q}^{(t+1)}}{\operatorname{argmin}}\ \|(\mathbf{P}(\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T - \mathbf{I})(\tau^* - \tau')\|^2 \tag{6}$$

where optimization variables $\mathbf{q}^{(t+1)}$ are suppressed for clarity.

Once the optimal next pose, $\mathbf{q}^{*(t+1)}$, is computed, we still need to incorporate it into the current time step. We do so by exploiting redundancy in control space, as described in the next paragraph.

Prioritized task force. The final control force $\bar{\tau}$ is the sum of multiple prioritized commanding forces. Using the operational space control framework ([Khatib et al. 2004]), we resolve potential interference among tasks by projecting the secondary commanding forces τ_s onto the nullspace of the primary task space: $\bar{\tau} = \tau_p + \mathbb{P}\tau_s$, where τ_p is the primary commanding force. In our formulation, the optimal torque τ^* required to achieve currently active tasks is the primary commanding force. Tracking the optimal next pose $\mathbf{q}^{*(t+1)}$ is not essential for the current tasks, but it will make it easier to multitask in the future. Therefore, we track $\mathbf{q}^{*(t+1)}$ as a secondary task so that it does not interfere with τ^*. In addition, we add a damping term for all the joints and make the damping force as a secondary task as well. The final commanding force can be written as

$$\bar{\tau} = \tau^* + \mathbb{P}(k_p(\mathbf{q}^{*(t+1)} - \mathbf{q}) - k_v\dot{\mathbf{q}}) \tag{7}$$

Additional forces. In addition to control forces, we also apply a gravity compensation force and a fictitious force to account for the effect of the lower body acceleration. We first compute the root acceleration, $\mathbf{a}_r$, by applying finite difference on the root positions generated by the locomotion synthesizer. The fictitious force, $-m_i\mathbf{a}_r$, is then added to each body node in the upper body, where m_i is the mass of body node i. Our simulation also considers the joint limits of the character. We compute the constraint force to enforce joint limits as a linear complementary problem.

5.2 Types of Tasks

This subsection provides the implementation details for the tasks we used to generate the examples in this paper. Using the following formulation, we can compute a particular solution τ' by solving Equation 3.

Tracking task: A task that moves a Cartesian point on the character toward a desired location $\bar{\mathbf{x}}$ in the world with desired speed $\bar{\mathbf{v}}$. We use a proportional-derivative (PD) controller to determine the commanding force: $\ddot{\mathbf{x}} = k_p(\bar{\mathbf{x}} - \mathbf{x}) + k_v(\bar{\mathbf{v}} - \dot{\mathbf{x}})$. A tracking task can also track the desired joint angle and joint velocity. In that case, the commanding force $\ddot{\mathbf{x}}$ represents the desired joint acceleration and $\mathbf{J}$ becomes an identity matrix.

Holding task: A task that maintains the current location of a Cartesian point $\mathbf{x}$ on the character against an external force $\mathbf{F}_x$ applied at $\mathbf{x}$. For example, if an object with mass m is held at $\mathbf{x}$, we set $\mathbf{b} = -\mathbf{AC}(\mathbf{q},\dot{\mathbf{q}}) + \dot{\mathbf{J}}\dot{\mathbf{q}} + \mathbf{AJ}^T m\mathbf{g}$. In addition, we set the commanding force to: $\ddot{\mathbf{x}} = -k_v\dot{\mathbf{x}}$ to avoid non-zero velocity at $\mathbf{x}$. A holding task can also maintain the current orientation of a body point $\mathbf{x}$ against an external force. Controlling the orientation can be done via a commanding torque $\dot{\omega} = -k_v\omega$, where ω is the angular velocity of the body node $\mathbf{x}$ resides. Equation 3 can be modified to: $\mathbf{A}\tau + \mathbf{b} = \dot{\omega}$, where $\mathbf{A} = \mathbf{J}_\omega\mathbf{M}^{-1}$ and $\mathbf{b} = -\mathbf{AC}(\mathbf{q},\dot{\mathbf{q}}) + \dot{\mathbf{J}}_\omega\dot{\mathbf{q}} + \mathbf{AJ}^T m\mathbf{g}$. The Jacobian $\mathbf{J}_\omega \in \mathbf{R}^{3\times n}$ relates the angular velocity of the Cartesian vector to joint velocity: $\omega = \mathbf{J}_\omega\dot{\mathbf{q}}$.

Transporting task: A task that combines the effort of holding and tracking to move an object to a desired location. We set $\ddot{\mathbf{x}} = k_p(\bar{\mathbf{x}} - \mathbf{x}) - k_v\dot{\mathbf{x}}$ and $\mathbf{b} = -\mathbf{AC}(\mathbf{q},\dot{\mathbf{q}}) + \dot{\mathbf{J}}\dot{\mathbf{q}} + \mathbf{AJ}^T m\mathbf{g}$. For both tracking and transporting tasks, we can gradually move the target point from the initial position to $\bar{\mathbf{x}}$ along a straight line with a bell-shape velocity profile to generate more natural human reaching motion.

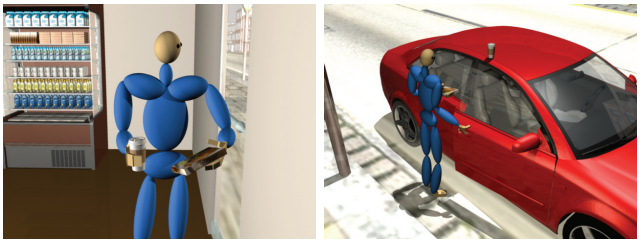

Figure 7: *Simulated motion in the coffee shop example.*

Attention task: A task that controls the look-at direction of the character by setting the commanding torque as $\dot{\omega} = k_p\theta(\mathbf{v}\times\mathbf{u}) - k_v\omega$, where $\mathbf{v}$ is the current look-at direction, $\mathbf{u}$ is the target look-at direction, and θ is the angle between $\mathbf{u}$ and $\mathbf{v}$. The attention task is initiated when the character starts to approach an object or an environment feature using one of the manipulation strategies. We define an attention zone as a sphere centered at the object of interest. A valid look-at direction is then defined as a vector from the location of the eyes to any point in the attention zone. Because real humans tend to look at the object carefully only at the beginning part of the reaching motion, we increase the radius of the attention zone as the character's manipulator gets closer to the object or the feature. This treatment introduces more overlap between attention zones of different tasks and allows the characters to manipulate multiple objects concurrently.

5.3 Locomotion and Finger Motion Synthesizer

We adopt existing work on motion blending and motion graphs to generate locomotion. A small set of mocap sequences containing a straight walk, single steps, and turning motions is used to create continuous walking sequences. We use the same method described by Kovar *et al.* [2002] to detect transition points in the dataset. Given the spatial path produced by the event planner, a sequence of walking cycles is selected and blended to follow the path. Additionally the locomotion synthesizer may refine its motion based on the proximity of active tasks. For example, if two events are very close to each other (e.g. E_3 and S_4 in Figure 2), the synthesizer may deviate from the original path and produce a small step toward the second one instead of a full walking and turning sequence.

The hand grasping motion is kinematically generated via a few keyframes and interpolation. When the hand is sufficiently close to grab the object, we stop simulating the object and rigidly attach the object to the hand. When the character releases the object, we resume the physical simulation on the object.

6 Results

We construct an articulated human character with 16 DOFs on the upper body and 18 DOFs on the lower body. The upper body motion is simulated using a rigid multibody simulator, DART [DAR]. We use a general optimization software, SNOPT [Gill et al. 1996], to solve for quadratic programs (Equation 4) and nonlinear programs (Equation 6). We create two examples to showcase the ability of the character to multitask in different scenarios: a baseline living room scenario and its variations, as well as a scenario in a coffee shop. The path planning and manipulation planning for the baseline living room example take 1.62 and 0.04 seconds respectively. On average, the simulation runs 3.5 times slower than real-time.

6.1 Living Room Example

Baseline scenario. Our baseline scenario is constructed from the example shown in Figure 2. According to the manipulation plans, the character picks up a book (S_1) with the right hand and a mug (S_3) with the left hand, and walks toward the bedroom while tucking the book under the left arm before opening the door with the right hand (F_2). The character then moves the book back to the right hand before placing the mug on the table (E_3), grabs a crumpled paper using the left hand (S_4), tucks the book back under the left arm to allow the right hand to pick up a bookbag (S_2), puts the bookbag on the right shoulder and uses the right hand to turn off the light (F_3). After walking toward the corner of the living room while moving the book back to the right hand, the character drops the paper from the left hand (E_4) and walks toward the front door. Finally, the character turns off the switch using the left hand (F_1). This complex plan is computed automatically by our manipulation planner.

Please view the resultant motion in the accompanying video. For this example, we modify the tracking task slightly for two occasions, reaching the torso and reaching the shoulder, to improve the aesthetics of the motion. Instead of setting a target position to $\bar{\mathbf{x}}$, we predefine a target trajectory such that the motion of the arm moves more naturally. Because both endpoints of the trajectory are determined in the character's local coordinates for these two special cases, we do not need to modify the trajectory for different objects or locations of the character.

Changing the environment map. We can modify any property on the environment map, delete or add objects, or change the manipulation graph for each object. Our algorithm then automatically produces a new manipulation plan for the multitask controller. In the example shown in Figure 2, we change the start configuration of the crumpled paper (S_4) and the end configuration of the mug (E_3). These two modifications drastically change the event sequence and manipulation plans. Please see the accompanying video for the resultant motion.

6.2 Coffee Shop Example

In the second scenario, the character drops by a coffee shop to buy lunch. The manipulated objects include a cup of coffee and a lunch box, both of which have the same manipulation graph as Object 3 and Object 4 in Figure 2. In addition, we introduce two different types of doors in this scenario: a door that can be pushed open with an elbow and a car door that can only be opened by a hand (Figure 7).

After taking the coffee in his right hand, the character picks up a lunch box from the refrigerator using his left hand. When the character walks toward the door, our planning algorithm prefers to use his left elbow to push the door open instead of letting him put one of the items on the ground. Finally, when the character reaches his car outside, he cannot open the car door using any manipulation strategies except for LH or RH, which are both occupied by other objects. The only option left is to temporarily leaves one of the items on the closest surface (i.e. choosing GR strategy). In our example, the character chooses to put the coffee cup on the roof of the car.

6.3 Evaluations

Changing object properties. One advantage of using physics simulation to generate manipulation motion is that the character can react differently to objects with different physical properties. To demonstrate the effect of dynamics, we modify the physical properties of the object and compare the changes in the motion. In one example, as shown in Figure 8, the character tries to put the bookbag on the shoulder while holding a mug with the other hand. We show that the character leans further to the side when picking up a heavier bookbag due to dynamics, but still manages to maintain the upright orientation of the mug. In reality, it takes less effort to pick up a heavy object if we lean away from the object. To simulate this effect, our optimization will need to have an additional term that minimizes joint torque usage.

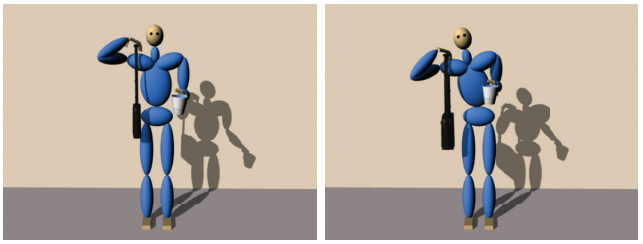

Figure 8: *Comparison between objects with different mass.*

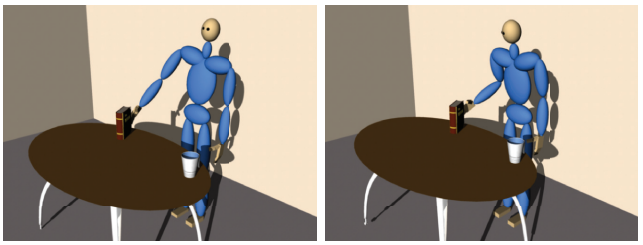

Figure 9: *Comparison between a simple pose tracking approach and our method.*

Comparison with a pose tracking approach. One simple method for generating multitasking motion is to apply inverse kinematics (IK) to solve for a target pose that satisfies multiple Cartesian constraints, and use a proportional-derivative (PD) control scheme to track the target pose. This method may work in some situations, but it has a few drawbacks compared to our method. First, the trajectory required to achieve the target pose highly depends on unintuitive parameters of the PD trackers, whereas the motion trajectory generated by our method depends on multitask consistency. Second, tracking a target pose alone does not take inactive candidate tasks into account. Our method, on the other hand, continuously adjusts the character's poses in anticipation of future tasks. Third, it is hard to produce a natural target pose using standard IK without exploiting many example poses, while our method does not require any upper body poses. We demonstrate the difference between our method and a pose tracking approach in an example shown in Figure 9 where the character tries to reach two objects in sequence. The motion generated by pose tracking is clearly less natural than our result as shown in the video. Motion capture data could be used as guidance for tasks involving tracking. However, it is difficult to acquire appropriate mocap data in advance for all manipulation tasks and their combinations.

Optimal next pose. We demonstrate the effect of an optimal next pose using two challenging scenarios involving a few inconsistent tasks. In the first example as shown in the top row of Figure 10, the character tries to reach an object on a lower surface using the right hand while keeping the bookbag strap from sliding down the right shoulder. At the same time, the character must maintain the orien-

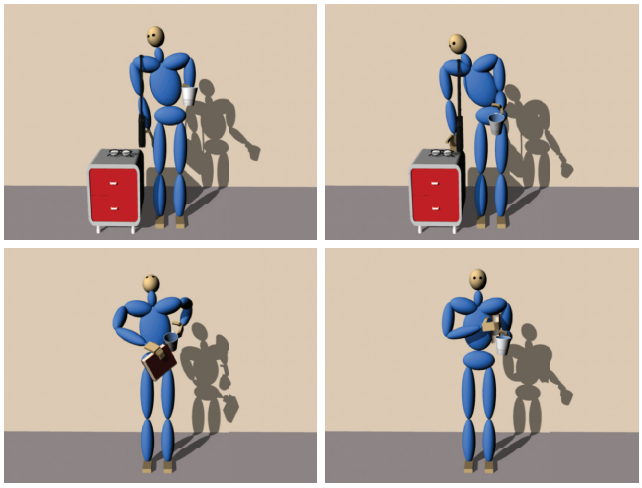

Figure 10: *Top row: Comparison between the results with an optimal next pose (right) and without (left). Bottom row: Comparison between the use of a unified large torque bound for all tasks (left) and a proper torque bound for each individual task.*

tation of the mug in the left hand. Without the optimal next pose, the character can satisfy the tasks to maintain the orientation of the right shoulder and the left hand, but it fails to reach the object. On the other hand, with the optimal next pose, the character continuously adjusts its pose to lean toward the right side and eventually reaches the object. In the second example, the character tries to maintain the orientation of the mug in the left hand while tucking a book under the left arm using the right hand. Due to high inconsistency between these tasks, the orientation task eventually becomes inactive. Without the optimal next pose, the character completely ignored the orientation of the mug, resulting even greater inconsistency between these tasks.

Effect of task-specific torque bounds. As an alternative to our formulation for task scheduling, we can directly compute a torque vector which is the closest in Euclidean distance to the intersection of all active tasks (in Figure 6, the purple arrow would point at the intersection of the red and the blue lines). The drawback of this method is that we can only set a single torque bound for the aggregate torque that combines all the active tasks (i.e. the bound on the magnitude of the purple vector in Figure 6). In contrast, our method provides flexibility to define different torque bounds for different tasks, resulting in much more natural motion for multitasking. We test the alternative method on the two examples used for testing the optimal next pose. In both examples, we find that it is difficult to determine a single torque bound for all active tasks. When the torque bound is too high, the character uses excessive torques to multitask, quickly leading to simulation blowup as shown in the bottom row of Figure 10. When the torque bound is too low, the character fails to achieve certain tasks that require a larger amount of torque. We define the torque bounds approximately based on the mass of the subtree rooted at each joint.

6.4 Limitations

Our current algorithm has a few limitations. First, we assume that the manipulation tasks are primarily done by the upper body and locomotion is done by the lower body. For simple pick-up and placement tasks that do not require much physical strength, this assumption can generate reasonable results. However, for more general whole-body manipulation tasks, such as pushing a heavy door or lifting a heavy object, coordination between locomotion and upper body manipulation is vital.

Our path planner implementation is very primitive and unable to handle extremely cluttered environments. Furthermore, we do not have a path planner at the level of upper body manipulation. When manipulating in a tight space, such as fetching an item in a packed refrigerator, the character's upper body is likely to collide with the environment or fail to move completely. To circumvent this issue, we plan to investigate a few broadly applied randomized algorithms proposed in previous work [Lavalle and Kuffner 2000; Kavraki et al. 1996].

Our algorithm only considers the shortest distance when planning the events. This is noticeable when we change the input of the baseline example, as the character carries both the mug and the book in and out of the smaller room before exiting, instead of picking them up on the way out. Other additional criteria, such as the amount of effort required for each task, could be taken into account during the DFS.

The walking motion in our examples can be largely improved if we use a larger mocap database and beter motion editing algorithms. Our motion graph currently only contains 13 short clips.

7 Conclusion

We introduce a physics-based technique to synthesize human activities involving concurrent full-body manipulation of multiple objects. To capture how humans deftly exploit different properties of body parts and objects for multitasking, we need to solve challenging planning and execution problems. Given an environment map along with the information about the objects and features in the environment, and manipulation graphs that describe all possible strategies to hold, move, push, or release an object, our algorithm generates a continuous animation of the character manipulating multiple objects and environment features concurrently at various locations in a constrained environment.

One interesting future direction is to integrate our manipulation controller with fully simulated biped locomotion. Physically simulating whole-body manipulation can raise new challenges in balance control. We are particularly interested in integrating our upper body controller with the biped walking controller developed by Coros *et al.* [Coros et al. 2010]. Their work has demonstrated a variety of walking related skills, such as picking up and carrying objects. In addition, we are interested in investigating robot control strategies for whole-body manipulation, such as lean-and-push or lift-and-push, to handle heavy and large objects.

The final piece to complete a fully simulated virtual character is dextrous manipulation. Many realistic multitask behaviors can be enabled by better prehension control algorithms. For example, the character can hook-grasp multiple coffee mugs and open a cabinet door all with one hand. Furthermore, a large repertoire of object manipulation tasks require precise collaboration between two hands. Developing coordinated bimanual controllers can be another exciting future research direction.

References

ABE, Y., AND POPOVIĆ, J. 2006. Interactive animation of dynamic manipulation. In *Eurographics/SIGGRAPH Symposium on Computer Animation*.

BRUDERLIN, A., AND WILLIAMS, L. 1995. Motion signal processing. In *SIGGRAPH*, 97–104.

CHOI, M. G., LEE, J., AND SHIN, S. Y. 2003. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Trans. Graph. 22* (April), 182–203.

COROS, S., BEAUDOIN, P., AND VAN DE PANNE, M. 2010. Generalized biped walking control. *ACM Trans. Graph. 29* (July), 130:1–130:9.

DART: Dynamic Animation and Robotics Toolkit, http://dart.golems.org/.

DE LASA, M., AND HERTZMANN, A. 2009. Prioritized optimization for task-space control. In *IEEE/RSJ international conference on Intelligent robots and systems*, IROS'09, 5755–5762.

DE LASA, M., MORDATCH, I., AND HERTZMANN, A. 2010. Feature-based locomotion controllers. *ACM Trans. Graph. 29*.

FENG, A. W., XU, Y., AND SHAPIRO, A. 2012. An example-based motion synthesis technique for locomotion and object manipulation. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*.

GILL, P., SAUNDERS, M., AND MURRAY, W. 1996. Snopt: An sqp algorithm for large-scale constrained optimization. Tech. Rep. NA 96-2, University of California, San Diego.

HARADA, K., KAJITA, S., KANEKO, K., AND HIRUKAWA, H. 2003. Pushing manipulation by humanoid considering two-kinds of zmps. In *IEEE International Conference on Robotics and Automation*, vol. 2, 1627–1632.

HUANG, Y., MAHMUDI, M., AND KALLMANN, M. 2011. Planning humanlike actions in blending spaces. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

KALLMANN, M., AUBEL, A., ABACI, T., AND THALMANN, D. 2003. Planning collision-free reaching motions for interactive object manipulation and grasping. *Computer graphics Forum 22*, 3 (Sept), 313–322.

KALLMANN, M. 2004. Interaction with 3-d objects. In *Handbook of Virtual Humans*. 303–322.

KALLMANN, M. 2005. Scalable solutions for interactive virtual humans that can manipulate objects. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment*, 69–74.

KAVRAKI, L., SVESTKA, P., LATOMBE, J.-C., AND OVERMARS, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. on Robotics and Automation 12*, 4, 566–580.

KHATIB, O., SENTIS, L., PARK, J., AND WARREN, J. 2004. Whole-body dynamic behavior and control of human-like robots. *International Journal of Humanoid Robotics 1*, 1, 29–43.

KHATIB, O. 1987. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal of Robotics and Automation 3*, 1 (february), 43 –53.

KOGA, Y., KONDO, K., KUFFNER, J., AND LATOMBE, J.-C. 1994. Planning motions with intentions. In *SIGGRAPH*, 395–408.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. In *SIGGRAPH*, 473–482.

LAMARCHE, F., AND DONIKIAN, S. 2001. The orchestration of behaviours using resources and priority levels. In *Eurographic workshop on Computer animation and simulation*, 171–182.

LAU, M., AND KUFFNER, J. J. 2005. Behavior planning for character animation. In *ACM SIGGRAPH/Eurographics symposium on Computer animation*, 271–280.

LAVALLE, S. M., AND KUFFNER, J. J. 2000. Rapidly-exploring random trees: Progress and prospects.

LIU, Y., AND BADLER, N. I. 2003. Real-time reach planning for animated characters using hardware acceleration. In *International Conference on Computer Animation and Social Agents*, 86–.

MISTRY, M., AND RIGHETTI, L. 2011. Operational space control of constrained and underactuated systems. In *Proceedings of Robotics: Science and Systems*.

NAKAMURA, Y., HANAFUSA, H., AND YOSHIKAWA, T. 1987. Task-priority based redundancy control of robot manipulators. *Int. J. Rob. Res. 6* (July), 3–15.

NISHIWAKI, K., YOON, W.-K., AND KAGAMI, S. 2006. Motion control system that realizes physical interaction between robot's hands and environment during walk. In *International Conference on Humanoid Robots*, 542–547.

RIJPKEMA, H., AND GIRARD, M. 1991. Computer animation of knowledge-based human grasping. *SIGGRAPH Comput. Graph. 25* (July), 339–348.

ROSE, C., COHEN, M. F., AND BODENHEIMER, B. 1998. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications 18*, 5 (Sept.).

SENTIS, L., AND KHATIB, O. 2005. Control of free-floating humanoid robots through task prioritization. In *IEEE International Conference on Robotics and Automation*.

SHAPIRO, A., KALLMANN, M., AND FALOUTSOS, P. 2007. Interactive motion correction and object manipulation. In *Symposium on Interactive 3D graphics and games*, ACM, 137–144.

STILMAN, M., SCHAMBUREK, J.-U., KUFFNER, J., AND ASFOUR, T. 2007. Manipulation planning among movable obstacles. In *IEEE International Conference on Robotics and Automation*.

TAKUBO, T., INOUE, K., AND ARAI, T. 2005. Pushing an object considering the hand reflect forces by humanoid robot in dynamic walking. In *IEEE International Conference on Robotics and Automation*, 1706 – 1711.

YAMANE, K., KUFFNER, J. J., AND HODGINS, J. K. 2004. Synthesizing animations of human manipulation tasks. *ACM Trans. Graph. 23* (Aug.), 532–539.

YOSHIDA, E., BELOUSOV, I., ESTEVES, C., AND LAUMOND, J.-P. 2005. Humanoid motion planning for dynamic tasks. In *IEEE International Conference on Humanoid Robotics*.