A Single-Planner Approach to Multi-Modal Humanoid Mobility

Andrew Dornbush\textsuperscript{1}, Karthik Vijayakumar\textsuperscript{1}, Sameer Bardapurkar\textsuperscript{1}, Fahad Islam\textsuperscript{1}, Maxim Likhachev\textsuperscript{1}

\textbf{Abstract}—In this work, we present an approach to planning for humanoid mobility. Humanoid mobility is a challenging problem, as the configuration space for a humanoid robot is intractably large, especially if the robot is capable of performing many types of locomotion. For example, a humanoid robot may be able to perform such tasks as bipedal walking, crawling, and climbing. Our approach is to plan for all these tasks within a single search process. This allows the search to reason about all the capabilities of the robot at any point, and to derive the complete solution such that the plan is guaranteed to be feasible.

A key observation is that we often can roughly decompose a mobility task into a sequence of smaller tasks, and focus planning efforts to reason over much smaller search spaces. To this end, we leverage the results of a recently developed framework for planning with adaptive dimensionality, and incorporate the capabilities of available controllers directly into the planning process. The resulting planner can also be run in an interleaved fashion alongside execution so that time spent idle is much reduced.

\section{Introduction}

Recent years have shown much interest in developing robust humanoid robots that can operate in environments that are often unstructured, cluttered, and unpredictable compared to controlled industrial settings. Furthermore, the structure that does exist is intended primarily for people, and not designed with the robots in mind. Structures such as staircases, ladders, railings, complement people’s mobility. This leads us to design humanoids so that they possess capabilities similar to people such as the ability to walk, climb, and use surfaces such as handrails for support.

The need for all these capabilities provides a number of challenge problems for motion planning. The most pronounced problem is the inherent high dimensionality of the robot’s configuration space. To guarantee that a plan safely and efficiently accomplishes a given task may require reasoning about all of the joints of the robot and the relationship between the robot and the various objects in its environment. These constraints are expensive to evaluate.

Luckily, a complicated mobility task can often be broken down into a sequence of smaller tasks. For example, a task for a robot to move from one end of a facility to the other might include traversing large areas by walking, climbing staircases or ladders and, in situations where the environment is hazardous, crawling under fallen structures or over debris. Examples of such environments are shown in Figure 1.

Typically, current approaches solve this problem hierarchically: a top-level planner decomposes the complete task into smaller tasks and then runs a different planner, specialized for each task, in isolation. Once plans are computed for each task, the top-level planner figures out how the robot will transition from one task to the next. This can be done with yet another specialized planner, or prescribed behaviors. The results of the Darpa Robotics Challenge, as demonstrated in \cite{1}, \cite{2}, \cite{3}, show the ubiquity of specialized task planners and behaviors. This common approach has shown to be brittle, as each task planner is constrained to satisfy the requirements of the original task decomposition, and must satisfy strict endpoint constraints to ensure that the transitions between tasks are feasible. In the worst case, where the top-level planner has chosen an incorrect decomposition, one of the planners may be unable to generate a solution at all or a transition between tasks is infeasible.

The approach presented here builds upon the notion of adaptive dimensionality. Rather than always search through a high-dimensional state space, adaptive dimensionality automatically figures out what dimensions are relevant in each region of the state space. This is tremendously beneficial to planning for humanoid mobility as there is lots of redundant motion that makes up the various modes of locomotion available to them. This paper presents how adaptive dimensionality can be applied to humanoid mobility, describes an implementation that yields real-time execution by interleav-
ing planning and execution, and presents experimental results showing the practicality of the approach.

II. RELATED WORK

Much work in humanoid mobility planning is focused on solving specific sub-problems of mobility. Examples of navigation planning using footsteps are shown in [4], [5]. These techniques plan in a low-dimensional space representing feasible footstep actions. They may rely on a controller to produce feasible joint trajectories or the planner may generate these trajectories online to ensure footstep validity. Example whole-body planning techniques have been explored in [6], [7], [8]. These approaches are generally intended for object interaction tasks, and don’t consider incorporating locomotion. Some example techniques specifically for climbing ladders are presented in [9] and [10].

Relatively less work has been done for humanoid robots on adaptively reasoning about the relevant dimensionality of the problem during the search process. Some examples of adaptive reasoning include [11] and [12]. Both works decompose the robot into appropriate subsystems based on kinematics, increase the dimensionality. The first RRT-based approach adaptively adds subsystems as the search gets closer to the goal. The second, optimization-based approach, plans iteratively, incorporating more descendant subsystems until a valid path is found. These approaches both iteratively increase the dimensionality of the entire search space, whereas our approach only increases dimensionality of the search space in regions where high-dimensional planning is required.

III. PLANNING FRAMEWORK

Our application to multi-modal humanoid navigation targets the humanoid robot shown in Figure 3. The robot has 4 symmetric limbs, each with 7 degrees of freedom, and additional joints for reorienting the attached sensors. At the end of each limb, is a dual-purpose end effector designed with a flat surface, both for walking and support, and a hook for latching onto cylindrical-shaped objects in the environment, such as ladder rungs and handrails. In this work, we explicitly plan for all of the joint variables of the limbs. The pose of the robot provides an additional 6 degrees of freedom for its global position and orientation. Together these degrees of freedom define a 34-dimensional search space.

We are also provided with a set of specialized controllers for performing various locomotion tasks. Currently, the robot is equipped with controllers for bipedal walking, crawling, and climbing ladders. Additionally, we are able to directly control each of the joint actuators to execute raw full-body paths. While it is possible to plan paths consisting of only raw joint motion, we are able to leverage the existence of the controllers both to improve planning efficiency and generate plans that can be executed more robustly on the actual robot.

We represent the planning problem as a search over a finite, discrete search space. The search space consists of a discrete state space \( S \), and a set of transitions \( T = \{(s_i, s_j) \mid s_i, s_j \in S\} \). Each pair \((s_i, s_j) \in T\) represents a feasible transition between two states. Each transition is associated with a scalar cost, \( c(s_i, s_j) > 0 \). We use the notation \( \pi(s_i, s_j) \) to denote a path from state \( s_i \) to state \( s_j \), and \( \pi^*(s_i, s_j) \) to denote an optimal, least-cost, path. This search space defines a graph \( G \), with vertex set \( S \) and edge set \( T \). The goal of the planner is to find a path in \( G \) from a given start state \( s_s \) to a goal state \( s_g \in S_G \), where \( S_G \subset S \).

To improve the efficiency of the search through this high-dimensional space, and to ease integration of specialized controllers, our algorithm takes much of its inspiration from the Planning with Adaptive Dimensionality framework presented in [13] and [14]. The framework for planning with adaptive dimensionality makes the observation that, in many areas of the search space, it is often not necessary to reason about the high dimensionality of the state space, as many of the resulting paths have a low-dimensional structure.

Section IV will begin with a brief overview of the framework for planning with adaptive dimensionality. Section V will describe extensions to the planning with adaptive dimensionality framework to enable planning with multiple low-dimensional planning representations simultaneously. Section VI will describe the details of the search algorithm used during the first phase of a single search iteration, with an emphasis on the use of multi-heuristic search. Section VII will describe the details of the search algorithm used during the second phase of a single search iteration, with an emphasis on incorporating user demonstrations to accelerate planning similar transitions in the high-dimensional space. Section VIII will describe how the search can be run in a resumable fashion to enable interleaving of planning and execution. Section IX will present a brief overview of the control architecture on the robotic platform, and specifically how plans are delivered to the appropriate controllers during execution. Section X will list the results of sample runs of the planner on targeted test environments.

IV. PLANNING WITH ADAPTIVE DIMENSIONALITY

This section provides a brief overview of the framework for planning with adaptive dimensionality. For detailed analysis of the adaptive dimensionality framework and additional applications, see [13].

For a complete planning solution, a search often needs to reason over a high-dimensional state space. However, we expect that large portions of a complete plan will exhibit a low-dimensional structure. For example, part of a humanoid mobility task might include large segments of bipedal walking. In these scenarios, it suffices to plan only for the footstep locations, and we reserve planning in the high-dimensional space for verifying that each footstep is feasible. The portions of the plan requiring high-dimensional reasoning are infrequent compared to portions that can be solved in this manner.

The planning with adaptive dimensionality framework leverages this low-dimensional structure by iteratively constructing a hybrid search space, composed primarily of low-dimensional states and transitions, and introducing high-
A. Graph Structure

The adaptive dimensionality framework considers two state spaces: the original high-degree-of-freedom state space that represents valid configurations of the robot, and a projection of the original state space to a low-dimensional representation, respectively labeled $S^{hd}$ and $S^{ld}$. A many-to-one mapping defined by

$$\lambda : S^{hd} \rightarrow S^{ld}$$

represents the projection from the high-dimensional space to the low-dimensional space. The inverse, one-to-many, mapping defined by

$$\lambda^{-1}(s_{ld}) = \{s \in S^{hd} | \lambda(s) = s_{ld}\}$$

represents the projection from a state in the low-dimensional space back to a subset of states in the high-dimensional space.

Both the high-dimensional and low-dimensional space can have its own set of transitions, $T^{hd}$ and $T^{ld}$ respectively. However, to guarantee completeness and bounded suboptimality, the following constraint is required:

$$c(\pi^*(s_i, s_j)) \geq c(\pi^*(\lambda(s_i), \lambda(s_j)), \forall s_i, s_j \in S^{hd} \ (1)$$

That is, the cost of the optimal path between any two states in the high-dimensional space must be at least the cost of the optimal path between their projections in the low-dimensional space.

The notation $G^{hd}$ and $G^{ld}$ represent the corresponding high-dimensional and low-dimensional graphs defined as $(S^{hd}, T^{hd})$ and $(S^{ld}, T^{ld})$, respectively.

B. Search Algorithm

Rather than search for a path in the original high-dimensional search space, $G^{hd}$, the adaptive dimensionality search algorithm prefers to search as much as possible in the low-dimensional search space, $G^{ld}$. To accomplish this, the search iteratively constructs a new hybrid search space $G^{ad}$, composed of an adaptive state space $S^{ad}$, and transition set $T^{ad}$. This new search space is composed primarily of states and transitions from $G^{ld}$ and is expanded to include regions of states and transitions from $G^{hd}$ as necessary.

Initially, the adaptive search space $G^{ad}$ includes all of $G^{ld}$. When a region of high-dimensional states is introduced, $G^{ad}$ is updated so that low-dimensional states $s$ that fall within the high-dimensional region are replaced by their high-dimensional equivalents in $\lambda^{-1}(s)$.

To be able to search this hybrid space, we must define a transition set that includes transitions between states from $S^{ld}$ and $S^{hd}$. The transition set for the adaptive search space is defined as follows. For a state $s \in S^{ad}$,

- If $s \in S^{hd}$ then for all transitions $(s, s') \in T^{hd}$, if $s' \in S^{ad}$ then $(s, s') \in T^{ad}$ otherwise $(s, \lambda(s')) \in T^{ad}$
- If $s \in S^{ld}$ then for all transitions $(s, s') \in T^{ld}$, if $s' \in S^{ad}$ then $(s, s') \in T^{ad}$. Additionally, for all transitions

This transition set includes transitions between low- and high-dimensional states, and only includes transitions to states in the adaptive state space $S^{ad}$. Notice that expanding or adding a new high-dimensional produces a new instance of $G^{ad}$.

The adaptive search algorithm begins by finding a path, $\pi_{ad}$, from the start to the goal in the current instance of $G^{ad}$. This path is allowed to contain states of differing dimensionalities, and so may not be executable. If no path is found during this phase, then no path exists from the start to the goal, and the search terminates. To construct an executable path from $\pi_{ad}$, another search is conducted within a tunnel surrounding $\pi_{ad}$. We define a tunnel $\tau$ of radius $w$ around an adaptively-dimensional path $\pi_{ad}$ as follows: $\tau$ is a subgraph of $G^{hd}$. A high-dimensional state $s \in \tau$ if there exists a state $s_{ad} \in \pi_{ad}$ such that the distance from $\lambda(s)$ to $s_{ad}$ (or $\lambda(s_{ad})$ if $s_{ad} \in S^{hd}$) is no larger than $w$ for some predefined distance metric on $S^{ad}$. All transitions $(s, s') \in T^{hd}$ are included such that $s, s' \in \tau$.

If the search fails to find a path from the start to the goal within $\tau$, high-dimensional regions are introduced where the search became stuck, and the adaptive search begins a new iteration on a newly constructed instance of $G^{ad}$. See [13] for details on how to identify locations to place high-dimensional regions.

V. Adaptive Dimensionality with Multiple Low-Dimensional Representations

In the domain of humanoid mobility planning, several useful low-dimensional representations are available. For our application, the humanoid robot is expected to utilize the available controllers for optimized bipedal walking, crawling, and ladder climbing. Each of these controllers has a natural low-dimensional representation. For crawling, the controller requires a 4-dimensional pose, $(x, y, z, \theta)$, of the robot. For bipedal walking, the controller requires paths that specify the 4-dimensional pose, $(x, y, z, \theta)$, of each foot. Finally, for ladder climbing, the controller requires only the 6-dimensional pose, $(x, y, z, \alpha, \beta, \gamma)$, for each of the four end-effectors. To be able to plan solutions that incorporate all of these representations, we need the ability to combine into a single search space.
Our approach maintains the separation between the high-dimensional and low-dimensional search spaces and their ability to define their own transition sets. Given \( n \) low-dimensional representations, we define low-dimensional discrete state spaces \( S_1, S_2, \ldots, S_n \), and their corresponding transition sets, \( T_1, T_2, \ldots, T_n \).

The mappings from the high-dimensional space to each low-dimensional space remain largely unchanged as well. For the \( i \)’th low-dimensional representation, a mapping defined by
\[
\lambda_i : S^{hd} \rightarrow S^i
\]
represents the mapping from states in \( S^{hd} \) to states in \( S^i \). Correspondingly, the inverse functions
\[
\lambda_i^{-1}(s_i) = \{ s \in S^{hd} | \lambda(s) = s_i \}, \forall i \in 1..n
\]
represent the mapping from states in \( S^i \) to subsets of \( S^{hd} \).

Additionally, we define functions
\[
\lambda_{i,j}(s_i) = \{ s \in S_j | \exists s_{hd} \in \lambda^{-1}(s_i) | \lambda(s_{hd}) = s \}
\]
to represent the mappings between states of low-dimensional representations. These mappings may be one-to-one or one-to-many depending on the dimensionality of the target representation.

The construction of \( S^{ad} \), and its graph representation, \( G^{ad} \), follows from its construction in the adaptive dimensionality framework. The initial instance of \( G^{ad} \) is the union of the search spaces \( G^i = (S^i, T^i) \) for each of the low-dimensional representations. The transition set, \( T^{ad} \) is extended to include projections from the high-dimensional representation to each low-dimensional representation. Additionally, \( T^{ad} \) also contains transitions that allow the search to effectively switch between low-dimensional representations. The complete transition set is defined as follows. For a state \( s \in S^{ad} \),

- If \( s \in S^{hd} \) then, for all high-dimensional transitions \((s, s') \in T^{hd} \), if \( s' \in S^{ad} \) then \((s, s') \in T^{ad} \), otherwise \((s, \lambda_i(s')) \in T^{ad} \) for each low-dimensional representation \( S^i \).
- If \( s \in S^i \) then, for all low-dimensional transitions \((s, s') \in T^i \), if \( s' \in S^{ad} \) then \((s, s') \in T^{ad} \). Additionally, for each representation \( S^j \in \{S^k | k \in 1..n, k \neq i \} \cup S^{hd} \), for all transitions \((s_j, s'_j) \in T^j \), where \( s_j \in \lambda_{i,j}(s) \), if \( s'_j \in S^{ad} \) then \((s, s'_j) \in T^{ad} \).

Thus far, we have only described how to incorporate multiple low-dimensional representations into the adaptive dimensionality framework. This indeed speeds up the search for finding a high-dimensional path, but we also desire to explicitly reason about the controller capabilities, to avoid high-dimensional planning wherever possible. Recall that during the second phase of each search iteration, the algorithm searches for a completely high-dimensional path, within the tunnel \( \tau \). To relieve the search of needing to perform high-dimensional planning, we extend the transition set of the high-dimensional representation to include all of the transitions that correspond to actions from the low-dimensional representations that are directly executable by an available controller.
that traverse up and down staircases, while the available
ladder representations are always directly executable by
own. representations, and may not produce successor states of their
from bipedal to crawling and from crawling to ladder. These
the ladder representation. We designed similar transitions
ladder rung, so that the resulting projection is relevant to
another state representation. For example, a successor state
transitions, whose resulting successor states project cleanly to
representations. To aid the planner, we append special tran-
sitions, depending on whether a particular heuristic is enabled
for that representation. This splitting of heuristics between
representations allows the search to explore simultaneously
across representations. The high-dimensional representation
has its own anchor and set of inadmissible heuristics, as
in vanilla MHA*, since the searches for the two adaptive
planning phases are independent. This is defined in the
InitializeHeuristicLists() method in lines 1 – 6 in the
algorithm.

2) Successor Generation: In MHA*, whenever a state is
expanded, its successors are inserted into all inadmissible
heuristic queues that are available to the search provided
it has not been expanded from either the anchor or any of
the inadmissible searches. This enables MHA* to effectively
Algorithm 1 MR-MHA*

1: procedure INITIALIZEHEURISTICLIST
2:     for d = 1 to max_dim do
3:         heuristicList[d].anchor = h_0
4:     for i = 1 to n do
5:         if h_i is enabled for dim d then
6:             heuristicList[d].inadm.append(h_i)
7:     return
8:     procedure KEYS(s, i)
9:     procedure INSERT(OPEN, s)
10:     procedure EXPANDS()
11:     procedure MR-MHA*
12:     g(s_goal) = ∞
13:     OPEN = null
14:     InitializeHeuristicList()
15:     for i = 0 to m do
16:         if inHeuristicList[s_start → dim].anchor then
17:             insert s_start into OPEN, with key(s_start, 0) as priority
18:         while OPEN not empty do
19:             for i = 1 to m do
20:                 if OPEN, MinKey() ≤ w_2 × OPEN_0, MinKey() then
21:                     if g(s_goal) ≤ OPEN, MinKey() then
22:                         terminate and return path pointed by bp(s_goal)
23:                     expand(s)
24:                     else
25:                         if g(s_goal) ≤ OPEN_0, MinKey() then
26:                             terminate and return path pointed by bp(s_goal)
27:                         expand(s)
28:     end

share paths between different heuristics that can help the search in different parts of the state-space.

However, when a state is expanded in MR-MHA*, the representation dimension of each successor is extracted, and accordingly are only inserted in heuristic queues which are available to that particular representation as defined in the heuristic lists. This allows MR-MHA* to effectively share paths within each representation without unnecessarily expanding states from irrelevant heuristic queues. This is shown in line 19 in the algorithm.

B. Implementation Details

Here we summarize the heuristics used for each state-space representation in our framework. As mentioned before, we perform full-body planning only in parts of the state-space where a controller is not executable. In our experimental setup, this only corresponds to the humanoid stepping on stairs. Hence, the heuristics we list for the full-body representation aid the search in humanoid stepping motion only.

A common approach to designing a heuristic function for a given state space is to first project it to a low-dimensional representation, and use the result of a search in the low-dimensional space as the heuristic value for a corresponding high-dimensional state. We designed several 3D grid searches with cost functions tuned for producing meaningful heuristic values, and computed those values online using a Dijkstra search from the goal to the state whose heuristic we are computing.

Bipedal Representation

1) Sum of grid search distances from both feet to the goal
2) Sum of grid search distances from both feet to the goal, with penalties for stepping close to the edges of staircase steps, to encourage alignment with the staircase direction
3) Sum of grid search distances from both feet to the goal, with penalties for using the ladder, to encourage staircase usage

Ladder Representation

1) A constant 0, to expand states in order of increasing g-values

Crawl Representation

1) Grid search distance from the COM of the robot to the goal
2) Difference in heading between the root of the robot and the feet of the robot (Fig.4)
3) Euclidean distance between COM of the current state and that of the target state
4) Euclidean distance between active feet of the current state and that of the target state (Fig.4)
5) Curve to provide guidance for stepping feet movement during the search (Fig.4)
6) Remaining number of steps on the lower dimensional path that need to be tracked

Fig. 4: Full body representation heuristics.
representation, which corresponds to the motion for the robot that mounts the ladder while standing in front it.

To speed up high-dimensional planning in these scenarios, we apply Experience Graphs, outlined in [16]. Experience Graphs, or E-Graphs, provide a way to incorporate prior experience, in our case from user demonstrations, to guide the search towards reusing paths with a good chance of leading to the goal. In our domain, these transitions are often similar and we can leverage previous solutions to generate modified transitions quickly.

A. Heuristic Computation

As discussed in [16], the E-Graphs approach defines the heuristic value for a state \( s_0 \) as

\[
h^E(s_0) = \min_{\pi} \sum_{i=0}^{N-1} \min \{ e^E h^G(s_i, s_{i+1}), c^E(s_i, s_{i+1}) \}
\]

where \( \pi \) is a path \( \langle s_0, \ldots, s_{N-1} \rangle \), \( s_{N-1} = s_{\text{goal}} \), and \( e^E \) is a scalar parameter \( \geq 1 \), which determines the degree to which the search is encouraged to reuse prior experience. The paths \( \pi \) consist of edges between any two states \( s_i \) and \( s_{i+1} \) with cost equal to the underlying heuristic \( h^G \), inflated by \( e^E \), and edges from the E-Graph with cost equal to the actual cost of the transition.

The underlying heuristic used with the E-Graph heuristic is computed by solving a lower-dimensional problem using dynamic programming. Similar to the heuristics used during the first phase of the search, we solve several 3D \( (x, y, z) \) Dijkstra searches, each from the goal position of one of the end effectors. The E-Graph transitions, as well as obstacles, are incorporated directly into these Dijkstra expansions to encourage obstacle avoidance and use of prior experience. When computing the heuristic value for a given full-body state, we sum up the contributions from these Dijkstra searches, using the current end effector positions.

B. Snap Motions

In many scenarios, the search can encounter difficulty using the path demonstrations, even if the path is representative of the expected path the robot should take. For example, consider a scenario where a user has demonstrated how the robot should mount a ladder. If, during planning, the search considers a state where the feet are slightly offset from the demonstration, it will still have to reason about the motion required to adjust the feet so that the demonstration can be reused. To alleviate this problem, we combine the ideas from [16] and [17] to create a set of adaptive motion primitives that reach partial states on the E-Graph. For example in the aforementioned scenario, the adaptive motion primitive will attempt to match partial waypoints for the arms and the torso, rather than trying to adjust the feet to match the demonstration completely.

IX. INTERLEASED PLANNING AND EXECUTION

Owing to the complexity of the planning tasks that we are addressing, the typical planning times to plan a path all the way from the start to the goal are significantly high. The key idea here is that instead of waiting for the planner to generate the entire executable path, we will interleave path planning and execution. The planner will return partial plan as it runs while the controller will start executing these plans on the robot in parallel. This idea has been widely studied by the real-time family of heuristic search algorithms. For the humanoid domain this approach provides significant speed ups in the overall planning and execution time because the path execution is generally slow.

Within the underlying framework of planning with adaptive dimensionality, we only employ the interleaving scheme in the tracking phase, because the tracking phase returns a path that is executable by the robot.

| Algorithm 2 Interleaved Planning and Execution |
|-----------------------------------------------|
| 1: Inputs: | lookahead |
| 2: while tracking time \( \neq \) lookahead do |
| 3: Run Tracking |
| 4: Reconstruct partial path |
| 5: Send partial path to the controller for execution |
| 6: Reset start state with the tail of the partial path |
| 7: Loop |

X. CONTROLLER FRAMEWORK

The path returned by the planner contains both transitions that correspond to executing an available controller, e.g. walking, crawling, climbing, and those that correspond to raw full-body joint motion. As described in Section 5 the low-dimensional transitions are directly executable by a controller. It is then the responsibility of the controller to compute full-body joint trajectories that robustly execute the desired motion. Each individual controller only accepts paths of their respective waypoint type. To interface the controllers with the planner, we developed a meta-controller for dispatching segments of the hybrid path to the correct
controller. After a hybrid path is received from planner, the meta-controller divides it into the individual segments of the same waypoint type and dispatches them sequentially to the corresponding controllers. After a segment is executed successfully, the controller signals the meta-controller to proceed. The meta-controller then dispatches the next segment to the corresponding controller, until the path is completed. The meta-controller is also responsible for updating the path, as additional waypoints are received during interleaved planning and execution.

### XI. Experimental Analysis

To demonstrate the effectiveness of the multi-heuristic adaptive planning approach, we tested the planner’s ability to plan paths in the sample environment from Figure 1c. The tests consisted of running the planner from numerous start locations, evenly distributed across (x, y) locations in the environment and from different start headings, to goal locations on the mid- and top-level platforms. In all cases, the planner was given 80 s to find a low-dimensional path, and 180 s to find the high-dimensional path. The results are shown in Table I.

| goal | success % | mean time (s) |
|------|-----------|---------------|
| top  | 89.6      | 57.5          |
| mid  | 89.6      | 65.7          |

**TABLE I**

The table lists the results across 231 different start locations, for each goal. Success rates are shown for both the low-dimensional phase of the search, and for the high-dimensional of the search. The success rates for the second “tracking” phase, are normalized with respect to the success of the first phase, thus the overall success rates for the planner are 51.52% and 58.87%, respectively for the two goals.

The columns containing planning times are mean times for the two phases of the search individually. The total average planning times are 101.3 and 91.3 seconds, respectively. Note that, outside of experiments, the planning is interleaved with execution of the path on the robot, so the time spent idle is only limited by the time taken by the search during the low-dimensional planning phase, plus a small lookahead for the high-dimensional search. The same is also true for the success rate, where the search is only limited by the speed of execution.

### XII. Conclusion

In this work, we have presented an approach to planning for multi-modal humanoid mobility using a single search algorithm. This approach is able to simultaneously reason about all the capabilities of the robot, incorporate the capabilities of available controllers, and automatically discover the transitions for switching between modes of locomotion. The resulting planner brings together planning with an adaptively-dimensional search space, using multiple heuristics, and incorporating user demonstrations to concentrate search efforts where they’re most needed. In future work we hope to incorporate planning for more complex interaction with the environment and address the robustness of scenarios requiring high-dimensional planning.

### REFERENCES

[1] S. J. Yi, S. McGill, L. Vadakethadu, Q. He, I. Ha, J. Han, H. Song, M. Rouleau, D. Hong, and D. D. Lee, “Thor-op humanoid robot for darpa robotics challenge trials 2013,” in 2014 11th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), Nov 2014, pp. 359–363.

[2] C. G. Atkeson, B. P. W. Babu, N. Banerjee, D. Berenson, C. P. Bove, X. Cui, M. DeDonato, R. Du, S. Feng, P. Franklin, M. Gennert, J. P. Graff, P. He, A. Jaeger, J. Kim, K. Knodeler, L. Li, C. Liu, X. Long, T. Padir, F. Polido, G. G. Tighie, and X. Xinjiefu, “No falls, no resets: Reliable humanoid behavior in the darpa robotics challenge,” in 2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids), Nov 2015, pp. 623–630.

[3] Y. Zhang, J. Luo, K. Hauser, H. A. Park, M. Paldhe, C. S. G. Lee, R. Ellenberg, B. Kilien, P. Oh, J. H. Oh, J. Lee, and I. Kim, “Motion planning and control of ladder climbing on drc-hubo for darpa robotics challenge,” in 2014 IEEE International Conference on Robotics and Automation (ICRA), May 2014, pp. 2086–2086.

[4] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, “Motion planning for humanoid robots,” Robotics Research, pp. 365–374, 2005.

[5] A. Hornung, A. Dornbush, M. Likhachev, and M. Bennewitz, “Anytime search-based footstep planning with suboptimality bounds,” in Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on. IEEE, 2012, pp. 674–679.

[6] F. Burget, A. Hornung, and M. Bennewitz, “Whole-body motion planning for manipulation of articulated objects,” in 2013 IEEE International Conference on Robotics and Automation, May 2013, pp. 1656–1662.

[7] A. Athar, A. M. Zafar, R. Asif, A. A. Khan, F. Islam, O. Hasan, et al., “Whole-body motion planning for humanoid robots with heuristic search,” in Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on. IEEE, 2016, pp. 4720–4727.

[8] M. Kanazawa, S. Nozawa, Y. Kakiuchi, Y. Kanemoto, M. Kuroda, K. Okada, M. Inaba, and T. Yoshiike, “Robust vertical ladder climbing and transitioning between ladder and catwalk for humanoid robots,” in 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Sep 2015, pp. 2202–2209.

[9] N. Vahrenkamp, C. Scheurer, T. Asfour, J. Kuffner, and R. Dillmann, “Adaptive motion planning for humanoid robots,” in 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, Sept 2008, pp. 2127–2132.

[10] C. Park, J. Pan, and D. Manocha, “High-dof robots in dynamic environments using incremental trajectory optimization,” International Journal of Humanoid Robotics, vol. 11, no. 02, p. 1441001, 2014.

[11] K. Gochev, B. Cohen, J. Butzke, A. Safonova, and M. Likhachev, “Path planning with adaptive dimensionality,” in Fourth annual symposium on combinatorial search, 2011.

[12] K. Gochev, A. Safonova, and M. Likhachev, “Planning with adaptive dimensionality for mobile manipulation,” in Robotics and Automation (ICRA), 2012 IEEE International Conference on. IEEE, 2012, pp. 2944–2951.

[13] S. Aine, S. Swaminathan, V. Narayanan, V. Hwang, and M. Likhachev, “Multi-heuristic a,” The International Journal of Robotics Research, vol. 35, no. 1-3, pp. 224–243, 2016.

[14] M. Phillips, B. J. Cohen, S. Chitta, and M. Likhachev, “E-graphs: Bootstrapping planning with experience graphs.” in Robotics: Science and Systems, vol. 5, no. 1. 2012.

[15] B. Cohen, G. Subramanian, S. Chitta, and M. Likhachev, “Planning for manipulation with adaptive motion primitives.” in Robotics and Automation (ICRA), 2011 IEEE International Conference on, 2011.