2008 8th IEEE-RAS International Conference on Humanoid Robots
December 1 ~ 3, 2008 / Daejeon, Korea

An Improved Hierarchical Motion Planner for Humanoid Robots

Salvatore Candido [1], Yong-Tae Kim [2], Seth Hutchinson [1]

[1] *University of Illinois at Urbana-Champaign, USA candido@illinois.edu, seth@illinois.edu*

[2] *Hankyong National University, South Korea ytkim@hknu.ac.kr*

***Abstract*—In our previous work [1], we proposed a hierarchical planner for bipedal and humanoid robots navigating complex environments based on a motion primitives framework. In this paper, we extend and improve that planner by proposing a different approach for the global and subgoal components of our planner. We continue to use a workspace decomposition that consists of a passage map, obstacle map, gradient map, and local map. We verify our approach using both simulation results and experimentally on a mechanical humanoid system.**

I. Introduction

Motion planning for humanoid robots is computationally expensive due to the high dimensionality of the robots' configuration spaces and the complexity of the environments in which they operate. In our previous work [1], we discussed a workspace decomposition and accompanying hierarchical planner for bipedal and humanoid robots based on a motion primitives framework. In summary, the hierarchical planner worked by planning a route through a two-dimensional map, generated by the workspace decomposition, using a classical motion planning algorithm. Subsequent levels of planning then attempted to construct a sequence of motion primitives to realize that path. This technique worked in some simulations but had some modes of failure. The problem centered on the fact that once a high level plan was generated, there was no mechanism by which to change it if lower levels of the planner could not find a sequence of motion primitives. Because of the greedy nature of the planning technique, we refer to this algorithm as the Greedy Hierarchical Planner (GHP).

In this paper, we propose an algorithm which overcomes some of the deficiencies of the GHP. We build a connectivity graph of the workspace decomposition and store it in a data structure. Querying this data structure gives us additional flexibility when planning sequences primitives. Furthermore, if no sequence of motion primitives can be found corresponding to a given path path through the connectivity graph, this approach can replan for another path through the graph. We refer to this algorithm as the Dijkstra Hierarchical Planner (DHP) due to the type of search used to plan paths through the connectivity graph of the workspace decomposition.

A. Robot Model

The robot's shape $\mathbf{s}$ is an ordered list of all the joint positions of the internal degrees of freedom of the robot. The robot is assumed to be a collection of rigid links whose positions and orientations, relative to one another, are defined by the positions of the joints linking them together. The set $\mathcal{A}_i$ denotes the set of points in the world occupied by the i^{th} link.

A configuration of the humanoid (or bipedal) robot is a complete specification of every point on the robot [2]. We denote the location of a coordinate frame rigidly attached to the robot in an arbitrary but defined place by $\mathbf{x} \in SE(3)$. This coordinate frame specifies a transformation between a fixed world coordinate frame and a specific link of the robot. The robot's shape $\mathbf{s}$ is an ordered list of all the joint positions of the internal degrees of freedom of the robot. We parameterize the configuration with the following representation:

$$\mathbf{q} = (\mathbf{x}, \mathbf{s}) \tag{1}$$

A configuration, $\mathbf{q} \in \mathcal{Q}$, encapsulates the shape of the robot and its position and orientation in the workspace. The notation $\mathcal{Q}$ refers to the configuration space, which is the set of all possible configurations. A parametrization of the configuration, when combined with a specification of the geometry of each link $\mathcal{A}_i$ can be used to uniquely determine the coordinates of every point on the robot. Commonly, the configuration space of a bipedal robot is $\mathcal{Q} = SE(3) \times T^m$ where T^m is the m-dimensional torus. This refers to a robot with m revolute and no prismatic joints. This could correspond to a robot built from m servo motors.

The robot's *state* is the information necessary to characterize the dynamics of the robot. It consists of the robot's configuration and the first time derivative of the configuration. The derivative is specified by scalar values for position and each internal degree of freedom and a matrix belonging to the Lie Algebra of 3×3 skew symmetric matrices of $SO(3)$ [3].

B. Motion Primitives

We consider a motion primitive to be a control policy that when activated at time t from a configuration with a specific initial shape, $\mathbf{s}(t) = \mathbf{s}_i$, takes the system to a configuration with a corresponding specific final shape, $\mathbf{s}(t+\tau) = \mathbf{s}_f$, after τ seconds have elapsed. Essentially, a motion primitive encodes a repeatable pattern of motion that the robot can perform any time its shape is $\mathbf{s}_i$. For example a motion primitive might be the motion associated with a single step.

978-1-4244-2822-9/08/$25.00 ©2008 IEEE

The generation and use of motion primitives has been demonstrated in [4], [5], [6]. The planning algorithm requires only specific parameters of each motion primitive to compute the sequence of motion primitives to be used to navigate the environment. Thus, each primitive is parametrized by the following set:

$$P = (\mathbf{x}, \mathbf{s}_i, \mathbf{s}_f, c, \tau, \mathbf{V}) \tag{2}$$

The net displacement of the robot's root frame in $SE(3)$ during execution of the motion primitive is $\mathbf{x}$. The initial and final shapes the robot will take while executing the motion primitive are $\mathbf{s}_i$ and $\mathbf{s}_f$. The cost of executing this primitive is c. The amount of time the robot takes to execute the motion primitive is τ. A bounding volume $\mathbf{V}$ contains the swept volume the robot moves through while executing this motion primitive. The cost can be based upon a combination of time, energy, risk (potential instability), and general desirability of a given motion.

The swept volume is the union of the set of all points on the robot at every instant of time during execution of the primitive. The bounding volume $\mathbf{V}$ is either the swept volume or a superset of the swept volume, and is used for fast collision checking. As the motion primitive produces a repeatable motion, the bounding volume can be precomputed and stored by the path planning algorithm. By placing the bounding volume at the location of the robot in the workspace, the path planning algorithm can quickly check for collisions with workspace obstacles without calculating any motions of the robot or recomputing the location of the robot's links while executing those motions.

In practice, $\mathbf{V}$ could be the swept volume [7], an approximate bounding volume [8], a bounding box, or an axis-aligned bounding box. Each one of these has trade-offs between computational expense and precision of the approximation. A swept volume is difficult to construct and testing for collisions against polygonal meshes may require a lot of computational operations, but the collision checking is exact. In contrast, an axis-aligned bounding box is simple to construct and testing for collisions is trivial. However, in many cases, too many potential collisions are reported where the robot would actually be far from any obstacle because the bounding volume is an overly conservative approximation. Requiring $\mathbf{V}$ only to bound the swept volume, in practice, allows an approximation that is arbitrarily close to the actual swept volume, although a high degree of precision is often unnecessary.

Use of motion primitives can be thought of as restricting the trajectories allowed in a continuous planning problem to turn it into a discrete problem with a finite action set. This defines a discrete transition function

$$\mathbf{q}_f = f(\mathbf{q}_i, P) \tag{3}$$

where $\mathbf{q}_f = (\mathbf{x}_f, \mathbf{s}_f)$, $\mathbf{q}_i = (\mathbf{x}_i, \mathbf{s}_i)$ and $\mathbf{x}_f = \mathbf{x}_i \cdot \mathbf{x}$ where $\mathbf{x} \in SE(3)$ is a constant transformation that is a parameter of P. A sequence of motion primitives is denoted as $\gamma = (P_1, P_2, \cdots P_K)$. By iteratively expressing the configuration using Equation (3) as each P_i is applied after P_{i-1}, the final configuration after applying the motion primitives of γ at $\mathbf{q}_i$ can be found. Thus, the transition function can also be defined for sequences of motion primitives

$$\mathbf{q}_f = f(\mathbf{q}_i, \gamma) \tag{4}$$

where $\mathbf{x}_f = \mathbf{x}_i \cdot \mathbf{x}_1 \cdot \mathbf{x}_2 \cdots \mathbf{x}_K$ and $\mathbf{x}_k$ is the $\mathbf{x}$ parameter of P_k.

C. Environment

A walking or humanoid robot operates in a workspace that is a subset of $\mathbb{R}^3$. We assume the floor to be mostly on a plane perpendicular to the gravity vector. Obstacles, ramps and holes in the floor may be sparsely scattered throughout the environment. Obstacles are assumed to have steep side walls. Basically, these assumptions correspond to typical indoor environments.

The motion planning algorithm is assumed to have a three-dimensional polygonal map of the robot's workspace denoted as $\mathcal{W} \subset \mathbb{R}^3$. This will be stored as a triangle mesh of the environment for collision checking against a simulation model of the robot. It is assumed that information about the height of the floor (or obstacles sitting on the floor), height of the ceiling, and orientation of the floor can be easily extracted from this map.

D. Problem Statement

The task of the motion planning algorithm is to find a sequence of motion primitives that, when applied, move the robot from a given initial configuration, $\mathbf{q}_{init}$, to a configuration in a desired goal region, $\mathcal{Q}_{goal} \subset \mathcal{Q}$. The set $\mathcal{Q}_{goal}$ will correspond to the robot's body-attached frame being in some region in the workspace, $\mathcal{B}_{goal} \subset SE(3)$, and the robot being able to maintain that position. Thus, every $\mathbf{q} = (\mathbf{x}, \mathbf{s}) \in \mathcal{Q}_{goal}$ will have the properties that $\mathbf{x} \in \mathcal{B}_{goal}$ and $\mathbf{s}$ is at an equilibrium. Given $\mathbf{q}_{init}$, $\mathcal{B}_{goal}$, the model of the robot $\mathcal{A}$, a set of motion primitives $\mathbf{P}$, and a model of the environment $\mathcal{W}$, the goal of the motion planner is to find a sequence of motion primitives γ that satisfies the following conditions:

1) $f(\mathbf{q}_{init}, \gamma) \in \mathcal{Q}_{goal}$.
2) The bounding volume $\mathbf{V}$ of P_k is not in collision with any obstacles when placed at $f(\mathbf{q}_{init}, \gamma)$.
3) For all P_k in γ, $\mathbf{s}_i$ of P_k is the same as $\mathbf{s}_f$ of P_{k-1} and $\mathbf{s}_1 = \mathbf{s}_{init}$.
4) The foot contact orientation of $\mathbf{s}_i$ and $\mathbf{s}_f$ of P_k match the support surface at $\mathbf{q}_i$ and $f(\mathbf{q}_i, P_k)$, respectively.

Furthermore, the planner should seek to minimize the cost of the path, the sum of the costs of each primitive in the sequence.

E. Previous Work

Much research has been done on generating motion primitives and a number of approaches are commonly taken. Gaits are generated manually in some cases by a human designer's intuition or by using motion capture to directly mimic human motion [9]. Computer algorithms such as neural networks, fuzzy logic, learning, and genetic algorithms can also be used to find gait trajectories for the robot [10], [11]. Control policies

based on the robot's zero moment point [12], [4] are common examples of using feedback to stabilize the robot and generate motion primitives. By looking at and controlling indicators of system behavior such as the center of mass, center of pressure, or zero moment point, controls are designed that produce (typically) statically stable gaits. Other interesting methods based on the tools of nonlinear control are virtual model control [13] and hybrid zero dynamics [14]. An energy-efficient approach to biped locomotion derived from the analysis of passive walking devices [15] combines the principles of passive walking together with controlled feedback [16]. In [16], a nonlinear control is used that replicates the principle of passive walking down an incline on arbitrary ground slope. These methods have been shown to be more efficient than strict trajectory tracking methods and it has been conjectured that this approach models the behavior of humans and animals (e.g., [17]).

Some approaches other than own are also based on a simplification of the humanoid motion planning model. Path planning for a humanoid robot on flat ground can be considered as a search for a sequence of feasible motion primitives rather than a search of a high dimensional configuration space for a trajectory [5], [6]. A search algorithm can, in many cases, quickly find a sequence of motions to take the robot to its destination. However, a search will require exponentially more checks as the number of motion primitives in that path increases. In environments with obstacles, stairs, inclines, and holes, the motions of humanoids are constrained by more than just placements of the robot on the plane. A large set of motion primitives is required to traverse these environments. An A* search or forward dynamic programming algorithm may take considerable time to compute a sequence of motion primitives or the problem may become computationally intractable. Some hierarchical planning approaches have been applied to control the growth of complexity and find an approximate solution to this problem [18], [19], [20].

II. Planning Algorithm

We propose a hierarchical algorithm consisting of a global plan, subgoal (or waypoint) plan, and local plans. A decomposition of the three-dimensional workspace is encoded into a set of two-dimensional maps and the higher level layers of the algorithm build heuristics that guide a series of A* searches that are small enough to compute quickly.

The higher levels of planning are conducted in $\mathbb{R}^2$ and $SE(2)$ but the local plan is made in $SE(3)$ with a discrete set of motion primitives that limit the movements of the robot. Local plans must be made in sequence as the initial position for a given local path depends on where in the subgoal region the previous local plan left the robot.

In our previous work, our global planner used a classical motion planning algorithm to overlay a path over our workspace decomposition. Then, our subgoal and local planners used the workspace decomposition, the overlayed global path, and a model of the environment to transform that decomposition into a feasible sequence of motion primitives.

In the algorithm we propose in this paper, the global planner builds a data structure from our workspace decomposition that represents the connectivity of adjacent regions of space that the robot can traverse. This is done as preprocessing. Then, either as preprocessing or online, the subgoal planner searches this data structure for a path to the goal and passes the next subgoal to the local planner.

A. Workspace Decomposition

We have discussed the construction of a workspace decomposition for the GHP in [1]. The same decomposition will be used here and we summarize the procedure for constructing it.

The motion planner is given as input a three-dimensional polygonal mesh modeling the robot's workspace. From this model, the height of the floor (or obstacles sitting on the floor), height of the ceiling and orientation of the floor is extracted into a set of two-dimensional maps. This is called a *2.5-dimensional representation* of the workspace [21] and can be used to simplify computation by providing pertinent information in an easily accessible data structure. The two-dimensional floor height map is referred to as M_f, and $M_f(i,j)$ refers to the element of the grid M_f, at location (i,j), which stores a rational number. Using the resolution of the grid and a knowledge of where the world coordinate frame projects onto a the grid, i and j can be converted to x and y, the projection of the robot's body-attached frame on the support surface in the world frame in $O(1)$ time. Thus, $M_f(i,j)$ stores the height of the support surface or an obstacle sitting on the floor at (x,y). Similarly, M_c, M_ϕ, and M_ψ store the height of ceiling and the orientation of the support surface with respect to the gravity vector. The maps M_ϕ and M_ψ store the angle made between the support surfaces and the x and y axes, respectively.

The robot can only move through regions where the free space between the ceiling and floor is larger than a constant k_p that is related to the robot's height and the clearance that is desired above the robot. The occluded passage map M_p is generated by checking that the difference between ceiling and floor height is greater than k_p. Rather than storing a rational number at each (i,j), M_p stores a Boolean value corresponding to whether or not the robot could stand at (x,y). Some obstacles have sufficient clearance above to allow the robot to stand on top of them, but they are tall enough that the robot has no motion primitives to place itself on top of that obstacle. If the maximum height difference between any point (x,y) and any other point belonging a fixed neighborhood of (x,y) is greater than the constant k_o, then the point is on the edge of a tall obstacle. The value of k_o is chosen with respect to the capabilities of the robot and the set of motion primitives with which it is equipped. For example, if using a set of motion primitives where the maximum height the robot can step onto an obstacle is one foot, k_o should be chosen equal to a foot. If an edge is taller than k_o with respect to its neighboring environment, then the robot will have no chance of stepping on top of it, and (x,y) is flagged in what is called

the obstacle map M_o as a constraint on motion. Also, if the change of orientation of the floor is greater than a different constant k_f, the robot for similar reasons should not attempt to step over that discontinuity, and these areas will also be flagged in the navigation map.

The navigation map M_{nav} is computed by combining the occluded passage map and obstacle map. The free workspace of this map is the set of candidate locations for a path for the robot. This map can be computed with a running time of $O\left(w^2\right)$ where w corresponds to the maximum number of grid cells in either the i or j direction of the map. The *terrain discontinuity map*, denoted M_d, is the map that will be used to choose subgoals for the robot's path. The terrain discontinuity map is constructed by a similar procedure to the navigation map choosing $k_o = k_f = 0$. Any discontinuous change in floor height or orientation will be flagged. The robot cannot step on, but can step over, some of the edges in the terrain discontinuity map. For explicit algorithms to compute M_{nav} and M_d see [22].

B. Global Planner

The objective of the global planner is to create a workspace decomposition that separates the free space of the terrain discontinuity map M_d into a set of regions and represents the ability of the robot to move between these regions in a graph structure. This workspace decomposition is used by the subgoal planner to find a route through the workspace and subgoals for the local planner to use as heuristics.

The first task of the global planner is to build M_d, the terrain discontinuity map. Once M_d is built, the planner partitions it into a set of disjoint regions. It would be desirable for all regions to be convex so planning within a region is simple. The individual regions of the decomposition are chosen as the maximal sets of adjacent grid cells in the free space of M_d (where there is no terrain discontinuity). Because the free regions of M_d are not necessarily convex, a trapezoidal decomposition [2] is performed on all regions that are not convex. Examples of this process can be seen in the simulation results.

The representation of the regions of M_d is stored as a set of polygons. The union of this set of polygons covers the entire two-dimensional projection of the workspace. Since the robot and $\mathcal{Q}_{goal}$ are always in the workspace, the regions onto which the robot's position and $\mathcal{Q}_{goal}$ project are determined by checking those projections against all polygons.

A connectivity graph, denoted $\mathcal{G}$, is built from this representation. A vertex in $\mathcal{G}$ represents a region in the decomposition M_d. Each vertex stores the polygon surrounding its region in the decomposition. An edge represents the robot's ability to move between adjacent regions of the graph using a given motion primitive. This means $\mathcal{G}$ is necessarily directed and multiple edges (directed in the same way) may exist between vertices.

The connectivity graph is initialized with an edge between every two vertices that share a common border in the decomposition of M_d. The set of motion primitives is then be checked to see if one or more motion primitives exist to transition from one region to another. If there is no such motion primitive, the edge is deleted, and the robot will not attempt to transition between these two regions in the workspace decomposition. If a motion primitive is found to facilitate the transition, the edge is retained and augmented with a reference to the motion primitive used to transition into the adjacent region, the cost of performing this transition and the subgoal region, $\mathcal{B}_i \subset SE(3)$, to which the robot's configuration must belong to make the transition. Additional edges are added if more than one motion primitive can be used to move between two regions.

For the edges representing transitions between regions that composed a larger region and that were split during trapezoidal decomposition, transitions do not correspond to a physical obstruction in the workspace. A large subgoal region is used, and a designation is given to the edge to mark that any motion primitive that could be used on the support surface containing this edge will be acceptable to make the transition between regions. Also, a transition cost of zero is given to this edge. Once this processing is complete, $\mathcal{G}$ is stored and passed to the subgoal planner.

Algorithm 1: Subgoal Planner

Input: $\mathcal{G}$, $\mathbf{q}_i$, $\mathcal{Q}_{goal}$
Output: $\mathcal{B}_i$ or failure
$v_{init} \leftarrow$ vertex of $\mathcal{G}$ containing projection of $\mathbf{q}_i$;
$v_{goal} \leftarrow$ vertex of $\mathcal{G}$ containing projection of $\mathcal{Q}_{goal}$;
$\text{Cost}(v_{init}) = 0$;
forall $v \neq v_{init}$ **do**
  $\text{Cost}(v_{init}) = \infty$;
Run Dijkstra's Algorithm from v_{init} *to* v_{goal}, Q *is a set of vertices*;
$Q \leftarrow \{v_{init}\}$;
while $|Q| > 0$ **do**
  $v \leftarrow \underset{v' \in Q}{\text{argmin}}\, \text{Cost}(v')$;
  $Q \leftarrow Q - \{v\}$;
  if $v = v_{goal}$ **then** break;
  forall $(v', e) \in \text{Neighbors}(v)$ **do**
    if $\text{Cost}(v) + \text{Cost}(v, v', e) < \text{Cost}(v')$ **then**
      $\text{Cost}(v') \leftarrow \text{Cost}(v) + \text{Cost}(v, v', e)$;
      $v'.\text{Path} \leftarrow (v.\text{Path}, e)$;
      $Q \leftarrow Q \cup \{v'\}$;
Dijkstra fails or returns v_{goal} *with a sequence of edges,* $(e_1, e_2, \cdots e_g)$ *leading from* v_{init} *to* v_{goal};
if Dijkstra fails **then**
  Return failure.;
else
  Return $\mathcal{B}_1$ stored on e_1 in $v.\text{Path}$;

C. Subgoal Planner

The task of the subgoal planner is to find a path through $\mathcal{G}$ between the vertex of $\mathcal{G}$ containing the projection of the

current configuration $\mathbf{q}_i$ and the vertex of $\mathcal{G}$ containing the project of the goal region, $\mathcal{Q}_{goal}$. The algorithm used by the subgoal planner is given by Algorithm 1.

First, the subgoal planner must determine the regions of the workspace decomposition to which $\mathbf{q}_i$ and $\mathcal{Q}_{goal}$ belong. This is done by projecting the body-attached frame of the robot at those configurations onto the two-dimensional map and searching through the regions stored in the vertices of $\mathcal{G}$. From the vertex representing the region that contains the initial configuration of the robot, v_{init}, Algorithm 1 performs a graph search. Algorithm 1 is a version of Dijkstra's algorithm [23] modified for the specific purpose of the subgoal planner.

The cost of the edges, the cost of transitioning between regions, plus the Euclidean distance between the centroids of the regions corresponding to a path through $\mathcal{G}$ is used to measure the cost of a given path through the graph. In Algorithm 1, the function Cost(v) refers to the cost labeled on the vertex, v. The function Cost(v, v', e) refers to the cost to get from vertex v to v' using the transition specified by e, the Euclidean distance between the centroids of v and v' plus a penalty stored on on e. The cost penalties on the edges are used to penalize certain transitions that are not desirable. For example, stepping onto a tall ledge costs more than a stepping down from a small ledge. Transitions between regions where the boundary is artificial, regions that were split to satisfy the convexity requirement, are given zero cost. More sophisticated cost functions could be used to measure the quality of paths through the graph but, in simulation, this simple one has given desirable results. In Algorithm 1, each vertice is augmented with a sequence of edges to traverse from v_{init} to v before being placed in the queue. This is referred to as v.Path. Once $v = v_{goal}$, the loop terminates and v stores a path from v_{init} to v_{goal}.

Once a path through $\mathcal{G}$ is found, $(e_1, e_2, \cdots e_l)$, the subgoal planner passes the subgoal region of the edge corresponding to the next region transition, $\mathcal{B}_1$ stored on e_1, to the local planner. The local planner attempts to turn the high level transition between vertices in $\mathcal{G}$, specified by e_1, into a sequence of motion primitives that allows the robot to move to the next region. If no path through the graph can be found, the hierarchical algorithm has exhausted all avenues through which to find a path and the algorithm returns failure.

If $\mathcal{Q}_{goal}$ projects onto a subset of a region in the decomposition, a final subgoal is added once the robot enters the region marked as v_{goal}. Similarly, if $\mathcal{Q}_{goal}$ projects onto portions of two or more regions in the decomposition, Algorithm 1 can be modified to search for the shortest path to one of the vertices whose region in the decomposition contain a portion of the projection of $\mathcal{Q}_{goal}$.

D. Local Planner

It is neither feasible nor desirable to attempt to exactly follow the global path, as it is chosen with minimal constraints on motion. The task of the local planner is to obtain a sequence of motion primitives that replaces $\mathcal{G}$ between successive subgoals. Starting at the robots current location, the goal of the local planner is to find a sequence of motion primitives such that after the final motion primitive is applied the robot is left in the subgoal region for the next subgoal and the robot is in an equilibrium state. Furthermore, the sequence of motion primitives should obey constraints on ordering and avoid causing the robot to collide with the environment.

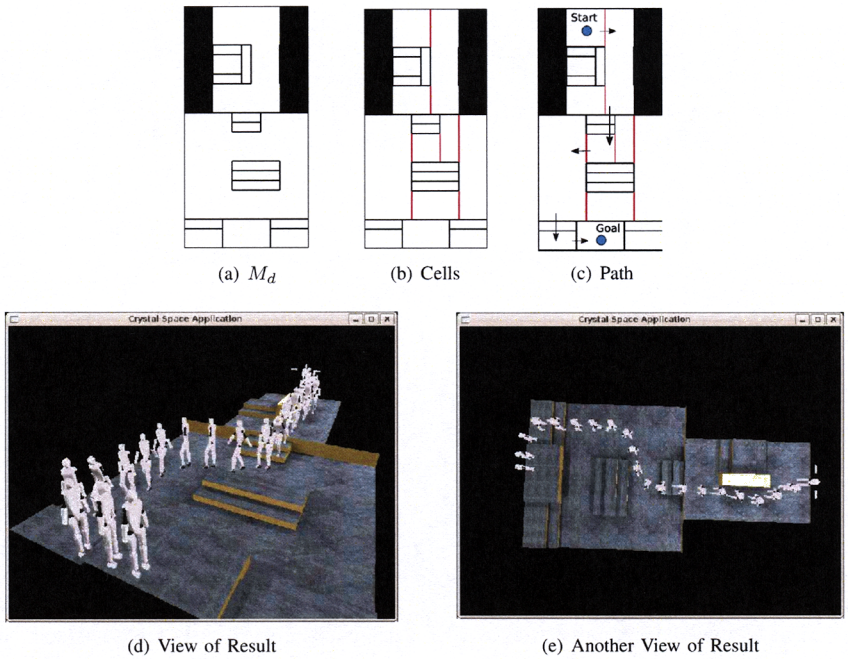
We use the same local planner as [1]. It is essentially a modified A* search that trims nodes from the search that violate the dynamic constraints of or artificial constraints imposed on the robot or that correspond to the robot colliding with an obstacle. The difference with the previous work is how the output of this component is treated. If the planner succeeds in finding a path from $\mathbf{q}_i$ to $\mathcal{B}_{i+1}$ and a sequence of motion primitives is returned, then the subgoal planner is given $f(\mathbf{q}_i, \gamma)$ as the initial condition. The subgoal planner then calculates a new sequence of subgoals that takes the robot to the goal. This continues until the robot reaches the final $\mathcal{B}_i$ in which all $\mathbf{q}$ have $\mathbf{x} \in \mathcal{B}_{goal}$.

If the local planner is unable to find a sequence of motion primitives to reach the subgoal region, this failure is reported to the global planner. The connectivity graph of the workspace decomposition, $\mathcal{G}$, is then augmented by removing the edge the local planner was attempting to traverse. This new connectivity graph is then passed to the subgoal planner and the subgoal planner attempts to replan a new route to the goal. This procedure continues until either the robot reaches the goal or no path remains in the connectivity graph. In the latter case, the hierarchical algorithm reports failure.

III. Simulation Results

To show that DHP is able to plan routes in some cases where GHP fails or gives an undesirable result, the subgoal and local planners of DHP were simulated in some of the same environments as GHP with the same simulated robot. The data structures $\mathcal{G}$ were constructed in a principled manner outside of the simulation environment. We found that the DHP was, in fact, able to improve upon the results of the GHP in some cases. Figures 1 (a), (b), and (c) show M_d, M_d after the trapezoidal decomposition, and the graph transitions chosen by DHP superimposed on M_d. The red lines in (b) indicate partitions inserted by the trapezoidal decomposition. The resulting plan is shown in Figure 1. The DHP result on this workspace shows the robot taking a route avoiding both sets of stairs crossed by the robot in the GHP result in the same workspace, improving on the result of GHP. In the simulation run used to generate Figure 1, the subgoal plans took an average of 0.502 ms and the local plans took an average of 250.3 ms. Note that we report average times as this corresponds to the average pause at subgoals if the robot is generating the local plans as needed. The plan required a sequence of 75 motion primitives to move the robot to the goal.

A second example environment is shown in Figure 2 (a) and (b) which shows M_d after the trapezoidal decomposition and the graph transitions chosen by DHP superimposed on M_d. For the same inputs GHP failed due to the global plan touching an



(a) M_d (b) Cells (c) Path

(d) View of Result (e) Another View of Result

Fig. 1. First Simulation Example

obstacle region in M_d, but not crossing it. The resulting DHP plan is shown in Figure 2 (c) and (d). This gives an example of a DHP success on a workspace and input where GHP fails to find any path. The subgoal plans took an average of 0.604 ms and the local plans took an average of 279.7 ms. This plan required a sequence of 69 motion primitives to reach the goal.

IV. Experimental Results

A twenty-four degree of freedom humanoid robot was developed to verify our motion planning method in a real environment. We implemented twenty motion primitives of the robot including forward step, sidestep, turning in place, angle step, step up and so on. Figure 3 (a), (b), and (c) shows the humanoid robot, an experimental environment and the trapezoidal decomposition of the workspace.

The snapshots of experimental results are shown in Figure 3 (d). A sequence of 105 motion primitives is required and it takes an average of 99 seconds to move the humanoid robot to the goal.

We found that individual execution of a motion primitive shows the displacement error and the direction deviation according to floor state and external disturbances. Also, we found that a long span of motion primitives can have disastrous effects on an open loop local plan.

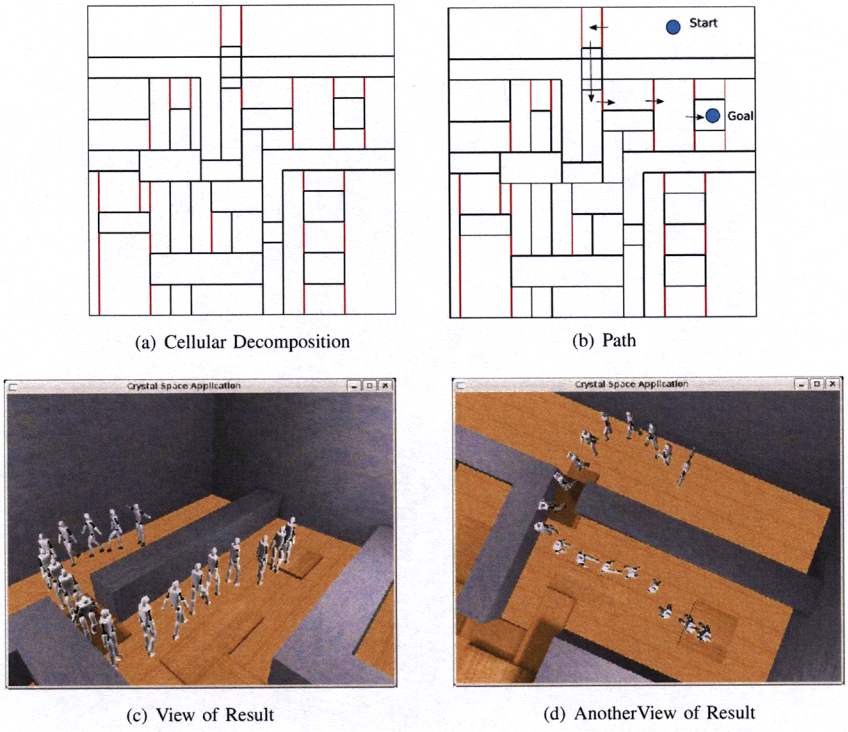
V. Conclusions and Future Work

In this paper, we discuss an extension and improvement of the GHP discussed in our previous paper [1]. We propose the Dijkstra Hierarchical Planner (DHP) which although it requires more preprocessing and computation as the algorithm runs, in some cases improves upon the failures of the GHP. We have shown simulation and experimental results for this algorithm.

A. Future Work

One of the biggest assumptions implicit in this work is the model of motion primitives. While an open loop local plan is ideal for character animation, it is less than desirable for actual bipedal robots. Small disturbances in the displacements individual executions of motion primitives can have disastrous effects over a long span of primitives concluding in stepping up onto or down off of a ledge. These deficiencies could arise due to unmodeled dynamics, external disturbances, modeling and measurement errors, and noise in the system. It will be important to develop methods to cope with some of these deficiencies of the model of the physical system.

One approach to deal with this is to take into account these deficiencies in the local planning phase of the algorithm. Guaranteeing bounds on the error or probabilistically ensuring success quantifies the risk to the robot hardware while allowing local plans to still be precomputed before each segment of the local path. Another approach is to integrate feedback into the local planning system. By having the robot replan its path after execution of each motion primitive (or every few motion primitives), the difference between the ideal execution and actual execution of previous motion primitives can be accounted for in future actions. This has been partially explored so far in simulation. A third approach may involve designing robust motion primitives that have guarantees on how far they might deviate from the nominal trajectory and



(a) Cellular Decomposition (b) Path

(c) View of Result (d) AnotherView of Result

Fig. 2. Second Simulation Example

tolerances on the initial shape from which they begin.

An important variable determining the quality of paths (or if paths can even be found) is the set of motion primitives. However, little work has been done to characterize what a desirable set of motion primitives would be with respect to the environment in which the robot is designed to operate. So far, the set of motion primitives has been chosen with respect to specific workspace features and on a trial and error basis. Perhaps sets of motion primitives could be characterized as deficient or redundant with respect to an environment. Another possibility would be to characterize how limiting a set of motion primitives is with respect the overall motions the robot is capable of performing. Analysis along these lines will be important for any robot employing the motion primitive concept as part of its movement strategy.

Finally, in order for a bipedal robot to be completely autonomous it must be able to navigate more than just a few workspaces. A truly autonomous robot using a motion primitives framework will need to be able to adapt its set of motion primitives to new environments. Combining high-level planning, such as the work presented in this paper, with systems that allow robots to adapt current motion primitives and learn new motion primitives will allow robots to function in the various environments we find all around us.

REFERENCES

[1] Y. Kim, S. Candido, and S. Hutchinson, "A workspace decomposition for hierarchical motion planning with humanoid robots," in *Proceedings of the IEEE International Conference on Advanced Robotics*, 2007.

[2] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA: MIT Press, 2005.

[3] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. Hoboken, NJ: John Wiley and Sons, 2006.

[4] K. Nagasaka, H. Inoue, and M. Inaba, "Dynamic walking pattern generation for a humanoid robot based on optimal gradient method," in *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics*, 1999, pp. 908–913.

[5] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, "Motion planning for humanoid robots," in *Proceedings of the International Symposium on Robotics Research*, 2003, pp. 365–374.

[6] K. Hauser, T. Bretl, K. Harada, and J. Latombe, "Using motion primitives in probabilistic sample-based planning for humanoid robots," in *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*, 2006.

[7] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha, "Fast proximity queries with swept sphere volumes," in *Proceedings of the IEEE Conference on Robotics and Automation*, 1999, pp. 3719–3726.

[8] J. Klosowski, M. Held, J. Mitchell, H. Sowizral, and K. Zikan, "Efficient collision detection using bounding volume hierarchies of k-dops," *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, no. 1, pp. 21–36, 1998.

[9] M. Choi, J. Lee, and S. Shin, "Planning biped locomotion using motion capture data and probabilistic roadmaps," *ACM Transactions on Graphics*, vol. 22, no. 2, pp. 182–203, 2003.

[10] D. Katić and M. Vukobratović, "Survey of intelligent control techniques for humanoid robots," *Journal of Intelligent and Robotic Systems*, vol. 37, no. 2, pp. 117–141, 2003.

[11] M. Cheng and C. Lin, "Genetic algorithm for control design of biped locomotion," *International Journal of Robotic Systems*, vol. 14, no. 5, pp. 365–373, 1997.

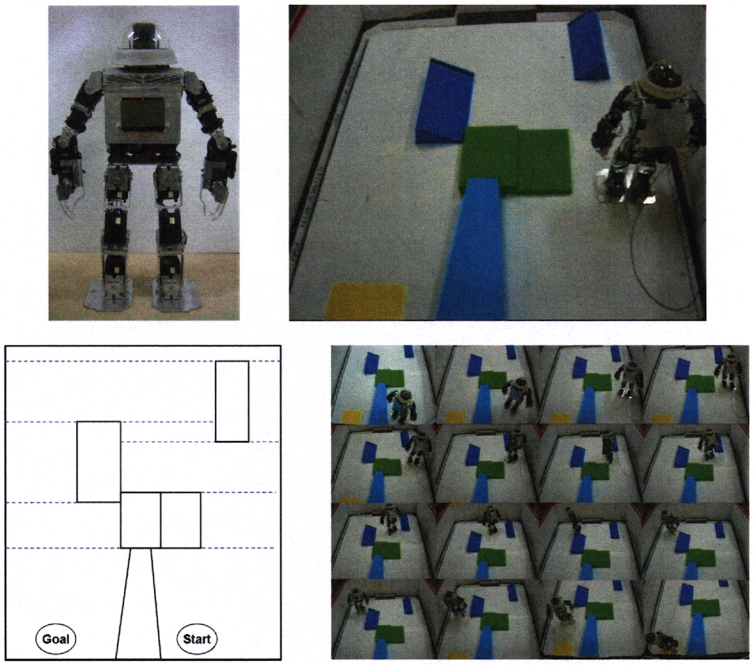


Fig. 3. Humanoid Robot, Experimental Environment, and Snapshots of Experimental Results

[12] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Yokoi, and H. Hirukawa, "A realtime pattern generator for biped walking," in *Proceedings of the IEEE Conference on Robotics and Automation*, 2002.
[13] J. Pratt, C. Chew, A. Torres, P. Dilworth, and G. Pratt, "Virtual model control: An intuitive approach for bipedal locomotion," *International Journal of Robotics Research*, vol. 20, no. 2, p. 129, 2001.
[14] E. Westervelt, J. Grizzle, and D. Koditschek, "Hybrid zero dynamics of planar biped walkers," *IEEE Transactions on Automatic Control*, vol. 48, no. 1, pp. 42–56, 2003.
[15] T. McGeer, "Passive dynamic walking," *International Journal of Robotics Research*, vol. 9, no. 2, pp. 62–82, 1990.
[16] M. Spong and F. Bullo, "Controlled symmetries and passive walking," *IEEE Transactions on Automatic Control*, vol. 50, no. 7, pp. 1025–1031, 2005.
[17] S. Collins, A. Ruina, R. Tedrake, and M. Wisse, "Efficient bipedal robots based on passive-dynamic walkers," *Science*, vol. 307, no. 5712, pp. 1082–1085, 2005.
[18] J. Chestnutt and J. Kuffner, "A tiered planning strategy for biped navigation," in *Proceedings of the IEEE/RAS Conference on Humanoid Robots*, 2004, pp. 422–436.
[19] T. Li, P. Chen, and P. Huang, "Motion planning for humanoid walking in a layered environment," in *Proceedings of the IEEE Conference on Robotics and Automation*, 2003, pp. 3421–3427.
[20] J. Gutmann, M. Fukuchi, and M. Fujita, "A modular architecture for humanoid robot navigation," in *Proceedings of the IEEE/RAS Conference on Humanoid Robots*, 2005, pp. 26–31.
[21] ——, "Real-time path planning for humanoid robot navigation," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2005, pp. 1232–1237.
[22] S. Candido, "Motion planning for bipedal walking robots," Master's thesis, University of Illinois at Urbana-Champaign, 2007.
[23] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA: MIT Press/McGraw-Hill, 2001.